Design a Simple and High Performance Switch Using a Two-stage Architecture

Chih-Ying Tu, Cheng-Shang Chang, Duan-Shin Lee, and Ching-Te Chiu
Institute of Communications Engineering
National Tsing Hua University
Hsinchu 300, Taiwan, R.O.C.
Email: cytu@gibbs.ee.nthu.edu.tw
cschang@ee.nthu.edu.tw
lds@cs.nthu.edu.tw
ctchiu@cs.nthu.edu.tw

Abstract*— Recently, there is tremendous interest in the research of two-stage switches. Unlike input-buffered switches, two-stage switches do not need to find matchings between inputs and outputs. As such, they are much easier to scale and much simpler to implement. However, two-stage switches usually suffer from the out-of-sequence problem. Though there are several methods proposed in the literature to solve such a problem, these proposed methods require either complex scheduling or additional hardware, which defeats the purpose of design simplicity. To design a simple and high performance switch using the two-stage architecture, we address three buffer design problems in this paper: re-sequencing buffers, central buffers and input buffers. We show that the size of the re-sequencing buffer needs to be proportional to the size of the central buffer to ensure that no packets are lost due to re-sequencing. Via simulations, we find that a moderate size of central buffer yields good throughput when traffic is not bursty. However, when the traffic is bursty, one needs to address the head-of-line blocking (HOL) problem at the input. We also find that using the round-robin service policy for multiple virtual output queues (VOQ) at inputs may exhibit a catastrophic phenomenon, called a non-ergodic mode. When a switch is trapped in a non-ergodic mode, its throughput is sharply reduced. To solve such a problem in input buffers, we show that one may introduce "randomness" into a switch to jump out of a non-ergodic mode.

keywords: two-stage switches, re-sequencing, input-buffered switches, non-ergodic mode, iSLIP

This research is supported in part by the National Science Council, Taiwan, R.O.C., under Contract NSC 92-2213-E-007-076, and the program for promoting academic excellence of universities NSC 93-2752-E-007-002-PAE.

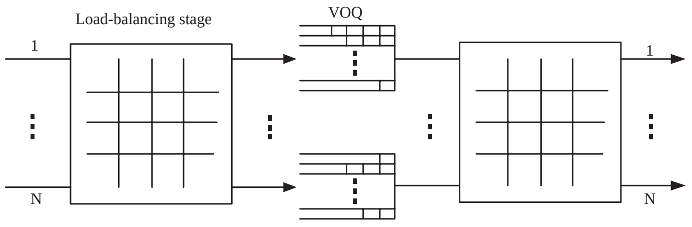

Fig. 1. The two-stage switch in [3]

I. Introduction

Recently, there is tremendous interest in the research of the two-stage switches (see e.g., [3], [4], [10], [11], [9], [7]). Unlike most input-buffered switches in the literature (e.g., PIM in [1], wave front arbitration in [16], SLIP in [13] and DRRM in [12]), two-stage switches do not need to find matchings between inputs and outputs. As such, the communication overhead and the computation overhead incurred by finding matchings in input-buffered switches could be minimized in two-stage switches.

The original design of the two-stage switch architecture in [3] consists of two crossbar switch fabrics with parallel central buffers between them (see Figure 1). The objective of the first stage is to perform load balancing. The first stage is a unbuffered crossbar switch with periodic connection patterns generated from a one cycle permutation matrix. The period of the connection patterns is equal to the number of input/output ports and every input-output pair is connected exactly once in a period. By so doing, packets that arrives at the first stage are distributed *evenly* to the central parallel buffers between the two switch fabrics. This makes the traffic coming to the second stage uniform.

The objective of the second stage is to perform switching. As the traffic coming to the second stage is uniform, switching can be easily done by time division multiplexing. As such, one can use the same connection patterns in the first stage for the second stage.

It is shown in [3] that such a two-stage switch has the following advantages:

(i) Scalability: the connection patterns in both switch fabrics are deterministic and periodic. There is no need to find matchings.

(ii) Low hardware complexity: only two crossbar switch fabrics and buffers between them are required. Neither internal speedup nor rate estimation is needed in the switch. Moreover, as one only needs to implement a set of connection patterns generated by a one cycle permutation matrix, there is no need to implement a full size crossbar. In fact, a Banyan type of multi-stage connecting network is enough (see e.g., [5]).

(iii) 100% throughput: under a mild technical condition on the input traffic, the two-stage switch in [3] achieves 100% throughput as an output-buffered switch for both unicast and multicast traffic with fan-out splitting.

(iv) Low average delay in heavy load and bursty traffic: when input traffic is bursty, load balancing is very effective in reducing delay, and the average delay of the two-stage switch is proven to converge to that of an output-buffered switch under heavy load.

(v) Efficient buffer usage: when both the two-stage switch and the corresponding output-buffered switch are allocated with the same finite amount of buffer at each port, the packet loss probability in the two-stage switch is much smaller than that in an output-buffered switch when the buffer is large.

One of the problems of the two-stage switch in [3] is that packets may be out of sequence. If the size of the parallel buffers is infinite, then the re-sequencing delay cannot be bounded in [3]. To solve the out-of-sequence problem, several methods have been proposed. The most common method is to schedule packets in a careful manner so that re-sequencing delay can be bounded [4], [10]. However, such an approach is at the cost of adding complicated hardware and computation overhead, which defeats the purpose of simplicity. The second method is to use the rate information to control incoming traffic entering the switch [6]. This is based on the assumption that the rate information is known. It is also difficult to adapt to traffic fluctuation. The third method, known as the mailbox switch in [5], uses a set of symmetric connection patterns to set up a feedback path from the central buffer to an input/output port. By feeding back the information of packet departure time, an input port can compute the waiting time needed for the next packet to depart in order. The mailbox approach does not have non-scalable communication and computation overheads, but its cost is the reduction of the throughput. By allowing limited re-sequencing delay, packets can be re-packed into mailboxes and it is shown in [5] that the throughput of a mailbox switch with limited re-sequencing delay can achieve 95% throughput.

Though the mailbox switch performs reasonably well without non-scalable communication and computation overheads, it is natural to ask the question whether one can build a simpler switch that has comparable performance to the mailbox switch. In this paper, we will propose a switch architecture that only requires to feedback one bit of information, i.e., whether a packet is successfully transmitted to a central buffer or not. There is no need for further communication and computation. We will do this by addressing three design problems: re-sequencing buffers in Section III, central buffers in Section IV, and input buffers in Section V. We show that the size of the re-sequencing buffer needs to be proportional to the size of the central buffer to ensure that no packets are lost due to re-sequencing. Via simulations, we find that a moderate size of central buffer yields good throughput when traffic is not bursty. However, when the traffic is bursty, one needs to address the head-of-line blocking (HOL) problem at the input. We also find that using the round-robin service policy for multiple virtual output queues (VOQ) at inputs may exhibit a catastrophic phenomenon, called a non-ergodic mode. When a switch is trapped in a non-ergodic mode, its throughput is sharply reduced. To solve such a problem in input buffers, we show that one may introduce "randomness" into a switch to jump out of a non-ergodic mode.

II. The Switch Architecture

In this paper, we assume that packets are of the same size. Also, time is slotted so that a packet can be transmitted within a time slot. In Figure 2, we propose a simple $N \times N$ two-stage switch. As in the original two-stage switch in [3], the switch consists of two crossbar switch fabrics. There are N *finite* central buffers between the two switch fabrics and N re-sequencing buffers in the N output ports. As to input buffers, the virtual Output Queue technique (VOQ) is adopted for each input port.

Both switch fabrics are running a series of periodic connection patterns and they produce the same connection pattern in every time slot. The period is equal to N.

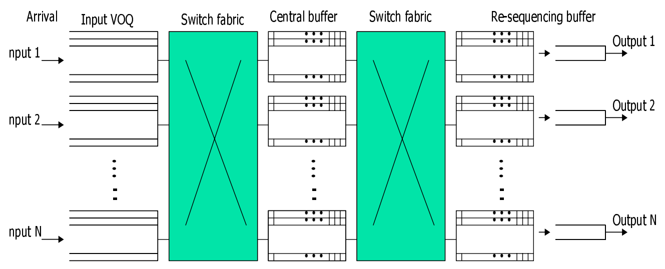

Fig. 2. The switch architecture

During the t^{th} time slot, the i^{th} input port is connected to the j^{th} output port if

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

As in the mailbox switch [5], this kind of connection patterns are symmetric and imply the following features.

- In every time slot, if the i^{th} input port is connected to the j^{th} output port, then the j^{th} input port is also connected to the i^{th} output port.
- The i^{th} input port is connected to the 1^{st} output at time i, the 2^{nd} output at time $i + 1$,…, the N^{th} output at time $i + N - 1$, the 1^{st} output at time $i + N$,…. That is to say, a specific input port is connected to one of the N output ports in the round-robin fashion.

Furthermore, due to the identical connection patterns of the two switch fabrics, the i^{th} input port is connected to the j^{th} central buffer which is connected to the i^{th} output port at the t^{th} time slot. As the connection patterns is periodic, each input/output port is connected to the each of the N central buffers exactly once in every N time slots.

One can take advantage of the symmetric connection patterns by folding the two switch fabrics into one. By so doing, we can save the cost of hardware implementation. However, the folded switch fabric needs to be accelerated twice the speed of the non-folded one.

Most importantly, we assume that both the input port and the output port are built in a line card. As such, the symmetric connection patterns set up a feedback path from the central buffer to the line card. This feedback path is used for notifying an input port whether a HOL packet is accepted by the connected central buffer or not.

The original two-stage switch in [3] needs infinite central buffers to achieve 100% throughput. **The question is then how one determines the size of central buffers to achieve high throughput.** Here we assume that there are N *finite* central buffers (indexed from 1 to N) between the two switch fabrics. Each central buffer have N bins (indexed from 1 to N), each bin is a First In First Out (FIFO) queue with F cells (indexed from 1 to F), and each cell has the capacity for storing exactly one packet. Hence the size of a central buffer is equal to NF. The packets destined for the i^{th} output port are stored in the cells of the i^{th} bin of a central buffer. We will show how the parameter F affects the throughput in Section IV.

Since the central buffer is finite, the bins of a central buffer may not have a empty cell to hold a new packet. When this happens, the packet is rejected and it needs to be queued at the input port. Therefore, input buffers are needed for the input ports. As such, there is the notorious head-of-line (HOL) blocking problem. **The question is then how one builds the input buffers to achieve high throughput.** In Section V, we will show a very interesting phenomenon. If one uses the simple round-robin scheduling in multiple VOQs at each input port, the switch might eventually get into a “non-ergodic” mode and the throughput is sharply reduced in such a mode. To avoid getting trapped in a non-ergodic model, we will propose several tentative solutions by introducing “randomness” into the switch.

As in the two-stage switch in [3], there is the out-of-sequence problem. To solve the problem, one needs to add a re-sequencing buffer. **The question is then how to build a *finite* re-sequencing buffer so that no packets are lost due to re-sequencing.** We will show in Section III that a re-sequencing buffer with size $N \times 2NF$ at each output port is enough to solve the problem.

III. First Design Problem: Re-sequencing Buffers

In this section, we propose a re-sequencing scheme so that no packets are lost due to re-sequencing. We show that a re-sequencing buffer with size $N \times 2NF$ at each output port is enough to solve the re-sequencing problem.

A. Maximum re-sequencing delay

First, we show the worst case that leads to the maximum re-sequencing delay. We define flow (i, j) as a sequence of packets that arrive at the i^{th} input port and are destined for the j^{th} output port. The out-of-sequence problem may occur while packets pass through the central buffer. We show the worst case in the Figure 3. Suppose that at time t, there are two consecutive packets (called packet k and packet $k + 1$) destined for output j at the i^{th} input port. ¿From the connection patterns specified in Section II, the i^{th} input is connected to the $h(i, t)^{th}$ central buffer. Suppose that packet k is placed in the last cell (i.e., the F^{th} cell) of the j^{th} bin of the $h(i, t)^{th}$ central buffer at time t. At time $t + 1$, the

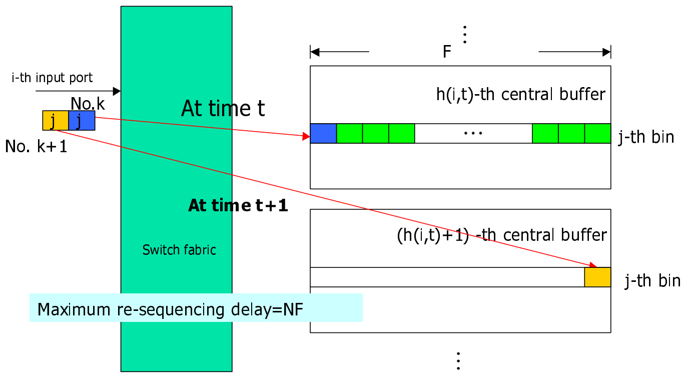

Fig. 3. The worst case of the re-sequencing delay

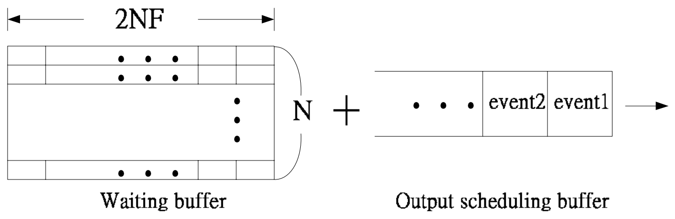

Fig. 4. The architecture of the re-sequencing buffer

i^{th} input is connected to the $h(i, t+1)^{th}$ central buffer. Now suppose that packet $k+1$ is placed in the first cell of the j^{th} bin of the $h(i, t+1)^{th}$ central buffer at time $t+1$. As the connection patters are deterministic and periodic, it takes N time slots to advance one position in a central buffer. It is easy to see that packet $k+1$ will be sent to output j before packet k and the worst case re-sequencing delay is bounded by NF time slots.

B. *The architecture of re-sequencing buffers*

Suppose that the bin size of each central buffer is F. In this section we propose a scheme that solves the out-of-sequence problem. In Figure 4, we show the architecture of the re-sequencing buffer. The re-sequencing buffer includes two parts. One is the waiting buffer and the other is the output scheduling buffer. The waiting buffer consists of N waiting queues (indexed from 1 to N) for N different flows. Each waiting queue has $2NF$ cells (indexed from 1 to $2NF$) for storing the packets from the same flow. The output scheduling buffer maintains an event list that schedules re-sequenced packets to depart from the output port.

Now we describe how the re-sequencing scheme works. Suppose a flow (i, j) packet with sequence number SN arrives at the re-sequencing buffer at the j^{th} output. Then the packet will be placed in the $[((SN - 1) \bmod (2NF)) + 1]^{th}$ cell of the i^{th} waiting queue of the waiting buffer. Each waiting queue keeps a *pointer* that points to the position of the next expected packet. Clearly, the position pointed by a pointer is empty. In the beginning, all pointers point to the first cells of their corresponding waiting queues. If the arriving packet is placed to the empty cell pointed by the pointer, then it will trigger an event for output scheduling buffer. Otherwise, nothing needs to be done after the packet is placed in the waiting buffer.

A triggered event consists of the following steps: the pointer is advanced to the next empty cell (wrapped around at the end of the waiting queue). Now the packets placed between the position pointed by the previous pointer and the position immediately before the current pointer are in sequence and they are eligible for transmission. As such, an event that contains the information of these packets (including the cell indices and the index of the waiting buffer) is added to the end of the event list for output scheduling buffer.

As long as the event list for output scheduling buffer is not empty, a packet is transmitted in every time slot (in the order of the sequence number) from the first event in the event list. When all the packets in the first event are transmitted, the event is removed from the event list.

By using the fact that the worst case re-sequencing delay (in the waiting buffer) is bounded by NF, we derive the following result for our re-sequencing scheme.

Theorem III.1 *For the re-sequencing scheme described in this section, there is no lost packet due to re-sequencing. Moreover, the number of packets stored in the re-sequencing buffer is bounded above by NF and the delay in the re-sequencing buffer is bounded above by $2NF$.*

The proof of Theorem III.1 will be given in the appendix of this paper.

IV. Second Design Problem: Central Buffers

It is known (see e.g., in [3]) that 100% throughput can be achieved if the size of the central buffer is infinite. However, as described in Theorem III.1, the size of the re-sequencing buffer is proportional to the size of the central buffer. If the center buffer is infinite, then there is no bound for re-sequencing delay. To limit the implementation complexity of the re-sequencing buffer, one has to consider a *finite* central buffer. On the other hand, if we choose a very small center buffer, then it is quite likely that packets will be blocked at the central buffer. This leads to very low throughput for the switch. **In short, if we increase the size of central buffer,**

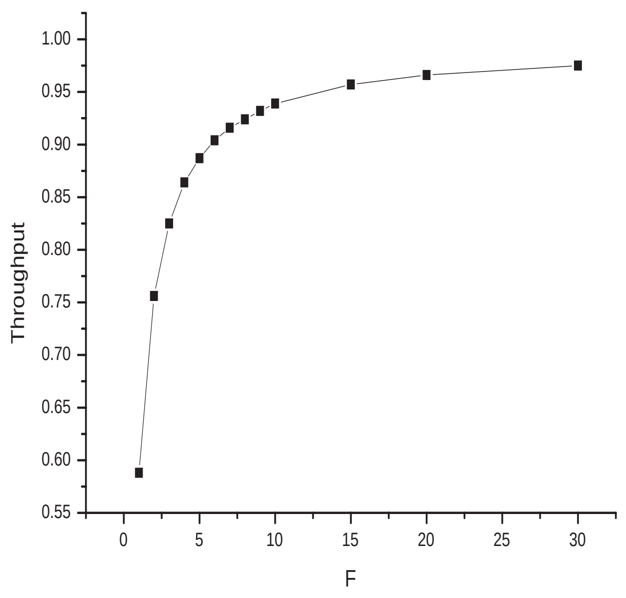

Fig. 5. The maximum throughput as a function of F in the first experiment

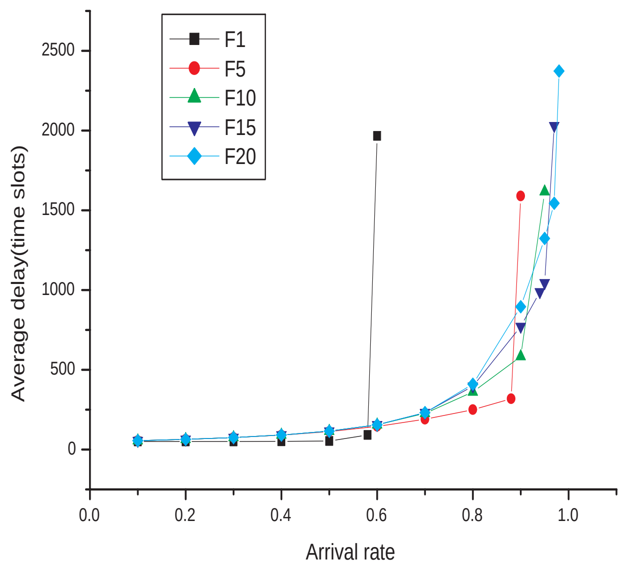

Fig. 6. The average delay for various F

we not only increase the throughput but also the re-sequencing delay. The key design problem is then to determine the right size of the central buffer so that one can have reasonably high throughput and tolerable re-sequencing delay.

The exact analysis for the finite buffer case is much more difficult than the infinite buffer case in [3]. Instead, we resort to computer simulations.

In all our simulations, we consider 100×100 two-stage switches, i.e., $N = 100$. The objective of the first experiment is to determine the size of the central buffer for the switch. The settings of the first experiment are as follows:

(1A) **Uniform i.i.d. traffic model:**
- Each arrival process at the input port is an independent and identical Bernoulli process.
- Each arrival process at the input port has the same arrival rate ρ_a.
- The destination of an arrival packet is uniformly distributed to the N output ports.

(1B) Each input buffer maintains a simple FIFO queue. As such, HOL blocking may occur.

(1C) The number of time slots in the simulation is 200000.

In our first experiment, we set the arrival rate of each input to 1, i.e., there is a packet arrival at each input port in every time slot. In Figure 5, we plot the (measured) maximum throughput as a function of F (the bin size of the central buffer). As shown in Figure 5, one can have throughput over 95% when $F = 15$. We also plot the average delay as a function of the arrival rate for $F = 1, 5, 10, 15, 20$ in Figure 6. As shown in Figure 6, the average delay is smaller for smaller F when the arrival rate is small. For instance, the case with $F = 1$ has the smallest average delay when the arrival rate is not greater than 0.4. However, as the arrival rate approaches to 0.58, the average delay for $F = 1$ is increased sharply. From Figure 6, it also shows that the case $F = 15$ has a reasonably good delay-throughput performance.

The first experiment suggests that $F = 15$ might be a good choice. To verify this, we set F=15 and measure the throughput for two additional traffic models.

(2A) Non-uniform i.i.d. traffic model:

- This model is similar to the uniform i.i.d. model, but the arrival packets are not uniformly distributed to the N output ports. Instead, the packets are routed to one specific destination (hot spot) with probability 0.5 and uniformly distributed to others. Every input has a different hot spot.

(2B) Uniform Pareto traffic model:

- This is the bursty traffic model described in [3], i.e., packets come as a burst with a random length.
- The burst lengths are chosen independently accord-

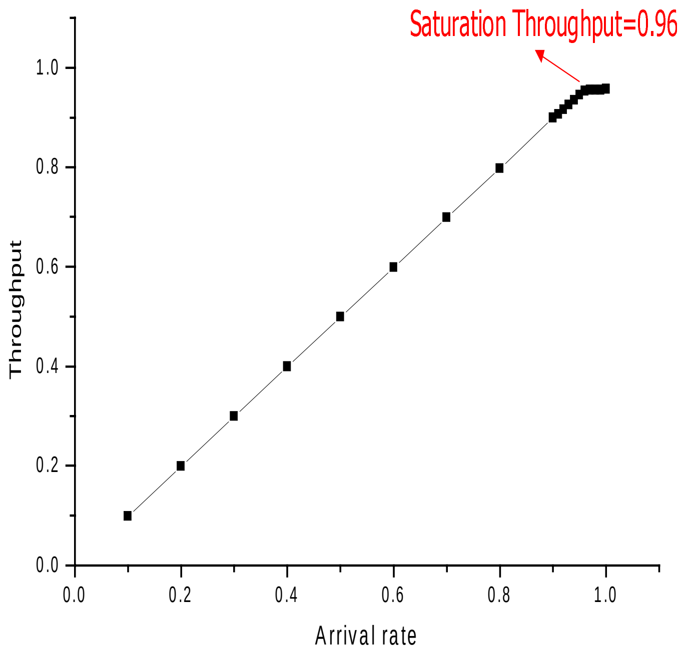

Fig. 7. Throughput as a function of the arrival rate ρ_a for the uniform i.i.d traffic model

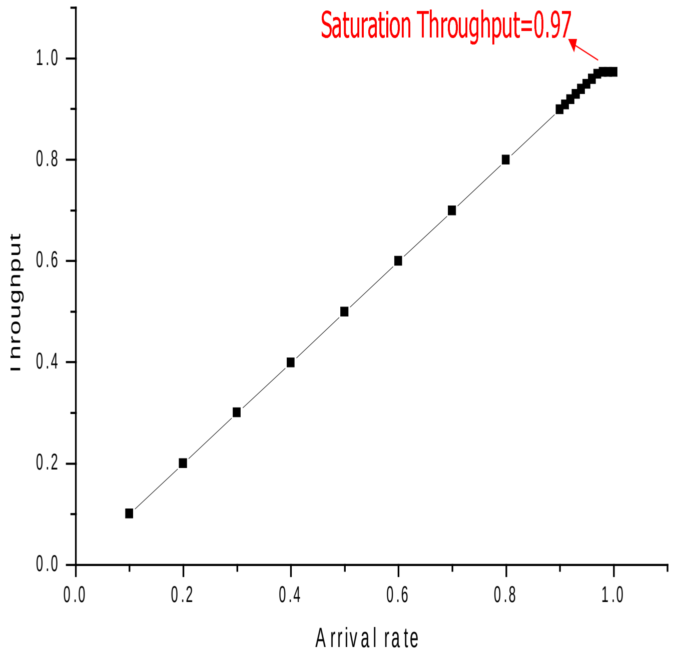

Fig. 8. Throughput as a function of the arrival rate ρ_a for the non-uniform i.i.d traffic model

ing to the following (truncated) Pareto distribution:

$$P(\text{A burst has length i}) = \frac{c}{i^{2.5}}, \quad \text{for } i = 1, 2, \dots, 10000,$$

where $c = \sum_{i=1}^{10000} (\frac{1}{i^{2.5}})^{-1}$ is the normalization constant.

- The destination of the packets in the same burst is uniformly distributed over N output ports.
- At a particular input port, after a burst, the probability that there is another arriving burst is ρ_a, the probability that there is no packets arriving in the next burst is $1 - \rho_a$.

In Figure 7, Figure 8, and Figure 9, we plot the throughput as a function of the arrival rate ρ_a for different traffic models. These figures show that the throughput increases with the arrival rate ρ_a linearly until it reaches its maximum throughput. Once the arrival rate is increased to its maximum throughput, it will almost maintain its maximum value even though the arrival rate ρ_a is increased further. The variation of the maximum throughput in Figure 9 is due to the measurement inaccuracy of the simulation for the bursty Pareto traffic. In Figure 7 and Figure 8, it shows that the maximum throughput is 96% for the uniform i.i.d. model and 97% for the non-uniform i.i.d. traffic model. But for the uniform Pareto traffic model, the maximum throughput is down to 70% as shown in Figure 9.

To gain intuition on these simulation results, we note from the queueing theory that a queue subject to a Bernoulli input has an exponential tail while a queue subject to a Pareto input has a Pareto tail (power law). Since a HOL packet is blocked when the bin in a central buffer is full, HOL blocking occurs much often for bins with large probabilities to be full. For the uniform Pareto traffic, the probability that a bin is full is governed by the power law, i.e., $1/F^\alpha$ for some $\alpha > 0$ (see e.g., [15], [8] and references therein). On the other hand, for the Bernoulli input, the probability that a bin is full is governed by the exponential law, i.e., $\exp(-\theta F)$ for some $\theta > 0$ (see e.g., [2] and references therein). **In short, HOL blocking is more severe for the bursty Pareto traffic than the Bernoulli traffic.**

For the HOL blocking problem caused by the bursty Pareto traffic, there are two general approaches to solve it. One is to increase the size of the central buffer. However, increasing the size of the central buffer causes another problem in the re-sequencing buffer as discussed in Section III. Moreover, if the bins are governed by the power law, then it is very inefficient to increase the throughput by increasing the size of the central buffer. The second approach is to use multiple virtual output queues (VOQs) at the input. This is the approach we will use in the next section.

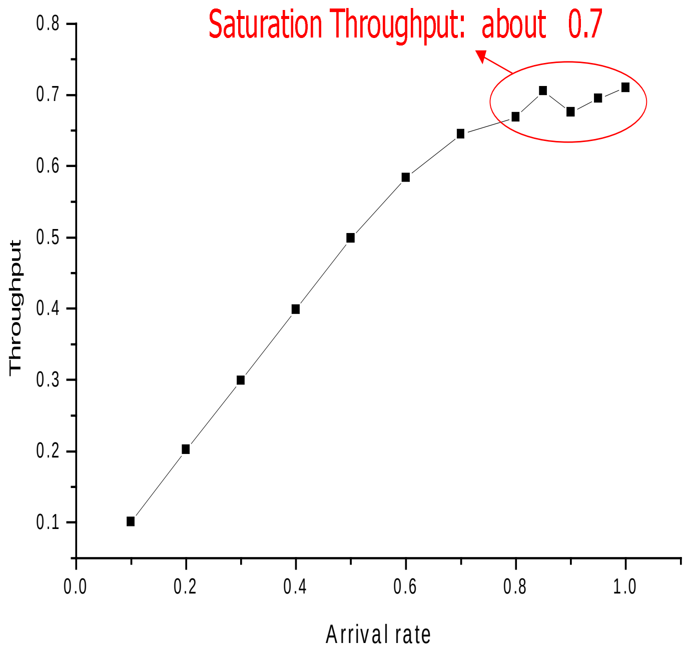

Fig. 9. Throughput as a function of the arrival rate ρ_a for the uniform Pareto traffic model

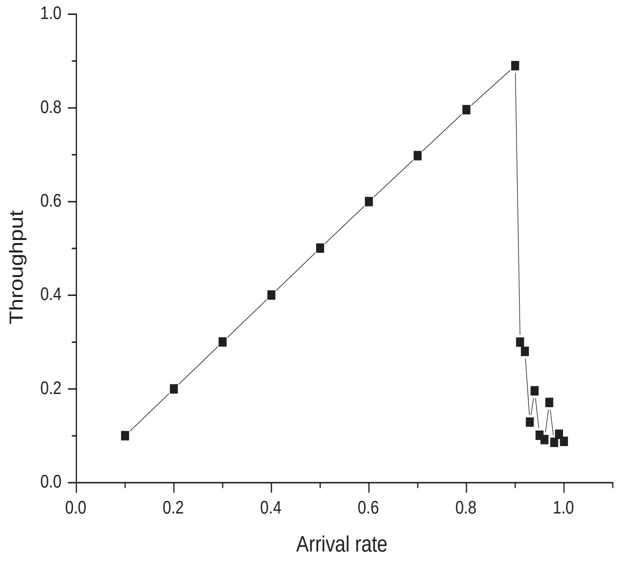

Fig. 10. The throughput of the switch with $m = 10$ and the RR service policy for the uniform Pareto traffic model

V. Third Design Problem: Input Buffers

Instead of using a single FIFO queue, in this section we will use multiple FIFO VOQs to solve the HOL problem.

A. *Round-robin policy for multiple VOQs*

Here we introduce the VOQ technique used in our architecture. Each input port has m VOQs and each VOQ is a FIFO queue. When $m = N$, we have a full size VOQ scheme and each VOQ corresponds to a flow. To reduce the implementation complexity, we may consider the case $m < N$. For this, we need to implement flow aggregation, i.e., a set of flows is assigned to a particular VOQ. To be specific, in our VOQ dispatching policy we examine the destination field of each arrival packet at the input port. If the destination of a packet is d, then the packet is dispatched to the $[((d-1) \bmod \text{ m}) + 1]^{th}$ VOQ.

As there are multiple VOQs at each input, we need to choose one of the m VOQs at each input port to send a packet in every time slot. Since our main objective is to have a simple and high performance switch architecture, we do not intent to use complicated matching polices that requires heavy communication or computation overheads. Here we adopt a very simple service policy, called the round-robin (RR) service policy. The RR service policy is described as follows:

1. Keep a pointer at each input port.
2. If not all the VOQs are empty, advance the pointer clockwise (in the round-robin fashion) to the first non-empty VOQ. Send the HOL packet from that VOQ.

In our third experiment, we replace the single FIFO queue (used in the second experiment) with the VOQ technique mentioned above. We set the number of VOQs to be 10 ($m = 10$), and then measure the throughput for the RR service policy by increasing the arrival rate ρ_a for two different traffic models: the uniform Pareto traffic model and the uniform i.i.d. traffic model.

In Figure 10, we plot the throughput as a function of the arrival rate ρ_a for the uniform Pareto traffic model. It shows that the maximum throughput is now increased to 90% from 70% of the case with single FIFO queue. However, while the arrival rate exceeds 90%, an unexpected catastrophic phenomenon occurs. The throughput not only cannot keep up with the arrival rate but also sharply reduces down to 10%. The throughput is even worse than the case of single FIFO queue! In Figure 11, it shows a similar result for the uniform i.i.d. traffic model. Both curves show that the switch under the RR service policy encounters an unexpected catastrophic phenomenon like that in ALOHA and CSMA (see e.g., [14]). By carefully examining our simulation results, we find that the patterns of pointer rotation fall into a deterministic and periodic circle. As such, a particular

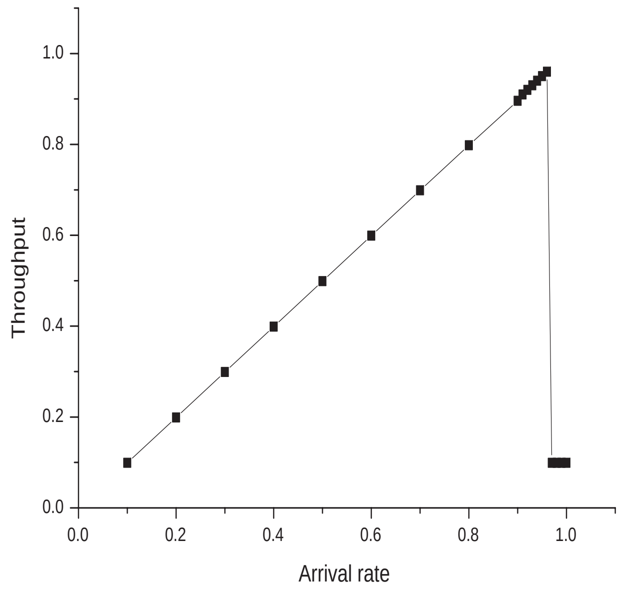

Fig. 11. The throughput of the switch with $m = 10$ and the RR service policy for the uniform i.i.d. traffic model

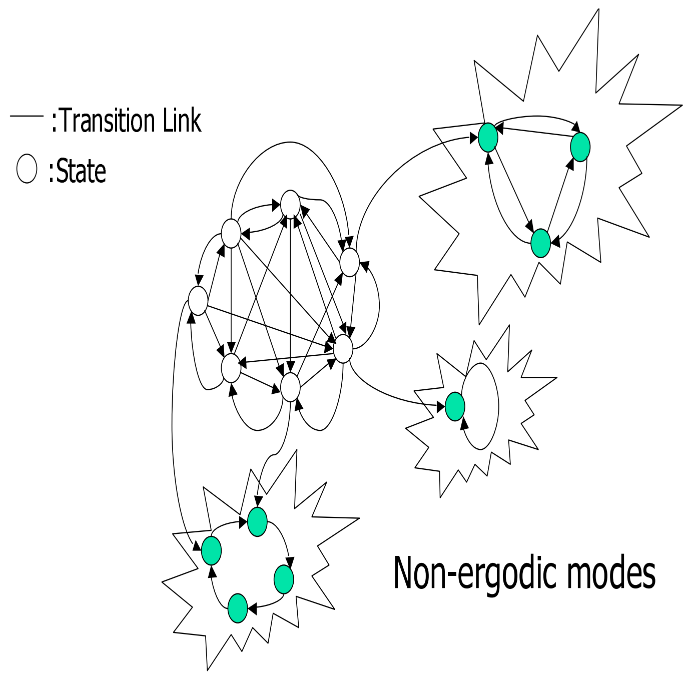

Fig. 12. The non-ergodic modes in a state transition diagram

central buffer always tries to fetch the same group of VOQs. Such a phenomenon is called a **non-ergodic mode** of the switch and it will be explained further in the next section.

B. Non-ergodic mode

To understand the non-ergodic mode in our switch, we note that **when the offered load exceeds the maximum (stable) throughout, all the VOQs start to grow. As the number of output ports is an integer multiple of the number of VOQs and the service policy is round-robin, the pointers at the input ports become deterministic and periodic when all VOQs become non-empty. Since the connection patterns of the switch fabrics are also deterministic and periodic, the "state" of the switch is non-ergodic, i.e., a particular central buffer will not be connected to all the VOQs in the long run. Instead, it is only connected to a certain subset of VOQs.**

In Figure 12, we illustrate concept of non-ergodic modes via a state transition diagram of a Markov chain. A non-ergodic mode is a group of states that have no transition link to other states outside of the group. If the system enters a non-ergodic mode, then it can only stay in one of the states in that non-ergodic mode.

¿From our simulation results, we observe that the flows are partitioned into several groups in a non-ergodic mode. **Once the switch enters a non-ergodic mode, a particular central buffer only has incoming traffic from a certain set of flows. Thus, for a particular central buffer, time slots are wasted for certain output ports as there are no incoming traffic for these output ports. As a result, the throughput is sharply reduced.** This phenomenon occurs for both the uniform i.i.d traffic and the bursty Pareto traffic. We also run another simulation which extends the number of VOQs to N, i.e., the full size VOQ scheme. For both the uniform i.i.d traffic and the uniform Pareto traffic, the throughput is also reduced to 0.6 in heavy load. This phenomenon also exists even when the number of VOQs is the same as the number of input ports, i,e., $m = N = 100$.

C. The effect of randomness for the non-ergodic mode

In this section, we provide several tentative solutions for avoiding the non-ergodic modes. The idea is to introduce randomness into the system so that the switch can jump out of a non-ergodic mode. **As shown in Figure 13, if a system is trapped in a non-ergodic mode, then providing a transition probability to permit the system to jump out of the non-ergodic mode is needed. The transition probability can be created by introducing randomness into the system so that the system will not be trapped in a fixed group of states.** We do so by modifying the scheme of

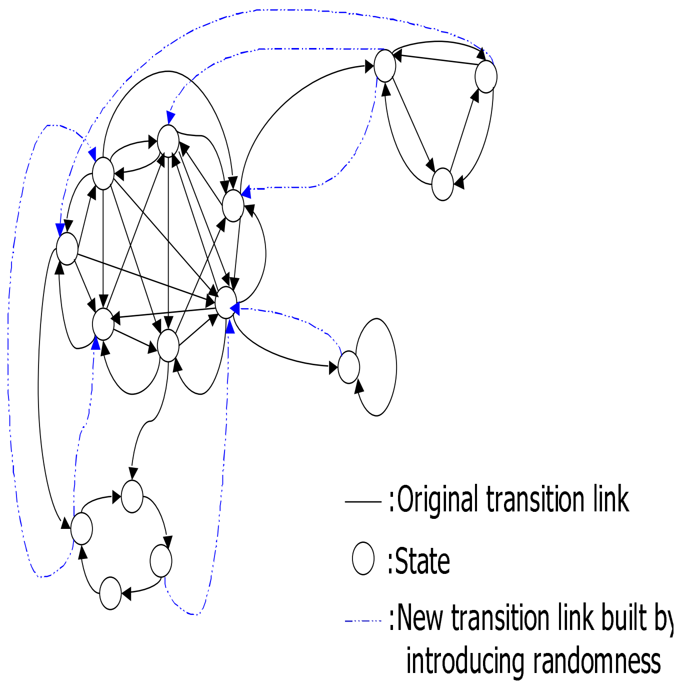
:Original transition link
:State
:New transition link built by introducing randomness

Fig. 13. Jumping out of a non-ergodic mode by introducing randomness

advancing pointers in the RR service policy.

[Solution 1.] Advance when not blocked: Advance the pointer when the transmission to the connected central buffer is not blocked. Otherwise keep the pointer at the same position.

[Solution 2.] Advance when blocked: Advance the pointer when the transmission to the connected central buffer is blocked. Otherwise keep the pointer at the same position.

[Solution 3.] Advance with probability 0.5: Advance the pointer with probability 0.5 in every time slot. (This is independent of the outcome of the transmission to the connected central buffer as in the RR service policy.)

[Solution 4.] Randomly setup the position of the pointers at the beginning of each time slot.

Note that the randomness in Solution 1 and Solution 2 relies on the event whether a transmission to a connected center buffer is successful or not. This may not be as "random" as that used in Solution 3 and Solution 4.

Under the same settings as the simulations used for the RR service policy, we also perform several experiments to compare the performance of the switches that use these solutions. In Figure 14 and 15, it shows that the throughput in heavy load has a great deal of improvement after modifying the RR service policy. In Figure 16 and 17, we enlarge the heavy load segments (for ρ_a from 0.80 to 1.00) in Figure 14 and 15. Now the improvement can be observed more clearly. ¿From these two figures, we show that Solution 3 and Solution 4 achieve 96% throughput for the uniform i.i.d. traffic model and 93% for the uniform Pareto traffic model. These two solutions are based on introducing randomness into the switch. However, the performance of Solution 1 and Solution 2, though greatly improved from the RR service policy, is not as good as that in Solution 3 and Solution 4. This is due to the fact that "randomness" introduced in Solution 1 and Solution 2 may not be independent of the switch. As such, it may not be random enough to provide sufficient transition links to enable the switch to jump out of non-ergodic modes. For Solution 3 and Solution 4, the random information is independent of the state of the switch.

Note that the throughput of Solution 3 is almost as good as that of Solution 4, even though only one bit of randomness is needed in Solution 3. In the regard of hardware complexity, Solution 3 is a better choice than Solution 4. For Solution 4, in general a pseudo random number generator is needed and this causes additional hardware complexity. For Solution 3, one only needs one bit information and this may be taken from a bit in the header or payload of an incoming packet. One then advances the pointer at an input port if the bit taken is 1. Otherwise keep the pointer at the same position.

We also extend our simulations to the full size VOQs for Solution 3 and Solution 4, i.e., $m = N = 100$. Our simulation results show that the throughput can achieve 96% for the uniform Pareto traffic model, which is higher than 93% of the case of $m = 10$. There is only 3% improvement of the throughput at the cost of expanding to the full size VOQs.

We note that non-ergodic modes can also be used for explaining the pointer synchronization problem observed in input-buffered switches with the Round-Robin Matching (RRM). The RRM used in input-buffered switch consists of the following three steps:

[**Step 1.**] *Request*. Each unmatched input sends a request to every output for which it has a non-empty VOQ.

[**Step 2.**] *Grant*. If an unmatched output receives any requests from the inputs, it grants to the one that is closest to its pointer. The pointer at that output is incremented *clockwise* to one location beyond the granted input.

[**Step 3.**] *Accept*. If an input receives a grant, it accepts the one that is closest to its pointer. The pointer at that input is incremented *clockwise* to one location beyond the accepted output.

It is observed by McKeown [13] that the pointers in RRM are trapped in a deterministic and periodic cycle for a certain traffic model. As such, only half of the inputs/outputs can be matched and that leads to only 50% throughput. To cope with the pointer synchronization

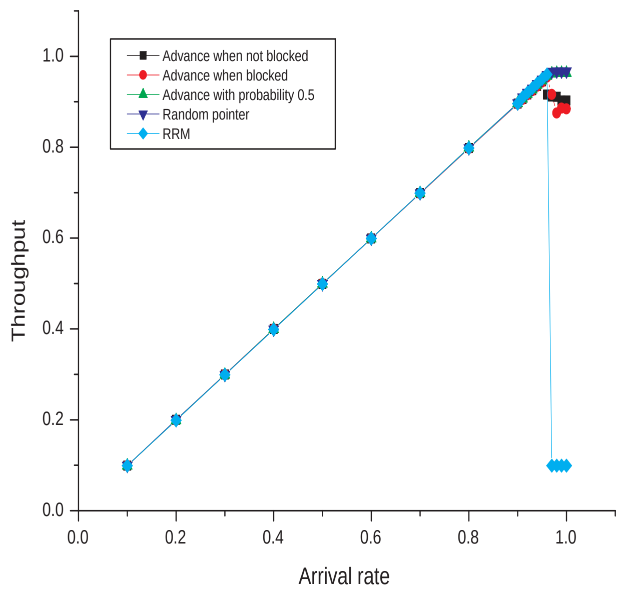

Fig. 14. The throughput of the switch with $m = 10$ under various service policies for the uniform i.i.d. traffic model

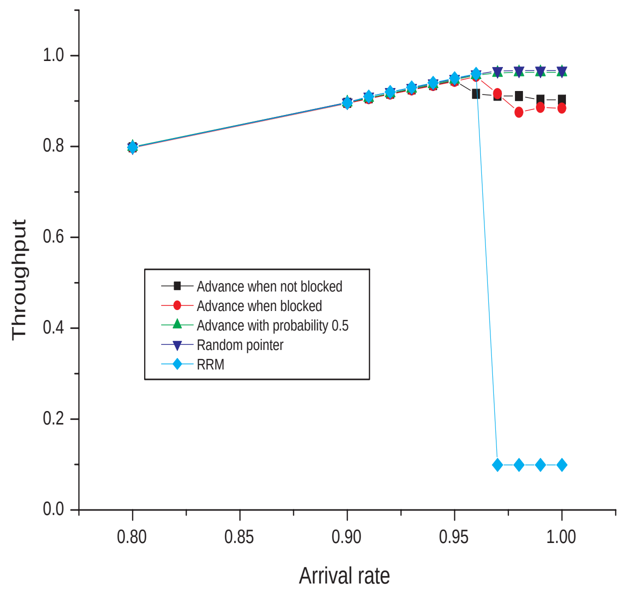

Fig. 16. Enlargement of Figure 14 under heavy load

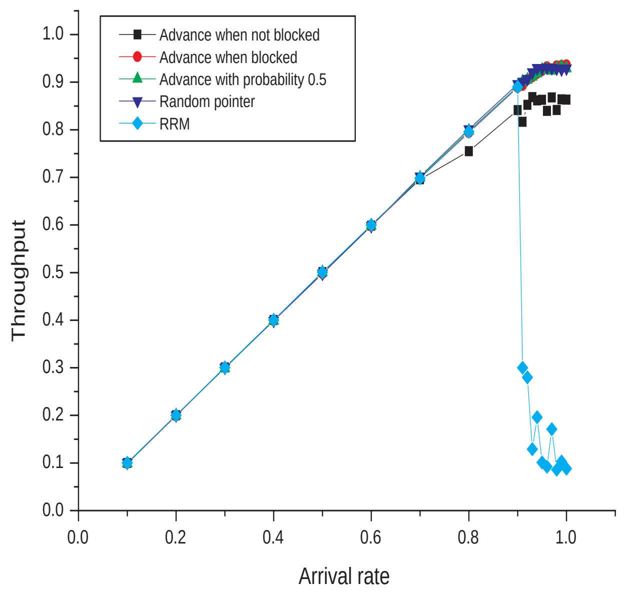

Fig. 15. The throughput of the switch with $m = 10$ under various service policies for the uniform Pareto traffic model

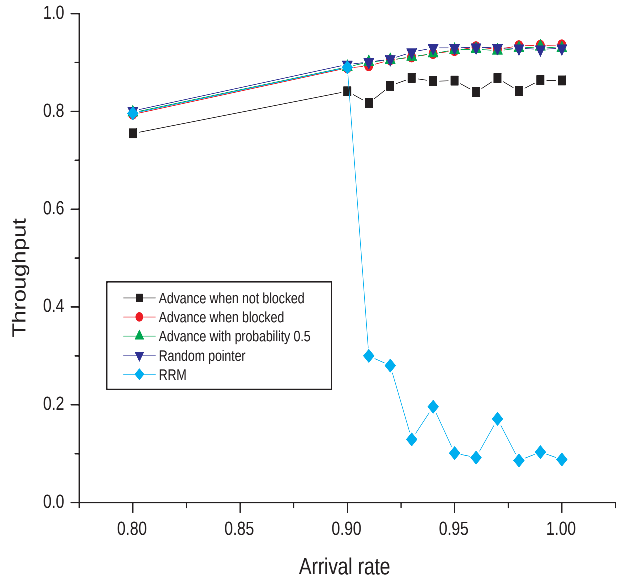

Fig. 17. Enlargement of Figure 15 under heavy load

problem, he proposed using iSLIP (see [13] for more details) by modifying Step 2 in RRM. The pointer at an output is incremented *clockwise* to one location beyond the granted input if and only if the grant is accepted in Step 3. Whether a grant is accepted is somehow like flipping a coin and the iSLIP algorithm can de-synchronize the pointers in RRM by using this one bit of ”hidden random” information. However, this one bit of information may not be random at all for a certain traffic model. In fact, as shown in [3], there is a deterministic traffic model that also leads iSLIP to a non-ergodic mode.

VI. Conclusions

In this paper, we proposed a simple and high performance switch using the two-stage architecture. In the following, we summarize our design principles for the proposed switch architecture.

(i) Re-sequencing buffer: the size of the re-sequencing buffer needs to be proportional to the size of the central buffer to ensure that no packets are lost due to re-sequencing. Thus, there is a trade-off between the throughput and the re-sequencing delay.

(ii) Central buffer: using moderate size of central buffers works fine when traffic is not bursty. However, when the traffic is Pareto, the behavior of central buffers is governed by the power law. In that case, one needs to address the head-of-line blocking (HOL) problem at the input.

(iii) Input buffer: increasing the number of VOQs may not increase the throughput in the two-stage switches if a simple robin-robin service policy is used. Be careful of the catastrophic phenomenon when the switch is trapped in a non-ergodic mode.

(iv) Non-ergodic mode: Falling in a non-ergodic mode is due to that fact the pointers at the input ports become deterministic and periodic. To jump out of a non-ergodic mode, one needs to provide new transition links by introducing “randomness” into the switch.

We note that there is also a recent paper [7] that proposed a simple design of the two-stage switch, called the Byte-focal switch. The key difference between theirs and ours is the method for the control of the re-sequencing delay. We use finite center buffers to control the re-sequencing delay, while they use flow spitters in [4] to control the re-sequencing delay. This results in different design for input buffers. For our input-buffers, we only used the modified round-robin service policies

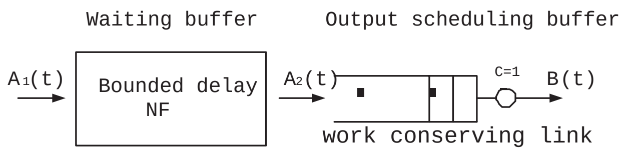

Fig. 18. The mathematical model of the re-sequencing buffer

to solve the problem incurred by non-ergodic modes. In [7], threshold type of control based on the queue length of each VOQ is used.

There are several problems that require further study in the following:

(i) For the case of variable length packets, determine the size of the re-sequencing buffer when packet reassembling is also performed in the re-sequencing buffer.

(ii) Perform mathematical analysis to find the relation between the size of the central buffer and the throughput.

(iii) Understand more thoroughly why the two-stage load balanced switch with multiple input VOQs may be trapped in a non-ergodic mode.

Appendix

In this section, we prove Theorem III.1. The proof is based on the “network calculus” (see e.g., [2] and references therein). Let $A_1(t)$ be the cumulative number of packets that arrive at the waiting buffer by time t, $A_2(t)$ be the cumulative number of packets that logically arrive at the output scheduling buffer by time t, and $B(t)$ be the cumulative number of packets that depart from the output scheduling buffer by time t. Note that packets that logically arrive at the output scheduling buffer are in fact physically in the waiting buffer. It is the information of its cell index that is passed on to the output scheduling buffer. Since the worst case re-sequencing delay is bounded above NF, the maximum delay for a packet in the waiting buffer is bounded above by NF. Thus, the waiting buffer can be viewed as a network element with bounded delay NF. Also, the output scheduling buffer sends out a packet per unit of time as long as the event list is not empty. As such, the output scheduling buffer is a work conserving link with capacity 1 (Section 1.3 in [2]). Thus, the architecture for re-sequencing buffer in Figure 4 is equivalent to the concatenation of a network element with bounded delay NF and a work conserving link as shown in Figure 18.

Now we show the total number packets stored in the re-sequencing buffer is bounded above by NF. From the input-output relation for a work conserving link (Lemma

1.3.1(ii) in [2]), we have

$$B(t) = \min_{0\le s\le t}[A_2(s) + (t-s)]. \tag{2}$$

As the maximum delay in the waiting buffer is bounded above NF, we also have

$$A_2(t) \ge A_1((t-NF)^+), \tag{3}$$

where $(x)^+ = \max(0,x)$. Using (3) in (2) yields

$$\begin{aligned} B(t) &\ge \min_{0\le s\le t}[A_1((s-NF)^+) + (t-s)] \\ &= \min_{0\le \tau\le t}[A_1(\tau) + (t-\tau-NF)^+] \end{aligned} \tag{4}$$

Since there is at most one packet arrival to the waiting buffer per unit of time, we have for all $\tau \le t$

$$A_1(t) - A_1(\tau) \le t - \tau. \tag{5}$$

Note that the number of packets stored in the re-sequencing buffer at time t is simply $A_1(t) - B(t)$. It then follows from (4) and (5) that

$$\begin{aligned} &A_1(t) - B(t) \\ &\le \max_{0\le\tau\le t}[A_1(t) - A_1(\tau) - (t-\tau-NF)^+] \\ &\le \max_{0\le\tau\le t}[t-\tau-(t-\tau-NF)^+] \\ &\le NF. \end{aligned}$$

Now we show that the maximum delay in the re-sequencing buffer is bounded by $2NF$. The delay of a packet in the re-sequencing buffer consists of two parts: the delay in the waiting buffer and the delay in the output scheduling buffer. The delay in the waiting buffer is bounded above by NF. Thus, it suffices to show that the delay in the output scheduling buffer is also bounded above by NF. Since the total number of packets stored in the re-sequencing buffer at time t is bounded above by NF, the number of packets (logically) in the output scheduling buffer is also bounded by NF. As the service policy of the output scheduling buffer is FIFO and there is a packet sending out from the output scheduling buffer per unit of time, the delay in the output scheduling buffer is also bounded by NF.

As the maximum delay in the re-sequencing buffer is bounded by $2NF$, it follows that the waiting buffer with size of $2NF$ is enough to ensure that on packets are lost inside the re-sequencing buffer.

Finally, we note that packets in the waiting buffer are not served in the FIFO order. This is the reason why we still need the waiting buffer to have the size of $2NF$ even though the total number of packets in the re-sequencing buffer is bounded above by NF.

REFERENCES

[1] T. Anderson, S. Owicki, J. Saxes and C. Thacker, "High speed switch scheduling for local area networks," *ACM Trans. on Computer Systems*, Vol. 11, pp. 319-352, 1993.

[2] C.S. Chang, *Performance Guarantees in Communication Networks*. London: Springer-Verlag, 2000.

[3] C.S Chang, D.S. Lee and Y.S. Jou, "Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering," *Computer Communications*, Vol. 25, pp. 611-622, 2002.

[4] C.S. Chang, D.S. Lee and C.M. Lien, "Load balanced Birkhoff-von Neumann switch, part II: Multi-stage buffering," *Computer Communications*, Vol. 25, pp. 623-634, 2002.

[5] C.S. Chang, D.S. Lee, and Y.J. Shih, "Mailbox Switch: A scalable two-stage switch Architecture for conclict resolution of ordered packets," *Proceedings of IEEE INFOCOM*, 2004.

[6] C.S. Chang, D.S. Lee, and C.Y. Yue, "Providing guaranteed rate services in the load balanced Birkhoff-von Neumann switches," *Proceedings of IEEE INFOCOM*, 2003.

[7] H. J. Chao, J. Song, N. S. Artan, G. Hu, and S. Jiang, "Byte-fcal: a practical load-balanced switch," *preprint*.

[8] P.R. Jelenković and A.A. Lazar, "Subexponential asymptotics of a Markov-modulated random walk with queueing applications," *J. Appl. Prob.*, June, 1998.

[9] I. Keslassy, S.-T. Chuang and N. McKeown, "A load-balanced switch with an arbitrary number of linecards," *Proc. of IEEE INFOCOM*, 2004.

[10] I. Keslassy and N. McKeown, "Maintaining packet order in two-stage switches," *Proceedings of IEEE INFOCOM*, New York, 2002.

[11] I. Keslassy, S.-T. Chung, K. Yu, D. Miller, M. Horowitz, O. Slogaard, N. McKeown, "Scaling Internet routers using optics," *ACM SIGCOMM 2003*, Karlsruhe, Germany, Sep. 2003.

[12] Y. Li, S. Panwar and H.J. Chao, "On the performance of a dual round-robin switch," *Proceedings of IEEE INFOCOM*, pp. 1688-1697, 2001.

[13] N. McKeown, "Scheduling algorithms for input-queued cell switches," *PhD Thesis. University of California at Berkeley*, 1995.

[14] R. Nelson, "Stochastic catastrophe theory in computer performance modeling," *Journal of the Association for Computing Machinery*, Vol. 34, pp. 661-685, 1987.

[15] A.G. Pakes, "On the tail of waiting-time distribution," *J. Appl. Prob.*, Vol. 12, pp. 555-564, 1975.

[16] Y. Tamir and H.C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, pp. 13-27, 1993.