CHAPTER 14

LOAD-BALANCED SWITCHES

Internet traffic continues to grow rapidly, and to keep pace with demand, there has been a significant research effort into high-speed large-capacity packet switch architectures that consume less power yet outperform current switch architectures. Because of memory speed constraints, most proposed large-capacity packet switches use input buffering alone or in combination with other schemes, such as output buffering or cross-point buffering. The issue of how to schedule packets efficiently to achieve high throughput and low delay for a large-capacity switch has been one of the main research topics in the past few years. Although several practical scheduling schemes have been proposed or implemented, for example, *i*SLIP [1], DRRM [2], and others, most of them require a centralized packet scheduler, an increase in the interconnection complexity between the line cards and the packet scheduler, and a speedup for the switch fabric to compensate for some deficiencies in packet scheduling. Most practical packet-scheduling schemes cannot achieve 100 percent throughput especially under some traffic distributions.

Recently, a novel switch architecture, the load-balanced Birkhoff–von Neumann (LB-BvN) switch proposed by Chang et al. [3], overcame the above-mentioned problems and opened up a new avenue for designing large-capacity packet switches without using a packet scheduler, and for achieving 100 percent throughput under nearly all traffic distributions.

14.1 BIRKHOFF–VON NEUMANN SWITCH

Before introducing load-balanced Birkhoff–von Neumann switches, let us look into the traditional Birkhoff–von Neumann switch. With reference to Figure 14.1, the Birkhoff–von

High Performance Switches and Routers, by H. Jonathan Chao and Bin Liu

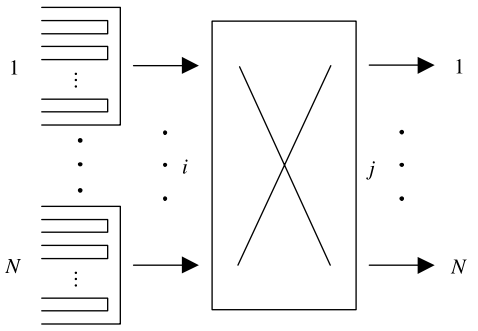

Figure 14.1 Birkhoff–von Neumann switch.

Neumann switch is an input-buffered switch. It uses the virtual output queuing (VOQ) technique to solve the head-of-line (HOL) blocking problem.

The principle behind solving output port contention in the Birkhoff–von Neumann switch is to use the capacity decomposition approach described by Birkhoff [4] and von Neumann [5] to schedule the connection patterns. Let $\mathbf{r} = [r_{i,j}]$ be the rate matrix with $r_{i,j}$ being the rate allocated to the traffic from input i to output j for an $N \times N$ input-buffered switch with the following conditions:

$$\sum_{i=1}^{N} r_{i,j} \leq 1, \qquad j = 1, 2, \ldots, N, \tag{14.1}$$

and

$$\sum_{j=1}^{N} r_{i,j} \leq 1, \qquad i = 1, 2, \ldots, N. \tag{14.2}$$

Then, there exists a set of positive numbers ϕ_k and permutation matrices $\mathbf{P}_k$, $k = 1, 2, \ldots, K$, for some $K \leq N^2 - 2N + 2$ that satisfies

$$\mathbf{r} \leq \sum_{k=1}^{K} \phi_k \mathbf{P}_k, \tag{14.3}$$

and

$$\sum_{k=1}^{K} \phi_k = 1. \tag{14.4}$$

When such a decomposition is obtained, it is simply a case of scheduling the connection pattern $\mathbf{P}_k$ proportional to its weight ϕ_k. For the details of the decomposition algorithm, refer to Chang et al. [6, 7].

For instance, a 4×4 input-buffered switch with the traffic rate matrix:

$$\begin{pmatrix} \frac{1}{3} & \frac{2}{3} & 0 & 0 \\ \frac{2}{3} & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Using the Birkhoff–von Neumann rate decomposition, we obtain the following two permutation matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ with corresponding weight ϕ_1 and ϕ_2:

$$\frac{1}{3}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad \frac{2}{3}\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

With these two permutation matrices in hand, it is easy to schedule the connection pattern according to its weight. The connection pattern corresponding to this example is shown in Figure 14.2.

If the allocated bandwidth is larger than the arrival rate for each input–output pair, Chang [6, 7] has shown that the Birkhoff–von Neumann input-buffered switch can achieve 100 percent throughput without framing and internal speedup. However, the complexity of Birkhoff–von Neumann decomposition is as high as $O(N^{4.5})$, and the number of permutation matrices deduced from the Birkhoff–von Neumann decomposition algorithm is $O(N^2)$, which may not scale for switches with a large number of input/output ports.

Thus, the Birkhoff–von Neumann input-buffered switch has attractive system performance but suffers from computational complexity and scalability. However, let us consider a special case of an input-buffered switch, in which all inputs have uniform traffic distribution.

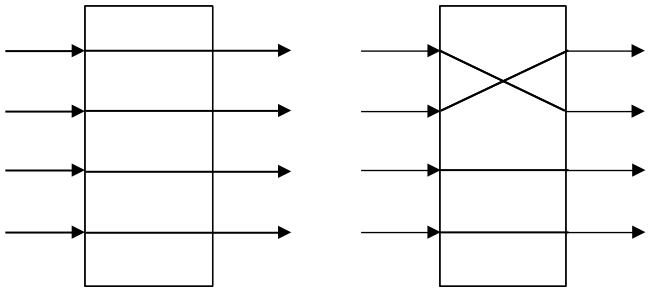

Figure 14.2 Connection pattern of example switch. (*a*) Connection pattern with weight 1/3; (*b*) Connection pattern with weight 2/3.

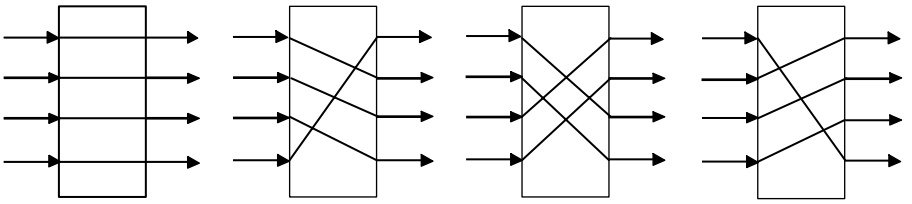

Figure 14.3 Connection patterns of a 4×4 input-buffered load-balanced switch.

We can easily construct a traffic rate matrix for such a 4×4 switch, as follows:

$$\begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix};$$

and the permutation matrices and corresponding weights for this load-balanced switch are:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The connection patterns according to these permutations are shown in Figure 14.3.

Notice that in an input-buffered switch with uniform traffic distribution, the Birkhoff–von Neumann decomposition gives us simply the periodic time division multiplexed (TDM) connection pattern. In other words, a deterministic TDM switch fabric can ensure a load-balanced switch to achieve 100 percent throughput with a scheduling complexity of $O(1)$. This is the motivation for designing a two-stage load-balanced Birkhoff–von Neumann switch.

14.2 LOAD-BALANCED BIRKHOFF–VON NEUMANN SWITCHES

This section presents the load-balanced Birkhoff–von Neumann switch architecture and its performance including throughput, delay, and buffer usage.

14.2.1 Load-Balanced Birkhoff–von Neumann Switch Architecture

With reference to Figure 14.4, the load-balanced Birkhoff–von Neumann (LB-BvN) switch consists of two crossbar switch stages and one set of VOQs between these stages. The first stage performs load balancing and the second stage performs switching. This switch does not need any schedulers since the connection patterns of the two switch stages are

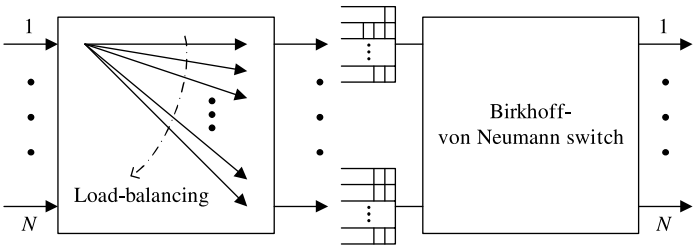

Figure 14.4 LB-BvN switch.

deterministic and are repeated periodically. The connection patterns should be selected so that in every consecutive N timeslot, each input should connect to each output exactly once with a duration of one timeslot.

The LB-BvN switch has the following advantages:

1. *Scalability.* On-line complexity of the scheduling algorithm of the switch is $O(1)$.
2. *Low Hardware Complexity.* Only two crossbar switch fabrics and buffers are required, and the two crossbar switch fabrics can be realized by the Banyan networks, due to deterministic and periodic connection patterns. Neither internal speedup nor rate estimation (as in the original Birkhoff–von Neumann switch) is needed in this switch.
3. *100 Percent Throughput.* Load-balanced Birkhoff–von Neumann switch achieves 100 percent throughput as an output-buffered switch for unicast and multicast traffic.
4. *Low Average Delay in Heavy Load and Bursty Traffic.* When input traffic is bursty, load balancing is very effective in reducing delay. The average delay of the load of the LB-BvN switch is proven to converge to that of an output-buffered switch under heavy load.
5. *Efficient Buffer Usage.* When both the LB-BvN switch and the corresponding output-buffered switch are allocated with the same finite amount of buffers at each port, the packet loss probability in the LB-BvN is much smaller than that in an output-buffered switch when the buffer is large.

14.2.2 Performance of Load-Balanced Birkhoff–von Neumann Switches

As mentioned in the earlier section, a single-stage input-buffered crossbar switch using deterministic sequence achieves 100 percent throughput for uniform Bernoulli i.i.d. (independent and identically distributed) traffic. In the LB-BvN switch, the first stage supplies the second stage with such a traffic distribution by performing load-balancing using deterministic connection patterns. Since the second-stage switch receives uniform Bernoulli i.i.d. traffic, the entire system can reach 100 percent throughput for nearly all input traffic patterns. A rigorous proof of the throughput for the LB-BvN switch and the conditions on input traffic requirements are given by Chang et al. [3].

With reference to Figure 14.5, the average delay of the LB-BvN switch is noticeably better than that of the single-stage Birkhoff–von Neumann switch, and converges to output-buffered switches at heavy load above 0.9.

This excellent average delay performance in the LB-BvN switch is due to the first-stage load-balancing switch. It efficiently reduces the burstiness of traffic from inputs to VOQs

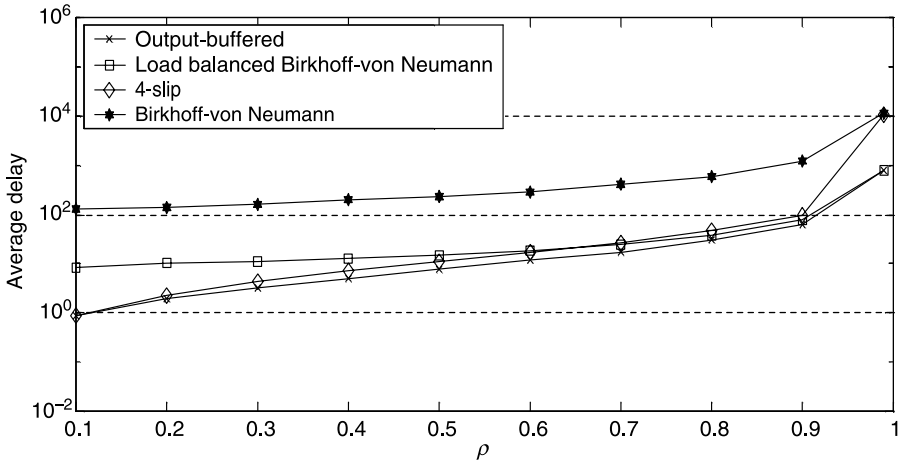

Figure 14.5 Average delay under the uniform bursty traffic.

between two stages. As a result, the second-stage Birkhoff–von Neumann switch always receives uniform Bernoulli i.i.d. traffic. Hence, the average delay is significantly better than the single stage Birkhoff–von Neumann switch and has convergence with output-buffered switches at high traffic load. Figure 14.6 illustrates the effect of the first-stage load-balancing switch under uniform bursty traffic. Notice the traffic at each VOQ is indeed uniform Bernoulli i.i.d.

Chang et al. [3] mathematically compared the average delay between the output-buffered switch and the LB-BvN switch. As Table 14.1 shows, under heavy load bursty traffic ($\rho \to 1$), the average delays of both switches are similar.

In addition to 100 percent throughput and outstanding average delay, the LB-BvN switch is more efficient in buffer usage than the corresponding output-buffered switch. With reference to Figure 14.7, the decay rate of the tail distribution of queue length in the LB-BvN

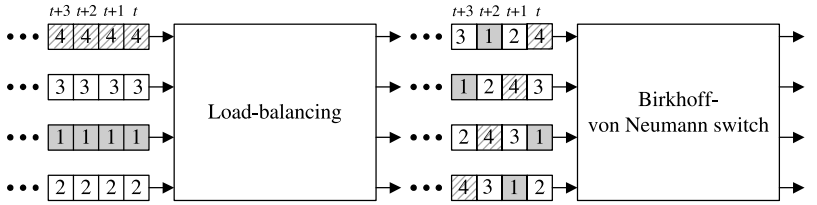

Figure 14.6 Burst reduction in the uniform bursty traffic model.

TABLE 14.1 Average Delay of Output-Buffered Switch and LB-BvN

Delay	Output-Buffered	Load-Balanced
i.i.d.	$(N-1)\cdot\rho/N\cdot 2(1-\rho)$	$(N-1)/2(1-\rho)$
Bursty	$(N-1)\cdot\rho/2(1-\rho)$	$(N-1)/2(1-\rho)$

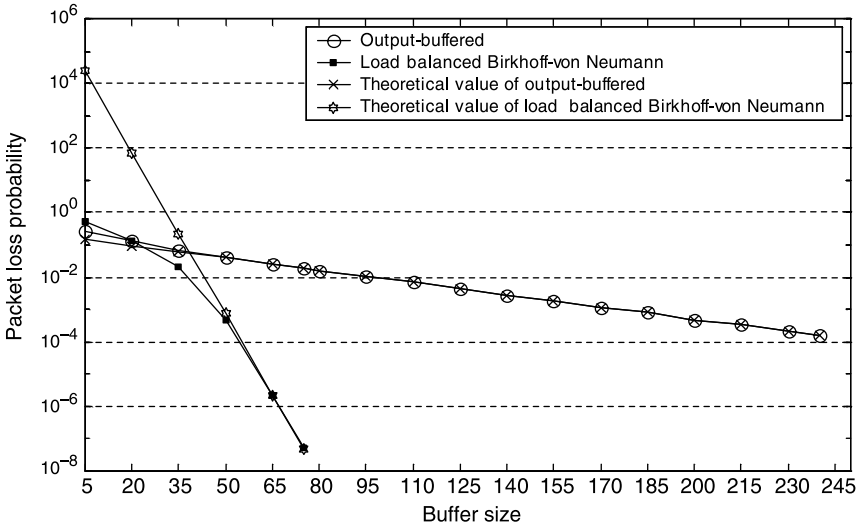

Figure 14.7 Packet loss probability under uniform bursty traffic ($N = 16$, $\rho = 0.8$).

is much smaller than that of the corresponding output-buffered switch. This implies that if we allocate the same finite amount of buffer in each port of both switches, the LB-BvN switch has a much smaller packet loss probability than that of the output-buffered switch.

14.3 LOAD-BALANCED BIRKHOFF–VON NEUMANN SWITCHES WITH FIFO SERVICE

A flow is defined as all of the packets[1] going from one input i to one output k and is denoted as S(i, k). Since the traffic at the input of the switch is not necessarily uniform, the number of packets from different flows can vary. This situation is reflected as the difference in queue lengths at the VOQs in the middle of the switch. Since those queues are served uniformly independent of their lengths, delays in addition to the queuing delay and out-of-sequence of packets are inevitable in the basic form of the LB-BvN switch architecture (Fig. 14.4).

The remainder of this chapter is dedicated to the techniques that have been proposed to solve the packet out-of-sequence problem that is inherent in the original LB-BvN switch. Why is this such a problem? Out-of-sequence packets make TCP trigger a fast recovery, and TCP's sliding window is reduced by half. This also reduces end-to-end throughput by half. Taking a closer look at how packets are transmitted out of sequence, we need to focus on the four flows S(1, 4), S(2, 3), S(3, 1), and S(4, 2) from Figure 14.6, each of which contains four packets (a, b, c, and d), and has an arrival order of a $<$ b $<$ c $<$ d. Each diagram in Figure 14.8 represents the switching stage of the LB-BvN switch at different times within a single connection cycle.

At time t in Figure 14.8*a*, four packets, 4a, 3a, 1a, and 2a, arrive at each of the four VOQs. Since there are no packets queued for the outputs determined by this connection pattern, no packets are transmitted.

[1]Packets and cells are interchangeable in the chapter.

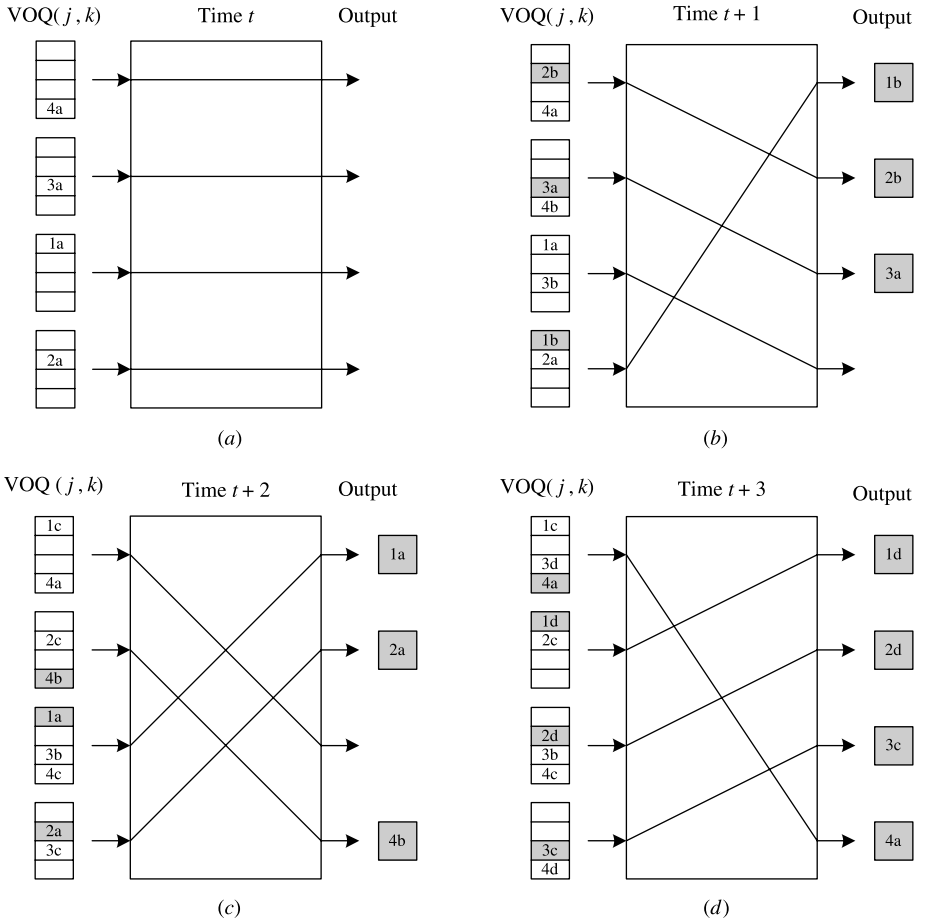

Figure 14.8 LB-BvN switching stages. (*a*) LB-BvN Switching stage at time *t*; (*b*) LB-BvN Switch stage at time $t + 1$; (*c*) LB- BvN Switching stage at time $t + 2$; and (*d*) LB-BvN Switching stage at time $t + 3$.

At time $t + 1$, four more packets (2b, 4b, 3b, and 1b) arrive at the VOQs, as shown in Figure 14.8*b*. The new connection pattern allows three packets from the inputs to be transmitted. Already we can begin to notice the out-of-sequence problem since packets 1b and 2b have been transmitted while packets 1a and 2a, which arrived one time slot earlier, are still waiting in the buffers.

At time $t + 2$, four new packets arrive and the current connection pattern transmits three more packets. Notice that packet 1a arrives at output 1 while in the previous time slot packet 1b has already been transmitted, as shown in Figure 14.8*c*. At time $t + 3$, the last packets of each of the four flows arrive at the second-stage inputs as shown in Figure 14.8*d*. The connection pattern then repeats itself in a periodic fashion, allowing buffered packets to be transmitted when they are connected to their desired output ports.

Seven schemes, proposed to solve the problem of out-of-sequence packets in the LB-BvN switches, are now described: (1) FCFS (first come first served) based scheduling policy [8], (2) EDF (earliest deadline first) based scheduling policy [8], (3) EDF-3DQ (EDF using a

three-dimensional queue) [9], (4) FFF (full frames first) [9], (5) FOFF (full ordered frames first) [10], (6) Mailbox switch [11], and (7) the Byte-Focal switch [12]. These proposed schemes can be categorized into two approaches. The second approach is to prevent packets from being received out-of-sequence at the outputs, for example, FFF and Mailbox switch. The second approach is to limit out-of-sequence packets to an upper bound, for example, $O(N^2)$, and then add a resequencing buffer (RB) at the output to reorder the packets. Such schemes include FCFS, EDF, EDF-3DQ, FOFF, and Byte-Focal switch.

14.3.1 First Come First Served (FCFS)

The FCFS scheme was proposed by Chang et al. [8], who are also the authors of the load-balancing Birkhoff–von Neumann switch. To tackle the out-of-sequence issue in the original LB-BvN switch, FCFS requires two additional buffers. With reference to Figure 14.9, the two additional buffers are: (1) the load-balancing buffer at every input of the switch, and (2) the RB at the output of the switch. The definitions of the terms used in Figure 14.9 are as follows:

1. FS(i, k). Flow splitter for flow S(i, k).
2. VCQ(i, j). Virtual central queue at input i, corresponding to output j of the first stage.
3. VOQ(j, k). Virtual output queue at input j, corresponding to output k of the second stage.

The operation of the switch is summarized as follows. Packets arriving at the switch are spread to virtual central queues (VCQs) for load balancing. This spreading is accomplished as follows. FCFS schemes have a flow-splitter for each flow at each input port that labels packets at each input as belonging to a particular flow S(i, k). Since there are N possible outputs, there are N possible flows, one corresponding to each of the N output ports. Next, a load balancer distributes all of the packets of a given flow to the N VCQs in a round-robin manner (there is a different load balancer for each flow) [10]. For each flow splitter, there is a pointer to keep track of the VCQ(i, j) that the previous packet from that flow has been sent to.

The first-stage switch follows a periodic deterministic pattern to connect an input i to an output j for one timeslot in each frame slot. A frame slot is defined as N timeslots. At timeslot t, the connection pattern (i, j) satisfies

$$i - 1 = (j - 1 + t) \bmod N, \tag{14.5}$$

where $t = 1, 2, \ldots, i = 1, 2, \ldots, N$, and $j = 1, 2, \ldots, N$.

The second-stage switch operates similar to the first one with a periodic deterministic pattern such that at time t, the connection pattern (j, k) satisfies

$$k - 1 = (j - 1 + t) \bmod N, \tag{14.6}$$

where $t = 1, 2, \ldots, j = 1, 2, \ldots, N$, and $k = 1, 2, \ldots, N$.

FCFS has jitter control in front of the second-stage buffers as shown in Figure 14.10. Packets from the same flow are distributed evenly in the first stage and arrive at the jitter control stage. Since packets of the same flow may experience different delays in the first-stage buffer due to the different lengths of VCQs, jitter control is used to restore packets'

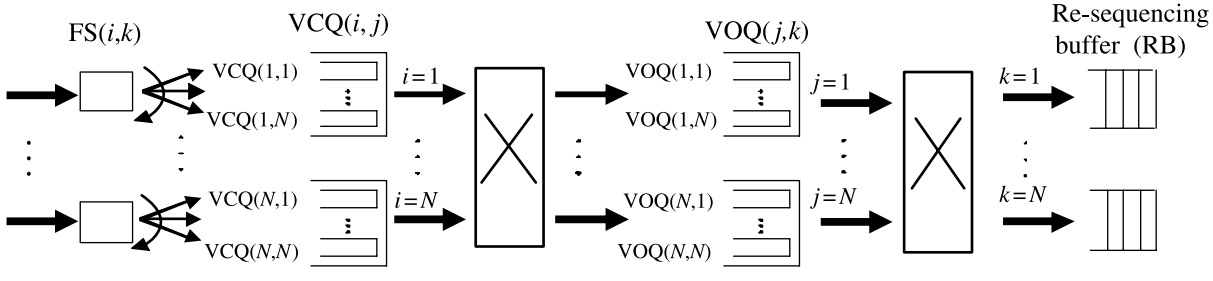

Figure 14.9 LB-BvN with out-of-sequence control.

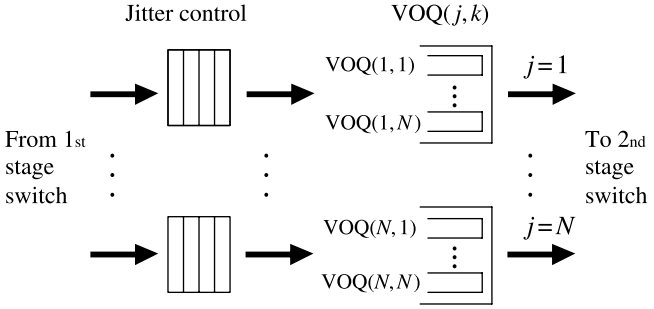

Figure 14.10 Second-stage buffers of FCFS.

arriving order at the second-stage buffer by imposing a delay, up to $N \times (N-1)$ timeslots, to each packet before joining the VOQs. In other words, every packet will experience the same amount of delay, $N \times (N-1)$ timeslots, from the time of their arrival at the VCQs to the time when they join the VOQs. However, this does not necessarily mean that their departures from the second-stage buffer will be in the same order as their arrivals. This is because they are likely to join VOQs with different lengths (although the difference is bounded by N cells). As a result, packets are out-of-sequence when they arrive at the output buffers. A RB is required at the outputs. For the FCFS, the jitter control increases the implementation complexity and increases every packet's delay by $N \times (N-1)$ timeslots.

The following example shows how the FCFS scheme resolves the packet out-of-sequence problem in the LB-BvN switch. Let us consider a flow S(1, 1), which implies a packet stream from input 1 to output 1. Packets Pa and Pb are two consecutive packets from flow S(1, 1) with sequence Pa $<$ Pb. As seen in the FCFS scheme described above in Figure 14.9, Pa and Pb split into two consecutive VCQ(1, j) and VCQ(1, $j+1$) in a round-robin fashion upon their arrival. Without losing generality, let us assume Pa and Pb are queued at VCQ(1, 1) and VCQ(1, 2), respectively, as shown in Figure 14.11. Since the queue length of each VCQ(i, j) is not uniform at all times, packets belonging to the same flow may depart from input VCQ(i, j) in an uncoordinated fashion. In this case, Pb will depart VCQ(1, 2)

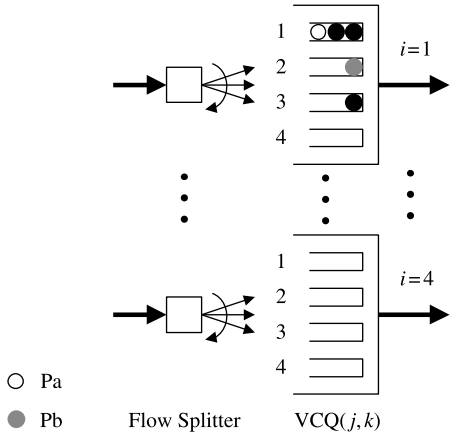

Figure 14.11 Pa and Pb are queued at VCQ(1, 1) and VCQ(1, 2), respectively.

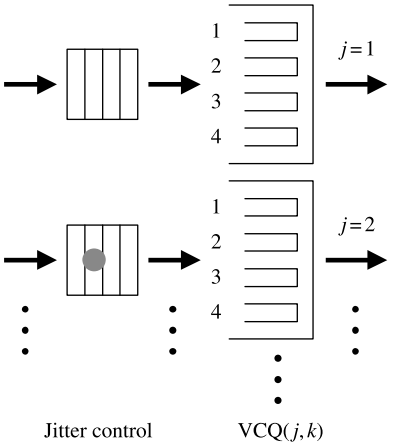

Figure 14.12 Pb is delayed in jitter control due to out-of-sequence caused by VCQ(i, j).

eight timeslots (two frame slots) earlier than Pa, which causes packet out-of-sequence problems. FCFS prevents this out-of-sequence problem by adding jitter control in front of VOQ(j, k). In this example, Pb is delayed in the jitter control stage by two frame slots (Fig. 14.12), so that it will enter VOQ(2, 1) at the same frame slot as Pa enters VOQ(1, 1). However, the queue lengths of VOQ(j, k) are not uniformly distributed from timeslot to timeslot either, as Figure 14.13 shows. Although the jitter-control stage resolves the out-of-sequence problem caused by input buffer VCQ(i, j), flow S(1, 1) may still experience the out-of-sequence problem due to VOQ(j, k). As a result, RBs are required at each output. The resequencing buffer reorders the packets so that packets of the same flow depart in the same order as they arrive. After resequencing, packets are stored in the output buffer until their transmission.

The FCFS scheme resolves the out-of-sequence problem in the original LB-BvN switch with the cost of: (1) two additional groups of buffers at inputs and outputs, respectively;

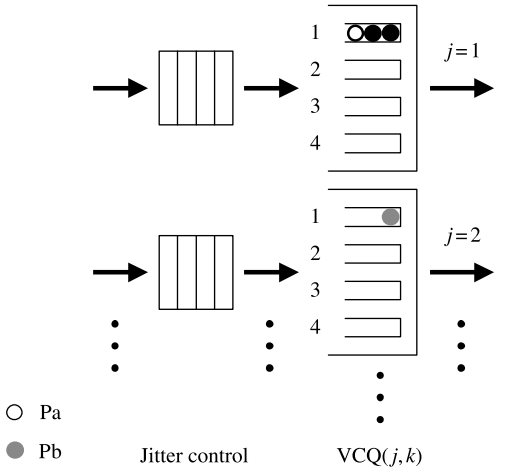

Figure 14.13 Pa and Pb are queued at VOQ(1, 1) and VOQ(2, 1) that have different queue lengths.

(2) additional average packet delay up to $N \times (N-1)$ timeslots due to the jitter control; and (3) increased implementation complexity by adding jitter control and the resequencing buffer. With jitter control, the FCFS is able to bound the resequence delay to N^2 cell. Thus, the RB size is bounded to N^2 cells.

14.3.2 Earliest Deadline First (EDF) and EDF-3DQ

The earliest deadline first (EDF) is another scheduling policy scheme proposed by Chang et al. [8] to resolve the packet out-of-sequence issue in the original LB-BvN switch. The EDF uses the same switch architecture and switch operation as the FCFS. However, the EDF scheme eliminates jitter control. Instead, it assigns a deadline to every packet to determine its departure from the second-stage buffer, VOQ(j, k). The deadline can be either the departure time of a corresponding output-buffered switch or simply the packet's arrival time. The packets in the second-stage buffer are served based on their deadline values. The earlier the deadline, the earlier the packet is served at the second-stage switch. Since packets arrive at VOQ(j, k)s in an uncoordinated fashion, the HOL packet is not necessarily carrying the earliest deadline. Searching the smallest timestamp in each VOQ is prohibitively complex and costly. Moreover, each flow S(i, k) may traverse different VOQ(j, k)s to reach the output port (k). Different lengths of VOQ(j, k) can still cause the mis-sequence problem. The EDF scheme requires a RB with a size of $2N^2 - 2N$ cells to reorder the cells.

EDF-3DQ [9] improves the EDF scheme by replacing the VOQs in the second-stage buffers with three-dimensional queues (3DQs). With reference to Figure 14.14, in the 3DQs structure, there is a different queue per (i, j, k); hence, there are a total of N^3 logical queues between first- and second-stage switches in EDF-3DQs. In the original VOQ structure, each VOQ(j, k) contains packets from multiple flows destined for output port k. In the 3DQ structure, each VOQ(j, k) has associated with it a total of N queues labeled 3DQ(i, j, k). Each of these N queues contains packets of the same flow. There are a total of N possible flows. Hence, there are N queues for each VOQ(j, k). For example, 3DQ(1, 1, 1) contains only packets of flow S(1, 1); 3DQ(2, 1, 1) only contains packets of flow S(2, 1). In the original VOQ structure, both of these flows may be placed in a single queue. So the objective of the 3DQ structure is to separate packets at each VOQ into their individual flows. With 3DQs, the earliest packet for (j, k) is always the HOL packet in its

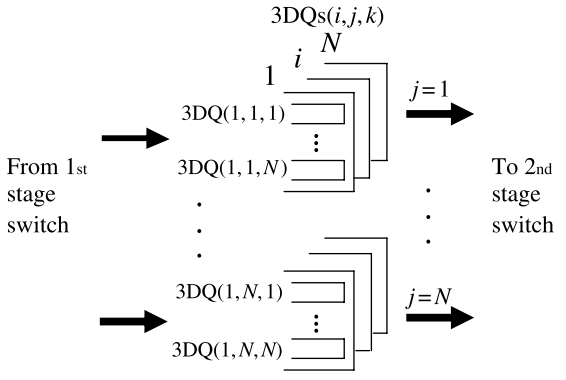

Figure 14.14 Three-dimensional queues (3DQs) in EDF.

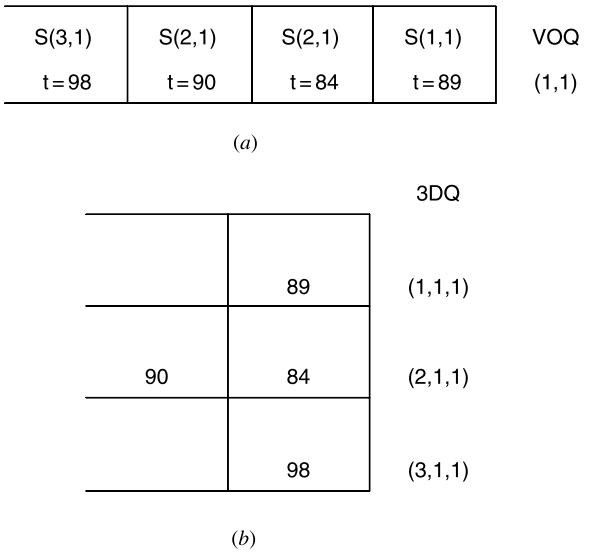

Figure 14.15 Comparison of (*a*) VOQ and (*b*) 3DQ structure.

queue. As a result, we only need to compare N HOL packets' timestamp to find the earliest deadline packet, instead of comparing $Q_{\max}$ timestamps in the original EDF scheme, where $Q_{\max}$ is one maximum queue length of the VOQ.

Figure 14.15 shows the effect of 3DQs in performing the EDF scheme. Figure 14.15*a* is an example of packet-queuing status in the original VOQ(j, k) structure. It can be easily seen that to find the earliest deadline packet, all packets in the queue need to be compared. In contrast, Figure 14.15*b* is the queuing status of the same packets in Figure 14.15*a*, but in a 3DQ format. Notice that the HOL packet of each 3DQ(i, j, k) always carries the earliest deadline. Therefore, to perform the EDF scheme, only a comparison of N timestamps is needed.

Although EDF-3DQ relaxes the constraints of searching the smallest timestamp in the complete queue, it still requires a comparison of N timestamps of N HOL packets of the 3DQs in each timeslot, limiting the switch size or the line rate. Moreover, EDF-3DQ still requires the same size of RB at each output port as in EDF, $2N^2 - 2N$.

14.3.3 Full Frames First (FFF)

This scheme uses a frame-based approach, called full frames first (FFF) [9], to solve the out-of-sequence problem in the LB-BvN switch. The FFF scheme is different from the other schemes in the sense that it completely eliminates the out-of-sequence problem and thus requires no RB buffers at the outputs. With reference to Figure 14.16, the switch architecture of the FFF scheme is similar to those of FCFS and EDF, except that FFF has 3DQs between two switch fabrics instead of VOQs in FCFS and EDF. Most importantly, FFF does not require any RBs.

Similar to FCFS and EDF, each traffic flow S(i, k) is split into N VCQ(i, j)s in a round-robin fashion upon their arrival at inputs. The first-stage switch fabric in FFF deterministically delivers packets from VCQ(i, j)s to 3DQ(i, j, k)s in a periodic manner.

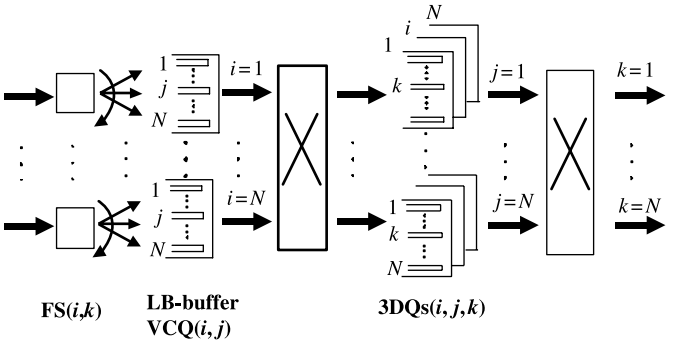

Figure 14.16 FFF scheme.

Likewise, the second-stage switch fabric also has a deterministic connection pattern for switching packets from 3DQ(i, j, k)s to their appropriate outputs. To prevent packet out-of-sequence due to the different queue lengths of the 3DQs, a scheduling algorithm is needed to serve packets from the 3DQs to the outputs.

In FFF, a candidate set of 3DQs for (i, k) consists of packets from (i, 1, k), (i, 2, k), and (i, N, k). It is important to remember that each 3DQ(i, j, k) contains packets from unique flows (packets from different flows will not be found in the same queue). Because of load balancing, a flow is uniformly distributed among all N 3DQs. Assume that the last serviced packet in the candidate set came from 3DQ(i, j_{last}, k). Because of the properties of the load-balancer and 3DQs, it is known that the next in-order packet for the flow S(i, k) will come from 3DQ($i, j_{(\text{last}+1) \bmod N}, k$). Let p_{ik} be the pointer to the 3DQ(i, j, k) of the next in-order packet: $p_{ik} = j_{(\text{last}+1) \bmod N}$. For instance, let Pa and Pb be two packets belonging to the same flow S(1, 1) with sequence Pa $<$ Pb. They will be queued at VCQ(1, 1) and VCQ(1, 2), respectively, upon their arrival at their inputs (Fig. 14.17*a*). Although Pa and Pb will transfer to 3DQs in different timeslots, one can expect they will be queued at 3DQ(1, 1, 1) and 3DQ(1, 2, 1), respectively. In other words, if Pa is from 3DQ(1, 1, 1), then the next in-order packet will necessarily be read from the 3DQ(1, 2, 1) with $p_{ik} = 1 + 1 = 2$, which is Pb in this example (Fig. 14.17*b*).

In FFF, a cycle is defined as N consecutive timeslots; a frame for flow (i, k) is defined as $f(i, k) = (i, p_{ik}, k), (i, p_{ik} + 1, k), \ldots, (i, N, k)$; and a frame is full if every 3DQ(i, j, k) for $j = p_{ik}, \ldots, N$ is non-empty. One can easily see that if the frame is full, then the second-stage switch fabric can continuously transfer in-order packets from 3DQ(i, p_{ik}, k) up until 3DQ(i, N, k). This is the key to preventing out-of-sequence packets in the FFF. Searching for full frames is performed once every cycle. An output reads all the packets in a full frame from one input, before moving on to read a full frame from the next input. Each output, k, uses a round-robin pointer $p_{ff}(k)$ to remember which input the last full frame came from, so that each output gives an opportunity to each input in turn to send a full frame to it. If there are no full frames, output k serves the non-full frames in a round-robin manner by using a pointer $p_{nff}(k)$.

More precisely, the FFF scheme consists of three computation steps for each output port k at the beginning of every cycle:

Step 1: Determine which of the frames f(i, k) is full.

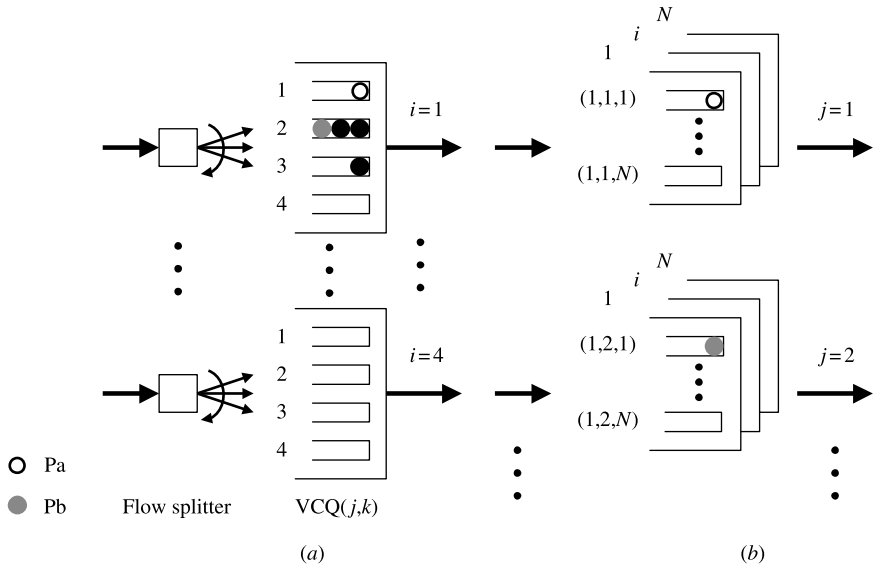

Figure 14.17 Property of flow splitter and 3DQs. (*a*) Pa and Pb at input VCQ(j,k); (*b*) Pa and Pb at 3DQ(i,j,k).

Step 2: Starting at $p_{ff}(k)$, find the first full frame and then update the full frame pointer $p_{ff}(k)$ to that input associated with the full frame.

Step 3: If there is no full frame, starting at $p_{nff}(k)$, find the first non-full frame, then update the non-full frame pointer $p_{nff}(k)$ accordingly.

Next, we use an example to illustrate FFF in more detail. Let us assume a 3×3 LB-BvN switch with the FFF scheduling algorithm. At the beginning of a cycle, the 3DQs for output 1 are in the state shown in Figure 14.18. To better explain the algorithm, we rearrange the 3DQs so that all of the queues containing packets from a given input are adjacent to each other (Fig. 14.19). The numbers indicate the packet sequence number within its flow. Let us also assume that $p_{ff}(1) = p_{nff}(1) = 3$; and the frame pointer $p_{11}(1)$, $p_{21}(1) = 3$, and $p_{31}(1) = 1$.

At the first timeslot, FFF serves the first full frame for input 3 because of $p_{ff}(1) = 3$. The first full frame consists of *ff* 1 = 136, 137, and 138. After serving this full frame, pointers are updated as $p_{ff}(1) = 1$ and $p_{31}(1) = 1$. In the next cycle, FFF serves the three packets from input 1 in frame *ff* 2 = 190, 191, and 192, then updates $p_{11}(1) = 1$ and $p_{ff}(1) = 2$. According to the definition, *ff* 3 = 57 is a full frame from input 2, though it only contains one packet. A frame is said to be full if and only if it is possible to transfer in-order cells from (i, p_{ik}, k) up until (i, N, k). The FFF serves it next and updates the pointers. After that, there is no full frame from input 3, but inputs 1 and 2 still have full frame. The FFF serves *ff* 4 and *ff* 5 in the next two cycles. Then, the pointers are updated as follows: $p_{ff}(1) = p_{nff}(1) = 3$, and $p_{11}(1) = p_{21}(1) = p_{31}(1) = 1$.

When there are no full frames left in the system, the FFF serves the non-full frames in round-robin order: *nff* 1, *nff* 2, and *nff* 3. Pointers are updated to $p_{ff}(1) = p_{nff}(1) = 3$, $p_{11}(1) = 2$, $p_{21}(1) = 3$, and $p_{31}(1) = 3$. Note that packet 198 from input 1 will not be

196	193	190	(1, 1, 1)
61	58		(2, 1, 1)
142	139	136	(3, 1, 1)

	194	191	(1, 2, 1)
62	59		(2, 2, 1)
143	140	137	(3, 2, 1)

198	195	192	(1, 3, 1)
	60	57	(2, 3, 1)
		138	(3, 3, 1)

Figure 14.18 Illustration of FFF algorithm in LB-BvN switch.

served due to missing packet 197. Similarly, packets 142 and 143 will not be served either, since $p_{31}(1) = 3$, the pointer is still waiting for packet 141.

The FFF features most of the benefits of the original LB-BvN switch with the benefit of the first in, first out (FIFO) service discipline. However, it requires complex 3DQs between

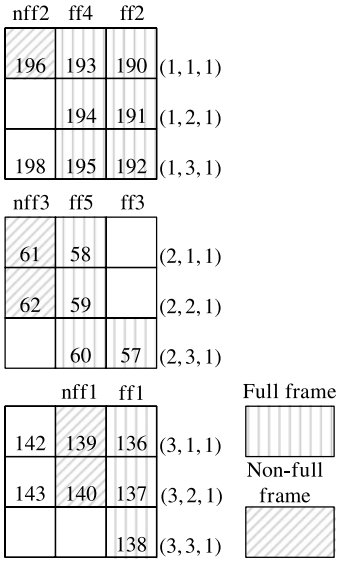

Figure 14.19 Illustration of FFF algorithm in LB-BvN switch.

two switch fabrics and a large amount of communication overhead flowing between the line cards to search for full frames.

14.3.4 Full Ordered Frames First (FOFF)

Full ordered frame first (FOFF) is another frame-based scheduling scheme to resolve the packet out-of-sequence problem in the LB-BvN switch [10]. Unlike FFF, FOFF allows packets to be out-of-sequence through the two switch stages. With reference to Figure 14.20, the switch architecture of the FOFF scheme consists of three groups of buffers and two deterministic TDM switch fabrics. The three groups of buffers are: (1) VOQ1(i, k) at every input, each of which associates with output port k; (2) VOQ2(j, k) between two switch fabrics, each of which associates with output port k; and (3) RB VCQ(j, k) at outputs, each of which is dedicated to the inputs of the second switch fabric.

Packets are queued into VOQ1(i, k) upon their arrival as they are in the traditional input-buffered switch. At the beginning of each frame slot (1 frame slot = N timeslots), each input selects a VOQ1 that will send packets in the next frame slot. Full frames in different VOQ1s are first selected and served in a round-robin manner. If there are no full frames, partial frames are selected and served in a round-robin manner. When a partial frame is chosen to send in the next frame slot, there will be some bandwidth waste in the first-stage switch.

If at least one full frame from each input can be found in every frame slot, there will be no out-of-sequence problem. However, if an input can only send a partial frame, then because of the difference in the occupancies at the VOQ2s, packets will be out-of-sequence as they arrive at the output. But, this out-of-sequence is bounded. It has been proven that a re-sequence buffer of size N^2 at each output of the switch is enough to re-sequence the packets [10]. In other words, when there are $N^2 + 1$ packets in the RB, at least one of the HOL packets of the VCQs is a head-of-flow (HOF) packet and can be selected to transmit.

Partial frames can cause another problem in the FOFF besides the out-of-sequence problem. It can cause bandwidth waste in the first-stage switch and thus increase the average delay of packets. Figure 14.21 illustrates the bandwidth waste. Assume input 1 selects a partial frame to send in the next frame slot ($T_f = 2$), where the partial frame has K packets, $K < N$. For simplicity, assume that input 1 sends its first packet of the frame to the first

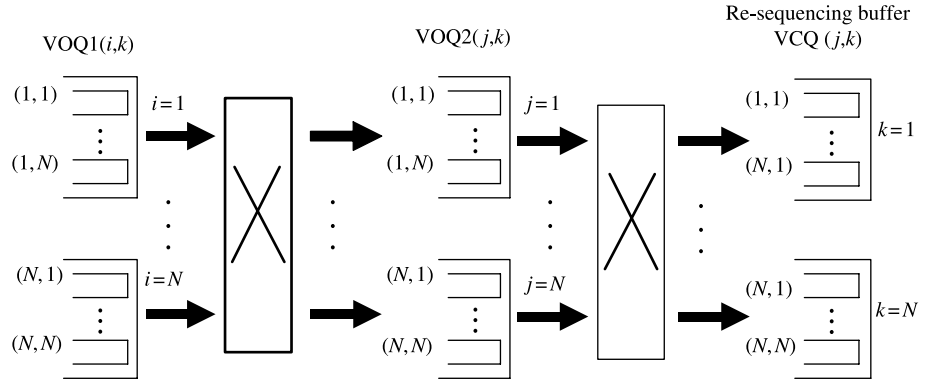

Figure 14.20 Queue structure of the FOFF Scheme.

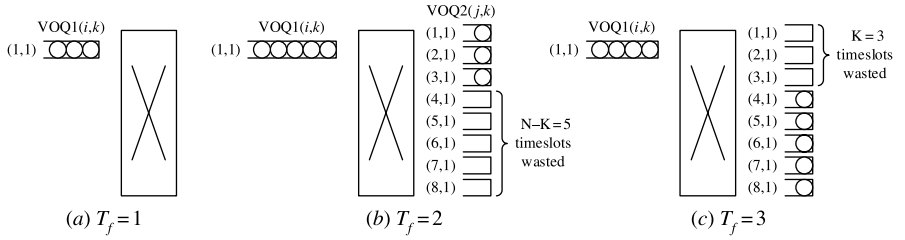

Figure 14.21 Example of bandwidth waste for the first-stage switch in FOFF. T_f shows the frame slot number. $N = 8$ timeslots are wasted to send one partial frame with $K = 3$ packets. Note that packets with circles constitute a full frame.

output of the first-stage switch (i.e., $j = 1$). At the end of this frame slot, input 1 sends K packets and it wastes $N - K$ slots without sending any packets. In the next frame slot, a frame with $N - K$ packets at input 1 is defined as a full frame and is selected to transmit at $T_f = 3$. In $T_f = 3$, only $N - K$ packets can be sent, even if there are packets waiting in the same VOQ1, for example, VOQ1(1, 1) in Figure 14.21c, wasting another K timeslots. As a consequence, any partial frame will waste up to N timeslots regardless of its partial frame size. Since the first stage is not work-conserving in terms of packets but only in terms of frames, no packets can be sent over the remaining timeslots for a partial frame. This increases the average delay.

The resequencing operation at the output could be quite challenging. It has been proven that there is at least one packet out of at most $N^2 + 1$ packets that is eligible to send. In each timeslot, we may need to search up to N HOL packets of the VCQs before finding a HOF packet. A HOF packet of a flow is determined by comparing its sequence number with an expected sequence number for the flow. In each timeslot, each output performs the following operations to find a HOL packet:

Step 1: Determine which flow the HOL packet belongs to.

Step 2: Access the sequence number of the flow's HOF packet, for instance, from a table.

Step 3: Compare it with the HOL packet's sequence number.

Step 4: If matched, send the HOL packet and update to the flow's next sequence number.

Step 5: Else, repeat steps 1 through 4 for the next VCQ's HOL packet.

In the worst case, there could be up to N repetitions of the above operations before finding a HOF packet to send.

14.3.5 Mailbox Switch

Another scheme to prevent the packet out-of-sequence problem is called the Mailbox switch [11]. As shown in Figure 14.22, the mailbox switch consists of two sets of buffers and two deterministic switch fabrics. The buffers at inputs are simply FIFO queues, and the buffers between the two switch fabrics are called mailboxes. There are a total of N mailboxes. Each mailbox contains N bins (indexed from 1 to N). Each bin contains F packets. For instance, a packet stored in the ith bin of a mailbox is destined for the ith output port. Two switch fabrics have symmetric connection patterns over time such that at time t, input

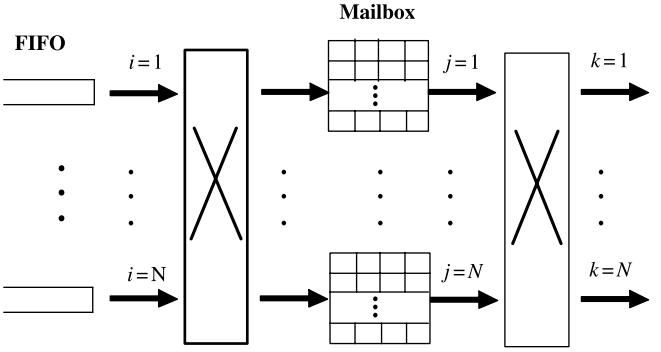

Figure 14.22 Mailbox switch architecture.

port i is connected to output port j if $(i + j) \bmod N = (t + 1) \bmod N$. From this condition, at time $t = 1$, input port 1 is connected to output port 1, input port 2 is connected to output port $N, \ldots$, and input port N is connected to output port 2. Specifically, input port i is connected to output port 1 at time i, output port 2 at time $i + 1, \ldots$, output port N at time $i + N - 1$. Note that input port i and output port j are connected if and only if input port j and output port i are connected. Any switch fabric implementing such a connection pattern is referred to as a symmetric TDM switch. One can solve j in $(i + j) \bmod N = (t + 1) \bmod N$ with the following function:

$$j = h(i, t) = [(t - i) \bmod N] + 1.$$

Thus, during the tth time slot, the ith input port is connected to the $h(i, t)$th output port of these two crossbar switch fabrics.

So, how do we solve the packet out-of-sequence problem? Since everything inside the switch is predetermined and periodic, the scheduled packet departure times can be fed back to inputs to compute the waiting time for the next packet so that packets can be scheduled in the order of their arrivals. This is possible because an input port of the first switch and the corresponding output port of the second switch are, in general, built on the same line card. The switch operation can be summarized as follows:

1. *Retrieve Mail.* At time t, the kth output port of the second switch is connected to the $h(k, t)$th mailbox. The packet in the first cell of the kth bin is transmitted to the kth output port. Packets in cells $2, 3, \ldots, F$ of the kth bin are moved forward to cells $1, 2, \ldots, F - 1$.
2. *Sending Mail.* Suppose that the HOL packet of the ith input port of the first switch is from flow $S(i, j)$. Note that the ith input port of the first switch is also connected to the $h(i, t)$th mailbox. To keep packets in sequence, this HOL packet is placed in the first empty cell of the jth bin of the $h(i, t)$th mailbox such that it will depart no earlier than $t + V_{i,j}(t)$. If no such empty cell can be found, the HOL packet is blocked and it remains the HOL packet of that FIFO.
3. *Updating Virtual Waiting Times.* All the flows that do not send packets at time t decrease their virtual waiting time by 1, including flows that have blocked transmissions. To update the virtual waiting time for flow $S(i, j)$, suppose that the

HOL packet is placed in the fth cell of the jth bin of the $h(i, t)$th mailbox. As the connection patterns are deterministic and periodic, one can easily verify that the $h(i, t)$th mailbox will be connected to the kth output port of the second-stage switch at $t + ((k - j - 1) \bmod N) + 1$. Thus the departure time for this packet is $t + (f - 1)N + [(k - j - 1) \bmod N] + 1$. As such, the number of timeslots that have to pass at $t + 1$ for flow (i, j) is $(f - 1)N + [(j - i - 1) \bmod N]$ and we have $V_{i,j}(t + 1) = (f - 1)N + [(j - i - 1) \bmod N]$.

Since the length of each mailbox is limited, and searching for an empty proper cell to place packets requires multiple tries, each unsuccessful try could result in backing off N timeslots for the packet departure time. Such back-off not only affects the packet being placed, but also affects all the subsequent packets that belong to the same flow because the virtual waiting time of that flow is also increased by N timeslots. To avoid such issues, it is better to limit the number of forward tries (δ) that a cell can attempt, such that, after searching δ cells beyond the virtual waiting time, the packet just simply gives up on that timeslot. However, δ is a very vital parameter in the Mailbox switch. As shown in Figure 14.23, when δ is small, the system throughput is limited by the HOL blocking at the FIFO queues of the first switch. On the other hand, when δ is large, the throughput is limited by the stability of the virtual waiting times.

To achieve better throughput, one may also search for an empty cell with a limited number of backward tries δ_b. By doing so, packets in the Mailbox switch might be out of sequence, but the resequencing delay is bounded. Figure 14.24 is the performance result of the Mailbox switch's limited forward tries and backward tries. We can see that the Mailbox switch can achieve a maximum throughput of 0.95.

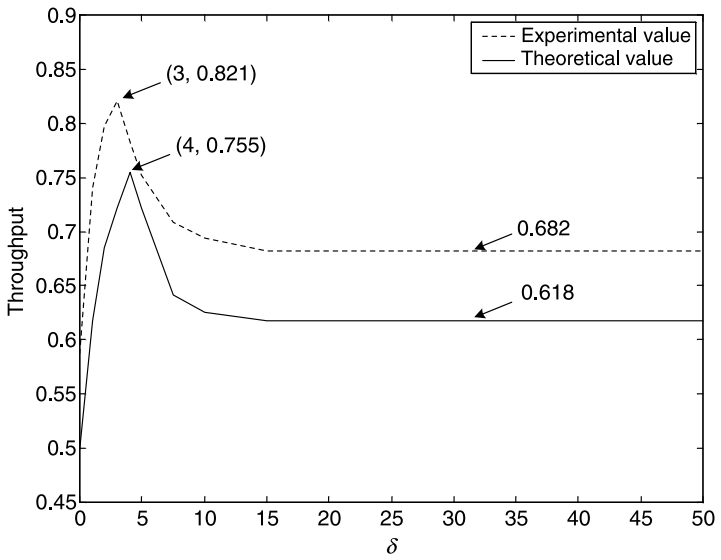

Figure 14.23 Mailbox switch throughput as a function of forward tries δ.

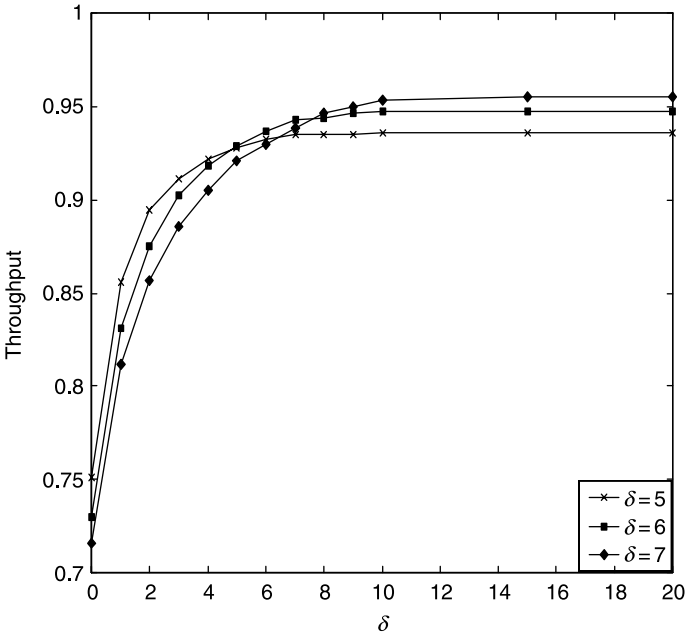

Figure 14.24 Maximum throughput as a function of δ_b in the mailbox switch.

14.3.6 Byte-Focal Switch

This section describes a practical load-balanced switch, called the Byte-Focal switch, which uses packet-by-packet scheduling to significantly improve the delay performance over switches of comparable complexity. It is called "Byte-Focal" to reflect the fact that packets of a flow (traffic from an input to an output) are spread to all line cards and brought to a focal point (the destined output). The Byte-Focal switch is simple to implement and highly scalable. It does not need a complex scheduling algorithm, or any communication between linecards, while achieving 100 percent throughput.

The Byte-Focal switch is based on packet-by-packet scheduling to maximize the bandwidth utilization of the first stage and thus improve the average delay performance. Figure 14.25 shows the Byte-Focal switch architecture. It consists of two deterministic switch fabrics and three stages of queues, namely, input queue i, center stage queue j, and output RB k, where $i, j, k = 1, 2, \ldots, N$. The deterministic switch fabrics operate the same way as the basic LB-BvN switch (Section 14.3.1), where both stages use a deterministic and periodic connection pattern.

There are two stages of VOQs in the Byte-Focal switch, VOQ1 and VOQ2 for the first- and second-stage switch, respectively. The flow f_{ik} is defined as the packets arriving at the input port i and destined to output port k. As shown in Figure 14.26, packets from f_{ik} are placed in VOQ1(i, k). Since at each time slot, the input port at the first stage is connected to the second stage cyclically, the packets in VOQ1(i, k) are sent to the N second-stage input ports in a round-robin manner and are placed in VOQ2$(1, k)$, VOQ2$(2, k)$, $\ldots$, VOQ2(N, k) according to their final destination. The Byte-Focal switch guarantees that the cumulative number of packets sent to each second-stage input port for a given flow differs by at most one. The VOQ2 is then served by the second fixed, equal-rate switch. Since the packets, in

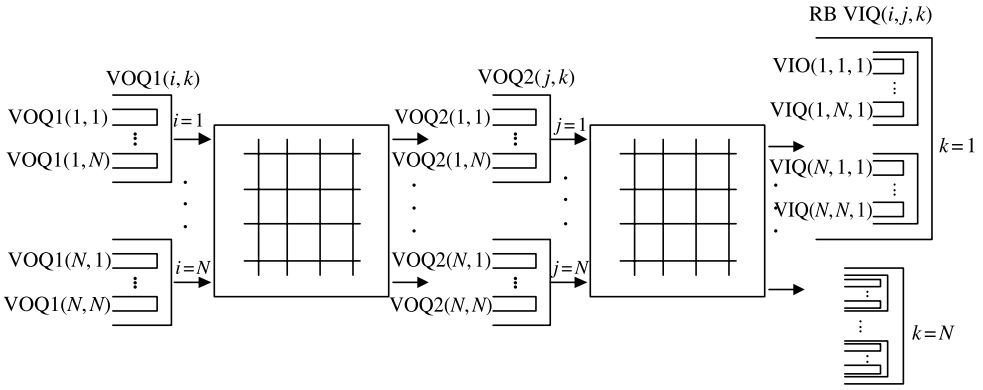

Figure 14.25 The Byte-Focal switch architecture.

general, suffer different delays in the second stage, they arrive at the output out-of-order (see Fig. 14.26 for an example).

The Byte-Focal switch uses the virtual input queue (VIQ) structure for the RB. At each output, there are N sets of VIQs where each set corresponds to an input port i. Within each VIQ set, there are N logical queues with each queue corresponding to a second-stage input j. VIQ(i, j, k) separates each flow not only by its input port i, but also by its second-stage queue j. Packets from input i destined to output k via second-stage input j are stored in VIQ(i, j, k). It is obvious that the packets in the same VIQ(i, j, k) are in order.

The HOF packet is defined as the first packet of a given flow that has not yet left the switch, and the HOL packet as the first packet of a given VIQ(i, j, k) queue. In each VIQ set, a pointer points to the VIQ(i, j, k) at which the next expected HOF packet will arrive. Because of the service discipline of the first-stage switch, each input port evenly distributes packets in a round-robin order into the second-stage queue j. This guarantees that the HOF packet appears as a HOL packet of a VIQ set in a round-robin order. Therefore, at each time slot, if the HOF packet is at the output, it is served and the pointer moves to the next HOF packet location VIQ(i, $(j + 1)$ mod N, k).

Since there are N flows per output, more than one HOF packet may be eligible for service in a given time slot. Therefore, in addition to the VIQ structure, there is a departure queue (DQ) with a length of at most N entries that facilitates the round-robin service discipline. The DQ is simply a FIFO logical queue. It stores the indices of the VIQ sets, but only one from each VIQ set. When the HOF packet of VIQ set i arrives, index i joins the tail of the DQ. When a packet departs from the DQ, its index is removed from the head of the DQ and

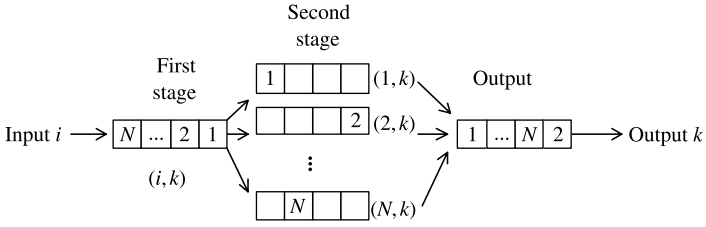

Figure 14.26 Example flow of packets from VOQ1(i, k) through the switch.

joins the tail of the DQ if its next HOF packet has arrived. The advantage of using the VIQ and the DQ structure is that the time complexity of finding and serving packets in sequence is $O(1)$. At each time slot, each VIQ set uses its pointer to check if the HOF packet has arrived, while the output port serves one packet from the head of the DQ.

As explained above, the VIQ structure ensures that the Byte-Focal switch will emit packets in order.

First-Stage Scheduling. In the effort to improve the average delay performance, the scheduling scheme at the first stage plays a very important role in the Byte-Focal switch. The packets in VOQ1(i, k) are cyclically distributed to the second stage. As a result, when the first-stage input port i is connected to the second-stage input port j, only some of the VOQ1s at i are eligible to send packets to j. As shown in Figure 14.27, this problem can be stated as follows:

> Each VOQ1(i, k) has a *J pointer* that keeps track of the last second-stage input to which a packet was transferred, and the next packet is always sent to the next second-stage input. As a HOL packet departs from a VOQ1(i, k), its J pointer value increases by one mod N. When input i is connected with j, each VOQ1(i, k) whose J pointer value is equal to j sends a request to the arbiter, and the arbiter selects one of them to serve.

The Byte-Focal switch performs the first-stage scheduling independently at each input port using locally available information. Thus, it does not need any communication between different linecards.

Let $P_{ik}(t)$ be the J pointer value for VOQ1(i, k) at time t. Define a set $S_j(t) = \{\text{VOQ1}(i, k) | P_{ik}(t) = j\}$, then $S_j(t)$ is the set of VOQ1s that can send packets to the second-stage input j at time t. Four ways of picking a VOQ1 to serve from the set $S_j(t)$ are considered next.

Round-Robin. To achieve a small delay while maintaining fairness among all traffic flows, an efficient arbitration is necessary to schedule the departure of the HOL packets of the VOQ1s. One simple way to do the first-stage scheduling is to use the round-robin scheme. In round-robin arbitration, in the set $S_j(t)$, the arbiter at each input port selects one of them in round-robin order. This scheme is simple and easy to implement.

Longest Queue First. Although the round-robin arbitration achieves fairness among all the traffic flows, under non-uniform traffic conditions, some congested VOQ1s could

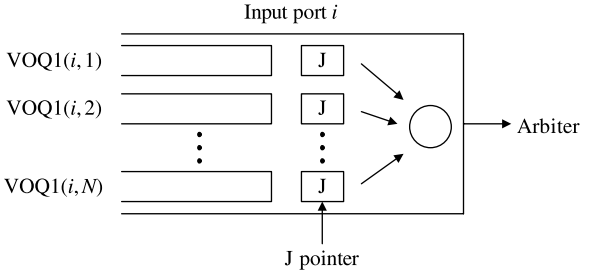

Figure 14.27 Scheduling schemes at the first stage.

overflow and the system becomes unstable (see the simulation results in Fig. 14.29). To stabilize the system, high priority is given to the congested VOQ1s. The longest queue first (LQF) algorithm ensures that, at each time slot, the arbiter at each input port chooses to serve the longest queue from the set $S_j(t)$.

Fixed Threshold Scheme. The longest queue first scheme can achieve good performance. However, finding the longest queue can be time-consuming and is not practical for high-speed large-scale switches. It is easier to identify the congested VOQ1s by observing if their queue length exceeds a predetermined threshold (TH), N. Let $q_{ik}(t)$ be the length of the VOQ1(i,k), and $q_{is}(t)$ be the length of the VOQ1(i,s) being served. Define a subset $S_j^{'}(t) = \{\text{VOQ1}(i,k) | \text{VOQ1}(i,k) \in S_j(t) \text{ and } q_{ik}(t) \geq TH\}$, then $S_j^{'}(t)$ is the set of VOQ1s that have more than TH cells and can send cells to j. The fixed threshold algorithm is:

1. At each time slot, if $q_{is}(t) \geq TH$, continue to serve this queue.
2. If not, the arbiter picks in a round-robin manner among the queues in set $S_j^{'}(t)$.
3. If $S_j^{'}(t)$ is empty and $q_{is}(t) > 0$, then it keeps serving the queue corresponding to $q_{is}(t)$.
4. If $q_{is}(t) = 0$, pick in a round-robin manner among the queues in set $S_j(t)$.

Dynamic Threshold Scheme. As the switch size becomes large, setting the threshold to switch size N causes large average delays. The reason is that the VOQ1 length has to reach a large value (N) before being identified as congested. Before reaching the threshold, it competes with other VOQ1s that have much smaller queue lengths. To better identify congested queues under different switch sizes and different traffic loadings, the dynamic threshold scheme is proposed. The dynamic threshold (TH) value is set to $Q(t)/N$, where $Q(t)$ is the total VOQ1 queue length at an input port at time t. $Q(t)/N$ is therefore the average VOQ1 queue length. The dynamic threshold scheme operates in the same way as the fixed threshold scheme except that the threshold is now set to the average queue length for that input.

In the simulation study, it is assumed that the switch size $N = 32$, unless otherwise noted. All inputs are equally loaded on a normalized scale $\rho \in (0, 1)$, and use the following traffic scenarios to test the performance of the Byte-Focal switch:

Uniform i.i.d. $\lambda_{ij} = \rho/N$.

Diagonal i.i.d. $\lambda_{ii} = \rho/2$, $\lambda_{ij} = \rho/2$, for $j = (i+1) \bmod N$. This is a very skewed loading, since input i has packets only for outputs i and $(i+1)$ *mod* N.

Hot-spot. $\lambda_{ii} = \rho/2$, $\lambda_{ij} = \rho/2(N-1)$, for $i \neq j$. This type of traffic is more balanced than diagonal traffic, but obviously more unbalanced than uniform traffic.

Normally, for single-stage switches, the performance of a specific scheduling algorithm becomes worse as the loadings become less balanced.

This section compares the average delay induced by different algorithms. As seen in Figure 14.28, the frame-based scheduling scheme, FOFF, has a much larger delay. The reason is that FOFF wastes bandwidth whenever a partial frame is sent. At low traffic load, many frames will be sent as partial frames, resulting in considerable bandwidth wastage at the first stage. From the figure, it can be seen that at low load, the delay difference between

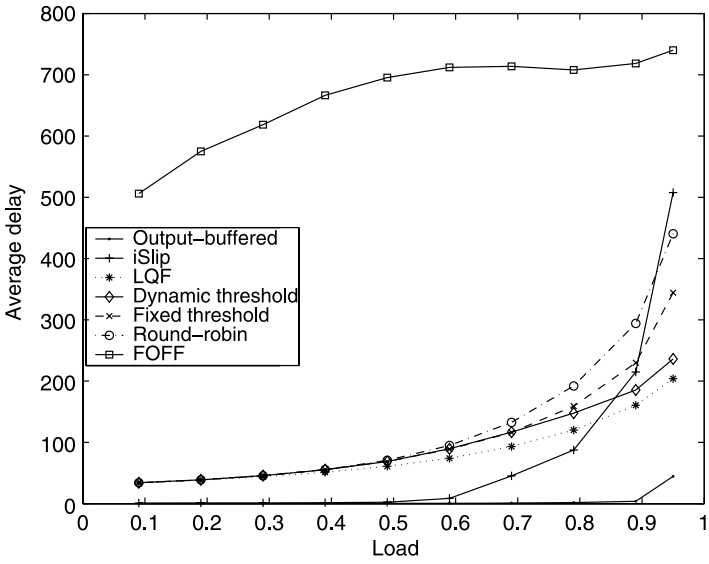

Figure 14.28 Average delay under uniform traffic.

FOFF and the Byte-Focal switch is quite large. The Byte-Focal switch performs packet-by-packet scheduling instead of frame-based scheduling, so it reduces the bandwidth wastage. At high traffic load, the Byte-Focal switch also achieves better performance than the FOFF. Compared to a single-stage algorithm, *i*Slip, when the loading is low, *i*Slip has a smaller average delay, but when the switch is heavily loaded, the Byte-Focal switch distributes the traffic evenly to the second stage, thus dramatically reducing the average delay.

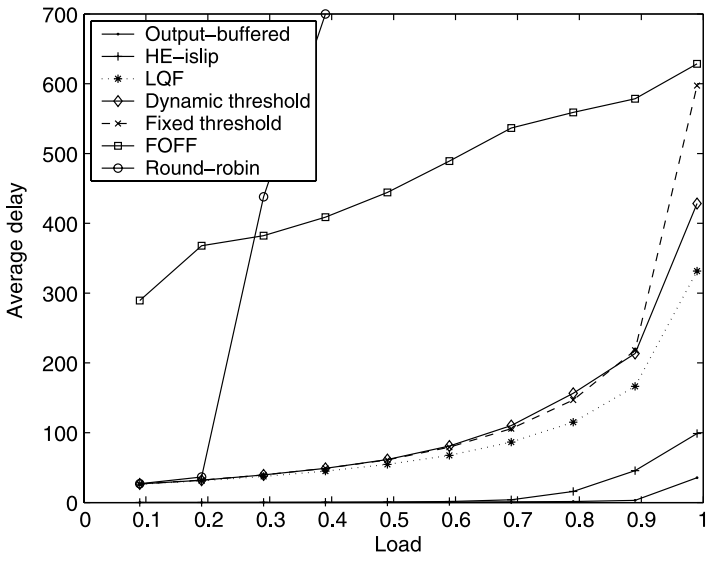

Figure 14.29 Average delay under hot-spot loading.

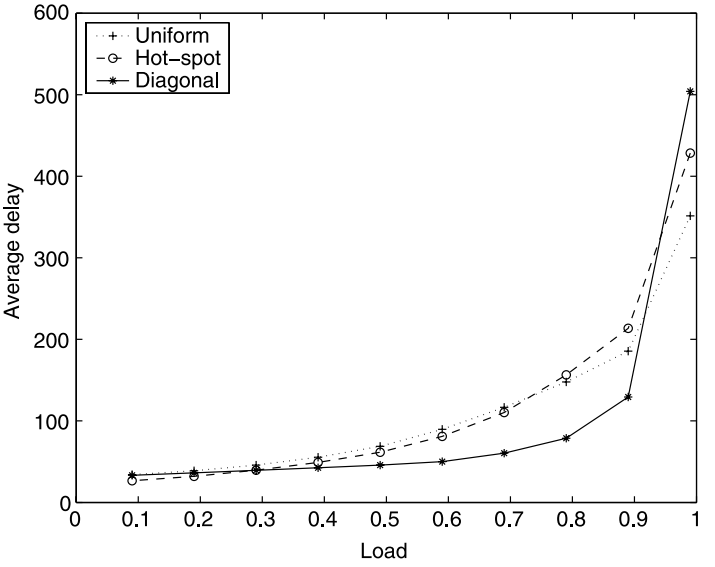

Figure 14.30 Average delay for the dynamic threshold scheme.

Figure 14.29 shows the average delay of various schemes under hot-spot loading. Although the round-robin scheme is simple to implement, it is not stable under non-uniform loadings (as seen in the figure, the throughput is only about 30 percent). For reference, the performances of a typical single-stage switch, HE-*i*Slip [13] and the ouput-buffered switch are also provided. The LQF scheme has the best delay performance among the Byte-Focal switch schemes, but, unlike the fixed and dynamic threshold schemes, it is not practical due to its high implementation complexity. Figure 14.29 shows that the dynamic threshold scheme performance is comparable to the LQF scheme. Compared to the fixed threshold scheme, the dynamic threshold scheme can adapt to the changing input loadings, thus achieving a better delay performance, while maintaining low complexity.

Figure 14.30 shows the average delay performance of the dynamic threshold scheme under different input traffic scenarios. As the input traffic changes from uniform to hot-spot to diagonal (hence less balanced), the dynamic threshold scheme can achieve good performance, especially for the diagonal traffic. The diagonal loading is very skewed and difficult to schedule using the centralized scheduling architecture. For the Logdiagonal traffic matrix [14], the delay performance is comparable to hot-spot loading. The Byte-Focal switch performs load-balancing at the first stage, thus achieving good performance even under extreme non-uniform loadings. This greatly simplifies the traffic engineering design.

Figure 14.31 shows the average delays for the dynamic threshold scheme with different switch sizes with the load kept fixed at 0.95. As shown in the figure, under the input traffic models that are considered, the delay increases as the switch size increases, and the average delays are almost linear with the switch size. Since the Byte-Focal switch does not use a centralized scheduler, it can scale well and achieve good performance even for very large switch sizes.

A cell in the Byte-Focal switch experiences queuing delays at the first and second stage, and resequencing delay at the output. Figure 14.32 shows the three components of the total delay. As can be seen, the first-stage queuing delay and the second-stage queuing delay are

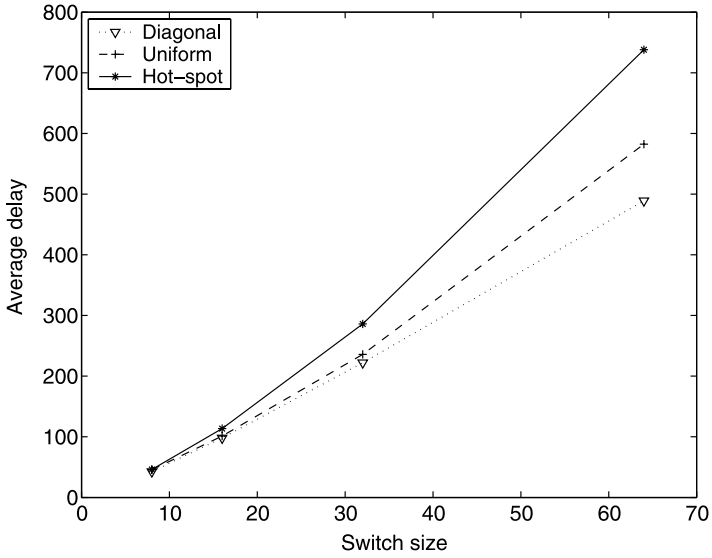

Figure 14.31 Average delay versus switch size N.

comparable, and the resequencing delay is much smaller compared to the other two delays. Figure 14.33 shows the resequencing delay increases as the switch size increases.

Since Internet traffic is bursty [15], let us consider the delay performance under bursty traffic. The burst length is set to be 10 cells. At a particular input port, after a burst, the probability that there is another burst arriving is μ, and the probability that there is no

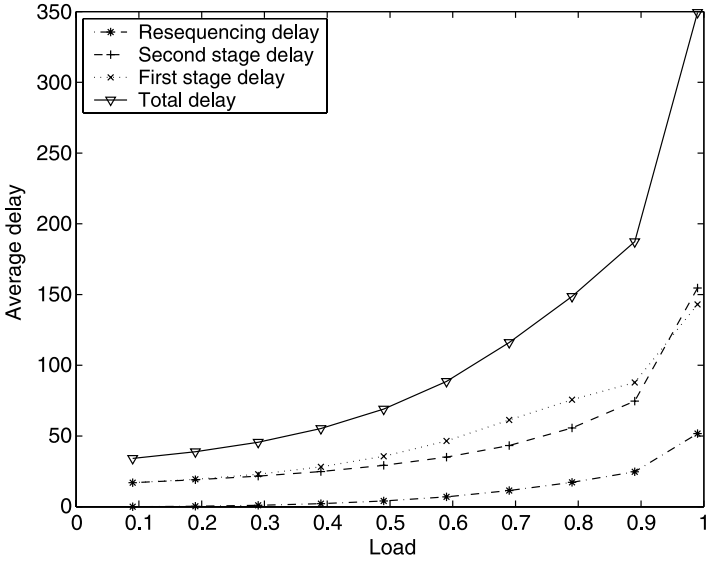

Figure 14.32 3-stage delays under uniform traffic for the dynamic threshold scheme.

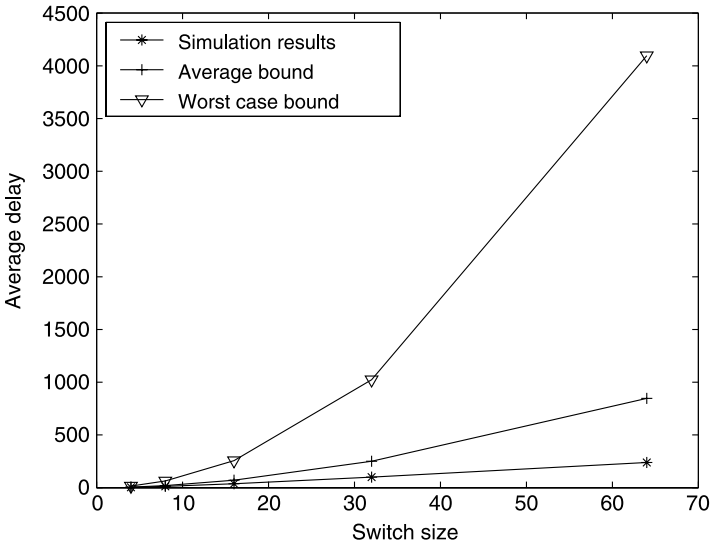

Figure 14.33 Resequencing delay with different switch size.

packet arriving corresponding to the next burst is $1 - \mu$. Then the loading to this input port is $\rho = 10\mu/(1 + 9\mu)$. Two scenarios are considered:

Bursty 1. Cells within the same burst are uniformly distributed to the N output ports.

Bursty 2. Cells within the same burst are destined to the same destination; however, bursts are uniformly distributed over N output ports.

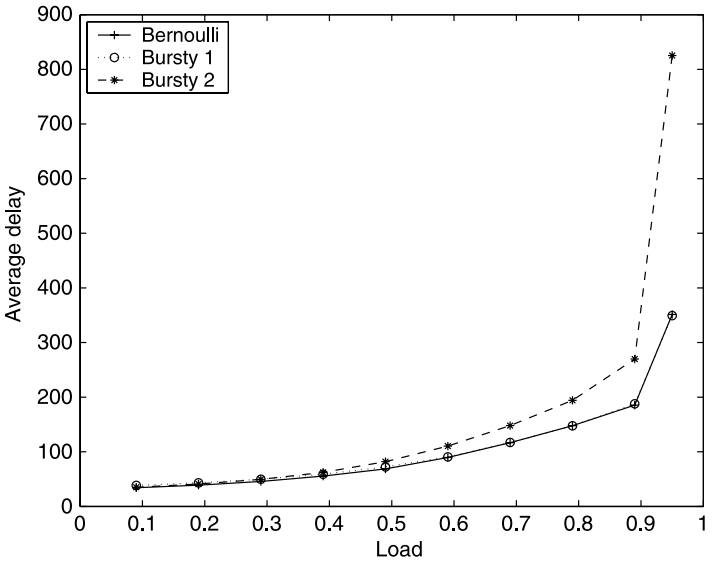

Figure 14.34 Average delay of the dynamic threshold scheme under bursty traffic.

Figure 14.34 shows the average delay of the Byte-Focal switch with the dynamic threshold scheme under the Bernoulli and bursty traffic models. It can be seen that the average delays under the Bernoulli and Bursty 1 traffic scenario are identical. In comparison with the single-stage switches, the Byte-Focal switch achieves considerable burst reduction. Therefore, it is very effective in reducing the average delay. From our simulations, the delay performance is worse for Bursty 2 as compared to Bursty 1 when the traffic load is high.

REFERENCES

[1] N. McKeown, "*i*SLIP: a scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201 (Apr. 1999).

[2] H. J. Chao, "Saturn: a terabit packet switch using dual round-robin," *IEEE Communications magazine*, vol. 38, no. 12, pp. 78–84 (Dec. 2000).

[3] C. S. Chang, D. Lee, and Y. Jou, "Load balanced Birkhoff-von Neumann switches. Part I: One-stage buffering," *Computer Communications*, vol. 25, no. 6, pp. 611–622 (2002).

[4] G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Univ. Nac. Tucum an Rev. Ser. A*, vol. 5, pp. 147–151 (1946).

[5] J. von Neumann, "A certain zero-sum two-person game equivalent to the optimal assignment problem," in *Contributions to the Theory of Games*. Princeton University Press, Princeton, NJ, vol. 2, pp. 5–12, 1982.

[6] C. Chang, W. Chen, and H. Huang, "On service guarantees for input buffered crossbar switches: a capacity decomposition approach by Birkhoff and von Neumann," in *Proc. IEEE IWQOS'99*, London, UK, pp. 79–86 (June 1999).

[7] C. S. Chang, W. J. Chen, and H. Y. Huang, "Birkhoff-von Neumann input buffered crossbar switches," in *Proc. IEEE INFOCOM'00*, Tel Aviv, Israel, pp. 1614–1623 (Mar. 2000).

[8] C. Chang, D. Lee, and Y. Jou, "Load balanced Birkhoff–von Neumann switche. Part II: Multi-stage buffering," *Computer Communications*, vol. 25, no. 6, pp. 623–634 (2002).

[9] I. Keslassy and N. McKeown, "Maintaining packet order in two-stage switches," in *Proc. IEEE INFOCOM'02*, New York, vol. 2, pp. 1032–1041 (June 2002).

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

[11] C. S. Chang, D. Lee, and Y. J. Shih, "Mailbox switch: A scalable two-stage switch architecture for conflict resolution of ordered packets," in *Proc. IEEE INFOCOM'04*, Hong Kong, vol. 3, pp. 1995–2006 (Mar. 2004).

[12] Y. Shen, S. Jiang, S. Panwar, and H. J. Chao, "Byte-Focal: a practical load balanced switch," in *Proc. IEEE Workshop on High Performance Switching and Routing* (HPSR'05), Hong Kong, pp. 6–12 (May 2005).

[13] Y. Li, S. Panwar, and H. J. Chao, "Exhaustive service matching algorithms for input queued switches," in *Proc. IEEE Workshop on High Performance Switching and Routing (HPSR 04)*, Phoenix, Arizona, pp. 253–258 (Apr. 2004).

[14] P. Giaccone, B. Prabhakar, and D. Shah, "Towards simple, high-performance schedulers for high-aggregate bandwidth switches," in *Proc. IEEE INFOCOM'02*, New York, vol. 3, pp. 1160–1169 (June 2002).

[15] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (Extended Version)," *IEEE/ACM Transactions on Networking*, vol. 2, issue 1, pp. 1–15 (Feb. 1994).