Cheng-Shang Chang and Duan-Shin Lee

Principles, Architectures and Mathematical Theories of High Performance Packet Switches

Feb. 2006

Preface

Due to the recent advances in optical transmission technologies, the transmission speed of optical links is much faster than the switching speed of current electronic Internet routers. The challenge is then to find switch architectures that scale with the transmission speed of fiber optics. There are two approaches for this: (i) the electronic approach and (ii) the optical approach. The electronic approach is to use parallel electronic devices to acquire the needed speedup for fiber optics. On the other hand, the optical approach is to explore the possibility of building intelligent logic control directly with optical devices. It is the intent of this book to introduce recent advances in switch architectures from both the electronic approach and the optical approach.

In this book, we will first give a detailed review of existing switch architectures in Chapter 2, including both the output-buffered switches and input-buffered switches that are currently used in most Internet routers. Output-buffered switches, though ideal in the mathematical sense, suffer from the memory accessing speed problem. As such, its speed is in general limited by the memory accessing speed of the state-of-the-art memory technology. To acquire the needed speedup, parallel buffers are needed. Input-buffered switches are built for that purpose. However, as there are parallel input buffers, coordination of of parallel buffers results in a problem of finding "good" matchings. It seems that the matching problem was solved by two great mathematicians, G. Birkhoff [17] and J. von Neumann [164], long before it became a problem in input-buffered switches. The Birkhoff-von Neumann switch, an input-buffered switch that implements the algorithm developed by Birkhoff and von Neumann, achieves the ideal 100% throughput for all admissible traffic.

The load-balanced Birkhoff-von Neumann switches in Chapter 3 eliminate the need for finding "good" matchings in input-buffered switches. They are, therefore, much more scalable than input-buffered

switches. The idea of the load-balanced Birkhoff-von Neumann switches is quite simple. In such a switch, parallel buffers are placed between two switch fabrics. The first switch fabric performs load balancing so that the traffic coming to the parallel buffers is uniform, and thus can be easily switched by the second switch fabric. As there are multiple routing paths through the load-balanced Birkhoff-von Neumann switches, packets in such switches may be delivered out of sequence. In Chapter 3, we shall introduce several variants in the literature that address the out-of-sequence problem in such switches.

Chapter 4 is a short chapter. There we introduce the concept of quasi-circuit switching. Traditionally, circuit switching is used for quality of service, while packet switching is used for bandwidth sharing. Quasi-circuit switching is a concept that falls in between packet switching and circuit switching, and thus can be viewed as a performance compromise between packet switching and circuit switching. The advantage of quasi-circuit switching is that quasi-circuit switches can be built with less complexity by using the load-balanced Birkhoff-von Neumann switches.

In Chapter 5, we introduce the optical approach. The key problem with optical packet switches is the lack of inexpensive memory. We start from an optical memory cell that is capable of storing one fixed-size packet. Then we use that as a basic building block to construct various optical queues, including time slot interchanges, First-In-First-Out (FIFO) multiplexers, FIFO queues, linear compressors, non-overtaking delay lines, and priority queues. The most interesting part of such a development is its connection to classical switching theory. For instance, linear compressors are connected to banyan networks, and non-overtaking delay lines and FIFO queues are connected to Benes networks.

This book is the result of courses developed in packet switch architectures at National Tsing Hua University. The material in this book can serve as a basis for a semester-long graduate level course that covers all the chapters in this book. Readers are recommended to take an undergraduate course in *computer networks* as a prerequisite. Also, for some of the mathematical theories in the book, it might be helpful to have some knowledge of *linear algebra (matrices)*, *differential equations*, *discrete math (graphs)* and *elementary probability*.

Several chapters of this book were rewritten from papers jointly coauthored with our colleagues and students in the last seven years.

For this, we gratefully thank Wen-Jyh Chen, Yi-Ting Chen, Jay Cheng, Ching-Te Chiu, Hsien-Chen Chiu, Chih-Chieh Chou, Hsiang-Yi Huang, Yi-Shean Jou, Issac Keslassy, Ching-Ming Lien, Nick McKeown, Ying-Ju Shih, Chih-Ying Tu, Chao-Kai Tu, Chao-Lin Yu, and Chi-Yao Yue. We give special thanks to Jay Cheng and Anne Bouillard for carefully reviewing an earlier draft and Yu-Hao Hsu for setting up the Web site

http://gibbs.ee.nthu.edu.tw/COM5353.htm

that contains slides and solutions for several sets of problems in the book. We are also grateful to the National Science Council and the Ministry of Education, Taiwan, R.O.C., for support of much of the work under the Program for Promoting Academic Excellence of Universities NSC 94-2752-E-007-002-PAE, and the National Innovative Communication Education Program.

Hsinchu, Feb. 2006 *Cheng-Shang Chang* and *Duan-Shin Lee*

Table of Contents

1. Introduction

1.1 The Internet

The whole world is connected via the Internet. The Internet has a layered structure and every layer has its specific objectives and functions (see Figure 1.1). When a person clicks a button to access a web page from the Internet, it evokes a sequence of operations that need to be carried out accurately. First, the web browser, e.g., Microsoft Internet Explorer, passes the web address to the Hyper Text Transfer Protocol (HTTP). Then HTTP evokes the Transmission Control Protocol (TCP) to connect the local host to the sever that contains the web page. TCP then sends and transmits Internet Protocol (IP) packets between the local host and the server. IP packets that contains the IP addresses of the local host and the server are routed through the Internet via several IP routers. Between two successive IP routers, IP packets are usually carried by a lower layer protocol, such as Ethernet or token ring. If the maximum packet size of the lower layer protocol is smaller than the size of an IP packet, then the IP packet needs to be further fragmented into several independent IP packets that fit the size of the packet of the lower layer protocol.

Every layer of protocol has its specific objective and its specific message format. For example, HTTP is in charge of setting up a connection between the local host and the server that handles html files. As it is impossible to set up a direct HTTP connection through the Internet, HTTP has to ask for its lower layer protocol, i.e., TCP, to do the job. To do this, HTTP has to specify how it communicates with TCP. Specifically, port 80 of TCP is reserved for HTTP so that any TCP segment with port 80 is forwarded to HTTP. HTTP then converts TCP segments into html files and forward it to the upper layer browser. Similarly, TCP cannot set up a TCP connection directly between the local host and the server, and it has to use IP. TCP, as its name suggests, also uses a window flow control mechanism that pre-

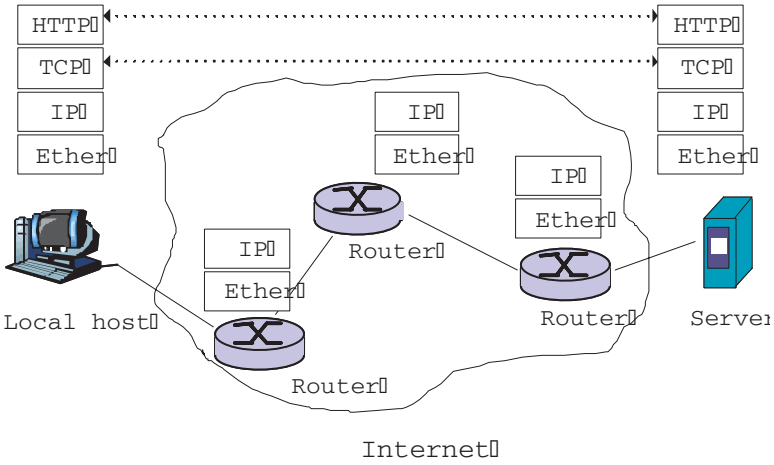


Fig. 1.1. An illustrating example of the Internet

vents both the Internet and the server into a congestion state. The main objective of IP is to route IP packets to the right place. Theoretically, every computer that connects to the Internet is assigned a *unique* IP address. In every IP packet, both the IP addresses of the local host and the server are added so that IP routers can use them to route packets. In addition to the IP addresses, the information of its upper layer protocol is also included in every IP packet (in this case, it is TCP). By so doing, an IP packet can be forwarded to the correct upper layer protocol at the right sever.

There are still mysteries that need to be resolved:

(i) How does the local host find out the IP address of the server via its web address?
(ii) How does an IP router route packets by the IP address?
(iii) Even when an IP router finds out the IP address of the next hop router, how does an IP router find out the physical address of the next hop router, e.g., the Ethernet address?

The first question is resolved by adding a network of Domain Name Servers (DNS) into the Internet. The network of DNS is a hierarchical network that is built upon the Internet. When the local host is not aware of the IP address of a web address, it will send a request to its local DNS server. If its DNS server cannot resolve the problem, the DNS server will send a request to its upper layer DNS server and resolve the problem in an iterative and recursive manner.

The third problem is resolved by introducing the Address Resolution Protocol (ARP). When an IP router does not have the physical address of the next hop router, it broadcasts a request for the physical address of the next hop router through its local area network, e.g., the Ethernet. The request contains both the physical address and the IP address of the router that sends out the request, and the IP address of the next hop router. By examining the IP address, the next hop router realizes that the request is addressed to it and it needs to send a reply that contains its physical address to the router that sends the request.

The second problem is much more complicated than the other two and it will be addressed in the next section.

1.2 IP routers

The main objective of an IP router is to route IP packets according to the IP address. An IP router usually contains several input ports and output ports. To route a packet from an input port to an output port, an IP router must contain a table that maps every IP address to the output port that links to the next hop router. The protocol that creates such a table is called a *routing* protocol and the table created is thus called a *routing table*. There are several routing protocols that can be used for routing (see e.g., [138, 103]), and these protocols are usually implemented as a piece of software in an IP router. Sometimes, a routing table is further converted into a *forwarding table* that is easier to map an IP address to the output port of its next hop router. The operation that forwards an IP packet from an input port to an output port is then called *forwarding*.

Forwarding consists of two steps: (i) table lookup: it finds the appropriate output port, and (ii) message copying: it copies packets from input ports to output ports. Table lookup can be carried out in a distributed manner. Thus, for high speed routers, a forwarding table is stored in every input port. When a packet is received from an input port, its output port can be found directly from the forwarding table. This can be done in *parallel* at every input port with affecting each other. However, message copying is much more difficult to do in *parallel*. This will be further discussed in the next section.

To perform routing and forwarding, an IP router is usually implemented by the following three components (see Figure 1.2):

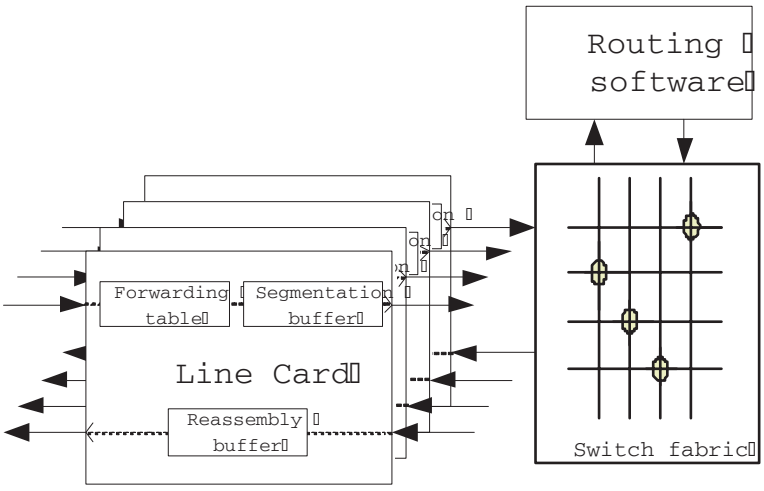


Fig. 1.2. An IP router

(i) A routing software: the routing software implements the routing protocol that creates the routing table and the associated forwarding table.
(ii) Line cards: a line card is associated with an input port and an output port. Every line card contains a forwarding table. When an IP packet is received from the input port in the line card, the destination IP address on the packet is used in the forwarding table to find out the appropriate output port. The IP packet is converted (and sometimes segmented) into the packet format used in the switch fabric. The packet is then sent through the switch fabric to the appropriate output port. When a packet is received from the switch fabric, the packet is buffered (and sometimes reassembled) and then sent out from the output port in the line card.
(iii) Switch fabric: the switch fabric performs message copying. It copies packets from input ports to output ports. Sometimes, multiple switch fabrics are installed in an IP router for the sake of reliability and speedup.

The bottleneck of an IP router is mostly in the switch fabric. Suppose that an IP router has N line cards (N input/output ports) and each line card is running at rate R bits/sec. The rate R is called the line rate (or line speed) of an IP router. The rate (or speed) of an IP router is generally referred to as the aggregated rate of all the line cards, i.e., $N \times R$ bits/sec. However, there is a catch for this. As the

switch fabric might be a bottleneck, not all the packets can be routed to the output ports successfully when each input port is running at the full rate R. A notion, called the throughput of an IP router, is defined to be the percentage of packets that are successfully delivered to output ports for a certain traffic coming to inputs ports with rate R. Not all commercially available IP routers yield 100% throughput for all kinds of traffic. As such, it might be very misleading if one compares the performance of IP routers by simply looking at their aggregated rates.

One might wonder why it is not called an IP switch. Switches and routers are basically the same. Traditionally, switches are used for circuit switching networks, such as telephone networks, that provide connection oriented services. On the other hand, routers are used for packet switching networks that provide connectionless services. However, the line becomes blur as more and more circuit switching networks and packet switching networks are integrated. Recently, switches are used more often than routers. For instance, a layer x switch implies a switch that uses layer x protocol for packet routing. In particular, a layer 2 switch is an Ethernet switch, a layer 4 switch is a switch that uses TCP for routing, and a layer 7 switch is a switch that uses web contents for routing.

1.3 Switch fabrics

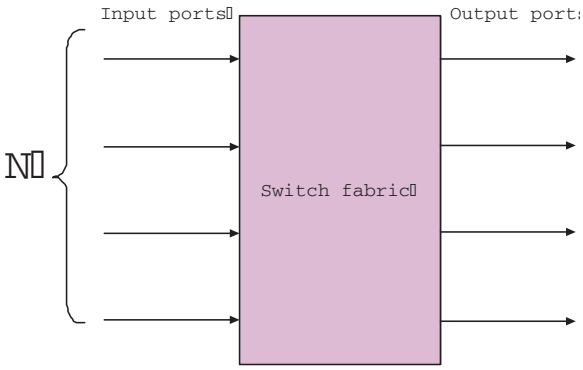


Fig. 1.3. A switch fabric

As mentioned in the previous section, the main objective of a switch fabric is to copy packets from input ports to output ports (see Figure 1.3). This will be addressed in details in Chapter 2. The easiest way to do this is to use a common shared memory for the switch fabric. A switch fabric (or simply a switch) that uses a common shared memory is known to be the shared memory switch architecture. Suppose that the incoming IP packets are segmented into packets with the same size at the line cards. Then one can partition time into time slots so that a packet (for the switch fabric) can be transmitted within a time slot from a line card. If there are N input ports and N output ports for the switch fabric, then within a time slot N packets have to be written into the common shared memory and read out from the common shared memory. If packets arriving at every input port is at the rate of R bits/sec, then in order to guarantee successful delivery of packets the common shared memory needs to be operated at the rate of $2NR$ bits/sec (as there are N write operations and N read operations within a time slot). Clearly, the main problem of the shared memory switch architecture is *scalability*. It does not scale when the number of input ports N is large.

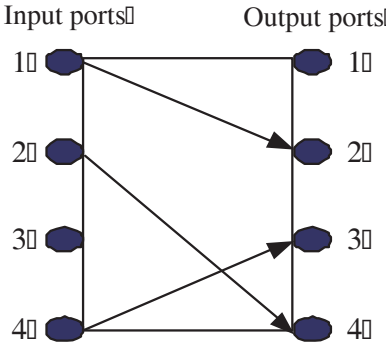


Fig. 1.4. Parallel transmissions

To cope with the scalability problem, parallel (and simultaneous) transmissions from input ports and output ports must be used (see Figure 1.4). However, as an input port (resp. output port) is allowed to transmit (resp. receive) a packet in a time slot, these parallel transmissions need to be coordinated so that every input port is connected to at most one output port and vice versa. Such a connection pattern

is known as a matching since one may view input ports as men and output ports as women. In Section 2.2, we will address several schemes that find matchings between input ports and output ports.

Coordinating parallel transmission takes efforts. There are two major overheads of doing this:

(i) Communication overhead: information needs to be exchanged among input/output ports for finding a matching.
(ii) Computation overhead: it takes time to compute a matching.

For a switch fabric with a large number of input ports, the communication overhead could be as large as transmitting the packet itself. This also results in the scalability problem.

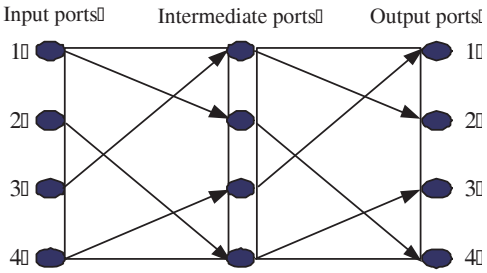


Fig. 1.5. A two-stage switch fabric

To solve the coordination problem for parallel transmissions, one may use a pre-determined schedule of parallel transmissions. However, a pre-determined schedule may not fit the traffic demand at the input ports. The idea is then to alter the traffic demand so that it fits the pre-determined schedule. This requires adding another switch fabric to form a two-stage switch fabric as shown in Figure 1.5. The first stage performs load balancing that distributes packets evenly to the intermediate ports. By so doing, the traffic demand coming to the intermediate ports is *uniform* in spite of the traffic demand at the input ports may be non-uniform. As the traffic demand coming to the intermediate ports is uniform, packets can be evenly distributed to output ports via the switch fabric at the second stage. Switch architectures like this are called load balanced Birkhoff-von Neumann switches and they will be addressed in details in Chapter 3.

1.4 Circuit switching, packet switching, and quasi-circuit switching

The Internet is a packet-switched network. On the other hand, telephone networks are based on circuit switching. In a circuit-switched network, resources, including bandwidth and buffers, are reserved along a path for the duration of communication. As such, quality of service (QoS) is easily guaranteed. However, as resources are reserved for dedicated use, resources are not used efficiently in circuit-switched networks. On the other hand, as there is no resource reservation in packet-switched networks, resources could be used more efficiently in packet-switched networks. However, it is much more difficult to provide QoS in packet-switched networks.

Providing QoS in the Internet in general requires implementing several mechanisms, such as traffic policing, packet scheduling and admission control (see e.g., [138, 103]). There are several theoretical frameworks developed in the literature for providing QoS. Among them, the network calculus developed by R. L. Cruz [53] (and later generalized in [26, 104, 2]) is the most popular one. It is shown in [132] that deterministic QoS can be guaranteed by implementing weighted fair queueing in routers. Such a framework was later proposed in the IETF RSVP/Intserv architecture [149]. For a complete introduction of the network calculus, we refer to the books [27, 105].

It would be nice to have QoS in a high speed IP router without implementing complicated mechanisms needed in the network calculus. For this, we introduce the concept of *quasi-circuit switching* in Chapter 4. Quasi-circuit switching is to find compromises between circuit switching and packet switching. The idea is not completely new and in fact it is a generalization and an abstraction of the stop-and-go queueing proposed by Golestani [67, 68]. As in [67, 68], time in a quasi-circuit switched network is partitioned into frames. Flows entering a quasi-circuit switched network are rate controlled so that the number of bits of each flow in every frame is always bounded. Via appropriate admission control of flows entering a quasi-circuit switched network, the links in the network never exceed their capacities, i.e., a quasi-circuit switched network is a congestion-free network like a circuit-switched network.

In circuit switching, traffic is completely isolated. On the other hand, traffic is completely mixed in packet switching. Quasi-circuit switching uses frames to isolate traffic. As such, it can be viewed as

circuit switching at the time scale of frames. Thus, it can still provide some guarantees of quality of services at the frame level. At the same time, quasi-circuit switching allows packets to be multiplexed within a frame. Thus, it can achieve statistical multiplexing gain within a frame. The concept of quasi-circuit switching is equivalent to the statistical line grouping in the circuit switching context (see e.g., [111]) and is known as the duration limited statistical multiplexing in [68].

1.5 Optical packet switches

Most of current IP routers use electronic memory. However, as the technology for fiber optics advances, the access speed of electronic memory is much slower than the transmission speed of fiber optics. As the links connecting IP routers are in general based on fiber optics, optical signals have to be converted into electric signals in IP routers. Such a conversion is very costly and that drives people to think whether IP packets can be directly switched in the domain of light.

As mentioned in Section 1.2, forwarding consists of two steps: (i) table lookup and (ii) message copying. To solve the problem of table lookup, traffic needs to be aggregated into flows. By so doing, a much shorter address than an IP address could be used. The short address is generally referred to as a *label* and it requires a much smaller forwarding table than the IP forwarding table. The technology that uses labels for switching is known as label switching (see e.g., [10]). However, abstracting the label from an optical packet for table lookup might still be too difficult to do. An idea is to send the label through a separate electronic network before the optical packet is transmitted. By so doing, table lookup is done in the electronic domain and it still has time to reserve the resource for the incoming optical packet. The technology based on such an idea is generally referred to as *optical burst switching* (see e.g., [160, 171]).

For message copying, one still needs to deliver an optical packet from an input port to an output port without colliding with other optical packets. For this, one still needs optical memory. Unlike electronic devices, it is very difficult to build memory using pure optical components. However, as it takes time for light to propagate, one may "store" light (and the information contained in the light) by circuiting the light in a fiber delay line and "release" the light when the information is retrieved. By so doing, one may use optical switches and fiber

delay lines to build an optical memory. An optical device built by optical switches and fiber delay lines is then called a Switched Delay line (SDL) element.

The main difficulty of using SDL elements to build optical device is the control of the optical switches. One has to release the light at the right time to the right place. In general, the decision is made based on the "state" of SDL elements which indicates whether light is stored in a particular section of a fiber delay line. To build a large memory, one has to use long fiber delay lines and this results in a huge number of states. Finding the right control for such an SDL element becomes a very complicated combinatoric problem.

Fortunately, there are certain types of memory, including time slot interchanges, and multiplexers with First In First Out (FIFO) queues, that can be recursively constructed by the SDL elements. The detailed constructions will be addressed in Chapter 5. These constructions can then be used for building optical packet switches.

2. Basic Architectures and Principles of Packet Switches

A *switch* is a network element with multiple input ports and output ports. We call a switch with M input ports and N output ports an $M \times N$ switch. The main objective of a switch is to transfer packets from an input port to one or more output ports. To achieve this, a switch has the following two basic functions: (i) table lookup: it finds the appropriate output port for a packet, and (ii) message copying: it copies packets from input ports to output ports. The first function is usually implemented by a routing table, and the second function is usually implemented by a switch fabric (with buffers). Readers interested in IPv4 table lookup may find implementable algorithms in P. Gupta, S. Lin and N. McKeown [70] and N.-F. Huang and S. M. Zhao [74]

The main objective of this chapter is to discuss switch architectures that perform message copying. For the ease of presentation, we assume that **packets are of the same size and that all the input/output links are of the same speed. Moreover, time is slotted into fixed and identical intervals (slots) so that a packet can be transmitted within a time slot at any input/output link**.

The contents of this chapter can be broadly classified into three topics:

– switch architectures, location of buffers and scheduling of packets;
– design of switch fabrics;
– exact emulation and approximation of output-buffered switches.

The first topic contains discussion on output-buffered, input-buffered and Birkhoff-von Neumann switches. For the second topic, we present three-stage constructions and two-stage constructions of switch fabrics in Section 2.4 and Section 2.5. The third topic is presented in Section 2.6 and Section 2.7.

2.1 Output-buffered switches

2.1.1 Shared memory switch and shared medium switch

The first architecture is called a *shared memory* switch. For such a switch (see Figure 2.1), packets from all input ports are read into a common shared memory. A central controller then writes all those packets to the destined output ports according to an address lookup table. All the read/write operations have to be done within a time slot. For an $N \times N$ shared memory switch, there are N write operations for the N input ports and N read operations for the N output ports per time slot. Thus, its memory access speed must be at least $2N \times$ *link speed*. As such, a shared memory switch is not *scalable* if the number of input/output ports is large.

Example 2.1.1. The memory access time for the current DRAM is roughly 10 ns (nano second). For a packet of 64 bytes (512 bits), the memory access speed of a shared memory switch is roughly 51.2 Gbits/sec. If the line speed is 2.48 Gbits/sec (OC48), then a shared memory switch can support up to 10 input/output ports.

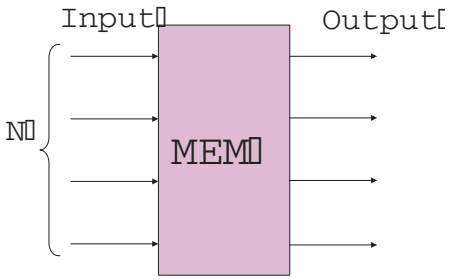


Fig. 2.1. A shared memory switch

The second one is a *shared medium* switch, most commonly known as a shared bus switch (see Figure 2.2). This kind of switch also requires a central controller to ensure that only one packet is transferred on the shared bus at any moment. Within each time slot, the central controller transmits a packet from each input in a round-robin manner onto the bus. Each output port listens to the bus all the time and accepts the

packet that is destined for it. For an $N \times N$ shared medium switch, the central controller needs to transmit N packet within a time slot. Thus, the memory access speed of an $N \times N$ shared medium switch must be at least $N \times$ *link speed*, which is one half of that for a shared memory switch. As a shared memory switch, a shared medium switch is not *scalable* if the number of input/output ports is large.

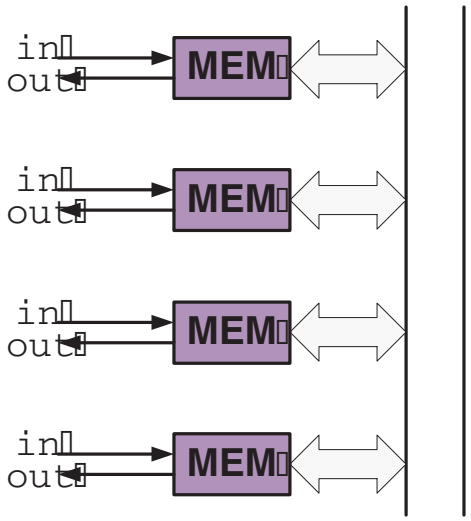


Fig. 2.2. A shared medium switch

Both the shared memory switch and the shared medium switch are called *output-buffered switches*. This is because an output port can transmit at most one packet within a time slot and there might be multiple packets that arrive at the same output port at the same time slot. When this happens, one needs buffers to solve such output conflicts. As such, there is a queue for each output port (even though they might share the same memory block).

2.1.2 Lindley equation

To perform mathematical analysis for an $N \times N$ output-buffered switch, we consider the following discrete-time queueing system. Let $a_i(t)$, $i = 1, 2, \ldots, N$, be the number of packets that arrive at the t^{th} time slot and are destined to the i^{th} output port. Also, let $q_i(t)$ be the number of packets stored in the i^{th} output buffer at the end of the t^{th} time slot. Then we have the following recursive equation:

$$q_i(t+1) = (q_i(t) + a_i(t+1) - 1)^+, \tag{2.1}$$

where $x^+ = \max(0, x)$. Equation (2.1) is intuitively clear as there is a departing packet as long as there are packets to be transmitted from the i^{th} output port. When this happens, the number of packets left at time $t+1$ is one less than the sum of the number of packets stored at time t and the number of packets that arrive at time $t+1$.

The equation in (2.1) is generally known as the Lindley equation [114] for a discrete-time queue. A discrete-time queue that satisfies the Lindley equation in (2.1) is also called a work conserving link (with capacity 1) in [27]. The Lindley equation can be recursively expanded to derive the following representation for $q_i(t)$.

Lemma 2.1.2. *Suppose that $q_i(0) = 0$.*

(i) Let $A_i(t) = \sum_{s=1}^{t} a_i(s)$ be the cumulative number of packets that arrive by time t and are destined to the i^{th} output port. Then

$$q_i(t) = \max_{0 \le s \le t} [A_i(t) - A_i(s) - (t-s)].$$

(ii) Let $B_i(t) = A_i(t) - q_i(t)$ be the cumulative number of packets that depart by time t from this queue. Then

$$B_i(t) = \min_{0 \le s \le t} [A_i(s) + (t-s)].$$

The proof for Lemma 2.1.2 is left as an easy exercise for the readers in Problem 1.

The next question is whether there is a steady state random variable $q(\infty)$ for $\{q(t), t \ge 1\}$ as $t \to \infty$. To answer this question, we need to introduce the concept of *stationarity* and *ergodicity*. A stochastic sequence $\{X(t), t \ge 1\}$ is *stationary* if its joint distribution is invariant with respect to time shift, i.e., for any time shift s and any k-dimensional joint distribution,

$$\begin{aligned} &\mathsf{P}(X(1) \le x_1, X(2) \le x_2, \ldots, X(k) \le x_k) \\ &= \mathsf{P}((X(1+s) \le x_1, X(2+s) \le x_2, \ldots, X(k+s) \le x_k). \end{aligned} \tag{2.2}$$

A stationary sequence $\{X(t), t \ge 1\}$ is *ergodic* if its long run average is the same as its ensemble average, i.e., for any function f with k variables

$$\lim_{t\to\infty}\frac{1}{t}\sum_{s=0}^{t-1} f\Big(X(1+s), X(2+s), \ldots X(k+s)\Big)$$
$$= \mathsf{E}\Big[f\Big(X(1), X(2), \ldots, X(k)\Big)\Big], \quad a.s., \tag{2.3}$$

where *a.s.* stands for "almost surely" and it means the convergence is almost surely true for every sample path of $\{X(t), t \geq 1\}$ (those sample paths that do not converge have probability 0). In particular, an ergodic sequence satisfies the following strong law of large number

$$\lim_{t\to\infty}\frac{1}{t}\sum_{s=1}^{t} X(s) = \mathsf{E}[X(1)], \quad a.s. \tag{2.4}$$

Clearly, if the sequence $\{X(t), t \geq 1\}$ is a sequence of independent and identically distributed (i.i.d.) random variables (r.v.'s), then $\{X(t), t \geq 1\}$ is ergodic according to the strong laws of large numbers for i.i.d. r.v's. However, not every stationary sequence is ergodic as shown in the following example.

Example 2.1.3. Consider a sequence of i.i.d. Bernoulli r.v.'s $\{X(t), t \geq 1\}$ with parameter 1/2, i.e., for all t

$$\mathsf{P}(X(t) = 0) = \mathsf{P}(X(t) = 1) = \frac{1}{2}\,.$$

Let $Y(t) = X(1)+X(t)$ for all t. Clearly, $\{Y(t), t \geq 1\}$ is still *stationary* with the mean rate $\mathsf{E}[Y(1)] = 1$. However, it is not ergodic as

$$\lim_{t\to\infty}\frac{1}{t}\sum_{s=1}^{t} Y(s) = X(1) + E[X(1)] = X(1) + \frac{1}{2} \neq 1, \quad a.s.$$

In fact, $\{Y(t), t \geq 1\}$ can be decomposed as two ergodic sequences, one with $X(1) = 0$ and the other with $X(1) = 1$.

The following theorem shows that there is a steady state random variable $q(\infty)$.

Theorem 2.1.4. *(Loynes [116]) Suppose that* $\{a_i(t), t \geq 1\}$ *is* stationary *and* ergodic *with mean* ρ*, i.e.,*

$$\lim_{t\to\infty}\frac{A_i(t)}{t} = \mathsf{E}a_i(1) = \rho, \quad a.s.$$

If $\rho < 1$*, then* $q_i(t)$ *converges to a steady state* random variable $q_i(\infty)$*.*

The result in Theorem 2.1.4 is known as the Loynes construction [116]. Details of the proof for such a construction can be found in [11, 27]. For an outline of the proof of Theorem 2.1.4, see Problems 2 and 3.

Note that there are two conditions in Theorem 2.1.4: (i) traffic characterization for the input, i.e., stationarity and ergodicity and (ii) the rate condition $\rho < 1$. The rate condition is intuitively clear as one must have the arrival rate ρ smaller than the departure rate 1. Otherwise, the queue goes to ∞ when the arrival rate exceeds the departure rate. The hard part is when the arrival rate is equal to the departure rate, and it would depend on how variable the input is. As such, for the engineering purpose, it is preferred that one keeps the arrival rate strictly smaller than the departure rate. The need of ergodicity is also clear. If the input is not ergodic, it may be decomposed as two or more ergodic sequences. One of them may have the arrival rate larger than the departure rate, while the expected arrival rate (over all the sample paths of the stochastic sequence) is smaller than the departure rate. The condition of stationarity may be weakened a little bit. It can be replaced by a *stable* sequence. A stable sequence is a stochastic sequence that becomes *stationary* when $t \to \infty$. For instance, if we consider two discrete-time queues in tandem. The output of the first queue is the input of the second queue. If we start the system from an empty system, then the input to the second queue is not stationary. However, it could be stable if the first queue satisfies the condition in Theorem 2.1.4. In fact, the stochastic sequence $\{q_i(t), t \geq 0\}$ in Theorem 2.1.4 can further be shown to be a stable and ergodic sequence. As such, one can further conclude the existence of a steady state random variable for the second queue if the condition in Theorem 2.1.4 is also satisfied for the second queue.

2.1.3 Average queue length

In the following proposition, we derive the average queue length for an output-buffered switch subject to independent and identically distributed input traffic.

Proposition 2.1.5. *Suppose that $\{a_i(t), t = 1, 2, \ldots\}$ is a sequence of independent and identically distributed (i.i.d.) random variables with mean ρ and variance σ^2. If $\rho < 1$, then as $t \to \infty$, $q_i(t)$ converges in distribution to a steady random variable with mean $\frac{\sigma^2+\rho^2-\rho}{2(1-\rho)}$.*

Note that ρ is the average arrival rate to the i^{th} output buffer. Proposition 2.1.5 shows that the mean (expected) queue length remains bounded as long as $\rho < 1$ (and $\sigma^2 < \infty$). Such a property is known as the 100% throughput property for output-buffered switches.

Proof. From Theorem 2.1.4, we know that the steady state random variable exists. Now we derive the expected queue length of the steady state random variable. Let $X(t) = q_i(t) + a_i(t+1)$. From (2.1), it follows that

$$X(t+1) = (X(t) - 1)^+ + a_i(t+2). \tag{2.5}$$

Rewrite (2.5) as follows:

$$\begin{aligned} X(t+1) &= (X(t)-1)^+ \mathbf{1}\{X(t) > 0\} \\ &\quad +(X(t)-1)^+ \mathbf{1}\{X(t) = 0\} + a_i(t+2), \end{aligned} \tag{2.6}$$

where $\mathbf{1}\{\mathcal{E}\}$ is the indicator random variable with value 1 if the event $\mathcal{E}$ is true and 0 otherwise. Note that $(X(t)-1)^+\mathbf{1}\{X(t)=0\}$ is simply 0. Also, since $X(t)$ is a nonnegative and integer valued random variable,

$$(X(t)-1)^+\mathbf{1}\{X(t) > 0\} = (X(t)-1)\mathbf{1}\{X(t) > 0\}.$$

Thus, we have from (2.6) that

$$X(t+1) = (X(t)-1)\mathbf{1}\{X(t) > 0\} + a_i(t+2). \tag{2.7}$$

Taking expectations on both sides of (2.7) yields

$$\mathsf{E}[X(t+1)] = \mathsf{E}[(X(t)-1)\mathbf{1}\{X(t) > 0\}] + \rho. \tag{2.8}$$

Note that

$$\begin{aligned} &\mathsf{E}(X(t)-1)\mathbf{1}\{X(t) > 0\} \\ &= \mathsf{E}X(t)\mathbf{1}\{X(t) > 0\} - \mathsf{E}\mathbf{1}\{X(t) > 0\} \\ &= \mathsf{E}X(t) - \mathsf{P}(X(t) > 0). \end{aligned} \tag{2.9}$$

As $t \to \infty$, $\mathsf{E}X(t)$ will be the same as $\mathsf{E}X(t+1)$. It then follows from (2.8) that $\mathsf{P}(X(t) > 0)$ converges to ρ as $t \to \infty$, i.e.,

$$\lim_{t\to\infty} \mathsf{P}(X(t) > 0) = \rho.$$

Now we square both sides of (2.7) to derive

$$\begin{aligned} X(t+1)^2 &= (X(t)-1)^2\mathbf{1}\{X(t) > 0\} \\ &+ 2(X(t)-1)\mathbf{1}\{X(t) > 0\}a_i(t+2) + a_i(t+2)^2. \end{aligned} \tag{2.10}$$

Since $a_i(t)$'s are i.i.d., $a_i(t+2)$ is independent of $X(t)$ (which is a function of $a_i(s), s \leq t+1$). Taking expectations on both sides of (2.10) yields

$$\begin{aligned}
&\mathsf{E}X(t+1)^2 \\
&= \mathsf{E}X(t)^2 - 2\mathsf{E}X(t) + \mathsf{E}\mathbf{1}\{X(t)>0\} \\
&\qquad +2\mathsf{E}X(t)\mathsf{E}a_i(t+2) \\
&\qquad -2\mathsf{E}\mathbf{1}\{X(t)>0\}\mathsf{E}a_i(t+2) + \mathsf{E}a_i(t+2)^2 \\
&= \mathsf{E}X(t)^2 - 2\mathsf{E}X(t) + \mathsf{P}(X(t)>0) + 2\rho\mathsf{E}X(t) \\
&\qquad -2\rho\mathsf{P}(X(t)>0) + \sigma^2 + \rho^2. \qquad (2.11)
\end{aligned}$$

As $t \to \infty$, $\mathsf{E}X(t)^2$ will be the same as $\mathsf{E}X(t+1)^2$. As $\mathsf{P}(X(t)>0)$ converges to ρ, we then have from (2.11) that

$$\lim_{t\to\infty} \mathsf{E}X(t) = \frac{1}{2}\frac{\sigma^2+\rho-\rho^2}{1-\rho}. \qquad (2.12)$$

Recall that $X(t) = q_i(t) + a_i(t+1)$. Thus,

$$\begin{aligned}
\lim_{t\to\infty} \mathsf{E}q_i(t) &= \frac{1}{2}\frac{\sigma^2+\rho-\rho^2}{1-\rho} - \rho \\
&= \frac{1}{2}\frac{\sigma^2+\rho^2-\rho}{1-\rho}. \qquad (2.13)
\end{aligned}$$

■

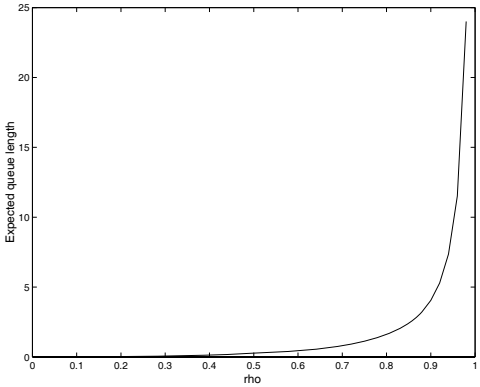


Fig. 2.3. The expected queue length as a function of the mean arrival rate

Example 2.1.6. (Output buffered switch with Poisson inputs) Suppose that $a_i(t)$'s are i.i.d. Poisson random variables with mean ρ. For a Poisson random variable, its variance is the same as its mean, i.e., $\sigma^2 = \rho$. Thus,

$$\lim_{t\to\infty} \mathsf{E}q_i(t) = \frac{1}{2}\frac{\rho^2}{1-\rho}. \tag{2.14}$$

In Figure 2.3, we plot (2.14). It shows that the expect queue length remains bounded when $\rho < 1$. Also, it approaches to ∞ when $\rho \to 1$.

2.1.4 Little's formula

We have derived the expected queue length for the discrete-time queue in (2.1) when the arrivals form a sequence of i.i.d. random variables. In this section, we will show how one uses Little's formula to derive the expected packet delay from the expected queue length.

A queueing system is generally referred to as a system that has customers arriving and departing. Little's formula [115] is a general formula that holds for most queueing systems. To explain Little's formula, consider a particular queueing system. Let $A(t)$ be the number of customers that arrive at the queueing system by time t, $D(t)$ be the number of customers that depart from the queueing system by time t, and $q(t)$ be the number of customers in the queueing system at time t. If $q(0) = 0$, then clearly we have

$$q(t) = A(t) - D(t).$$

The delay of a customer is the time difference between its arrival and its departure, i.e., the amount of time a customer stays in the queueing system. Let $d(n)$ be the delay of the n^{th} customer. Suppose that the followings hold for the queueing system:

$$\lim_{t\to\infty} \frac{1}{t}\int_0^t q(s)ds = L, \quad a.s., \tag{2.15}$$

$$\lim_{n\to\infty} \frac{1}{n}\sum_{m=1}^{n} d(m) = W, \quad a.s., \tag{2.16}$$

and

$$\lim_{t\to\infty} \frac{A(t)}{t} = \lim_{t\to\infty} \frac{D(t)}{t} = \lambda, \quad a.s., \tag{2.17}$$

for some constants L, W and λ. Note that L is the time average of queue length, W is the time average of customer delay, and λ is the average arrival rate and average departure rate. Then these three quantities are related by Little's formula as follows:

$$L = \lambda W. \tag{2.18}$$

The proof of Little's formula is based on a very simple sample path argument. Without loss of generality, assume that the queueing system is started from an empty system, i.e., $q(0) = 0$. Also, we assume that the arrival rate is nonzero, i.e., $\lambda > 0$. Let $q_n(t)$ be the indicator random variable such that $q_n(t) = 1$ if the n^{th} customer is in the queueing system at time t and $q_n(t) = 0$ otherwise. Thus, for all $s \leq t$, $q(s)$ is simply the sum of all the customers in the queueing system at time s, i.e.,

$$q(s) = \sum_{n=1}^{A(t)} q_n(s). \tag{2.19}$$

If the n^{th} customer departs before time t, then $\int_0^t q_n(s)ds$ is exactly the amount of time that the n^{th} customer stays in the queueing system, i.e.,

$$\int_0^t q_n(s)ds = d(n). \tag{2.20}$$

If the n^{th} customer departs after time t, then we still have

$$\int_0^t q_n(s)ds \leq d(n). \tag{2.21}$$

As $D(t) \leq A(t)$, we also have for all $s \leq t$

$$q(s) \geq \sum_{n=1}^{D(t)} q_n(s). \tag{2.22}$$

From (2.19) and (2.21), it follows that

$$\int_0^t q(s)ds \leq \sum_{n=1}^{A(t)} d(n). \tag{2.23}$$

On the other hand, we have from (2.22) and (2.20) that

$$\int_0^t q(s)ds \geq \sum_{n=1}^{D(t)} d(n). \tag{2.24}$$

Using the two inequalities in (2.23) and (2.24) yields

$$\begin{aligned}
&\lim_{t\to\infty}\frac{D(t)}{t}\lim_{t\to\infty}\frac{1}{D(t)}\sum_{n=1}^{D(t)}d(n)\\
&\leq \lim_{t\to\infty}\frac{1}{t}\int_0^t q(s)ds\\
&\leq \lim_{t\to\infty}\frac{A(t)}{t}\lim_{t\to\infty}\frac{1}{A(t)}\sum_{n=1}^{A(t)}d(n) \qquad (2.25)
\end{aligned}$$

From (2.17) and $\lambda > 0$, we know that

$$\lim_{t\to\infty} A(t) = \lim_{t\to\infty} D(t) = \infty, \quad a.s.$$

Using (2.15)-(2.17) in (2.25) yields the desired Little's formula.

As a direct application of Little's formula and Proposition 2.1.5, we derive the average packet delay for an output-buffered switch with i.i.d. arrivals in the following proposition.

Proposition 2.1.7. *For an output-buffered switch, let $d_i(n)$ be the packet delay for the n^{th} packet arriving at the i^{th} output buffer. Suppose that the arrival process $\{a_i(t), t = 1, 2, \ldots\}$ is a sequence of independent and identically distributed (i.i.d.) random variables with mean ρ and variance σ^2. If $\rho < 1$, then*

$$\lim_{n\to\infty}\frac{1}{n}\sum_{m=1}^{n} d_i(m) = \frac{\sigma^2+\rho^2-\rho}{2\rho(1-\rho)}, \quad a.s.$$

Unlike the stochastic sequence $\{q_i(t), t \geq 0\}$, the sequence of random variables $\{d_i(n), n \geq 1\}$ may not converge to a steady state random variable $d_i(\infty)$. For instance, consider the arrival process $\{a_i(t), t \geq 1\}$ with $\mathsf{P}(a_i(t) = 2) = 0.3$ and $\mathsf{P}(a_i(t) = 0) = 0.7$. Then the rate condition in Theorem 2.1.4 is satisfied, and $q_i(t)$ converges to a steady state random variable $q_i(\infty)$. However, this is not the case for the stochastic sequence $\{d_i(n), n \geq 1\}$. As packets arrive in pairs, obviously we have $d_i(2n) = d_i(2n-1) + 1$ under the First Come First Serve (FCFS) policy. As such, there is no steady state random variable for $\{d_i(n), n \geq 1\}$. In view of this, the average packet delay is in form of time average as shown in Proposition 2.1.7.

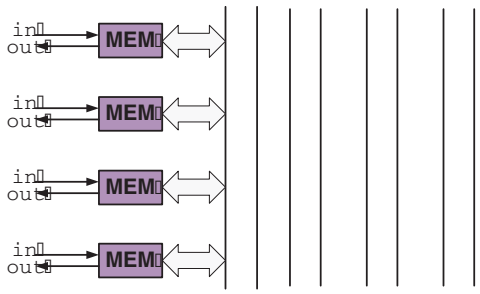


Fig. 2.4. Adding parallel buses to a shared medium switch

2.2 Input-buffered switches

To solve the memory access speed problem, one can simply add more parallel buses to the shared medium (bus) switch to allow parallel read/write operations (see Figure 2.4). For an $N \times N$ switch, at most N parallel buses are needed as there are at most N packets transmitting at the same time. By so doing, the memory access speed only needs to match the link speed, i.e., an input port selects a bus to transmit a packet within a time slot and an output port selects the appropriate bus to receive a packet within a time slot. However, there are limitations for these parallel read/write operations. As an input port can transmit at most a packet within a time slot and an output port can receive at most a packet within a time slot, all these parallel read/write operations are limited to *distinct* input-output pairs (see Figure 2.5). In the case that two or more input ports would like to transmit a packet to a particular output port, only one of them is allowed to do so and the rest of them need to buffer their packets at the input ports. Thus, there is a queue at each input port and such a switch architecture is called an *input-buffered switch* in Figure 2.6. Also, as it has the connection patterns that can be represented by a crossbar, it is also known as a *crossbar* switch.

2.2.1 Fundamental limits on achievable rates of input-buffered switches

The constraints on the parallel read/write operations limit the rates that can be achieved by input-buffered switches. To see this, consider the following $N \times N$ matrix $P(t) = (P_{i,j}(t))$ that represents the connection pattern of an $N \times N$ input-buffered switch at time t: $P_{i,j}(t)$ is

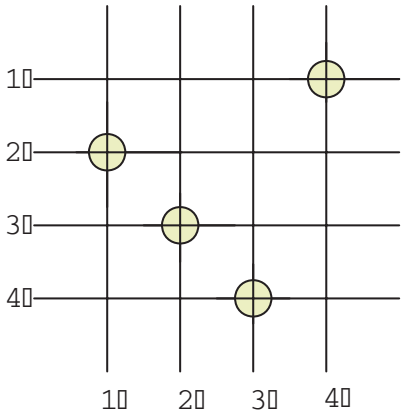


Fig. 2.5. A connection pattern of a crossbar switch

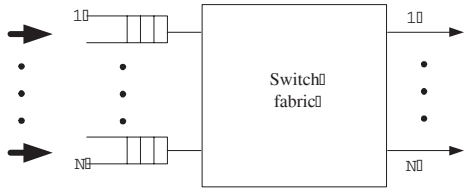


Fig. 2.6. An input-buffered switch

set to 1 if the i^{th} input port is transmitting a packet to the j^{th} output port at the t^{th} time slot, and 0 otherwise. Such a matrix is called a *connection matrix*. The limitations imposed on the parallel read/write operations are equivalent to that the connection matrix $P(t)$ must be a sub-permutation matrix, i.e., there is at most a 1 in each row or column. For example, the connection pattern in Figure 2.5 can be represented by the following permutation matrix

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Define $r_{i,j}$ as the long run average rate from the i^{th} input port to the j^{th} output port, i.e.,

$$r_{i,j} = \lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} P_{i,j}(s), \quad a.s. \tag{2.26}$$

(Note that the long run average rate is the same as the expected rate if the sequence is stationary and ergodic.) Let $R = (r_{i,j})$ be the rate matrix that contains $r_{i,j}$'s as its elements. As $P(t)$ is a sub-permutation matrix (there is at most a 1 in each column), for all $j = 1, 2, \ldots, N$,

$$\begin{aligned} \sum_{i=1}^{N} r_{i,j} &= \sum_{i=1}^{N} \lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} P_{i,j}(s) \\ &= \lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} \sum_{i=1}^{N} P_{i,j}(s) \\ &\leq 1. \end{aligned} \tag{2.27}$$

Similarly, one also has

$$\sum_{j=1}^{N} r_{i,j} \leq 1, \quad i = 1, 2, \ldots, N. \tag{2.28}$$

The two conditions in (2.27) and (2.28), known as **"no overbooking" conditions** in A. Hung, G. Kesidis and N. McKeown [79], are intuitively clear as they simply state that neither the total rate to an output port nor the total rate coming out from an input port can be larger than 1. The no overbooking conditions set the limits on the rates that can be achieved by input-buffered switches. The question is then whether any rate matrix is achievable if the no overbooking conditions

are satisfied. We will show in the next section this may not be true due to the effect of head-of-line blocking.

2.2.2 Head-of-line blocking

Input-buffered switches may suffer from the head-of-line (HOL) blocking problem and that results in degradation in throughput. HOL blocking occurs under the following situation: for an input-buffered switch with each input maintaining a single First In First Out (FIFO) queue, if there are multiple input queues with their head-of-line packets (the first packets in the FIFO queues) destined for the same output port in a time slot, only one of them can be transferred and the others are blocked. For those blocked queues, the packets queued behind the head-of-line packets and destined for available output ports are also blocked.

Example 2.2.1. (Head-of-line blocking) For example, consider the 3×3 input-buffered switch in Figure 2.7. The FIFO queue at input 1 has two packets: the first one is destined to output 1 and the second one is destined to output 2. The FIFO queue at input 2 has two packets destined to output 1. The FIFO queue at input 3 has two packets: the first one is destined to output 1 and the second one is destined to output 3. As all the head-of-line packets are all destined to output 1, only one of them can get through. Suppose that the first packet at input 2 is selected. In this case, the second packet at input 1 and the second packet at input 3 are blocked by the packets ahead of them. The throughput in the case is only 33%. However, if we allow the second packet at input 1 and the second packet at input 3 to take over the packets ahead of them, then we can have three packets transmitted at the same time and the throughput will be 100%!

As shown in Example 2.2.1, head-of-line blocking decreases the throughput. For an input-buffered switch with a large number of input ports, i.e., $N >> 1$, the maximum throughput is limited by $2 - \sqrt{2} \approx 58.6\%$ (under a uniform traffic assumption) [92]. To see this, consider an $N \times N$ input-buffered switch with each buffer being operated under the FIFO policy. To find the maximum throughput, assume there are already infinite many packets queued at the input buffers. As such, there are exactly N HOL packets. Also, assume that

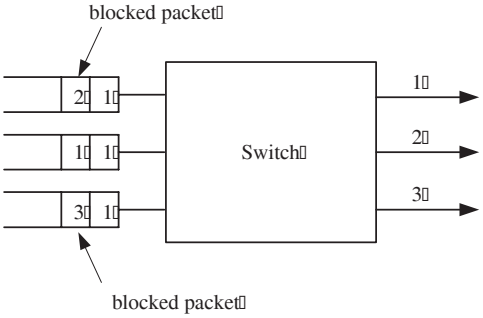


Fig. 2.7. Head-of line blocking

a packet chooses its destination uniformly and independently. Since the traffic is uniform, it suffices to look at a particular output port due to symmetry. Let $X(t)$ be the number of HOL packets that are destined to output port 1 at time t. As there are exactly N HOL packets at any time t, we then have from symmetry that

$$\mathsf{E}(X(t)) = 1. \tag{2.29}$$

Let $a(t)$ be the number of packets that become HOL packets and choose output port 1 as their destination at time t. As there is at most one packet that can be transmitted to output port 1, we then have

$$X(t+1) = (X(t) - 1)^+ + a(t). \tag{2.30}$$

As shown in the proof of Proposition 2.1.5, we have

$$\lim_{t\to\infty} \mathsf{P}(X(t) > 0) = \rho, \tag{2.31}$$

$$\lim_{t\to\infty} \mathsf{E}X(t) = \frac{1}{2}\frac{\sigma^2 + \rho - \rho^2}{1-\rho}, \tag{2.32}$$

where ρ and σ^2 are the mean and the variance of $a(t)$. Note that ρ is also the maximum throughput that we would like to find out under the assumption of infinite many packets queued at the input ports.

Finally, note that $a(t)$ is a (random) sum of Bernoulli random variables as every packet chooses its destination uniformly and independently. Thus, $a(t)$ converges to a Poisson random variable with mean ρ when $N \to \infty$. Recall that the variance of a Poisson random variable is the same as its mean. We then have from (2.29) and (2.32) that

$$\frac{1}{2}\frac{2\rho - \rho^2}{1-\rho} = 1. \tag{2.33}$$

As $0 \le \rho \le 1$, we have from (2.33) that $\rho = 2 - \sqrt{2}$.

2.2.3 Virtual output queueing

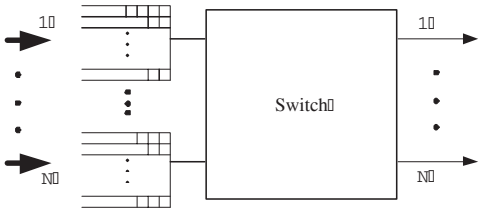


Fig. 2.8. Virtual output queueing

One quick way to solve the HOL blocking problem is known as the *virtual output queueing* (VOQ) technique. The VOQ technique (see Figure 2.8) maintains a separate queue for each output port at each input port, instead of having a single FIFO queue shared by all output ports at each input port. For instance, for an $N \times N$ switch, there are N queues at each input port and this is tantamount to a total number of N^2 queues at the input ports. Now we have (at most) N^2 HOL packets and we need to choose (at most) N of them to transmit through the switch fabric at any time slot. These HOL packets need to be chosen under the following two constraints: (i) no more than one packets can be from the same input port, and (ii) no more than one packets can be sent to the same output port.

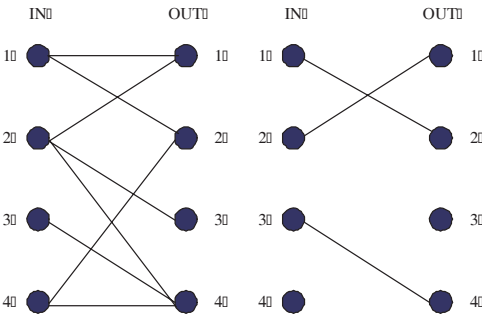


Fig. 2.9. A bipartite graph (left) and a matching of the bipartite graph (right)

Such a problem can be formulated as a bipartite matching problem. Recall that a bipartite graph $G = (V, E)$ (see Figure 2.9) is a graph in which the vertex set V is partitioned into two sets IN and OUT

such that every edge in E has one end in IN and the other in OUT. A *matching* in a bipartite graph is a set of edges of G such that no of them have a vertex in common. In our setting, the set IN contains N vertices with each vertex representing an input port. Similarly, the set OUT also contains N vertices with each vertex representing an output port. If the j^{th} VOQ at the i^{th} input port is not empty, then we set an edge from the i^{th} vertex from the IN set to the j^{th} vertex in the OUT set. The choice of the HOL packets is then equivalent to finding a matching in the corresponding bipartite graph.

To gain the intuition of a matching in a bipartite graph, one may consider the set IN as a set of men and the set OUT as a set of women. An edge between a man and a woman indicates that the man and the woman are fond of each other and eligible for a marriage. A matching is then to find a set of marriages (edges) such that a man is allowed to marry a woman and vice versa. We will further classify matchings as follows:

(i) A *maximal* matching is a matching that no more edges can be added.
(ii) A *maximum* matching is a matching that has the largest number of edges among all matchings.
(iii) A *perfect* matching is a matching that contains all the vertices in the bipartite graph.

Note that not every bipartite graph has a perfect matching. For example, the bipartite graph in Figure 2.9 does not have a perfect matching. On other hand, a bipartite graph may have more than one perfect matching. A perfect matching is always a maximum matching in a bipartite graph, and a maximum matching is always a maximal matching.

As indicated in [122], a good matching algorithm needs to have the following four properties:

(i) High throughput: ideally, once the no overbooking conditions in (2.27) and (2.28) are satisfied, then the expected queue length of each VOQ should be finite.
(ii) Starvation free: it is not desirable to have a non-empty queue to remain unserved indefinitely.
(iii) Fast: the computational complexity of finding the matching should be low.

(iv) Simple to implement: there should be an easy way to implement the algorithm, preferably in hardware.

In [6], a simple algorithm, called Parallel Iterative Matching (PIM), was proposed. There are three steps in each iteration:

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, it grants to one by randomly selecting a request uniformly.

Step 3. *Accept.* If an input receives a grant, it accepts one by randomly selecting a grant uniformly.

Example 2.2.2. (Parallel Iterative Matching) For example, consider the bipartite graph in Figure 2.9. Step 1 will send out requests as shown in Figure 2.10. In Step 2, output 1 grants input 2, output 2 grants input 1, output 3 grants input 2, and output 4 grants input 3. Finally, in Step 3, input 1 accepts the grant from output 2, input 2 accepts the grant from output 4.

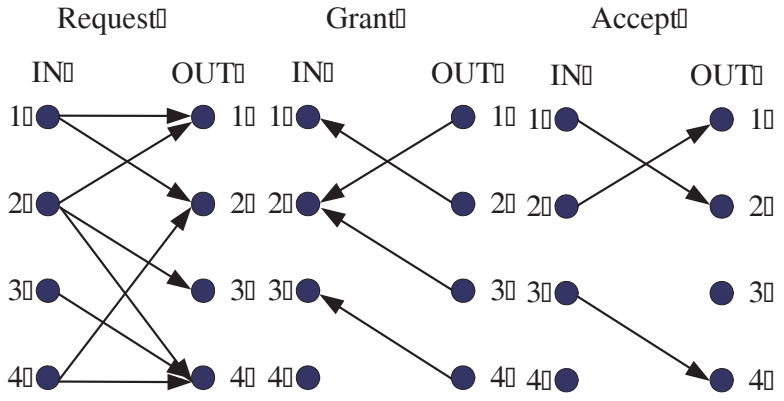


Fig. 2.10. A PIM iteration

Note that PIM uses randomness for arbitration. It also uses randomness to avoid starvation. As it can be carried out in parallel, it is also fast. By iterating the three steps above, PIM attempts to reach a

maximal matching, where no more edges can be added (in Figure 2.10, it reaches a maximal matching in one iteration). However, as pointed out in [122], it is not easy to implement *randomness* in high speed.

2.2.4 Round-robin matching

One simple way to avoid using random arbitration is to use *round-robin matching* (RRM). To do this, each input and each output keeps a pointer and the pointer rotates *clockwise* as described below.

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.

Note that there is no randomness in RRM and it is considerably simpler than PIM in the hardware design. However, as there is no randomness in RRM, the pointers in the RRM might become deterministic and periodic if all the VOQs are non-empty. As a result, the pointers in RRM are synchronized and RRM might be trapped in a bad mode with low throughput. Such a problem is known as the synchronization of grant pointers in [122]. We illustrate the problem by the following example in [122].

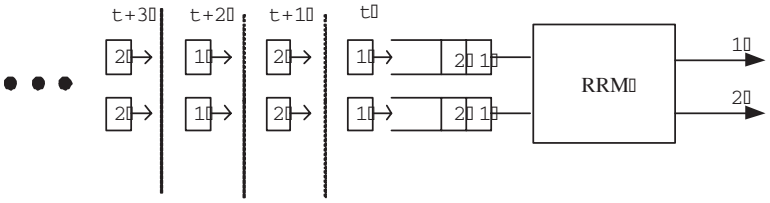


Fig. 2.11. The input traffic to a 2×2 RRM switch with one iteration

Example 2.2.3. (Synchronization of grant pointers in RRM) Consider a 2×2 switch that uses the round-robin matching with one

iteration (see Figure 2.11). At time $t-1$, both the first input buffer and the second input buffer have two packets destined to output ports 1 and 2. At time t, a packet destined to output port 1 arrives at the first buffer and the second buffer. At time $t+1$, another packet destined to output port 2 arrives at the first buffer and the second buffer. The input pattern then repeats itself from time $t+2$ onward as shown in Figure 2.11.

Now suppose all the pointers (at both the inputs and the outputs) are pointing at 1 at time t (see Figure 2.12). At the first step input 1 sends requests to output 1 and output 2, and input 2 sends requests to output 1 and output 2. As the pointer of output 1 is at 1, it grants input 1 at the second step and advances its pointer to input 2. Similarly, output 2 also grants input 1 at the second step and advances its pointer to input 2. Now input 1 receives grants from output 1 and output 2. As the pointer of input 1 is at 1, it accepts output 1 at the third step and advances its pointer to output 2.

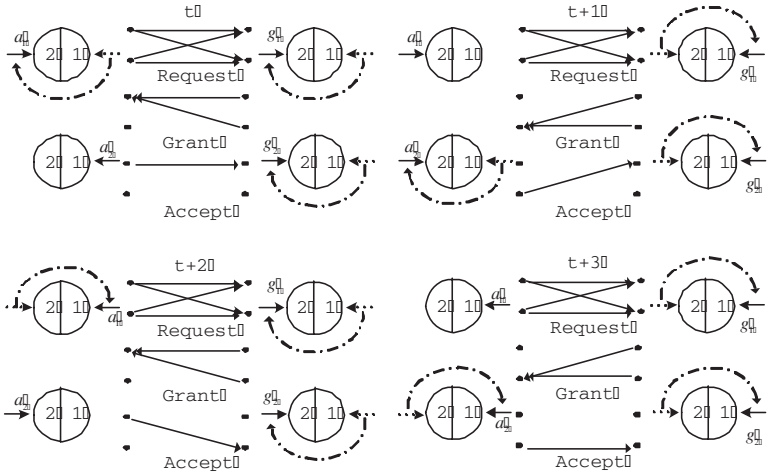


Fig. 2.12. An illustrating example for synchronization of grant pointers in a RRM switch with one iteration

At time $t+1$, both inputs still send requests to both outputs at the first step. As both the pointers of output 1 and output 2 are pointing at input 2, both outputs grant input 2 and advance their pointers to input 1 at the second step. At the third step, input 2 accepts output 1 and advance its pointer to output 2.

Following the same argument, it is easy to see that input 1 accepts output 2 at time $t+2$, input 2 accepts output 2 at time $t+3$. The connection patterns are then repeated every four time slots. As there is only one matched input-output pair for each time slot, the throughput is 50% in this example.

2.2.5 SLIP

To solve the pointer synchronization problem in RRM, McKeown [122] proposed the SLIP algorithm by not moving the grant pointers unless the grant is accepted. The second step in RRM is modified as follows:

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 if and only if the grant is accepted in Step 3.

The intuition that SLIP can solve the pointer synchronization problem in RRM is that SLIP introduces additional "randomness" on the movement of the grant pointers. Note that if the incoming traffic is random, then the event that an output after sending out an grant is accepted becomes a random event. As such, the grant pointers are less likely to be synchronized. Moreover, as its round-robin nature, SLIP will behave like a TDM system under heavy load for the uniform traffic in Example 2.2.3.

As in PIM, the above three steps can be carried out iteratively. The n-SLIP algorithm will be used to indicate that n SLIP iterations are carried out.

Even though SLIP has the desired four properties for the uniform traffic, it is still possible for SLIP to be trapped in a bad mode if the traffic is heavy, bursty and non-uniform. This is due to the fact that the event that an output after sending out an grant is accepted could become deterministic and periodic if the incoming traffic is not "random." We illustrate this by the following example in [34].

Example 2.2.4. (A bad mode of SLIP) Consider a 3×3 SLIP switch with the periodic input traffic shown in Figure 2.13. At time $t-1$, the first (resp. second, third) input buffer has two packets destined

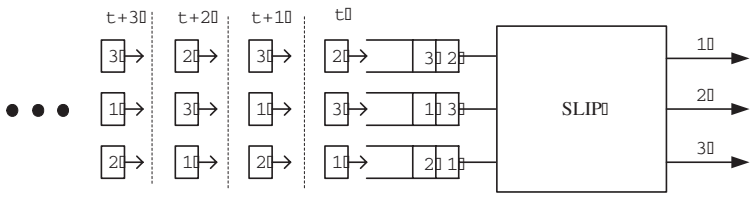


Fig. 2.13. The input traffic to a 3×3 SLIP switch

to output ports 2 and 3 (resp. 3 and 1, 1 and 2). At time t, a packet destined to output port 2 (resp. 3, 1) arrives at the first (resp. second, third) buffer. At time $t + 1$, another packet destined to output port 3 (resp. 1, 2) arrives at the first (resp. second, third) buffer. The input pattern then repeats itself from time $t + 2$ onward as shown in Figure 2.13.

Now suppose all the pointers (at both the inputs and the outputs) are pointing at 1 at time t. As shown in Figure 2.14, at the first step input 1 sends requests to output 2 and output 3, input 2 sends requests to output 1 and output 3, and input 3 sends requests to output 1 and output 2. Note that output 1 receives requests from input 2 and input 3. As the pointer of output 1 is at 1, it grants input 2 at the second step. Similarly, output 2 grants input 1 and output 3 grants input 1 at the second step. Now input 1 receives grants from output 2 and output 3. As the pointer of input 1 is at 1, it accepts output 2 at the third step. Both pointers of input 1 and output 2 need to be updated. The pointer of input 1 is moved to 3 (a_1 in Figure 2.14) and the pointer of output 2 is moved to 2 (g_2 in Figure 2.14). Similarly, as input 2 receives the only grant from output 1, it accepts output 1 at the third step. Now the pointer of input 2 is moved to 2 (a_2 in Figure 2.14) and the pointer of output 1 is moved to 3 (g_1 in Figure 2.14). It is clear that we already have a maximal matching and there is no need to run further SLIP iterations at time t. For this input traffic, readers can easily check that the SLIP algorithm (with as many iterations as possible) produces the connection patterns as shown in Figure 2.14. Note that all the pointers at $t + 2$ and $t + 5$ are the same and they yield the same connection pattern. As a result, SLIP is trapped in a periodical sequence of connection patterns and all these connection patterns can send two packets per time slot. Thus, the throughput in this example is only 66.667%, instead of 100% in an output-buffered

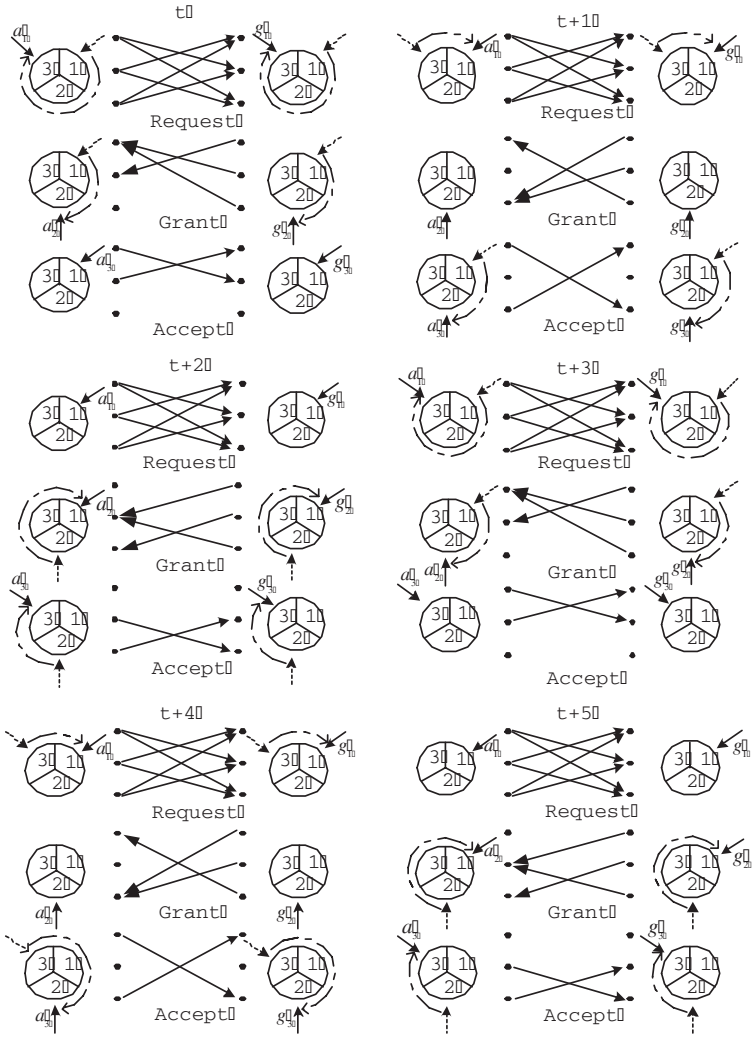


Fig. 2.14. An illustrating example of a bad mode in a SLIP switch

switch. For SLIP to get out of the trap, the traffic needs to be changed, and this might take a long time when the traffic is heavy and bursty.

2.3 Birkhoff-von Neumann switches

Both PIM and SLIP fail to provide 100% throughput for non-uniform traffic under the no overbooking conditions in (2.27) and (2.28). In this section, we show that if the rates $r_{i,j}$'s are known, then one can achieve 100% throughput for all non-uniform traffic in input-buffered crossbar switches (equipped with VOQs). The idea, first proposed in C.-S. Chang, W.-J. Chen and H.-Y. Huang [28], is to use the capacity decomposition approach by Birkhoff [17] and von Neumann [164]. To explain the idea, let $R = (r_{i,j})$ be the rate matrix with $r_{i,j}$ being the rate requested by the traffic from input i to output j for an $N \times N$ input-buffered crossbar switch. Then under the no overbooking conditions in (2.27) and (2.28), there exists a set of positive numbers ϕ_k and permutation matrices P_k, $k = 1, \ldots, K$ for some $K \leq N^2 - 2N + 2$ that satisfies

$$R \leq \sum_{k=1}^{K} \phi_k P_k, \quad \text{and} \tag{2.34}$$

$$\sum_{k=1}^{K} \phi_k = 1. \tag{2.35}$$

Such a decomposition may be obtained off-line and it only needs to be recomputed when the requested rates change. Once we obtain such a decomposition, we can simply schedule the connection pattern P_k proportional to its weight ϕ_k, $k = 1, \ldots, K$. One on-line scheduling algorithm to do this is the Packetized Generalized Processor Sharing (PGPS) algorithm in Parekh and Gallager [132] (or the Weighted Fair Queueing (WFQ) in Demers, Keshav, and Shenkar [58]).

2.3.1 The decomposition algorithm

In this section, we describe the Birkhoff-von Neumann algorithm to obtain the decomposition in (2.34).

A nonnegative matrix $R = (r_{i,j})$ that satisfies the conditions in (2.27) and (2.28) is known to be *doubly substochastic* (see e.g., Marshall and Olkin [120] for further properties of such matrices). If, furthermore, both inequalities in (2.27) and (2.28) are equalities, then the matrix $R = (r_{i,j})$ is called *doubly stochastic*. The decomposition basically consists of two steps: (i) it first finds a doubly stochastic matrix that is not smaller than the original rate matrix, and (ii) it then finds a decomposition for the doubly stochastic matrix obtained from the first step. The first step is based on the following result by von Neumann.

Proposition 2.3.1. (von Neumann [164]) *If a matrix $R = (r_{i,j})$ is doubly substochastic, then there exists a doubly stochastic matrix $\tilde{R} = (\tilde{r}_{i,j})$ such that*

$$r_{i,j} \leq \tilde{r}_{i,j}, \quad \forall i, j.$$

This can be constructed by the following algorithm.

Algorithm 1:

(i) If the sum of all the elements in R is less than N, then there exists an element (i, j) such that $\sum_n r_{i,n} < 1$ and $\sum_m r_{m,j} < 1$.

(ii) Let $\epsilon = 1 - \max[\sum_n r_{i,n}, \sum_m r_{m,j}]$. Add ϵ to the element in the $(i, j)^{th}$ position to obtain a new matrix R_1. Then in R_1, the number of row sums and column sums that are strictly smaller than 1 is at least one less than that in R.

(iii) Repeat this procedure until the sum of all the elements is equal to N. In this case, we derive a doubly stochastic matrix.

Clearly, the above algorithm will be ended after at most $2N - 1$ steps as the last step will make one row sum and one column sum to 1. As the matrix is $N \times N$, the computational complexity of finding an element (i, j) with $\sum_n r_{i,n} < 1$ and $\sum_m r_{m,j} < 1$ is $O(N^2)$. Thus, the computational complexity of Algorithm 1 is $O(N^3)$.

Example 2.3.2. (von Neumann algorithm) Consider the following doubly substochastic matrix

$$R = \begin{bmatrix} \underline{0.3} & 0.2 & 0.1 \\ 0.3 & 0.1 & 0.4 \\ 0.1 & 0.5 & 0.2 \end{bmatrix}.$$

We show how one uses the above algorithm to convert it into a doubly stochastic matrix. Note that the row sum that includes $r_{1,1}$ is 0.6 and the column sum that includes $r_{1,1}$ is 0.7. Thus, we can add 0.3 to $r_{1,1}$ and yields the following substochastic matrix

$$\begin{bmatrix} 0.6 & \underline{0.2} & 0.1 \\ 0.3 & 0.1 & 0.4 \\ 0.1 & 0.5 & 0.2 \end{bmatrix}.$$

By so doing, now the column sum of the first column is 1. Repeating the same steps by adding 0.1 to the $(1,2)^{th}$ element yields

$$\begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.3 & \underline{0.1} & 0.4 \\ 0.1 & 0.5 & 0.2 \end{bmatrix}.$$

Now the row sum of the first row is 1. Similarly, operating on the element at the $(2,2)^{th}$ element to have

$$\begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.3 & 0.2 & \underline{0.4} \\ 0.1 & 0.5 & 0.2 \end{bmatrix}.$$

Operating on the element at the $(2,3)^{th}$ element to have

$$\begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.3 & 0.2 & 0.5 \\ 0.1 & 0.5 & \underline{0.2} \end{bmatrix}.$$

Finally, operating on the element at the $(3,3)^{th}$ element to yield the following doubly stochastic matrix

$$\tilde{R} = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.3 & 0.2 & 0.5 \\ 0.1 & 0.5 & 0.4 \end{bmatrix}.$$

Note that the last step makes both the row sum of the third row and the column sum of the third column equal to 1.

The second step is based on Birkhoff's result on doubly stochastic matrices [17].

Proposition 2.3.3. (Birkhoff [17]) *For a doubly stochastic matrix $\tilde{R}$, there exists a set of positive numbers ϕ_k and permutation matrices P_k such that*

$$\tilde{R} = \sum_k \phi_k P_k.$$

Let e be the column vector with all its elements being 1. As $\tilde{R}$ is doubly stochastic,

$$e = \tilde{R}e = \sum_k \phi_k (P_k e) = (\sum_k \phi_k)e.$$

This shows that

$$\sum_k \phi_k = 1. \tag{2.36}$$

Basically, Proposition 2.3.3 says that doubly stochastic matrices are within the convex hull of the permutation matrices.

In the following, we present an algorithm to obtain the set of positive numbers ϕ_k and permutation matrices P_k. As indicated in Marshall and Olkin [120], the algorithm was originated by Dulmage and Halperin [60]. We note that the same algorithm was rediscovered by Inukai [85] for the time slot assignment problem in a satellite-switched time-division multiple access (SS/TDMA) system.

Algorithm 2:

(i) For a doubly stochastic matrix $\tilde{R}$, let $(i_1, i_2, \ldots, i_N)$ be a permutation of $(1, 2, \ldots, N)$ such that $\prod_{k=1}^{N} \tilde{r}_{k,i_k} > 0$. The existence of such a permutation for a doubly stochastic matrix is given by Mirsky [125], pp. 185 and Berge [15], pp. 105 (see Problem 10 for a proof by the Hall theorem [71]).

(ii) Let P_1 be the permutation matrix corresponding to $(i_1, i_2, \ldots, i_N)$, and $\phi_1 = \min_{1 \le k \le N}[\tilde{r}_{k,i_k}]$. Define the matrix R_1 by

$$R_1 = \tilde{R} - \phi_1 P_1.$$

(iii) If $\phi_1 = 1$, then

$$R_1 e = \tilde{R}e - P_1 e = \mathbf{0},$$

where $\mathbf{0}$ is the column vector with all its elements being zero. In this case, the matrix R_1 is the zero matrix as all the row sums are zero. Thus, we have completed the representation.

(iv) If $\phi_1 < 1$, then the matrix $\frac{1}{1-\phi_1}R_1$ is doubly stochastic and we can continue the decomposition from (i).

As each iteration reduces at least one element to zero, it is easy to see that the algorithm will be stopped by at most N^2-N+1 steps (the last one will reduce N elements to zero). In fact, as shown in [120] and references therein, the algorithm will be stopped by at most N^2-2N+2 steps. The number N^2-2N+2 cannot be reduced by a smaller number for the worse case. The main computation complexity of Algorithm 2 is to find a permutation $(i_1, i_2, \ldots, i_N)$ such that $\prod_{k=1}^{N} \tilde{r}_{k,i_k} > 0$. This can be done by converting the matrix $\tilde{R}$ into a 0-1 valued adjacency matrix of a bipartite graph and finding a maximum matching in that bipartite graph (note that a maximum matching is one of the largest maximal matchings in terms of the number of edges in the matching). In this case, a maximum matching is ensured to be perfect, i.e., every vertex is associated with exactly one edge. It is known (see e.g., Papadimitriou and Steiglitz [131], Theorem 10.2) that the computational complexity for the bipartite matching problem is $O(N^3)$ via the alternating path algorithm, and can be improved to $O(N^{2.5})$ via mapping the problem to the maximum flow problem. Thus, the computational complexity of Algorithm 2 is $O(N^{4.5})$.

Example 2.3.4. (Birkhoff decomposition) Consider the doubly stochastic matrix

$$\tilde{R} = \begin{bmatrix} \underline{0.6} & 0.3 & 0.1 \\ 0.3 & \underline{0.2} & 0.5 \\ 0.1 & 0.5 & \underline{0.4} \end{bmatrix}$$

obtained in Example 2.3.2. For such a matrix, the (identity) permutation $(1, 2, 3)$ contains all positive elements (with $\tilde{r}_{1,1} = 0.6$, $\tilde{r}_{2,2} = 0.2$ and $\tilde{r}_{3,3} = 0.4$). The smallest element of these three is $\tilde{r}_{2,2}$. Thus, we have

$$R_1 = \begin{bmatrix} \underline{0.4} & 0.3 & 0.1 \\ 0.3 & 0 & \underline{0.5} \\ 0.1 & \underline{0.5} & 0.2 \end{bmatrix} = \tilde{R} - 0.2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Now R_1 has one less zero elements than $\tilde{R}$. Moreover, $\frac{1}{1-0.2}R_1$ is still a doubly stochastic matrix. As the permutation $(1, 3, 2)$ contains all positive elements in R_1, following the same procedure yields

$$R_2 = \begin{bmatrix} 0 & \underline{0.3} & 0.1 \\ \underline{0.3} & 0 & 0.1 \\ 0.1 & 0.1 & \underline{0.2} \end{bmatrix} = R_1 - 0.4 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Now R_2 has one less zero elements than R_1 and two less zero elements than $\tilde{R}$. Following the same procedure, the readers should be able to show that

$$\begin{aligned} \tilde{R} &= 0.2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + 0.4 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} + 0.2 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &\quad + 0.1 \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} + 0.1 \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \end{aligned}$$

Thus, we then have for the Birkhoff decomposition that $\phi_1 = 0.2$, $\phi_2 = 0.4$, $\phi_3 = 0.2$, $\phi_4 = 0.1$ and $\phi_5 = 0.1$, and

$$P_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, P_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, P_3 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P_4 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, P_5 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

2.3.2 The on-line scheduling algorithm

In this section, we present the on-line scheduling algorithm used in the Birkhoff-von Neumann switch. From the constructions in Algorithms 1 and 2, we can find a set of positive numbers ϕ_k and permutation matrices P_k, $k = 1, \ldots, K$ for some $K \leq N^2 - 2N + 2$ that satisfies

$$R \leq \sum_{k=1}^{K} \phi_k P_k, \quad \text{and} \tag{2.37}$$

$$\sum_{k=1}^{K} \phi_k = 1. \tag{2.38}$$

In view of (2.37) and (2.38), the amount of time that the crossbar switch sets its connection pattern according to the permutation matrix P_k should be proportional to ϕ_k. As time is slotted, we can use

the Packetized Generalized Processor Sharing (PGPS) algorithm in Parekh and Gallager [132] (or the Weighted Fair Queueing (WFQ) in Demers, Keshav, and Shenkar [58]) to approximate the scheme.

Algorithm 3:

(i) Assign a class of tokens for each permutation matrix P_k, $k = 1, \ldots, K$.

(ii) At time 0, generate a token for each class. Assign the virtual finishing time of the first class k token by

$$F_k^1 = \frac{1}{\phi_k}.$$

Sort these K tokens in an increasing order of the virtual finishing times.

(iii) The switch serves the tokens in an increasing order of the virtual finishing times. Once a class k token is served in a time slot, the switch sets its connection pattern according to the permutation matrix P_k for that time slot.

(iv) After the $(\ell - 1)^{th}$ class k token is served, the switch generates the ℓ^{th} class k token and assign the virtual finishing time by

$$F_k^\ell = F_k^{\ell-1} + \frac{1}{\phi_k}.$$

Insert the newly generated token in the sorted token list and repeat the algorithm from (iii).

Note that Algorithm 3 is in fact a special case of PGPS. It corresponds to the case that there is an infinite number of packets at time 0 for PGPS. As the algorithm generates a new token of the same class after a token is served, there are exactly K classes of tokens in the sorted token list. Since the tokens are always backlogged, there is no need to keep track of the virtual finishing time. Thus, Algorithm 3 is independent of the buffer contents at any time.

The complexity of Algorithm 3 is to insert every new token into the sorted token list. As there are K tokens in the sorted list and $K \leq N^2 - 2N + 2$, the computational complexity of Algorithm 3 is $O(\log N^2)$. Since it takes $N \log_2 N$ bits to describe a permutation matrix and there are $O(N^2)$ permutation matrices needed to be stored in Algorithm 3, the memory complexity of Algorithm 3 is $O(N^3 \log N)$.

Example 2.3.5. (Packetized Generalized Processor Sharing (PGPS)) Consider the decomposition in Example 2.3.4. Let

$$V(t) = (V_1(t), V_2(t), V_3(t), V_4(t), V_5(t))$$

with $V_k(t)$, $k = 1, 2, 3, 4, 5$, be the virtual finishing time of the class k token at time t. At time 0, $V_k(0) = 1/\phi_k$. Since $\phi_1 = 0.2$, $\phi_2 = 0.4$, $\phi_3 = 0.2$, $\phi_4 = 0.1$ and $\phi_5 = 0.1$, we have $V(0) = (5, 2.5, 5, 10, 10)$. The smallest virtual finishing time is 2.5, which is the virtual finishing time of the class 2 token. We then schedule at time 1 the permutation matrix P_2 and increase the virtual finishing time of class 2 token from 2.5 to 5. Now $V(1) = (5, 5, 5, 10, 10)$. The minimum virtual finishing time is 5 and there are three of them. For ease of our presentation, we break the tie by choosing the one with the smallest index. Thus, we choose the class 1 token and schedule P_1 at time 2. Now increase the virtual finishing time of class 1 token from 5 to 10. This leads to $V(2) = (10, 5, 5, 10, 10)$. The emulation results for the first 10 time slots are summarized in Table 2.1. Note that the assignment of the permutation matrices repeats every 10 slots (as $V(10) = V(0) + (10, 10, 10, 10, 10)$). Moreover, in every 10 time slots, there are two time slots for P_1, four time slots for P_2, two times slots for P_3, one time slot for P_4, and one time slot for P_5. Thus, the proportional amount of time that P_k is scheduled is exactly ϕ_k.

In Figure 2.15, we show the time line for the tokens generated up to $t = 20$. To provide more intuition for Algorithm 3, one may think that the tokens for each class are generated periodically in advance along the time line. The algorithm simply sorts the tokens according to their generating time and sets up the corresponding connection patterns as shown in Figure 2.15.

2.3.3 Rate guarantees

Example 2.3.5 shows that the Birkhoff-von Neumann switch is capable of providing rate guarantees. We will formalize this in Theorem 2.3.6 below. Let $C_{i,j}(t)$ be the cumulative number of slots that are assigned to the traffic from input i to output j by time t. Also, let $p_{k,i,j}$ be the element at the (i, j) position of the permutation matrix P_k obtained from Algorithm 2. Note that $p_{k,i,j}$ only takes values 0 and 1. Define $E_{i,j} = \{k : p_{k,i,j} = 1\}$ to be the subset of $\{1, 2, \ldots, K\}$ such that for

Table 2.1. Emulation of the on-line scheduling algorithm

time slot	virtual time	permutation
0	(5,2.5,5,10,10)	
1	(5,5,5,10,10)	P_2
2	(10,5,5,10,10)	P_1
3	(10,7.5,5,10,10)	P_2
4	(10,7.5,10,10,10)	P_3
5	(10,10,10,10,10)	P_2
6	(15,10,10,10,10)	P_1
7	(15,12.5,10,10,10)	P_2
8	(15,12.5,15,10,10)	P_3
9	(15,12.5,15,20,10)	P_4
10	(15,12.5,15,20,20)	P_5

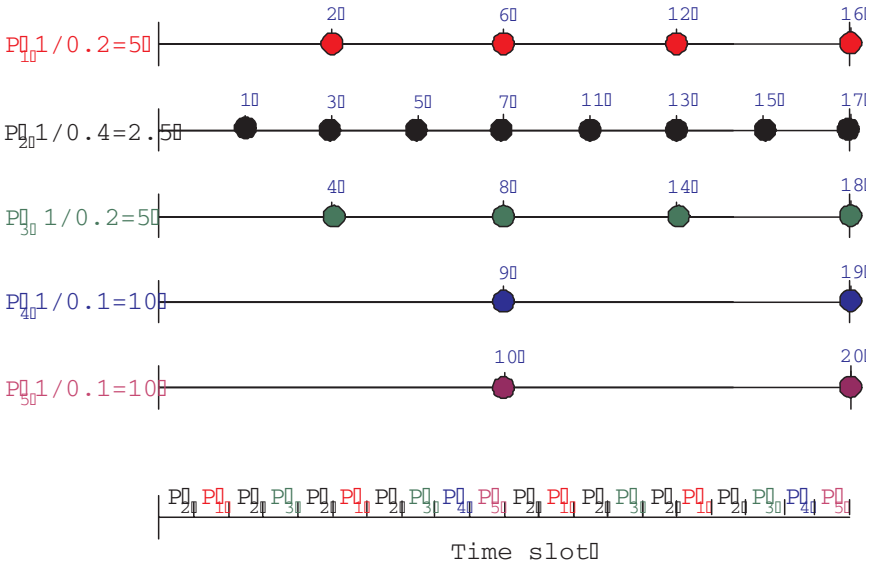


Fig. 2.15. The time like for the tokens generated up to $t = 20$ in Example 2.3.5

all k in $E_{i,j}$, the permutation matrix P_k has a nonzero element at the (i,j) position. In other words, $E_{i,j}$ contains the set of permutation matrices that allow input i to transmit a packet to output j.

Theorem 2.3.6. *For any rate matrix $R = (r_{i,j})$ that satisfies (2.27) and (2.28), Algorithms 1,2 and 3 generates a scheduling policy that guarantees*

$$\sum_{k\in E_{i,j}} \phi_k(t-s) - s_{i,j} \le C_{i,j}(t) - C_{i,j}(s) \le \sum_{k\in E_{i,j}} \phi_k(t-s) + s_{i,j}, \quad (2.39)$$

where

$$s_{i,j} = \min[K, |E_{i,j}| + \sum_{k\in E_{i,j}} \phi_k(K-1)], \quad (2.40)$$

and $|E_{i,j}|$ is the number of elements in the set $E_{i,j}$.

Note that K, the number of permutation matrices in (2.34), and ϕ_k's are the parameters obtained from Algorithm 2. As $K \le N^2 - 2N + 2$ in Algorithm 2, we have $s_{i,j} \le N^2 - 2N + 2$. Also, we have from (2.37) that

$$\sum_{k\in E_{i,j}} \phi_k = \sum_{k=1}^{K} \phi_k p_{k,i,j} \ge r_{i,j}. \quad (2.41)$$

This in turn implies

$$C_{i,j}(t) - C_{i,j}(s) \ge r_{i,j}(t-s) - N^2 + 2N - 2, \quad (2.42)$$

for all i, j, and $s \le t$. Ideally, we would like to achieve fluid type of rate guarantees, i.e.,

$$C_{i,j}(t) - C_{i,j}(s) = r_{i,j}(t-s).$$

However, this is unlikely as $r_{i,j}(t-s)$ may not be an integer. The constant $N^2 - 2N + 2$ is the difference between the rate guarantees in the Birkhoff-von Neumann switch and the ideal rate guarantees. Note that the constant is independent of the rate matrix r.

We will need the following lemma for the proof of Theorem 2.3.6. Let τ_k^ℓ be the index of the time slot that the ℓ^{th} class k token is served, and $D_k(t)$ be the cumulative number of class k tokens served by time t. These two quantities are related via the following inversion formula

$$D_k(t) = \sup\{\ell : \tau_k^\ell \le t\}, \quad (2.43)$$

i.e., $D_k(t) = \ell^*$ if ℓ^* is the largest ℓ such that $\tau_k^\ell \le t$.

Lemma 2.3.7. *For all $k = 1, \ldots, K$,*

$$\lfloor \phi_k t \rfloor \leq D_k(t) \leq \lfloor \phi_k (t + K - 1) \rfloor, \tag{2.44}$$

where $\lfloor x \rfloor$ is floor function of x, i.e., the largest integer that is less than or equal to x.

Using the inequality $x - 1 < \lfloor x \rfloor \leq x$, one has

$$\phi_k t - 1 < D_k(t) \leq \phi_k (t + K - 1). \tag{2.45}$$

Proof. As Algorithm 3 serves the tokens in an increasing order of the virtual finishing times, in the worst case the ℓ^{th} class k token is served after all the tokens with virtual finishing times not greater than F_k^ℓ are served. Note from (iv) in Algorithm 3 that the class k token is generated every $1/\phi_k$ unit of time. Thus, we have

$$F_k^\ell = \ell / \phi_k.$$

The number of class k tokens that have virtual finishing times not greater than F_k^ℓ is simply ℓ. On the other hand, the number of class j tokens (with $j \neq k$) that have virtual finishing times not greater than F_k^ℓ is

$$\left\lfloor \frac{F_k^\ell}{\frac{1}{\phi_j}} \right\rfloor = \lfloor \phi_j F_k^\ell \rfloor.$$

Thus, the total number of tokens that have virtual finishing times not greater than F_k^ℓ is

$$\ell + \sum_{j \neq k} \lfloor (\ell \phi_j)/\phi_k \rfloor. \tag{2.46}$$

As the algorithm serves a token per time slot, we have

$$\begin{aligned} \tau_k^\ell &\leq \ell + \sum_{j \neq k} \lfloor (\ell \phi_j)/\phi_k \rfloor \\ &\leq \ell + \sum_{j \neq k} (\ell \phi_j)/\phi_k = \ell/\phi_k = F_k^\ell, \end{aligned}$$

where we use the fact that $\lfloor x \rfloor \leq x$ in the second inequality. Thus, every token is served not later than its virtual finishing time.

Now we show that for all $k = 1, \ldots, K$,

$$D_k(t) \geq \lfloor \phi_k t \rfloor.$$

To see this, note from Algorithm 3 that the number of class k tokens that have virtual finishing times not greater than t is

$$\lfloor \phi_k t \rfloor.$$

All these class k tokens are served not later than their virtual finishing times and thus they are served not later than t.

We prove the second inequality in (2.44) by contradiction. Suppose that for some k and some t,

$$D_k(t) > \lfloor \phi_k(t + K - 1) \rfloor.$$

As $D_k(t)$ is an integer, this is equivalent to

$$D_k(t) \geq \lfloor \phi_k(t + K - 1) \rfloor + 1.$$

From the inversion formula in (2.43), the $\lfloor \phi_k(t + K - 1) \rfloor + 1^{th}$ class k token must be served not later than t, i.e.,

$$\tau_k^{\lfloor \phi_k(t+K-1) \rfloor + 1} \leq t.$$

As the algorithm serves the tokens in an increasing order of their virtual finishing times, those tokens with virtual finishing times less than $F_k^{\lfloor \phi_k(t+K-1) \rfloor + 1}$ must also be served not later than t. From the strict inequality $\lfloor x \rfloor + 1 > x$, it follows the number of class k tokens served not later than t is strictly larger than $\phi_k(t + K - 1)$. Also,

$$F_k^{\lfloor \phi_k(t+K-1) \rfloor + 1} = (\lfloor \phi_k(t + K - 1) \rfloor + 1)/\phi_k > t + K - 1.$$

Note that the number of class j tokens ($j \neq k$) with virtual finishing time not greater than $t + K - 1$ is

$$\lfloor (t + K - 1)\phi_j \rfloor,$$

and these tokens are also served not later than t as their virtual finishing times are strictly smaller than $F_k^{\lfloor \phi_k(t+K-1) \rfloor + 1}$. Thus, the total number of tokens served not later than t is strictly larger than

$$\phi_k(t + K - 1) + \sum_{j \neq k} \lfloor (t + K - 1)\phi_j \rfloor.$$

Since there are exactly t tokens served by time t, we have

$$\begin{aligned} t &> \phi_k(t + K - 1) + \sum_{j \neq k} \lfloor (t + K - 1)\phi_j \rfloor \\ &> \phi_k(t + K - 1) + \sum_{j \neq k} ((t + K - 1)\phi_j - 1) \\ &= t, \end{aligned}$$

and we reach a contradiction. ■

Proof. (Theorem 2.3.6) We first prove the lower bound in (2.39). Note that

$$C_{i,j}(t) = \sum_{k \in E_{i,j}} D_k(t).$$

Thus, it suffices to show that

$$\sum_{k \in E_{i,j}} D_k(t) - \sum_{k \in E_{i,j}} D_k(s) \geq (\sum_{k \in E_{i,j}} \phi_k)(t-s) - K, \tag{2.47}$$

and that

$$\begin{aligned} &\sum_{k \in E_{i,j}} D_k(t) - \sum_{k \in E_{i,j}} D_k(s) \\ &\geq (\sum_{k \in E_{i,j}} \phi_k)(t-s) - |E_{i,j}| - \sum_{k \in E_{i,j}} \phi_k(K-1). \end{aligned} \tag{2.48}$$

We now show (2.47). As exactly one token is served in a time slot,

$$\sum_{k=1}^{K} D_k(t) = t, \ \ \forall t. \tag{2.49}$$

Thus, we have from (2.49) and the lower bound in (2.45) that

$$\sum_{k \in E_{i,j}} D_k(t) = t - \sum_{k \notin E_{i,j}} D_k(t) \leq \sum_{k \in E_{i,j}} \phi_k t + (K - |E_{i,j}|), \ \ \forall t. \tag{2.50}$$

Using the lower bound in (2.45) and (2.50) yields

$$\begin{aligned} &\sum_{k \in E_{i,j}} D_k(t) - \sum_{k \in E_{i,j}} D_k(s) \\ &\geq \sum_{k \in E_{i,j}} (\phi_k t - 1) - \sum_{k \in E_{i,j}} \phi_k s - (K - |E_{i,j}|) \\ &= \sum_{k \in E_{i,j}} \phi_k (t-s) - K. \end{aligned}$$

To see (2.48), note from the lower bound and the upper bound in (2.45) that

$$\begin{aligned} &\sum_{k \in E_{i,j}} D_k(t) - \sum_{k \in E_{i,j}} D_k(s) \\ &\geq \sum_{k \in E_{i,j}} (\phi_k t - 1) - \sum_{k \in E_{i,j}} (\phi_k (s + K - 1)) \\ &= \sum_{k \in E_{i,j}} \phi_k (t-s) - |E_{i,j}| - \sum_{k \in E_{i,j}} \phi_k (K-1). \end{aligned}$$

To prove the upper bound in (2.39), we simply interchange the upper bound and the lower bound used for s and t in the proof for the lower bound in (2.39). ■

2.3.4 Framing

In this section, we consider a special case of the Birkhoff-von Neumann switch. We assume that there is an integer f so that the matrix $f\tilde{R}$ contains all integer-valued elements. Under such an assumption, there are several important properties as shown in Lemma 2.3.8.

Lemma 2.3.8. *For a doubly stochastic matrix $\tilde{R}$, suppose there is an integer f so that the matrix $f\tilde{R}$ contains all integer-valued elements.*

(i) Algorithm 2 is stopped by at most f steps.
(ii) $f\tilde{R}$ can be written as a sum of f permutation matrices.
(iii) Algorithm 3 is periodic with period f.

Proof. (i) The first property can be argued by induction. Note that if $f = 1$, then the doubly stochastic matrix $\tilde{R}$ must be a permutation matrix, and (i) is trivially satisfied. Suppose that (i) holds for any integer that is smaller than f as the induction hypothesis. Also, suppose that we have from Step (ii) in Algorithm 2 that $R_1 = \tilde{R} - \phi_1 P_1$ for some $\phi_1 > 0$ and some permutation matrix P_1. As ϕ_1 is one of the nonzero elements in $\tilde{R}$ and $f\tilde{R}$ is an integer-valued matrix, $f\phi_1$ is a positive integer. Since $R_1 = \tilde{R} - \phi_1 P_1$, we have

$$(f - f\phi_1)(\frac{1}{1-\phi_1}R_1) = f\tilde{R} - f\phi_1 P_1. \tag{2.51}$$

Note that both matrices on the right side of (2.51) are matrices with integer-valued elements. Thus, for $\phi_1 < 1$, the matrix

$$(f - f\phi_1)(\frac{1}{1-\phi_1}R_1)$$

is a matrix with all integer-valued elements. Since $f\phi_1$ is an integer not less than 1, we have from the induction hypothesis that Algorithm 2 will take at most another $f - 1$ steps to decompose $(\frac{1}{1-\phi_1}R_1)$. Thus, Algorithm 2 will be stopped by at most f steps for the decomposition of $\tilde{R}$.

(ii) The second property also follows from induction. The case for $f = 1$ is trivial as $\tilde{R}$ is a permutation matrix. Suppose that (ii) holds for any integer that is smaller than f as the induction hypothesis. From the induction hypothesis, the integer-valued matrix

$$(f - f\phi_1)(\frac{1}{1-\phi_1}R_1) = fR_1$$

can be written as a sum of $f - f\phi_1$ permutation matrices. As $f\tilde{R} = f\phi_1 P_1 + fR_1$, $f\tilde{R}$ can then be written as the sum of f permutation matrices.

(iii) As $f\tilde{R}$ can be written as a sum of f permutation matrices, $\tilde{R}$ can be written as a sum of f permutation matrices with a common weight $1/f$. Thus, Algorithm 3 is periodic with period f. ■

Example 2.3.9. (Framed Birkhoff-von Neumann switch) For instance, consider the matrix $\tilde{R}$ in Example 2.3.4. Note that $10\tilde{R}$ contains all integer-value elements. Moreover,

$$10\tilde{R} = 2P_1 + 4P_2 + 2P_3 + P_4 + P_5$$

is the sum of 10 permutation matrices. As shown in the emulation in Example 2.3.5, Algorithm 3 is periodic with period 10.

As Algorithm 3 is periodic with period f, there is no need to run Algorithm 3 any more. We may simply store all the f permutation matrices in the memory (if f is not too large). This results in a framing structure used in [79, 108].

Framing structure has its own advantages and disadvantages. Its main disadvantage is the granularity problem. The rate assigned to an input-output pair must be an integer multiple of *line speed*$/f$ (in our examples the line speed is normalized to 1), where f is the frame size (the number of slots in a frame). To achieve finer granularity, one has to increase the frame size f. However, this also increases the complexity of the switch as the switch has to memorize all the f permutation matrices.

The main advantage of the framing structure is the *incremental assignment* property. To see this, consider the rate matrix

$$R = \begin{bmatrix} 0.3 & 0.2 & 0.1 \\ 0.3 & 0.1 & 0.4 \\ 0.1 & 0.5 & 0.2 \end{bmatrix}.$$

in Example 2.3.2. As explained in Example 2.3.9, we can provide rate guarantees for the rate matrix R by using a framing structure with $f = 10$. Note that we increase some rates in R in order to convert R into the doubly stochastic matrix $\tilde{R}$ during Algorithm 1. Now we can deduct these back after the Birkhoff decomposition in Algorithm 2. This implies that we can use sub-permutation matrices to provide rate guarantees instead of the permutation matrices in Example 2.3.9. Specifically, we can use the following 10 sub-permutation matrices to guarantees the rates in R

$$\tilde{P}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tilde{P}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tilde{P}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

$$\tilde{P}_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \tilde{P}_5 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \tilde{P}_6 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

$$\tilde{P}_7 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tilde{P}_8 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tilde{P}_9 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix},$$

$$\tilde{P}_{10} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tag{2.52}$$

Note that $\tilde{P}_1$ and $\tilde{P}_2$ are the sub-permutation matrices from P_1, $\tilde{P}_3$, $\tilde{P}_4$, $\tilde{P}_5$ and $\tilde{P}_6$ are the sub-permutation matrices from P_2, $\tilde{P}_7$ and $\tilde{P}_8$ are the sub-permutation matrices from P_3, $\tilde{P}_9$ is the sub-permutation matrix from P_4, and $\tilde{P}_{10}$ is the sub-permutation matrix from P_5. Also,

$$\sum_{k=1}^{10} \tilde{P}_k = \begin{bmatrix} 3 & 2 & 1 \\ 3 & 1 & 4 \\ 1 & 5 & 2 \end{bmatrix} = 10R.$$

Now suppose that we would like to increase the rate $r_{2,2}$ from 0.l to 0.2. For this, we can simply change $\tilde{P}_2$ to

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \underline{1} & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

However, if we would like to increase the rate $r_{3,1}$ from 0.1 to 0.2. (the rate matrix is still doubly substochastic), we cannot add this directly to one of the $\tilde{P}_k$'s. One might wonder whether one needs to carry out the Birkhoff-von Neumann decomposition again to find out all the sub-permutation matrices needed to guarantee the rates. Fortunately, there is a much easier way to do this, called the Slepian-Duguid algorithm [151, 59] (see also Section 2.4.2 and the book by Hui [76] for its applications on rearrangeable networks).

First, we need a more compact representation for the f sub-permutation matrices, called Paull's matrix in [135, 76]. Paull's matrix is the $N \times N$ matrix H with $H_{i,j}$ being the set of indices of the sub-permutation matrices in which the $(i,j)^{th}$ element is 1, i.e.,

$$H_{i,j} = \{k : \tilde{p}_{k,i,j} = 1\},$$

where $\tilde{p}_{k,i,j}$ is the $(i,j)^{th}$ element of $\tilde{P}_k$. For example, Paull's matrix for the 10 sub-permutation matrices $\tilde{P}_i$, $i = 1, 2, \ldots, 10$, is

$$H = \begin{bmatrix} \{1,2,3\} & \{7,8\} & \{10\} \\ \{7,8,10\} & \{1\} & \{3,4,5,6\} \\ \{9\} & \{3,4,5,6,10\} & \{1,2\} \end{bmatrix}. \tag{2.53}$$

Let $|H_{i,j}|$ be the number of elements in $H_{i,j}$ and $|H|$ be the matrix that contains $|H_{i,j}|$ in its $(i,j)^{th}$ element. Since $|H_{i,j}| = fr_{i,j}$, we have

$$|H| = fR. \tag{2.54}$$

Thus, Paull's matrix H contains the decomposition of the sub-permutation matrices for the matrix fR.

Now we introduce the Slepian-Duguid algorithm for incremental assignment. Suppose that we would like to increase the rate $r_{i,j}$ by $1/f$ (the minimum rate granularity) and such an increase does not violate the no overbooking conditions in (2.27) and (2.28). There are only two cases that can happen.

Case 1 There exists an index that is not found in row i and column j of H.

Case 2 There exists an index c in row i that is not found in column j, and an index d in column j that is not found in row i.

To see this, note from (2.54) and the no overbooking conditions in (2.27) and (2.28) that

$$\sum_{j=1}^{N} |H_{i,j}| < f \tag{2.55}$$

and

$$\sum_{i=1}^{N} |H_{i,j}| < f. \tag{2.56}$$

These two are strict inequalities as H is Paull's matrix before we increase the rate. Let H_1 be the union of the sets $H_{i,j}$, $j = 1, 2, \ldots, N$, and H_2 be the union of the sets $H_{i,j}$, $i = 1, 2, \ldots, N$. Clearly, H_1 is the set that contains all the indices in row i, and H_2 is the set that contains all the indices in column j. If Case 1 does not happen, then the union of H_1 and H_2 equals to $\{1, 2, \ldots, f\}$. In view of (2.55), H_2 cannot be a subset of H_1 (otherwise the union of H_1 and H_2 has less than f indices). Similarly, from (2.56), it follows that H_1 cannot be a subset of H_2. Thus, there exists an index c in row i that is not found in column j, and an index d in column j that is not found in row i.

Slepian-Duguid algorithm:

(i) If Case 1 happens, then we can simply place the unfound index in $H_{i,j}$.

(ii) In Case 2, we look at the row where d appears in column j to see whether the index c appears in that row. If such a c is found, we then check the column of that c to see whether d can be found. Continue the process alternatively until we cannot find a c or d. Now we place d in $H_{i,j}$. Replace all c with d and all d with c for all c, d found in the process (see Figure 2.16).

Note that there are at most $2N - 2$ columns and rows (excluding row i and column j) that can be visited by the process in the Slepian-Duguid algorithm. Thus, the algorithm will be ended by at most $2N-2$ steps. Also, only the sub-permutation matrices $\tilde{P}_c$ and $\tilde{P}_d$ need to be changed.

Example 2.3.10. (The Slepian-Duguid algorithm) Suppose that we would like to increase the rate $r_{3,1}$ from 0.1 to 0.2 for the rate matrix R in Example 2.3.2. With Paull's matrix in (2.53), we cannot find an index that is not in row 3 and column 1. However, index 4 in $H_{3,2}$ is in row 3, but not in column 1. Index 7 in $H_{2,1}$ is column 1, but not in row 3. So we check row 2 to see whether index 4 can be found in

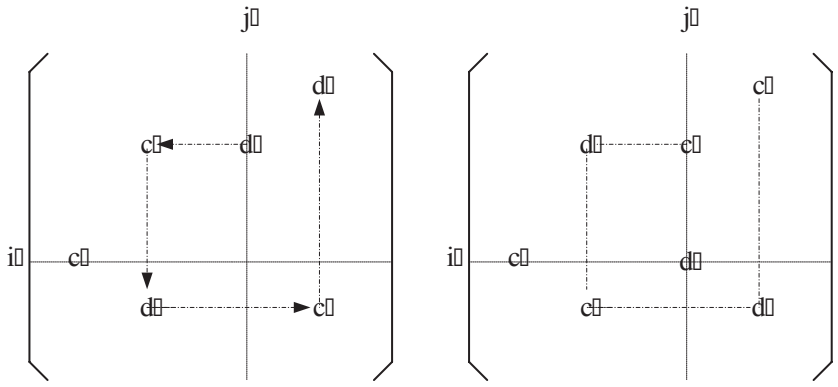


Fig. 2.16. Search and replace in the Slepian-Duguid algorithm

row 2. Since $H_{2,3}$ contains index 4, we need to continue the process and check whether index 7 can be found in column 3. As index 7 is not in column 3, the process stops. Now we add index 7 to $H_{3,1}$ and the modified $H_{3,1}$ is $\{7, 9\}$. Replace index 7 in $H_{2,1}$ by index 4 and the modified $H_{2,1}$ is $\{4, 8, 10\}$. Replace index 4 in $H_{2,3}$ by index 7 and the modified $H_{2,3}$ is $\{3, 5, 6, 7\}$. Only $\tilde{P}_4$ and $\tilde{P}_7$ are changed. They are now

$$\tilde{P}_4 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \qquad \tilde{P}_7 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

2.3.5 Maximum weighted matching algorithm

There are other scheduling algorithms that achieve 100% throughput as the Birkhoff-von Neumann switch. In this section, we briefly discuss the Longest Queue First (LQF) policy in the paper by McKeown, Anantharam and Walrand [123] and the paper by Dai and Prabhakar [56]. As in PIM and SLIP, the LQF policy establishes the bipartite graph by assigning a link from the i^{th} input to the j^{th} output if the j^{th} VOQ of the i^{th} input is non-empty. In addition to this, it also assigns a weight to each link. The weight of a link is the number of packets stored in the corresponding VOQ of that link. Thus, a longer queue carries a larger weight. The LQF policy then schedules the sub-permutation matrix (matching) that yields the maximum sum of weights.

To be precise, let $q_{i,j}(t)$ be the number of packets stored in the j^{th} VOQ of the i^{th} input at time t for an $N \times N$ switch. Formulate the following optimization problem:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N} p_{i,j}q_{i,j}(t) \\ \text{Subject to} \quad & \sum_{j=1}^{N} p_{i,j} \le 1,\ i = 1, 2, \ldots, N, \\ & \sum_{i=1}^{N} p_{i,j} \le 1,\ j = 1, 2, \ldots, N, \\ & p_{i,j} \in \{0, 1\},\ i, j = 1, 2, \ldots, N. \end{aligned} \tag{2.57}$$

The constraints above are equivalent to that $P = (p_{i,j})$ is a sub-permutation matrix. The LQF policy schedules the sub-permutation matrix that achieves the maximum in the optimization problem.

The optimization problem in (2.57) is known as the maximum weighted matching algorithm. As stated in [123], the complexity of solving the maximum weighted matching problem is $O(N^3 \log N)$. Since the maximum weighted matching problem needs to be solved every time slot, it is in general too complicated to be implemented in high speed switches.

Example 2.3.11. (Maximum weighted matching) Consider a 4×4 input-buffered switch. Suppose that

$$(q_{i,j}(t)) = \begin{bmatrix} 3 & 4 & 0 & 0 \\ 5 & 0 & 2 & 2 \\ 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 1 \end{bmatrix}.$$

In Figure 2.17, we show the associated weighted bipartite graph and its maximum weighted matching. The maximum of the optimization problem is achieved when

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

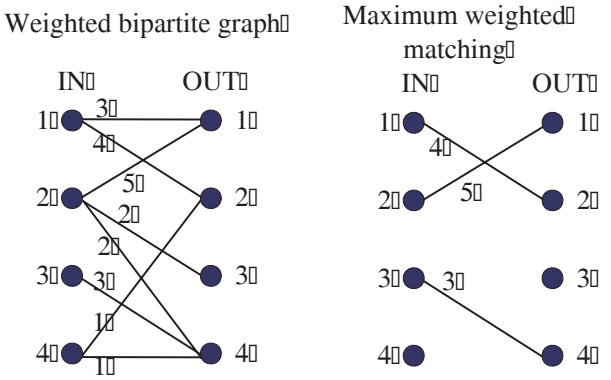


Fig. 2.17. A weighted bipartite graph and its maximum weighted matching

Here we provide an intuitive argument for the reason why the weighted matching algorithm provides 100% throughput under the no overbooking conditions. The formal proofs can be found in [56]. Consider the case that the time slot is extremely small so that one can view the system as a *continuous-time fluid model*. The input rate from the i^{th} input port to the j^{th} output port is a constant $r_{i,j}$. Let $P(t) = (p_{i,j}(t))$ be the sub-permutation matrix scheduled at time t by the maximum weighted matching algorithm. In the fluid model, $p_{i,j}(t)$ is the output rate of the j^{th} VOQ at the i^{th} input port at time t. Thus, instead of having the Lindley recursion in the discrete-time model, we have in the continuous-time fluid model the following differential equation:

$$q'_{i,j}(t) = r_{i,j} - p_{i,j}(t), \quad q_{i,j}(t) > 0. \tag{2.58}$$

Let

$$V(t) = \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{1}{2} (q_{i,j}(t))^2. \tag{2.59}$$

The function $V(t)$ is known as the Liapunov function in [56]. Note that Liapunov function is nonnegative and convex in $q_{i,j}(t)$'s. Thus, it has a global minimum at $q_{i,j}(t) = 0$ for all i, j. Moreover, if the Liapunov function $V(t)$ is bounded, then all the $q_{i,j}(t)$'s are also bounded. The idea is then to show that the Liapunov function $V(t)$ is always decreasing in time so that it cannot grow unbounded. Note from (2.58) that

$$V'(t) = \sum_{i=1}^{N}\sum_{j=1}^{N} q_{i,j}(t)q'_{i,j}(t)$$
$$= \sum_{i=1}^{N}\sum_{j=1}^{N} q_{i,j}(t)r_{i,j} - \sum_{i=1}^{N}\sum_{j=1}^{N} q_{i,j}(t)p_{i,j}(t). \tag{2.60}$$

As we assume that the rate matrix $R = (r_{i,j})$ satisfies the no overbooking conditions in (2.27) and (2.28), it then follows from the Birkhoff-von Neumann decomposition that

$$R \le \sum_{k=1}^{K} \phi_k P_k,$$

for some positive ϕ_k (with $\sum_{k=1}^{K} \phi_k = 1$) and permutation matrices $P_k = (p_{k,i,j})$. Thus,

$$\sum_{i=1}^{N}\sum_{j=1}^{N} q_{i,j}(t)r_{i,j} \le \sum_{i=1}^{N}\sum_{j=1}^{N} q_{i,j}(t) \sum_{k=1}^{K} \phi_k p_{k,i,j}$$
$$= \sum_{k=1}^{K} \phi_k [\sum_{i=1}^{N}\sum_{j=1}^{N} q_{i,j}(t)p_{k,i,j}]$$
$$\le \max_{1\le k\le K} [\sum_{i=1}^{N}\sum_{j=1}^{N} q_{i,j}(t)p_{k,i,j}]$$
$$\le \sum_{i=1}^{N}\sum_{j=1}^{N} q_{i,j}(t)p_{i,j}(t),$$

as $P(t) = (p_{i,j}(t))$ is chosen to maximize the weighted sum among all the sub-permutation matrices. From (2.60), we have $V'(t) \le 0$. As such, $V(t)$ cannot grow unbounded (in fact, $V(t) = 0$ for all t if we start from $V(0) = 0$). So are $q_{i,j}(t)$'s.

Instead of finding the maximum weight matching, one can simply find a maximal matching, as in PIM and (iterated) SLIP. However, as shown in Example 2.2.4, the throughput for a maximal matching algorithm is not 100% in general. An interesting result, as formally proved in [56], is that the throughput for maximal matching algorithms is at least 50%. To see the intuition behind this, suppose that

$$\sum_{j=1}^{N} r_{i,j} \le \frac{1}{2}, \quad i = 1, 2, \ldots, N, \tag{2.61}$$

$$\sum_{i=1}^{N} r_{i,j} \le \frac{1}{2}, \quad j = 1, 2, \ldots, N. \tag{2.62}$$

Let $P(t) = (p_{i,j}(t))$ be the sub-permutation matrix scheduled at time t under a maximal matching algorithm. Clearly, (2.58) still holds. Now consider the Liapunov function

$$\tilde{V}(t) = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}[q_{i,k}(t) + q_{k,j}(t)]. \tag{2.63}$$

Note that

$$\begin{aligned}
\tilde{V}'(t) &= \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}[q'_{i,k}(t) + q'_{k,j}(t)] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}[r_{i,k} - p_{i,k}(t) + r_{k,j} - p_{k,j}(t)] \\
&\le \sum_{i=1}^{N}\sum_{j=1}^{N}\Big[\frac{1}{2} + \frac{1}{2} - \sum_{k=1}^{N} p_{i,k}(t) - \sum_{k=1}^{N} p_{k,j}(t)\Big].
\end{aligned} \tag{2.64}$$

Note that if $q_{i,j}(t) > 0$, then a maximal matching ensures that there is at least one edge connected to either input port i or output port j (as otherwise we can simply connect input i to output j and the matching would not be maximal). Thus, if $q_{i,j}(t) > 0$,

$$\sum_{k=1}^{N} p_{i,k}(t) + \sum_{k=1}^{N} p_{k,j}(t) \ge 1.$$

From (2.64), we have $\tilde{V}'(t) \le 0$ if $q_{i,j}(t) > 0$ for all i, j. Thus, $\tilde{V}(t)$ cannot be unbounded. So are $q_{i,j}(t)$'s.

2.4 Three-stage constructions of switch fabrics

As shown in Figure 2.5, the number of cross points in an $N \times N$ crossbar switch fabric is N^2, which may not scale for switches with a large number of input/output ports. The idea is to build a large crossbar switch fabric by connecting smaller crossbar switch fabrics. This has been extensively studied in the area of circuit switching (see e.g., [14, 76, 147, 84, 111]). In circuit switching, a crossbar switch fabric allows connections to be established in an incremental manner, i.e., a new

connection (from a free input to a free output) can be added without affecting existing connections. A switch fabric with such a property is known as a *strictly nonblocking* (or simply nonblocking) switch. In Section 2.4.1, we will show how one builds nonblocking switches using three-stage constructions.

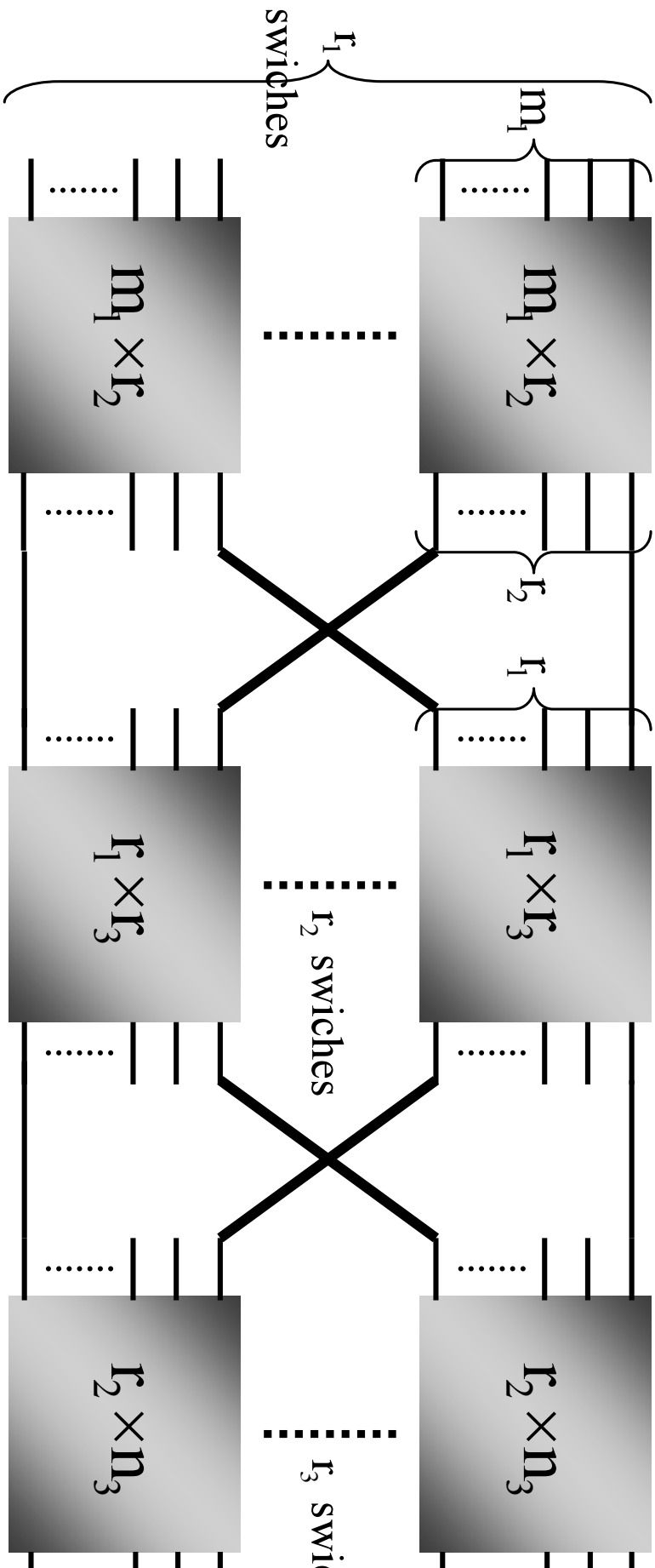
A weaker concept of nonblocking is *rearrangeable nonblocking*. A new connection (from a free input to a free output) can be added by altering the routes of existing connections. Switch fabrics with such a property are called rearrangeable nonblocking switches (or simply rearrangeable switches). As new connections can be added from free inputs to free outputs, a rearrangeable nonblocking switch realizes all the connection patterns that correspond to sub-permutations. In Section 2.4.2, we will introduce three-stage rearrangeable networks and their algorithms for setting up new connections.

2.4.1 Clos networks

In Figure 2.18, we show a three-stage Clos network. The first stage consists of r_1 $m_1 \times r_2$ switches. Every one of the r_2 outputs from a switch at the first stage is connected to an input of the r_2 $r_1 \times r_3$ switches at the second stage. Every one of the r_3 outputs from a switch at the second stage is connected to an input of the r_3 $r_2 \times n_3$ switches at the third stage. In this three-stage switch, there are $r_1 \times m_1$ inputs and $r_3 \times n_3$ outputs.

Theorem 2.4.1. *(Clos [51]) Suppose that every switch inside the three-stage Clos network is nonblocking. If $r_2 \geq m_1 + n_3 - 1$, then the three-stage Clos network in Figure 2.18 is also nonblocking.*

Proof. To show that the three-stage Clos network is nonblocking, we need to show that there is a non-conflicting path from a free input to a free output. Note that there are exactly r_2 paths from an input to an output (with each path passing through a switch at the second stage). Thus, if we would like to connect a free input of switch a at the first stage to a free output of switch b at the third stage, then we must find a switch at the second stage that is not being used by the other $m_1 - 1$ inputs of switch a and the other $n_3 - 1$ outputs of switch b. The worst case is when the other $m_1 - 1$ inputs of switch a and the other $n_3 - 1$ outputs of switch b all use different switches at the second stage. When this happens, there are $m_1 + n_3 - 2$ conflicting switches


r1
swiches
m1
m1 ×r2
r2
r1
r1 ×r3
r2 swiches
r2 ×n3
r3 swi

at the second stage. As $r_2 \geq m_1 + n_3 - 1$, there is always a free one at the second stage and one can connect a free input to a free output without conflicting other existing connections. ■

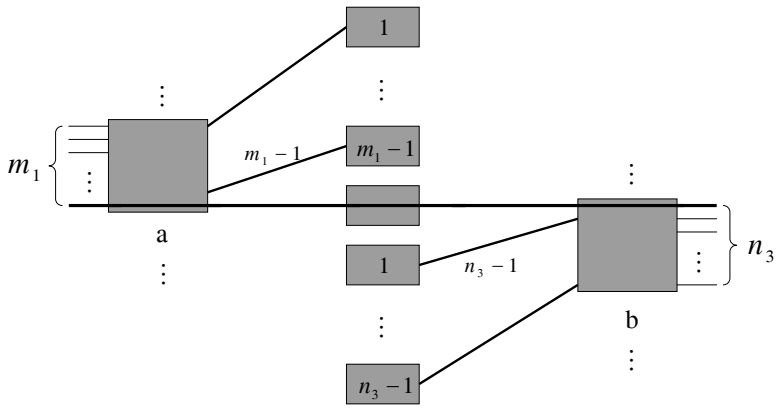


Fig. 2.19. The worst case of a nonblocking Clos network

Consider the special case that $m_1 = n_3 = r_1 = r_3 = n$ and $r_2 = 2n - 1$. From Theorem 2.4.1, it is a nonblocking $n^2 \times n^2$ switch. Note that the switches at first stage are all $n \times (2n-1)$. Thus, every n inputs are expanded into $2n-1$ inputs at the second stage. The "expanding" factor is then $(2n-1)/n = 2 - 1/n$. Such a factor is also the speedup factor needed for exact emulation in Section 2.6.2.

The complexity of the three-stage Clos network is considerably smaller than that of a crossbar switch fabric. To see this, note that the number of cross points for this special case is

$$(n \times (2n-1)) \times n + (n \times n) \times (2n-1) + ((2n-1) \times n) \times n = O(n^3),$$

which is smaller than $O(n^4)$ for an $n^2 \times n^2$ crossbar switch fabric.

2.4.2 Rearrangeable networks

A special case of the three-stage Clos network is when $m_1 = r_2 = n_3 = m$ and $r_1 = r_3 = k$ as shown in Figure 2.20. Let $N = m \times k$. Thus, the three-stage network in Figure 2.20 is an $N \times N$ switch fabric.

Fig. 2.20. A three-stage rearrangeable network

Theorem 2.4.2. *Suppose that every switch inside the three-stage construction in Figure 2.20 is rearrangeable nonblocking, i.e., each $m \times m$ (resp. $k \times k$) switch realizes all the $m \times m$ (resp. $k \times k$) sub-permutation matrices. Then the three-stage network in Figure 2.20 is also rearrangeable nonblocking, i.e., it realizes all the $N \times N$ sub-permutation matrices. Equivalently, one can find non-conflicting paths from all inputs to all outputs as long as the connection pattern is a sub-permutation matrix.*

Proof. We give a constructive proof for this. Number the switches at the first stage from 1 to k. Do the same for the switches at the third stage. As such, we can use the $(i, \ell)^{th}$ input, $i = 1, 2, \ldots, k$, $\ell = 1, 2, \ldots, m$, to denote the ℓ^{th} input of the i^{th} switch at the first stage. Similarly, we use the $(j, \ell)^{th}$ output, $j = 1, 2, \ldots, k$, $\ell = 1, 2, \ldots, m$, to denote the ℓ^{th} output of the j^{th} switch at the third stage. We use the indicator variables $p_{(i,\ell_1),(j,\ell_2)}$, $i, j = 1, 2 \ldots, k$, $\ell_1, \ell_2 = 1, 2, \ldots, m$, to represent a connection pattern between the N inputs and the N outputs, i.e., we set $p_{(i,\ell_1),(j,\ell_2)} = 1$ if there is a connection from the $(i, \ell_1)^{th}$ input to the $(j, \ell_2)^{th}$ output and 0 otherwise.

Now we show how one finds non-conflicting paths between the N inputs and the N outputs. For $i, j = 1, 2, \ldots, k$, let

$$q_{i,j} = \sum_{\ell_1=1}^{m} \sum_{\ell_2=1}^{m} p_{(i,\ell_1),(j,\ell_2)}. \tag{2.65}$$

The quantity $q_{i,j}$ represents the number of paths required from the i^{th} switch at the first stage to the j^{th} switch at the third stage. Note that the number of paths from the i^{th} switch at the first stage cannot be larger than the number of its inputs. Thus,

$$\sum_{j=1}^{k} q_{i,j} \leq m, \quad i = 1, 2, \ldots, k. \tag{2.66}$$

Similarly, the number of paths to the j^{th} switch at the third stage cannot be larger than the number of its outputs. Thus,

$$\sum_{i=1}^{k} q_{i,j} \leq m, \quad j = 1, 2, \ldots, k. \tag{2.67}$$

These two conditions are similar to the no overbooking conditions in (2.27) and (2.28). From Lemma 2.3.8(ii) (the Birkhoff-von Neumann decomposition for an integer-valued matrix), it follows that the

$k \times k$ matrix $Q = (q_{i,j})$ can be decomposed as the sum of m $k \times k$ sub-permutation matrices, $\tilde{P}_1, \tilde{P}_2, \ldots, \tilde{P}_m$. By setting the connection patterns of the m $k \times k$ switches at the second stage with these m sub-permutation matrices, we then have enough number of paths from any switch at the first stage to any stage at the third stage. For instance, if the $(i,j)^{th}$ element of some sub-permutation matrix $\tilde{P}_{m_0}$ is 1, then a connection from the $(i,\ell_1)^{th}$ input to the $(j,\ell_2)^{th}$ output can be routed through the ${m_0}^{th}$ switch at the second stage. ■

Example 2.4.3. (Rearrangeable networks) Consider a three-stage network in Figure 2.20 with $m = 10$ and $k = 3$. Suppose that

$$Q = \begin{bmatrix} 3 & 2 & 1 \\ 3 & 1 & 4 \\ 1 & 5 & 2 \end{bmatrix}.$$

Then we can represent Q as the sum of the 10 sub-permutation matrices in (2.52) and use them as the connection patterns for the 10 switches at the second stage. The three paths require from the first switch at the first stage to the first switch at the third stage can be routed through the first, the second and the third switches at the second stage.

Note that the implementation complexity of the three-stage network in Figure 2.20 is much smaller than the nonblocking three-stage network in Theorem 2.4.1. However, finding the paths between the inputs and the outputs is much more difficult than that in Theorem 2.4.1. More importantly, if one would like to add one more connection, one does not need to re-route existing paths in the nonblocking three-stage network in Theorem 2.4.1. However, it might require carrying out another decomposition in Theorem 2.4.2 and thus re-routing existing paths. As such, the three-stage Clos networks in Figure 2.20 are called *rearrangeable* networks. An efficient algorithm that does re-routing existing paths is the Slepian-Duguid algorithm described in Section 2.3.4.

Example 2.4.4. (The Slepian-Duguid algorithm for rearrangeable networks) Continue from Example 2.4.3. Suppose that we

would like to add another connection from the 3rd switch at the first stage to the 1st switch at the third stage. This problem is equivalent to the rate increase problem in Example 2.3.10. The Slepian-Duguid algorithm in Example 2.3.10 shows that two existing connections need to be rearranged. One connection from the 2nd switch at the first stage to the 1st switch at the third stage is moved from the 7th switch at the second stage to the 4th switch at the second stage. The other connection from the 2nd switch at the first stage to the 3rd switch at the third stage is moved from the 4th switch at the second stage to the 7th switch at the second stage.

Instead of using the Birkhoff-von Neumann decomposition, here we introduce a more efficient method, called the Lee-Hwang-Capinelli algorithm [106, 84], for finding the sub-permutation matrices in the central stage in Figure 2.20. For this, we need to extend the $k \times k$ Paull matrix to a $k \times m$ specification matrix that explicitly specifies every sub-permutation in the Paull matrix. In the $k \times m$ specification matrix, the ℓ^{th} column represents the sub-permutation for the ℓ^{th} sub-permutation matrix. For instance, the specification matrix for the 10 sub-permutation matrices in (2.52) is

$$\begin{bmatrix} 1 & 1 & 1 & - & - & - & 2 & 2 & - & 3 \\ 2 & - & 3 & 3 & 3 & 3 & 1 & 1 & - & 1 \\ 3 & 3 & 2 & 2 & 2 & 2 & - & - & 1 & 2 \end{bmatrix}.$$

In this specification matrix, the fourth column represents the sub-permutation

$$\begin{aligned} 1 &\mapsto - \\ 2 &\mapsto 3 \\ 3 &\mapsto 2 \end{aligned}$$

and the sub-permutation matrix

$$\tilde{P}_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

The Lee-Hwang-Capinelli algorithm:

(i) As defined in (2.65), let $q_{i,j}$ be the number of paths required from the i^{th} switch at the first stage to the j^{th} switch at the third switch. In [84], the integer-valued $k \times k$ matrix $Q = (q_{i,j})$ is called the frame

matrix. As shown in the proof of Theorem 2.4.2, all the row sums and column sums are bounded by m. As such, one can use the von Neumann algorithm to convert the frame matrix Q to a matrix $\tilde{Q} = (\tilde{q}_{i,j})$ so that all the row sums and column sums in the matrix $\tilde{Q}$ are equal to m.

(ii) From the matrix $\tilde{Q}$, construct an initial $k \times m$ specification matrix $H = (h_{i,\ell})$ by arbitrarily assigning j to the entries of the i^{th} row of H for $\tilde{q}_{i,j}$ times, $j = 1, 2 \ldots, k$. As every column sum of $\tilde{Q}$ is m, the total number of j's in the initial specification matrix is also m for all $j = 1, \ldots, k$. Also, as the assignment is arbitrary, the columns of the initial specification matrix may not be permutations. The objective of the algorithm is then to carry out a sequence of pairwise interchanges (swaps) of two assignments so that every column of the specification matrix is a permutation. For this, a column is called j-excessive if it contains more than one j, and j-deficient if it contains no j. An element j is called *balanced* if no j-excessive column exists. Note that when j is balanced, there is also no j-deficient column as the number of j's in the specification matrix is equal to m, the number of columns in the specification matrix.

(iii) Find the **smallest** j such that a j-excessive column ℓ_1 and a j-deficient column ℓ_2 exist. Suppose that $h_{i,\ell_1} = j$ and $h_{i,\ell_2} = x$ (for some row i). Pairwise interchange these two elements on the same row, i.e., set $h_{i,\ell_1} = x$ and $h_{i,\ell_2} = j$. If $j < x$, then go back to (iii).

(iv) Suppose $j > x$. From (iii), we know that x is balanced before the pairwise interchange. However, after the pairwise interchange, column ℓ_1 becomes x-excessive and column ℓ_2 become x-deficient. Thus, there exists a row $i_1 \neq i$ such that $h_{i_1,\ell_1} = x$. Suppose that $h_{i_1,\ell_2} = y$. Pairwise interchange these two elements on row i_1, i.e., set $h_{i_1,\ell_1} = y$ and $h_{i_1,\ell_2} = x$. If $j < y$, then go back to (iii). If $j > y$, then (iv) has to be repeated as y is now not balanced. Since x, y, ... are all distinct, the number of pairwise interchanges in (iv) is at most j until we find an element $z > j$. When we find an element $z > j$, we then go back to (iii).

(v) Deduct the assignments added from the von Neumann algorithm so that the columns of the specification matrix H contains sub-permutations needed for the frame matrix Q.

To compute the complexity of this algorithm, note that for each j-excessive column in (iii), it takes at most j pairwise interchanges. As the number of j-excessive columns is bounded by m, the number

of pairwise interchanges needed for j to be balanced is bounded by mj. Summing up $j = 1, 2, \ldots, k$ shows that the complexity of this algorithm is $O(mk^2) = O(Nk)$, where $N = mk$ is the total number of inputs of the $N \times N$ three stage switch.

Example 2.4.5. (The Lee-Hwang-Capinelli algorithm) As in Example 2.4.3, we consider a three-stage network in Figure 2.20 with $m = 10$ and $k = 3$. Suppose that

$$Q = \begin{bmatrix} 3 & 2 & 1 \\ 3 & 1 & 4 \\ 1 & 5 & 2 \end{bmatrix}.$$

From the von Neumann algorithm (see Example 2.3.2), we have

$$\tilde{Q} = \begin{bmatrix} 6 & 3 & 1 \\ 3 & 2 & 5 \\ 1 & 5 & 4 \end{bmatrix}.$$

Thus, we need to assign six 1's, three 2's, and a 1 in the first row of the initial specification matrix. The trivial way to do this is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 \end{bmatrix}.$$

For the second row, there should be three 1's, two 2's and five 3's, and it can be chosen as follows:

$$\begin{bmatrix} 2 & 2 & 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 \end{bmatrix}.$$

For the third row, there should be a 1, five 2's and four 3's, and it can be chosen as follows:

$$\begin{bmatrix} 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \end{bmatrix}.$$

This yields the following initial specification matrix

$$H = \begin{bmatrix} \underline{1} & 1 & 1 & 1 & 1 & 1 & \underline{2} & 2 & 2 & 3 \\ 2 & 2 & 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \end{bmatrix}.$$

Now the first column is 1-excessive and the 7^{th} column is 1-deficient. Thus, we pairwise interchange these two element (according to (iii)) and yield

$$H = \begin{bmatrix} \underline{2} & 1 & 1 & 1 & 1 & 1 & \underline{1} & 2 & 2 & 3 \\ 2 & 2 & 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \end{bmatrix}.$$

Up to this point, the element 1 is balanced. The next smallest element that has not been balanced is 2. Note that the first column is 2-excessive and the 7^{th} column is 2-deficient. Thus, we pairwise interchange these two element (according to (iii)) and yield

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 \\ 2 & 2 & 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 \\ \underline{1} & 2 & 2 & 2 & 2 & 2 & \underline{3} & 3 & 3 & 3 \end{bmatrix}.$$

From (iv), we know that 1 is not balanced any more, and we find another 1 in the third row of the first column. We then interchange that with the 3 in the third row of the 7^{th} column. This yields

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 \\ 2 & 2 & 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 \\ 3 & \underline{2} & 2 & 2 & 2 & 2 & 1 & 3 & 3 & \underline{3} \end{bmatrix}.$$

Now we go back to (iii) and find the second column is 2-excessive and the 10^{th} column is 2-deficient. We then pairwise interchange the two elements in the third row and this leads to

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 \\ 2 & 2 & 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 \\ 3 & 3 & 2 & 2 & 2 & 2 & 1 & 3 & 3 & 2 \end{bmatrix}.$$

Now every column of H is a permutation. Finally, we deduct the elements added from the von Neumann algorithm to derive the following specification matrix

$$H = \begin{bmatrix} 1 & 1 & 1 & - & - & - & - & 2 & 2 & 3 \\ 2 & - & 3 & 3 & 3 & 3 & - & 1 & 1 & 1 \\ 3 & 3 & 2 & 2 & 2 & 2 & 1 & - & - & 2 \end{bmatrix}.$$

This corresponds to the 10 sub-permutation matrices in (2.52) (with the indexes renumbered).

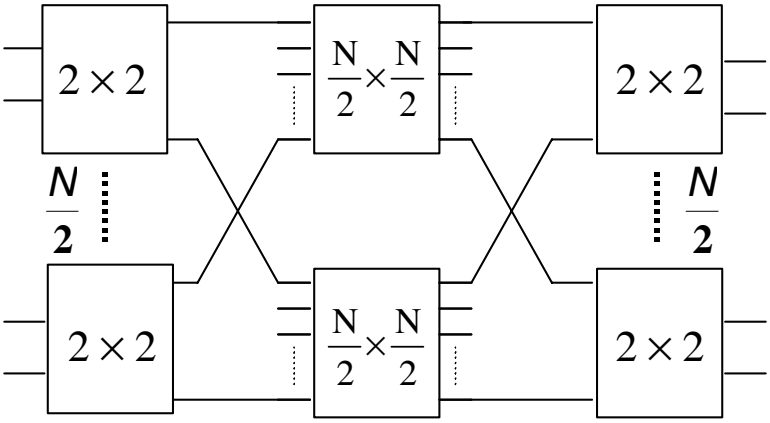


Fig. 2.21. Recursive construction of the Benes network

2.4.3 Benes networks

One interesting case of the rearrangeable networks is when N is a power of 2. In this case, one can construct a rearrangeable network by using 2×2 switches. In Figure 2.21, one first constructs a three-stage Clos network with 2×2 switches at the first stage and the last stage. The second stage consists of two $\frac{N}{2} \times \frac{N}{2}$ switches. As N is a power of 2, each of the two $\frac{N}{2} \times \frac{N}{2}$ switches in Figure 2.21 can be further implemented by a three-stage Clos network with 2×2 switches at the first stage and the last stage, and two $\frac{N}{4} \times \frac{N}{4}$ switches at the second stage. One can expand the $\frac{N}{4} \times \frac{N}{4}$ switches recursively and we obtain a multi-stage switch with all 2×2 switches. Such a switch architecture is known as the Benes network. In Figure 2.22, we show an 8×8 Benes network. For an $N \times N$ Benes network, the number of stages is $2 \log_2 N - 1$ and the number of 2×2 switches is $N \log_2 N - \frac{N}{2}$.

The multi-stage Benes network is a very efficient design in terms of hardware complexity. To see this, note that for an $N \times N$ switch to realize all the permutation matrices, the minimum number of 2×2 switches needed for an $N \times N$ switch is $\log_2(N!)$ (as there are $N!$ permutation matrices). From the Stirling formula, i.e.,

$$N! \approx N^{N+1/2} e^{-N} \sqrt{2\pi},$$

it follows that the minimum number of 2×2 switches needed is $O(N \log_2 N)$ and the Benes network achieves the same order of complexity. On the other hand, finding the connection patterns of the 2×2

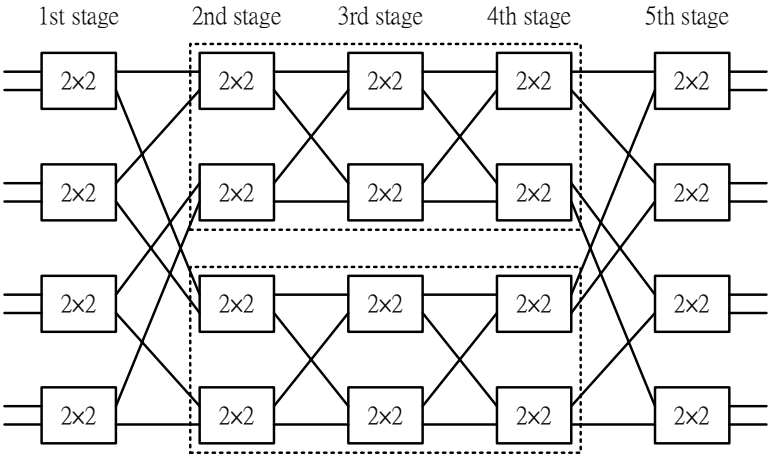


Fig. 2.22. An 8×8 Benes network

switches becomes difficult. As the Benes network is constructed recursively from the three-stage Clos network, the Birkhoff-von Neumann decomposition used for finding the connection patterns also needs to be carried out recursively.

Example 2.4.6. (Connection patterns in a Benes network) Consider the 8×8 Benes network in Figure 2.22. Number the input/output ports from 1 to 8. Number the switches at each stage from 1 to 4. Suppose that we would like to connect the i^{th} input to the $(9-i)^{th}$ output, $i = 1, 2, \ldots, 8$. This is equivalent to realizing the following 8×8 permutation matrix

$$\left[\begin{array}{cc|cc|cc|cc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right] \tag{2.68}$$

in the 8×8 Benes network via finding the connection patterns for all the 2×2 switches. We say a 2×2 switch is set to the "bar" state if

its upper input is connected to its upper output and its lower input is connected to its lower output. On the other hand, a 2×2 switch is set to the "cross" state if its upper input is connected to its lower output and its lower input is connected to its upper output.

By viewing the 2nd stage, the 3rd stage and the 4th stage as two 4×4 rearrangeable networks, we then have a three stage rearrangeable network. Let $Q = (q_{i,j})$ with $q_{i,j}$ being the number of paths from the i^{th} switch at the first stage to the j^{th} switch at the 5^{th} stage, $i, j = 1, 2, 3, 4$. Note that $q_{i,j}$'s can be found by counting the number of 1's in each 2×2 sub-matrix in (2.68). Thus, we have

$$Q = \begin{bmatrix} 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

From the trivial Birkhoff-von Neumann decomposition above, we know that we need to realize the following 4×4 permutation for both the 4×4 rearrangeable networks (formed by the switches at the 2nd stage, the 3rd stage and the 4^{th} stage),

$$\left[\begin{array}{cc|cc} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array}\right]. \tag{2.69}$$

In view of the permutation in (2.69), we can now set the connection patterns for the 2×2 switches at the first stage and the 5^{th} stage. Since we would like to connect input 1 to output 8 and input 2 to output 7, we can simply route input 1 via the upper 4×4 rearrangeable network and input 2 via the lower 4×4 rearrangeable network. As such, we set the first switch at the first stage to the "bar" state and the 4^{th} switch at the 5^{th} stage to the "cross" state. In fact, we can simply set all the 2 switches at the 1st stage to the "bar" state and all the 2×2 switches at the 5^{th} stage to the "cross" state to realize the 8×8 permutation matrix in (2.68) (note that one alternative is to set all the 2 switches

at the 1st stage to the "cross" state and all the 2×2 switches at the 5^{th} stage to the "bar" state).

It remains to find the connection patterns for the 2×2 switches at the 2nd stage, the 3rd stage and the 4^{th} stage. As both the 4×4 rearrangeable networks realize the same 4×4 permutation matrix in (2.69), we only need to consider the upper 4×4 rearrangeable network. Let $\hat{Q} = (\hat{q}_{i,j})$ with $\hat{q}_{i,j}$ being the number of paths from the i^{th} switch at the 2nd stage to the j^{th} switch at the 4^{th} stage, $i, j = 1, 2$. As before, $\hat{q}_{i,j}$'s can be found by counting the number of 1's in each 2×2 sub-matrix in (2.69). Thus, we have

$$\hat{Q} = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Thus, the first and the second 2×2 switches at the 3rd stage need to realize the permutation matrix

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

and they are set to the "cross" state. Similarly, the third and the 4^{th} 2×2 switches at the 3rd stage are set to the "cross" state. As argued for the 8×8 case, we can simply set all the 2 switches at the 2nd stage to the "bar" state and all the 2×2 switches at the 4^{th} stage to the "cross" state to realize the 4×4 permutation matrix in (2.69) (another alternative is to set all the 2 switches at the 2nd stage to the "cross" state and all the 2×2 switches at the 4^{th} stage to the "bar" state). In Figure 2.23, we show all the connection patterns in the 8×8 Benes network for this example.

2.5 Two-stage constructions of switch fabrics

We have shown that the three-stage rearrangeable networks are capable of realizing all the permutations (and sub-permutations). If one only uses two-stage construction, it is intuitive that one can only realize a certain subset of permutations. The objective of this section is to introduce certain subsets of permutations that can be realized by two-stage construction. A very good reference for the theory of two-stage constructions is Li's book [111].

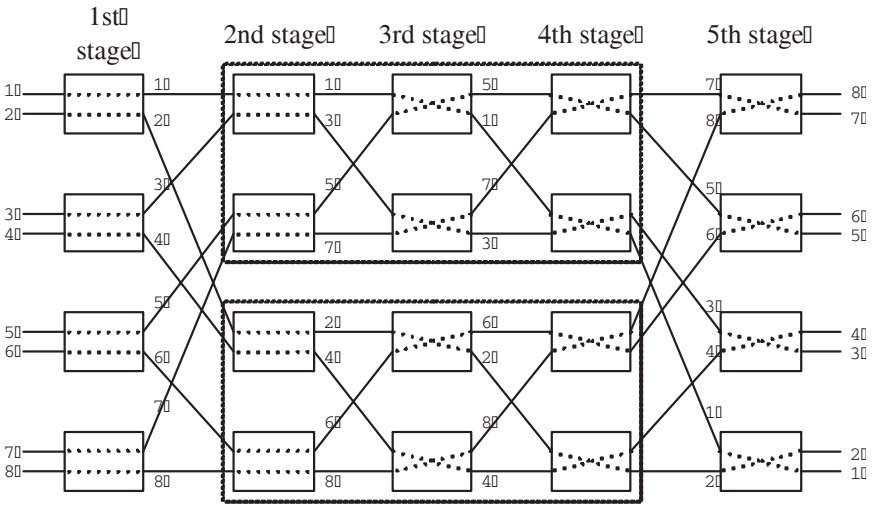


Fig. 2.23. Connection patterns in the 8×8 Benes network for Example 2.4.6

2.5.1 The X2 construction

First, we give a brief overview on the two-stage construction. Suppose that $N = pq$. As shown in Figure 2.24, the first stage consists of p $q \times q$ switches (indexed from $1, 2, \ldots, p$) and the second stage consists of q $p \times p$ switches (indexed from $1, 2, \ldots, q$). These two stages of switches are connected by the perfect shuffle, i.e., the ℓ^{th} output of the k^{th} switch at the first stage is connected to the k^{th} input of the ℓ^{th} switch at the second stage. Also, index the N inputs and outputs from 1 to N. The N inputs of the $N \times N$ switch are connected to the inputs of the switches at the first stage by the perfect shuffle. To be precise, let

$$\ell(i) = \lfloor \frac{i-1}{p} \rfloor + 1, \tag{2.70}$$

and

$$k(i) = i - (\ell(i) - 1) * p. \tag{2.71}$$

Note that for $i = 1, 2, \ldots, N$, $\ell(i)$ is an integer between 1 and q and $k(i)$ is an integer between 1 and p. The i^{th} input of the $N \times N$ switch is connected to the $\ell(i)^{th}$ input of the $k(i)^{th}$ switch at the first stage. Also, we note that the j^{th} output of the $N \times N$ switch is the $k(j)^{th}$ output of the $\ell(j)^{th}$ switch at the second stage. Such a two-stage construction is known as the X2 construction in [111] as it is a concatenation of a shuffle exchange (X) and a two-stage network (2).

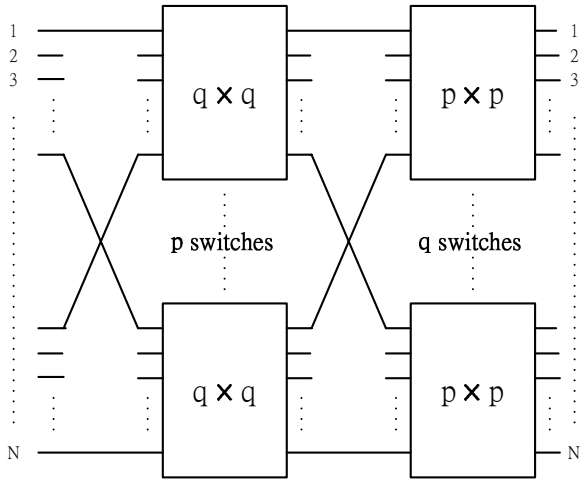


Fig. 2.24. A two-stage (X2) construction of an $N \times N$ switch

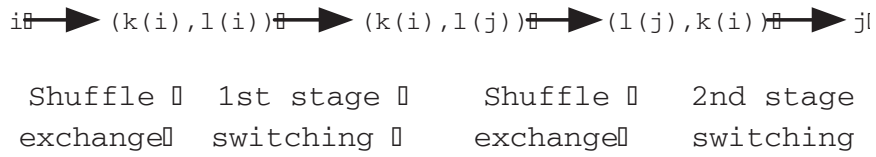


Fig. 2.25. Unique routing path in a two-stage (X2) construction of an $N \times N$ switch

One key property of the two stage construction is the **unique routing path property**. For instance, suppose that we would like to connect the i^{th} input to the j^{th} output (see Figure 2.25). Via the shuffle exchange, the i^{th} input is connected to the $\ell(i)^{th}$ input of the $k(i)^{th}$ switch at the first stage (denoted by $(k(i), \ell(i))$ in Figure 2.25). The $k(i)^{th}$ switch at the first stage then connects its $\ell(i)^{th}$ input to its $\ell(j)^{th}$ output (denoted by $(k(i), \ell(j))$ in Figure 2.25). Via the shuffle exchange between the two stages of switches, the $\ell(j)^{th}$ output of the $k(i)^{th}$ switch at the first stage is connected to the $k(i)^{th}$ input of the $\ell(j)^{th}$ switch at the second stage (denoted by $(\ell(j), k(i))$ in Figure 2.25). The $\ell(j)^{th}$ switch at the second stage then connects its $k(i)^{th}$ input to its $k(j)^{th}$ output, which is exactly the j^{th} output of the switch.

2.5.2 Banyan networks

Like the Benes networks, the banyan networks are also constructed by 2×2 switches.

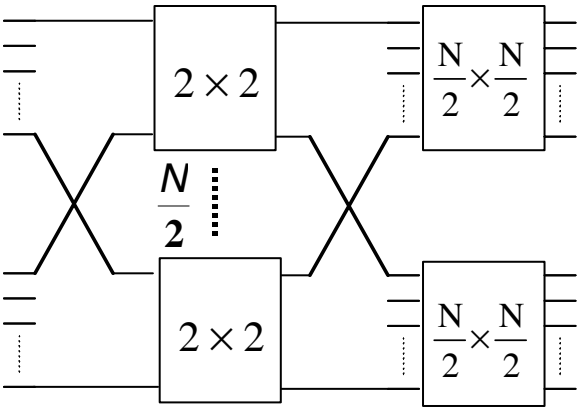


Fig. 2.26. Recursive construction of an $N \times N$ banyan network

Banyan networks are constructed recursively as follows:

(i) A 2×2 switch is a 2×2 banyan network.
(ii) The $N \times N$ banyan network is constructed by the X2 construction with $q = 2$ and $p = N/2$ (see Figure 2.26).

In Figure 2.27, we show an 8×8 banyan network. By recursively expanding the $N \times N$ banyan network, one can show that the number of stages is $\log_2 N$ and the number of 2×2 switches is $\frac{N}{2} \log_2 N$.

From the unique routing path property of the X2 construction, the banyan networks has the self-routing property that finds the connection patterns of the 2×2 switches in a very simple manner. To illustrate the self-routing property of the banyan network, we number both the inputs and the outputs from $0, 1, \ldots, N-1$, with the binary representation. Thus, each output is represented by $\log_2 N$ bits. To send a packet to an output (from any input), we simply follow the binary representation by taking the upper (resp. lower) output at the k^{th} stage if the k^{th} most significant bit is 0 (resp. 1). For instance, suppose that we would like to send a packet from input 0 to output 3 (with the binary representation 011) in Figure 2.27. As the dotted path shown in Figure 2.27, we take the upper output at the first stage as the most significant bit of the output is 0. We then take the lower outputs at both the second stage and the third stage as the last two bits are 1. The packet is then sent to the desired output.

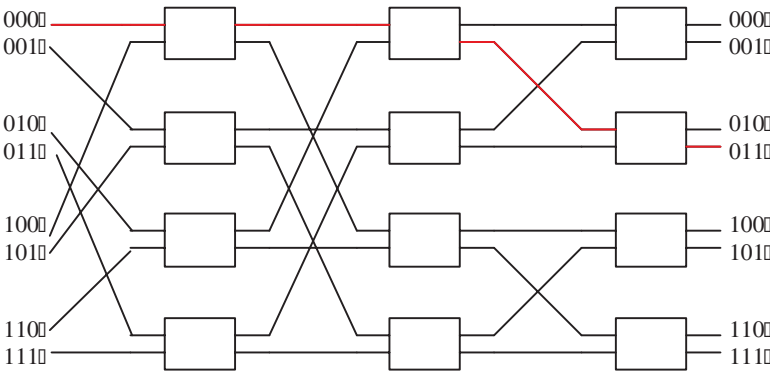


Fig. 2.27. An 8×8 banyan network

The problem of the banyan network is that it does not realize all the sub-permutation matrices. There might be internal conflicts for paths to different outputs. For instance, if input 4 (with binary representation 100) would also like to send a packet to output 0 (with the binary representation 000), then it will conflict with the path for the packet from input 0 to output 3 at the upper output of the first stage.

A sufficient condition for the banyan networks to be free of internal conflicts is known as the *monotone* and *consecutive* condition (see the shuffle exchange network in Problems 21 and 22 for a proof). Call an input *active* if it has a packet to send. The consecutive condition ensures that there is no gap between two active inputs. The monotone condition says that the output addresses of the packets in the consecutive active inputs are non-decreasing. A more general result will be introduced in the next section of CU nonblocking switches.

2.5.3 CU nonblocking switches

As illustrated in the previous section, the banyan networks (and the X2 construction) cannot realize all the permutations. In this section, we show that the X2 construction can be used for constructing switches that realize all the circular unimodal (CU) permutations. Such switches are called CU nonblocking switches.

Definition 2.5.1. *(i) A permutation* $\pi : \{1, 2, \ldots, N\} \mapsto \{1, 2, \ldots, N\}$ *is called a circular shift if there exists an integer* $1 \leq m \leq N$ *such that*

$$\pi(i) = \begin{cases} i + m & \text{if } i + m \leq N \\ i + m - N & \text{otherwise} \end{cases} .$$

(ii) A permutation $\pi : \{1, 2, \ldots, N\} \mapsto \{1, 2, \ldots, N\}$ *is called circular unimodal (CU) if there exists a circular shift permutation* σ *such that* $\pi(\sigma(i))$ *is unimodal in* $\sigma(i)$*, i.e., there exists an* i^* *such that* $\pi(\sigma(i))$ *is increasing in* $\sigma(i)$ *for* $\sigma(i) \leq \sigma(i^*)$ *and decreasing in* $\sigma(i)$ *for* $\sigma(i) \geq \sigma(i^*)$*.*

(iii) An $N \times N$ *switch is called CU nonblocking if it can realize all the* $N \times N$ *CU permutations.*

Example 2.5.2. (Circular unimodal permutation) Consider the following permutation $\pi : \{1, 2, \ldots, 20\} \mapsto \{1, 2, \ldots, 20\}$

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 8 & 6 & 4 & 2 & 1 & 3 & 5 & 7 & 9 & 13 & 16 & 19 & 20 & 18 & 17 & 15 & 14 & 12 & 11 & 10 \end{pmatrix}.$$

Note that $\pi(5) = 1, \pi(6) = 3, \ldots, \pi(13) = 20$. Thus, π is increasing from 5 to 13. On the other hand, $\pi(13) = 20, \pi(14) = 18, \ldots, \pi(20) = 10, \pi(1) = 8, \ldots, \pi(4) = 2, \pi(5) = 1$, and π is (circular) decreasing from 13 to 5. Thus, we can choose the circular shift σ with

$$\sigma(i) = \begin{cases} i+4 & \text{if } i+4 \le 20 \\ i+4-20 & \text{otherwise} \end{cases} .$$

By so doing, we have $\sigma(1) = 5$ and $\sigma(9) = 13$. This implies that $\pi(\sigma(i))$ is increasing from $\sigma(1)$ to $\sigma(9)$ and it is decreasing from $\sigma(9)$ to $\sigma(20)$.

There are several facts for the circular unimodal permutations.

(i) Let π^{-1} be the inverse mapping of a circular unimodal permutation π. Then π is (circular) increasing from $\pi^{-1}(1)$ to $\pi^{-1}(N)$ and (circular) decreasing from $\pi^{-1}(N)$ to $\pi^{-1}(1)$. In other words, in every circular unimodal permutation there is a unique minimum and a unique maximum that divides the permutation into two (circular) monotone segments. This is also the easiest way to verify whether a permutation is a circular unimodal permutation. For instance, in Example 2.5.2, one has $\pi^{-1}(1) = 5$ and $\pi^{-1}(20) = 13$.

(ii) Once $\pi(i)$'s are known for all i in the (circular) increasing segment, then the rest of $\pi(i)$ can be uniquely determined. For instance, suppose that all the values of π in Example 2.5.2 for $i = 5$ to 13 are specified as in Example 2.5.2. From the fact that the rest of $\pi(i)$ forms a decreasing sequence, we know that $\pi(14)$ must be 18 as 18 is the largest number that have not been specified. Following the same argument yields $\pi(15) = 17$.

One of the most important theorems of the two-stage construction is that it preserves the property of CU nonblocking as stated in the following theorem.

Theorem 2.5.3. *Consider the two-stage construction in Figure 2.24, i.e., the X2 construction. If all the $q \times q$ switches at the first stage and the $p \times p$ switches at the second stage are CU nonblocking, then the constructed $N \times N$ switch is also CU nonblocking.*

The outline for the proof of Theorem 2.5.3 will be given in Problem 26.

Example 2.5.4. (Two-stage construction for a circular unimodal permutation) In Figure 2.28, we show how one constructs a 20×20 CU nonblocking switch by using the $X2$ construction. For this example, we have $N = 20$ and $p = 4$ and $q = 5$. As such, there are four 5×5 switches at the first stage and five 4×4 switches at the

second stage. According to Theorem 2.5.3, the 20×20 switch is CU nonblocking if all the four 5×5 switches at the first stage and the five 4×4 switches at the second stage are CU nonblocking.

To gain more intuition on Theorem 2.5.3, let us consider the circular unimodal permutation in Example 2.5.2. From the unique routing path property of the X2 construction, we illustrate in Figure 2.28 all the paths from the inputs to the outputs. For instance, as $\pi(1) = 8$, the first input is connected to the first input of the first 5×5 switch (at the first stage). The first 5×5 switch connects its first input to the second output. Its second output is then connected to the first input of the second 4×4 switch (at the second stage). The second 4×4 switch then connects its first input to the fourth output.

Let π_{ij} be the connection pattern (permutation) realized at the j^{th} switch of the i^{th} stage. From Figure 2.28, it is easy to see that the connection pattern of the first 5×5 switch at the first stage is

$$\pi_{11} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 3 & 5 & 4 \end{pmatrix},$$

which is a 5×5 circular unimodal permutation. Similarly, one can verify that all the connection patterns for the rest three switches at the first stage are

$$\pi_{12} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 4 & 5 & 3 \end{pmatrix},$$

$$\pi_{13} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 4 & 5 & 3 \end{pmatrix},$$

and

$$\pi_{14} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 5 & 4 & 3 \end{pmatrix}.$$

Also, the connection patterns for the five 4×4 switches at the second stage are

$$\pi_{21} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 4 & 2 \end{pmatrix},$$

$$\pi_{22} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 1 & 3 \end{pmatrix},$$

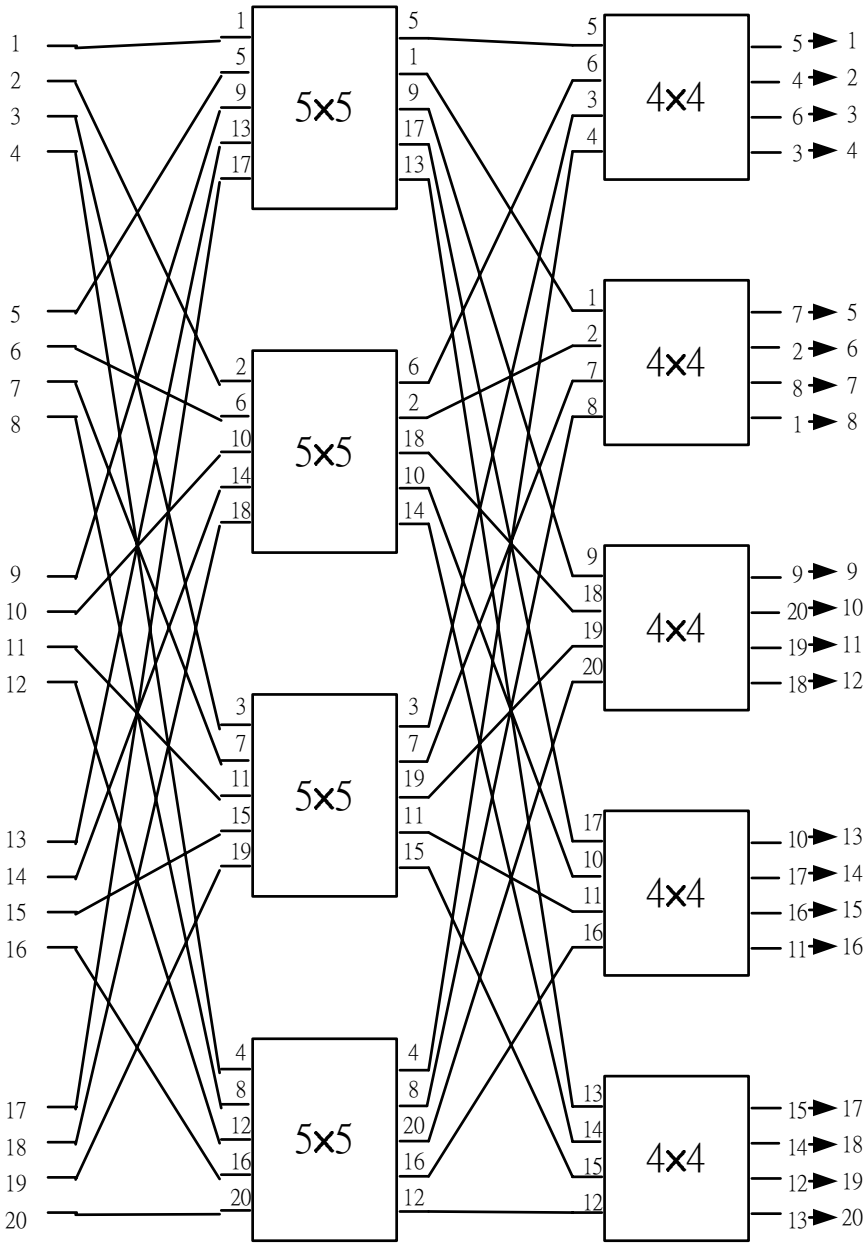


Fig. 2.28. An example for a CU nonblocking switch

$$\pi_{23} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 3 & 2 \end{pmatrix},$$

$$\pi_{24} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix},$$

and

$$\pi_{25} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 1 & 3 \end{pmatrix},$$

It is left to the readers to verify that all the connection patterns in the 5×5 switches at the first stage are 5×5 circular unimodal permutations and all the connection patterns in the 4×4 switches at the second stage are 4×4 circular unimodal permutations.

A CU nonblocking switch can realize all the circular unimodal permutations. As such, it can also realize the set of sub-permutations that satisfy the monotone consecutive condition (as stated in the banyan network), i.e.,

(i) the active inputs are consecutive, and
(ii) the mapping between active inputs and outputs is monotonically increasing.

To see this, note that once a sub-permutation that satisfies the monotone consecutive condition is specified, there is a unique way to extend this sub-permutation to a circular unimodal permutation (as commented in (ii) for circular unimodal permutations).

A switch that realizes all the sub-permutations that satisfy the monotone consecutive condition is known as a linear decompressor in [111]. If the monotone condition in (ii) is replaced from "increasing" to "decreasing," a switch that realizes all the sub-permutations that satisfy the modified monotone consecutive condition is known as an *upturned* linear decompressor in [111]. Clearly, a CU nonblocking switch is both a linear decompressor and an upturned linear decompressor.

As mentioned in Section 2.5.2, a banyan network can realize all the sub-permutations that satisfy the monotone consecutive condition. Thus, a banyan network is a linear decompressor. To see this, note that an $N \times N$ banyan network is constructed by recursively expanding the $X2$ construction with $q = 2$ and $p = N/2$. By so doing, it can be built using only 2×2 switches. Thus, the banyan network is in fact a CU nonblocking switch. However, there are several ways for carrying out

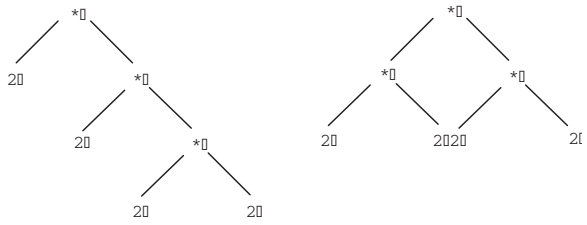


Fig. 2.29. Binary trees for the recursive expansions of 16

the recursive expansion. For instance, if $N = 16$, then one can first use the $X2$ construction with $q = 2$ and $p = 8$. One then further expands the two 8×8 switches with $q = 2$ and $p = 4$. Finally, one expands the four 4×4 switches with $q = 2$ and $p = 2$. Such an expansion can be denoted by the following notation that indicates the order of the expansion

$$16 = 2 * 8 = 2 * (2 * 4) = 2 * (2 * (2 * 2)).$$

Another way of doing the expansion is

$$16 = 4 * 4 = (2 * 2) * (2 * 2).$$

Viewing each multiplication as a binary operator, one can represent both the expressions as binary trees (see Figure 2.29). It is clear that the tree obtained by the second expansion (the right one in Figure 2.29) is a more balanced one. It is shown in Theorem 4.1.18 in [111] that all these expansions are topologically equivalent (as they are all CU nonblocking switches) and known as the banyan-type networks (that realize all the circular unimodal permutations). The difference is that a more balanced expansion yields a better VLSI design in terms of layout complexity.

2.5.4 Bitonic sorters and Batcher sorting networks

A bitonic list with length N is a list that is monotonically increasing to some k and then monotonically decreasing to N. To be precise, a bitonic list $a = (a_1, a_2, \ldots, a_N)$ is a list that satisfies $a_1 \leq a_2 \leq \ldots \leq a_k$ and $a_k \geq a_{k+1} \geq \ldots a_N$ for some $1 \leq k \leq N$. A circular bitonic list is a list that can be turned into a bitonic list by circular shifting. For instance, $(a_3, a_4, \ldots, a_N, a_1, a_2)$ is a circular bitonic list when $(a_1, a_2, \ldots, a_N)$ is a bitonic list. A bitonic sorter is a device that can

sort any circular bitonic list into a monotonically increasing or decreasing sequence. Specifically, a bitonic down (resp. up) sorter can sort any circular bitonic list into an increasing (resp. decreasing) sequence. One interesting observation is that **every circular bitonic list is associated with a circular unimodal permutation**. For instance, consider a list $a = (a_1, a_2, \ldots, a_{N-1}, a_N)$. Let $R(a_i)$ be the rank of a_i among these N elements, i.e., $R(a_i) = N$ if a_i is the largest element, $R(a_i) = N - 1$ if a_i is the second largest element, ..., and $R(a_i) = 1$ if a_i is the smallest element. If the list a is a circular bitonic list, then the permutation

$$\pi = \begin{pmatrix} 1 & 2 & \ldots & N-1 & N \\ R(a_1) & R(a_2) & \ldots & R(a_{N-1}) & R(a_N) \end{pmatrix}.$$

is a circular unimodal permutation (note that one needs to break ties in the way so that the relative order can be preserved). This suggests that one can use CU nonblocking switches for bitonic sorters.

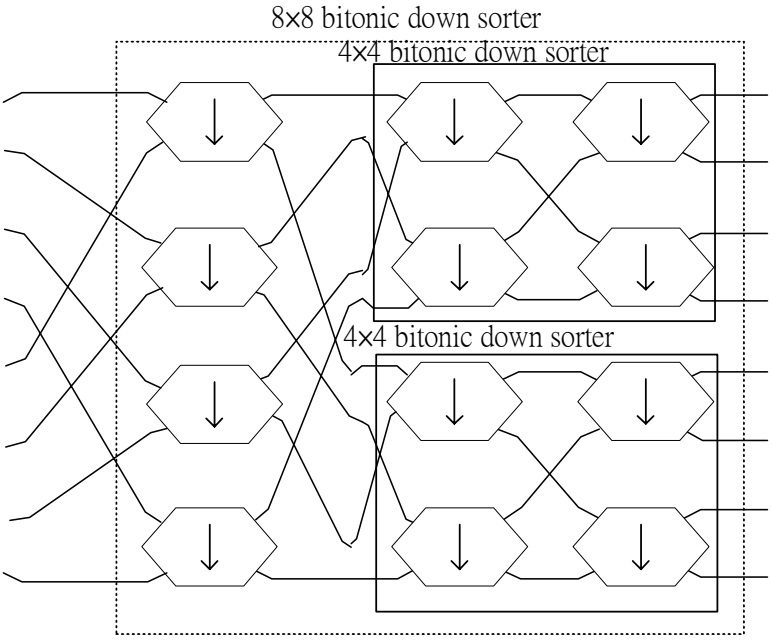


Fig. 2.30. An 8×8 bitonic down sorter

As described in the previous section, one way to construct a large CU nonblocking switch is to use the X2 construction and expand such

a construction recursively. Following the X2 construction, one can construct an $N \times N$ bitonic sorter recursively as follows:

(i) A down (resp. up) sorter sorts a list into a monotonically increasing (resp. decreasing) list. A 2×2 down (resp. up) sorter is a 2×2 bitonic down (resp. up) sorter.

(ii) The $N \times N$ bitonic down (resp. up) sorter is constructed by cascading $N/2$ 2×2 down (resp. up) sorters at the first stage and two $\frac{N}{2} \times \frac{N}{2}$ bitonic down sorters at the second stage. The N inputs are connected to $N/2$ 2×2 down sorters by a shuffle exchange. The upper (resp. lower) output of the k^{th} 2×2 down sorter at the first stage is connected to the k^{th} input of the upper (resp. lower) $\frac{N}{2} \times \frac{N}{2}$ bitonic down sorter, $k = 1, \ldots, N/2$.

In Figure 2.30, we show how one constructs an 8×8 bitonic down sorter via the X2 construction.

Now view each 2×2 down sorter as a 2×2 switch. Then the above construction corresponds to an $N \times N$ CU nonblocking switch. As described above, every circular bitonic list $a = (a_1, a_2, \ldots, a_N)$ can be mapped to a circular unimodal permutation by using the rank function $R(\cdot)$. By so doing, input i is connected to output $R(a_i)$ for all $i = 1, 2, \ldots, N$. As the rank function yields a circular unimodal permutation, such a connection pattern can be realized by the corresponding CU nonblocking switch. From the unique routing path property (cf. the routing in a banyan network), it then follows that the outputs of the bitonic down sorter are the results of the rank function $R(a_i)$, $i = 1, 2, \ldots, N$. As such, the bitonic down sorter can sort any circular bitonic list into a monotonically increasing sequence. This is stated as the Batcher sorting theorem in the following.

Theorem 2.5.5. *(***Batcher sorting theorem [13]***)* *An $N \times N$ bitonic down sorter can sort a circular bitonic list into a monotonically increasing list.*

Instead of using the results from the CU nonblocking switches, a direct proof for the Batcher sorting theorem is given in Problems 24 and 25 by using the unique cross over property of a circular bitonic list.

Now we show how one uses the Batcher sorting theorem to construct a Batcher sorting network that can sort any list into a monotonically increasing list. A Batcher sorting network can be constructed recursively as follows:

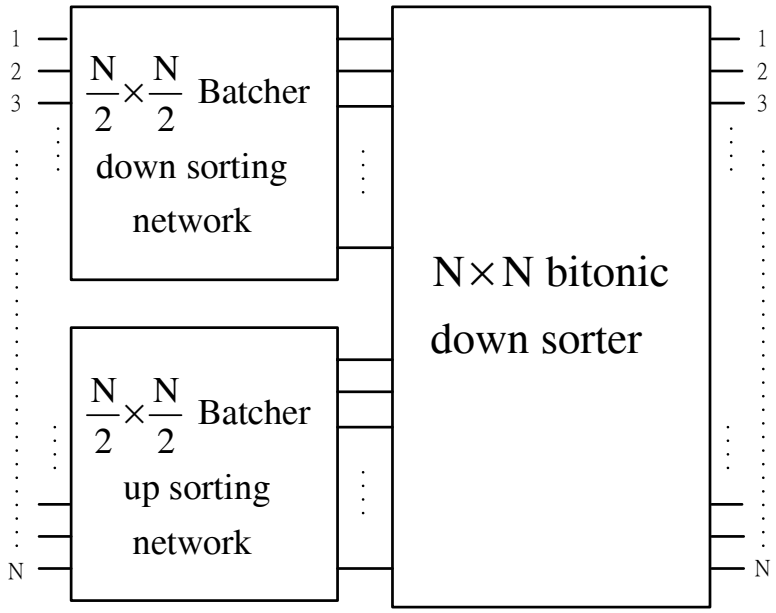


Fig. 2.31. Recursive construction of $N \times N$ Batcher down sorting network

(i) A 2×2 down (resp. up) sorter is a 2×2 Batcher down (resp. up) sorting network.
(ii) The $N \times N$ Batcher down sorting network consists of two stages (see Figure 2.31). The first stage consists of two $\frac{N}{2} \times \frac{N}{2}$ Batcher sorting networks. The upper one is a down sorting network, while the lower one is an up sorting network. The second stage is an $N \times N$ bitonic down sorter. The $N/2$ outputs of the down (resp. up) sorting network are connected to the upper (resp. lower) $N/2$ inputs of the $N \times N$ bitonic down sorter.

Use the Batcher sorting theorem, we known that the outputs from the two $\frac{N}{2} \times \frac{N}{2}$ Batcher sorting networks form a bitonic list. As a bitonic list is special case of circular bitonic list, it can be sorted by the $N \times N$ bitonic sorter. This shows that the Batcher sorting network can sort any list into a monotonically increasing list.

Example 2.5.6. (Batcher sorting network) In Figure 2.32, we show how one constructs an 8×8 Batcher down sorting network by recursively expanding the construction above. As shown in Figure 2.32, there are six active inputs. The value at each input is the destined

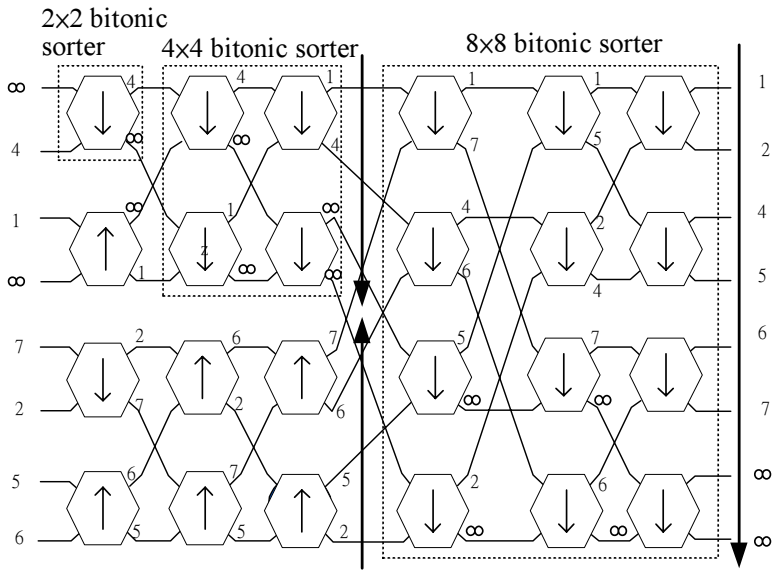


Fig. 2.32. An 8×8 Batcher down sorting network

output port number of the packet at that input. The values of the idle inputs are assigned with ∞. As shown in Figure 2.32, the 2×2 switches with an up arrow are the 2×2 up sorters and those with a down arrow are the 2×2 down sorters. For instance, the first four outputs of the first stage are two monotone sequences: the first two are increasing, and the last two are decreasing. These two monotone sequences, each with length 2, are merged into the upper 4×4 bitonic sorter and thus generate an increasing sequence of length 4. Similarly, the lower 4×4 bitonic sorter generates a decreasing sequence of length 4. These two monotone sequence are then merged into the 8×8 bitonic sorter to generate an increasing sequence of length 8. Note that the outputs from the Batcher sorting network satisfy the monotone and consecutive condition needed for the banyan network.

2.5.5 Batcher-banyan networks and three-phase switches

We have shown that the banyan network can realize all the sub-permutations that satisfy the monotone and consecutive condition. To generate monotone and consecutive inputs, one can add a Batcher sorting network [13] in front of the banyan network. Such a network

is called the Batcher-banyan network and it can realize all the sub-permutation matrices. Unlike the Benes network, finding routing paths in the Batcher-banyan network is easy. In the first stage of the Batcher sorting network, the output addresses are being compared at each 2×2 sorter. In the second stage of the banyan network, the output addresses are used for self routing at each 2×2 switch. However, the hardware complexity of the Batcher-banyan network is higher than that of the Benes network. To see this, note that an $N \times N$ bitonic sorter consists of $\log_2 N$ stages. Thus, the number of stages in an $N \times N$ Batcher sorting network is

$$\sum_{k=1}^{\log_2 N} k = \frac{1}{2}\log_2 N(\log_2 N + 1),$$

and the number of 2×2 switches needed is

$$\frac{N}{4}\log_2 N(\log_2 N + 1).$$

Adding the number of 2×2 switches in the $N\times N$ banyan network, one can show that the number of 2×2 switches needed for the Batcher-banyan network is

$$\frac{N}{4}\log_2 N(3 + \log_2 N),$$

which is $O(N(\log_2 N)^2)$.

We have shown that a switch fabric implemented by a Batcher-banyan network is capable of realizing all the sub-permutations. Now we introduce the so-called three phase switch [76] that also uses the Batcher-banyan network for output conflict resolution (finding a matching). Consider an $N \times N$ switch with its switch fabric implemented by a Batcher-banyan network. Suppose that for all $i = 1, , 2, \ldots, N$, there are two feedback paths connected to the i^{th} input of the switch: the first one is from the i^{th} output of the Batcher sorting network, and the second one is from the i^{th} output of the switch.

The three phases are as follows:

(i) *Request and Purge.* Each input sends a request that contains the information of its input address and the desired output address. The Batcher sorting network then sorts these requests according to the output addresses of these requests. After the Batcher sorting network, requests with the same output addresses appear at adjacent outputs. Thus, requests that conflict with each other can be

easily purged by comparing the output addresses of the requests appeared at adjacent outputs of the Batcher sorting network.

(ii) *Acknowledgement.* Those un-purged requests (at the outputs of the Batcher sorting network) are forwarded to the inputs of the switch via the first feedback path. Suppose that the i^{th} input of the switch receives an un-purged request that is sent from the j^{th} input of the switch (via the first feedback path). The i^{th} input then sends an acknowledgement through the Batcher-banyan network to the j^{th} output. The j^{th} output of the switch then relays the acknowledgement via the second feedback path to the j^{th} input of the switch.

(iii) *Packet transmission.* Once an input receives an acknowledgement, it then sends a packet to the desired output.

Example 2.5.7. In Figure 2.33, we depict how the three-phase switch works. In this example, Input 1 (I1) would like to send a packet to Output 2 (O2), Input 2 (I2) would like to send a packet to Output 3 (O3), Input 3 (I3) would like to send a packet to Output 2 (O2), and Input 4 (I4) would like to send a packet to Output 1 (O1). In the first phase, these four inputs send requests that contain the information of the input addresses and the output addresses. At the second output and the third output of the Batcher sorting network, one finds that these two adjacent requests have the same output address and one of them (the request from I3) is purged. Via the feedback paths from the outputs of the Batcher sorting network to the inputs of the switch, those un-purged requests are relayed back to the inputs. In the second phase, those un-purged requests are sent back to their inputs as acknowledgements. The input address at a request is now used for the output address of the acknowledgment. In this example, Input 1 sends an acknowledgement to Output 4, Input 2 sends an acknowledgement to Output 1, and Input 4 sends an acknowledgement to Output 2. Via the feedback paths from the outputs of the switch to the inputs of the switch, acknowledgements are relayed to the inputs that sent requests. In this example, Inputs 1,2 and 4 received acknowledgments. Up to this point, output conflicts have been resolved (and a matching has been found). Thus, in the third phase, Inputs 1,2 and 4 send packets to their outputs.

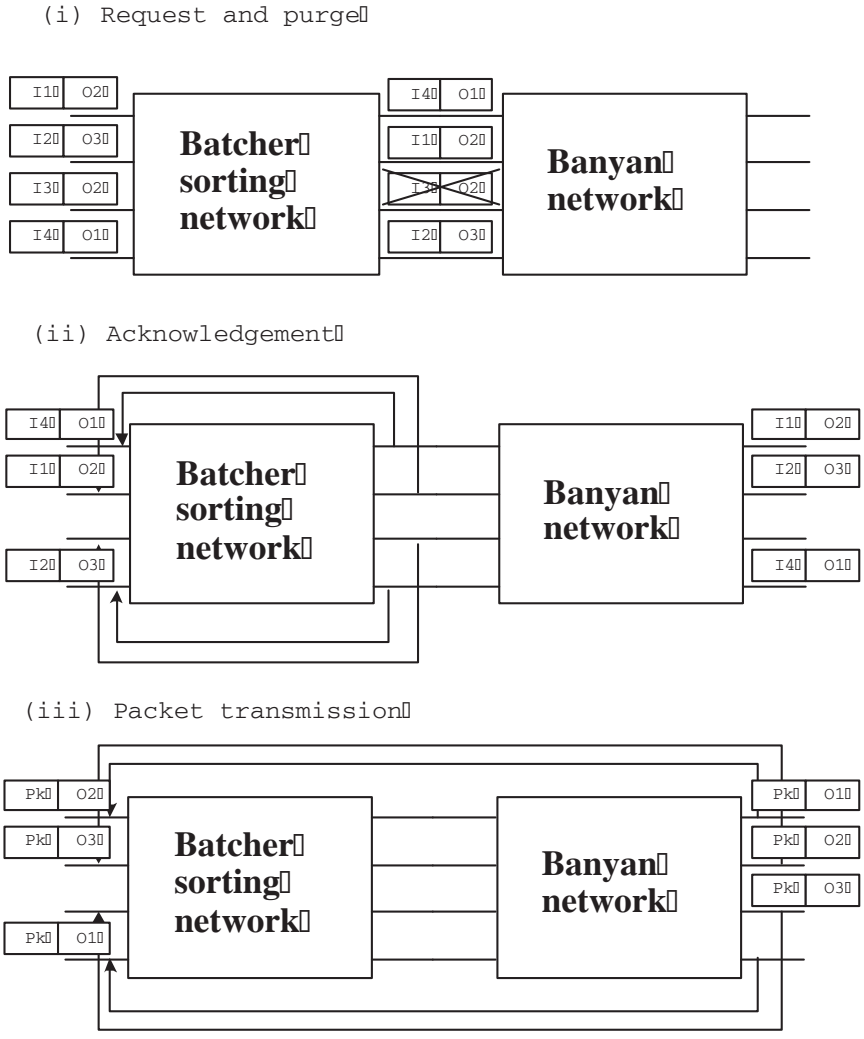


Fig. 2.33. An illustration for the three-phase switch

Like PIM or SLIP, phases 1 and 2 can be carried out iteratively to reach a maximal matching between inputs and outputs. Also, as each unacknowledged input sends only one request to the switch fabric, there is no need for the acceptance step in PIM and SLIP.

2.5.6 Concentrators

An $N \times N$ down sorter is an $N \times N$ switch that can sort any input list into a monotonically increasing list. For instance, an $N \times N$ Batcher sorting network described in Section 2.5.4 is an $N \times N$ sorter. An M-to-N concentrator is an $M \times N$ switch that selects the N largest elements from the M inputs. Certainly, one can implement an M-to-N concentrator by an $M \times M$ sorter. Here we show a way to implement an M-to-N concentrator without the full complexity of a sorter.

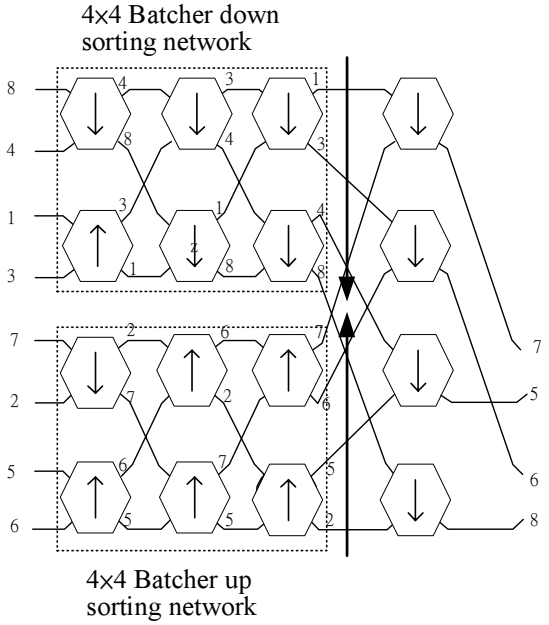


Fig. 2.34. An 8-to-4 concentrator

The key observation is that a bitonic down sorter can sort a circular bitonic list into a monotonically increasing list. As stated in the previous section, an $N \times N$ bitonic sorter can be recursively constructed using the X2 construction with $N/2$ 2×2 down sorters at the first

stage and two $\frac{N}{2} \times \frac{N}{2}$ bitonic down sorters at the second stage. Note that there is no link between these two $\frac{N}{2} \times \frac{N}{2}$ bitonic down sorters. Since the output of the $N \times N$ bitonic sorter is a monotonically increasing list, **the inputs to the lower $\frac{N}{2} \times \frac{N}{2}$ bitonic down sorter must be the larger half of the N inputs**. A direct proof of such a statement is given in Problem 24.

Suppose that M is an even number and $M/2 \geq N$. Analogous to the way that we construct a Batcher sorting network, we can construct an M-to-N concentrator as follows:

(i) The M-to-N concentrator consists of two stages. The first stage consists of two $\frac{M}{2} \times \frac{M}{2}$ Batcher sorting networks. The upper one is a down sorting network, while the lower one is an up sorting network.

(ii) Select the N largest outputs from each of these two $\frac{M}{2} \times \frac{M}{2}$ Batcher sorting networks to form a bitonic list with length $2N$. Now connect these $2N$ elements to N 2×2 down sorters via a shuffle exchange. As these N 2×2 down sorters are in fact the first stage of a $2N \times 2N$ bitonic sorter, the larger outputs of these N 2×2 down sorters are then the N largest elements among the M inputs of the concentrator.

In Figure 2.34, we show an 8-to-4 concentrator via the construction described above. One can see from Figure 2.34 that the 8 outputs from the two Batcher sorting networks form a bitonic list with length 8. The four largest elements, i.e., $7, 5, 6, 8$, appear at the four outputs of the concentrator. Note that the relative order of these four largest elements are irrelevant to the objective of the concentrator.

One of the most important applications of a concentrator, as its name suggests, is for traffic concentration. For an active input, we may mark a bit 1 in a packet header. On the other hand, the bit is set to 0 if an input is idle. By sorting this bit, an M-to-N concentrator selects N active inputs among M inputs. Note that if there are more than N active inputs, an M-to-N concentrator in general does not specify which N active inputs should be selected. In Definition 5.3.3, we will define a prioritized concentrator that specifies the priority order for the selection of active inputs.

For the purpose of traffic concentration, an M-to-N concentrator constructed in this section can be further simplified (if $N << M/2$). In Section 2.7.3, we will introduce a fast knockout algorithm that greatly simplifies the design of traffic concentration.

2.5.7 Mirror image of two-stage constructions

A mirror image of a switch is a switch with the inputs (resp. outputs) that are the outputs (resp. inputs) of the original switch. In this section, we consider the mirror image of the X2 construction. As shown in Figure 2.35, the mirror image of the X2 construction is now a 2X construction, i.e., two stages of switches (2) followed by a shuffle exchange (X). The X2 construction is used for realizing all the circular unimodal permutations. Clearly, the 2X construction, having the inputs and outputs interchanged, can now be used for realizing all the *inverse* circular unimodal permutations. To be precise, a permutation is called an *inverse* circular unimodal permutation if its inverse permutation is a circular unimodal permutation. In other words, for an inverse circular unimodal permutation, if we circularly shift the outputs (for some k), then the inputs becomes a unimodal list.

Example 2.5.8. (Inverse circular unimodal permutation) Consider the following permutation $\pi : \{1, 2, \ldots, 20\} \mapsto \{1, 2, \ldots, 20\}$

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 4 & 6 & 3 & 7 & 2 & 8 & 1 & 9 & 20 & 19 & 18 & 10 & 17 & 16 & 11 & 15 & 14 & 12 & 13 \end{pmatrix}.$$

Rewriting the permutation in the order of the outputs yields

$$\begin{pmatrix} 8 & 6 & 4 & 2 & 1 & 3 & 5 & 7 & 9 & 13 & 16 & 19 & 20 & 18 & 17 & 15 & 14 & 12 & 11 & 10 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \end{pmatrix}.$$

This is exactly the inverse permutation of the circular unimodal permutation in Example 2.5.2. Thus, the permutation π in this example is indeed an inverse circular unimodal permutation.

As discussed in Section 2.5.3, there are several facts for the inverse circular unimodal permutations.

(i) Let π^{-1} be the inverse mapping of an inverse circular unimodal permutation π. Then π^{-1} is (circular) increasing from $\pi(1)$ to $\pi(N)$ and (circular) decreasing from $\pi(N)$ to $\pi(1)$.
(ii) Once $\pi^{-1}(i)$'s are known for all i in the (circular) increasing segment, then the rest of $\pi^{-1}(i)$ can be uniquely determined.

An $N \times N$ switch is called UC nonblocking if it can realize all the $N \times N$ inverse circular unimodal permutations. Clearly, if a switch is CU nonblocking, then its mirror image is UC nonblocking. As the 2X construction is the mirror image of the X2 construction, one has the following theorem from Theorem 2.5.3.

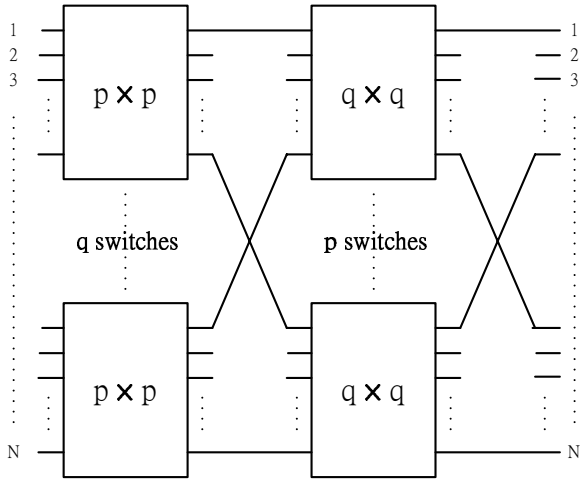


Fig. 2.35. The 2X construction

Theorem 2.5.9. *Consider the two-stage construction in Figure 2.35, i.e., the 2X construction. If all the $p \times p$ switches at the first stage and the $q \times q$ switches at the second stage are UC nonblocking, then the constructed $N \times N$ switch is also UC nonblocking.*

Both the terms CU nonblocking and UC nonblocking were introduced in [111]. The letter C stands for "circular" and the letter U stands for "unimodal." Also, the first (resp. second) letter corresponds to the operation or property of the inputs (resp. outputs). Thus, CU can be interpreted as follows: if we perform circular shift for the inputs, the outputs become unimodal. Similarly, UC means the other way around. That is, if we perform circular shift for the outputs, the inputs become unimodal.

A UC nonblocking switch can realize all the inverse circular unimodal permutations. As such, it can also realize the set of sub-permutations that satisfy the inverse monotone consecutive condition, i.e.,

(i) the active **outputs** are consecutive, and
(ii) the mapping between active inputs and outputs is monotonically increasing.

A switch that realizes all the sub-permutations that satisfy the inverse monotone consecutive condition is known as a linear compressor in [111]. If the monotone condition in (ii) is replaced from "increasing"

to "decreasing", a switch that realizes all the sub-permutations that satisfy the modified inverse monotone consecutive condition is known as an *upturned* linear compressor in [111]. As commented for linear decompressors, there is one-to-one mapping between a sub-permutation that satisfies the inverse monotone consecutive condition and an inverse circular unimodal permutation. Thus, a UC nonblocking switch is both a linear compressor and an upturned linear compressor.

2.6 Exact emulation

In this section, we will address switch architectures that yield exactly the same departure processes as those from the corresponding output-buffered switches. Such switch architectures are called *exact emulation* of output-buffered switches. In other words, if one considers a switch as a black box, then one cannot tell whether the switch is based on the output-buffered architecture by only looking at the inputs and the outputs.

2.6.1 Crosspoint buffers

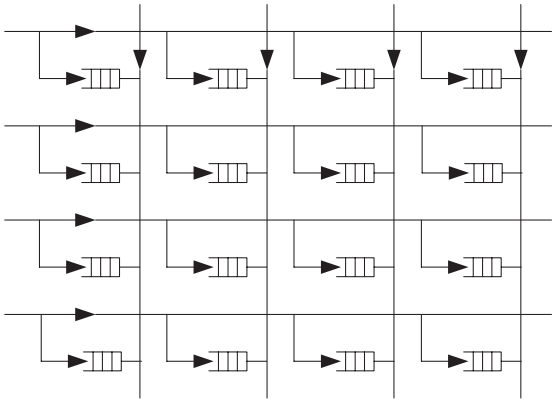


Fig. 2.36. A crossbar switch with crosspoint buffers for exact emulation of an $N \times N$ output-buffered switch

In Figure 2.36, we consider a crossbar switch. There are separate buffers at the crosspoints. For an $N \times N$ switch, the number of separate

buffers is N^2. As the VOQs of an input-buffered switch, each buffer at a crosspoint stores packets from one input to one output. Call the $(i,j)^{th}$ buffer for the buffer at the crosspoint of the i^{th} input and the j^{th} output. When a packet destined for the j^{th} output arrives at the i^{th} input, it is placed in the $(i,j)^{th}$ buffer. If the departure time of that packet from the corresponding output-buffered switch is known, one can simply transmit the packet to the j^{th} output at its departure time from the $(i,j)^{th}$ buffer. Clearly, one can achieve exact emulation of an output-buffered switch by this architecture.

As pointed out in the survey paper by Ahmadi and Denzel [3], the architecture based on crosspoint buffers does not scale for a large number of input/output ports because of the square growth. It is only good either for small switches, or as basic architecture for the building blocks of modular multi-stage switches. In the next section, we will show a parallel buffer architecture that reduces the number of buffers from N^2 to $2N-1$.

2.6.2 Parallel buffers

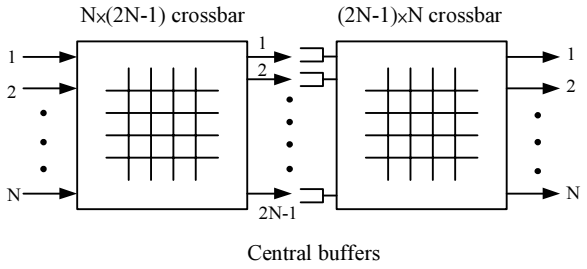


Fig. 2.37. A switch architecture with parallel buffers for exact emulation of an $N \times N$ output-buffered switch

In Figure 2.37, we show the parallel-buffered switch architecture. Such a switch architecture was previously addressed in the paper by Wai and Kumar [165] and the paper by Iyer, Awadallah and McKeown [86]. The first stage is an $N \times (2N-1)$ crossbar switch fabric and the second stage is an $(2N-1) \times N$ crossbar switch fabric. There are $2N-1$ central buffers (parallel buffers) between these two switch fabrics.

Suppose that the departure time of a packet from the corresponding output-buffered switch is known when the packet arrives at the input. Consider a packet p that arrives at time t_a and departs at time t_d from

the corresponding output-buffered switch. At time t_a, we can switch packet p (via the first crossbar) to one of the central buffers that does not have a packet to send at time t_d. When time reaches t_d, we can then switch packet p (via the second crossbar) to its output and we have an exact match of the departure time. The magic number $2N-1$ makes sure that we can always find one of the central buffers that does not have a packet to send at time t_d. This is because there are at most $N-1$ packets that will depart at time t_d from the other $N-1$ output ports (output conflicts) and there are at most $N-1$ packets that arrive at time t_a from the other $N-1$ input ports (input conflicts). In the worst case, all these $2N-2$ packets use distinguished central buffers. As there are $2N-1$ center buffers, there is always a free one (without any conflicts)! The argument above is completely parallel to that used in the three-stage nonblocking Clos networks (see Section 2.4.1). The ratio of the number of buffers to the number of input/output ports is $2-1/N$. Such a factor may be regarded as the speedup factor needed for exact emulation.

2.6.3 Combined input output queueing

Another alternative is called the Combined Input Output Queueing (CIOQ) switch in Figure 2.38. In the CIOQ architecture, there is only one $N \times N$ crossbar switch fabric. Also, there are buffers at both the inputs and outputs. As in the previous approach, one needs internal speedup. A speedup factor of S indicates that the crossbar switch fabric is capable of transmitting S packets per slot time. It is shown in [50, 153] that a speedup factor of $2-1/N$ is necessary and sufficient for exact emulation of FCFS output-buffered switches. For the ease of presentation, we follow the development in [50] by using a speedup factor of 2.

As in the previous approach, assume that the departure time of a packet from the corresponding output-buffered switch is known when the packet arrives at the input. **The basic idea of the CIOQ switch is then to move packets from input buffers to output buffers before their desired departure times**. To achieve this, each input port maintains a priority list of the packets in its buffer at each time slot. The relative order between two packets stays the same with respect to time. An arriving packet is inserted into the input priority list according to a *critical packet first* policy that will be specified later. Each output port also maintains a priority list of the packets

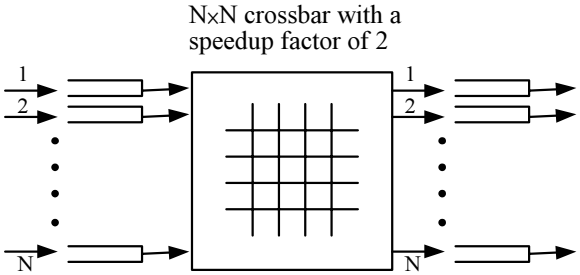


Fig. 2.38. Combined Input Output Queueing switch architecture

in all the input buffers and destined to it. The priority list at an output is ordered by the packet departure times from the corresponding output-buffered switch. The connection pattern for the crossbar is set by finding a *stable* matching between the input ports and the output ports (note that there are two connection patterns per time slot as the speedup factor is 2).

Definition 2.6.1. (Stable matching) *A matching of input ports to the output ports is said to be stable if for each packet waiting in an input buffer, one of the following holds:*

(i) The packet is part of the matching (the packet is transmitted from its input to its output).
(ii) Another packet that is ahead of the packet in its input priority list is part of the matching.
(iii) Another packet that is ahead of the packet in its output priority list is part of the matching.

A direct consequence from the stable matching conditions is that a packet is part of the matching if it is on the top of both the input priority list and the output priority list. The conditions for a stable matching can be achieved by the Gale-Shapley algorithm for the stable marriage problem. The basic Gale-Shapely algorithm goes as follows:

The Gale-Shapely algorithm

Step 1. *Request.* Each unmatched output that has not been rejected by all the inputs sends a request to the input of the next packet in its priority list that has not rejected it.

Step 2. *Grant.* Each input grants the request to the output that is the highest one in its priority list. Reject all the other (and previous) requests.

Step 3. *Iterations.* Repeat from Step 1 until there is no unmatched output that has not been rejected by all the inputs.

In Step 2 it might happen that an input granted some output before and receives a request from another output higher in its priority list. When this happens, the output higher in its priority list is granted, and the output that is granted before is rejected and becomes unmatched. The complexity of the Gale-Shapely algorithm is $O(N^2)$. Readers might like to compare the stable matching algorithm with the maximal matching algorithm such as PIM and SLIP. The complexity of finding a maximal matching is only $O(N)$, which is simpler than that for finding a stable matching. As shown in Section 2.3.5, the maximal matching algorithm achieves 50% throughput. With a speedup factor of 2, it can reach 100% throughput. However, it does not guarantee exact emulation of the output-buffered switch as the CIOQ switch.

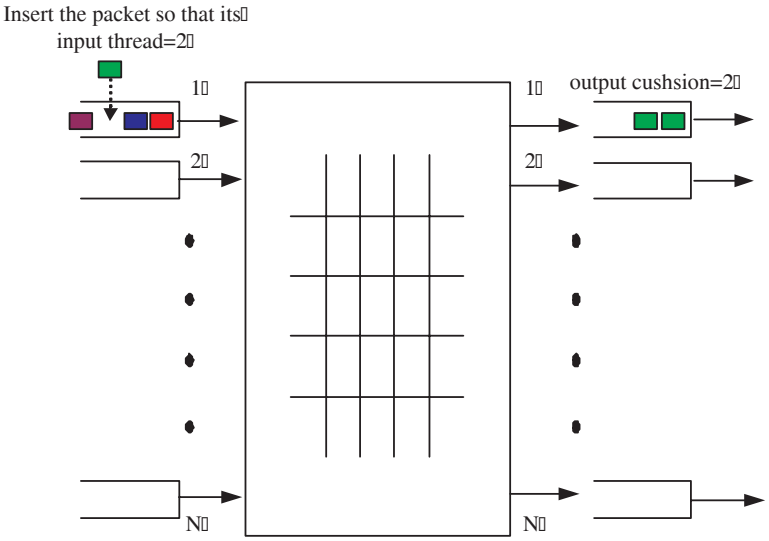


Fig. 2.39. An illustrating example for the critical packet first policy

Now we specify the critical packet first policy used for the insertion of an arriving packet into the input priority list. Define the *output cushion* of a packet by the number of packets waiting in its output buffer with smaller desired departure times. Also, define the *input thread* by the number of packets ahead of the packet in the input priority list (the input thread of a packet is zero once it is moved to the output buffer).

The *slackness* of a packet is defined to be the difference between its output cushion and its input thread. There are two cases to insert an arriving packet in the input priority list. If the output cushion of an arriving packet is smaller than the number of packets in its input buffer when the packet arrives, the packet is inserted in the input priority list so that its slackness is zero. For instance, as shown in Figure 2.39, a packet that arrives at input 1 and is destined to output 1 has output cushion 2. The packet is then placed in its input priority list so that its input thread is 2. By so doing, its slackness is 0. On the other hand, if the output cushion of an arriving packet is not smaller than the number of packets in its input buffer, then the arriving packet is placed at the end of the input priority list. In either case, the slackness is nonnegative when a packet arrives.

Now we show that the stable matching algorithm in the CIOQ switch with a speedup factor of 2 achieves exact emulation. The first observation is that the slackness of a packet waiting at an input buffer is non-decreasing in time. To see this, consider a particular packet p at an input buffer. As there is at most one packet arrival to the input buffer of packet p in a time slot, the input thread of packet p can be increased by at most one. Similarly, there is at most one packet departure from the output port of packet p in a time slot, the output cushion of packet p can be decreased by at most 1. However, as there are two stable matchings in a time slot, conditions (ii) and (iii) ensure that the slackness will be increased by (at least) 2 from these two stable matchings (if the packet is not transferred to the output). Thus, the slackness of a packet waiting at an input buffer is non-decreasing in time. As the slackness of a packet is nonnegative when it arrives, we know that the slackness of a packet is always nonnegative.

Suppose that the CIOQ switch achieves exact emulation up to time $t-1$ as the induction hypothesis. Consider a packet p that departs from the corresponding output-buffered switch at time t. Packet p is either in its input buffer or its output buffer of the CIOQ switch. If it is in the output buffer, it may depart on time. Thus, we only need to consider the case that packet p is still in the input buffer. Since the CIOQ switch achieves exact emulation up to time $t-1$, all the packets ahead of packet p in its output priority list must have departed by time $t-1$. Thus, the output cushion of packet p must be zero. Since the slackness of a packet is always nonnegative, the input thread of packet p is also zero. As such, packet p is on the top of both the input priority

list and the output priority list. The stable matching algorithm then guarantees that packet p is part of the matching and it is transferred from its input to its output at time t.

2.7 Knockout switches

The Knockout switch in [170, 93] is a clever approximation of an output-buffered switch. As shown in Figure 2.40, the $N \times N$ Knockout switch consists of N broadcast buses (with each input having its own bus) and N bus interfaces (with each output having its own bus interface). At each time slot, each input can transmit at most one packet onto its own bus. Through the bus interface, each output selects the packets destined to it and sends them out one by one from the output link. Within a time slot, there might be N packets arriving at a bus interface. As there is at most one packet departing from a bus interface within a time slot, buffering is required at each bus interface. If we would like to achieve exact emulation of an output-buffered switch, a speedup factor of N is required for the buffer at each interface (cf. the shared medium switch in Figure 2.2). However, if we are only interested in obtaining a good approximation of an output-buffered switch, we may not need to speed up N times.

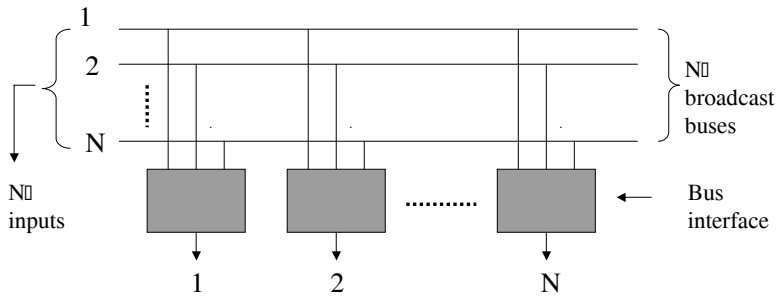


Fig. 2.40. The Knockout Switch architecture

2.7.1 The Knockout principle

In Figure 2.41, we show the clever design of the bus interface of the Knockout switch. The bus interface consists of three parts. The first one is the packet filter that receives the packet destined to the output

and discards others. The second one is a concentrator that only selects at most L packets from the packets destined to the output. The last one is a buffered L-to-1 multiplexer.

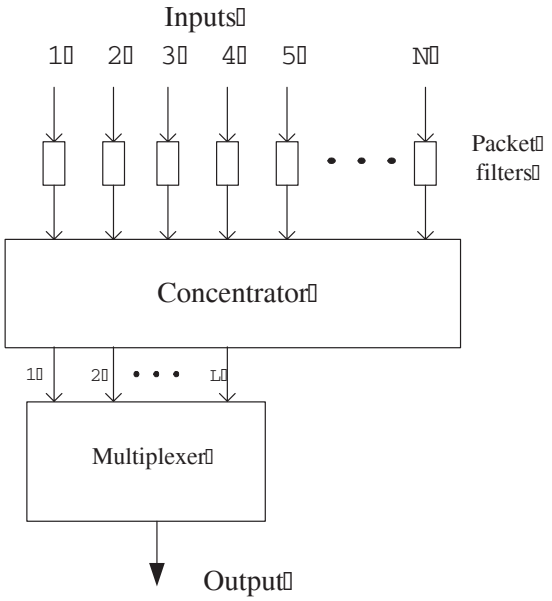


Fig. 2.41. The bus interface in the Knockout Switch

As there are only L outputs for the N inputs at the concentrator, packets could be lost at the concentrator if there are more than L packets destined to a given output at the same time slot. The famous Knockout principle is that the parameter L of the concentrator could be chosen (independent of the size of switch N) so that packets lost at the concentrator would not be a concern. The intuition behind this goes as follows: for a stable output-buffered switch, the expected number of packet to a given output is at most 1. Thus, the probability that there are more than $L >> 1$ packets to a given output at the same time is very small.

To illustrate the Knockout principle, consider the uniform i.i.d. traffic model. With probability ρ, assume that a packet arrives at each input for every time slot. This is independent of everything else. The destination of an arriving packet is chosen uniformly among the N output ports. This is also independent of everything else. Denote by

p_k the probability that there are k packets destined to a given output. As the number of arrivals to a given output at a time slot is the sum of N Bernoulli r.v.'s with parameter ρ/N, we then have

$$p_k = \frac{N!}{k!(N-k)!}(\frac{\rho}{N})^k(1-\frac{\rho}{N})^{N-k}. \tag{2.72}$$

It follows that the probability of a packet being dropped in a concentrator with N inputs and L outputs is

$$\begin{aligned} p_{\text{loss}} &= \frac{\text{the expected number of losses in a time slot}}{\text{the expected number of arrivals in a time slot}} \\ &= \frac{1}{\rho}\sum_{k=L+1}^{N}(k-L)p_k. \end{aligned} \tag{2.73}$$

When $N \to \infty$, the sum of N Bernoulli r.v.'s with parameter ρ/N converges to a Poisson r.v. with mean ρ. Thus,

$$\lim_{N\to\infty} p_k = e^{-\rho}\frac{\rho^k}{k!}. \tag{2.74}$$

Replacing (2.74) in (2.73) yields

$$\begin{aligned} \lim_{N\to\infty} p_{\text{loss}} &= \frac{1}{\rho}\sum_{k=L+1}^{\infty}(k-L)\frac{e^{-\rho}\rho^k}{k!} \\ &= (1-\frac{L}{\rho})(1-\sum_{k=0}^{L}\frac{e^{-\rho}\rho^k}{k!}) + \frac{e^{-\rho}\rho^L}{L!}. \end{aligned} \tag{2.75}$$

Using (2.75), one can show that p_{loss} is at the order of 10^{-9} when $L = 11$. Thus, when N is large, one can choose L large enough such that packet lost at the concentrator would not be a concern.

2.7.2 *L*-to-1 multiplexer

Within a time slot, the L-to-1 multiplexer might receive up to L packets. As there is at most one packet that can depart from the multiplexer within a time slot, buffering is needed. If one implements the L-to-1 multiplexer by a shared memory switch in Figure 2.1, then one might need a speedup factor of $L + 1$ for such an implementation. In the Knockout switch, the required speedup is achieved by a clever design that uses parallel buffers.

To explain the design of the multiplexer, we first introduce a well known queueing result. Consider a system with parallel queues as

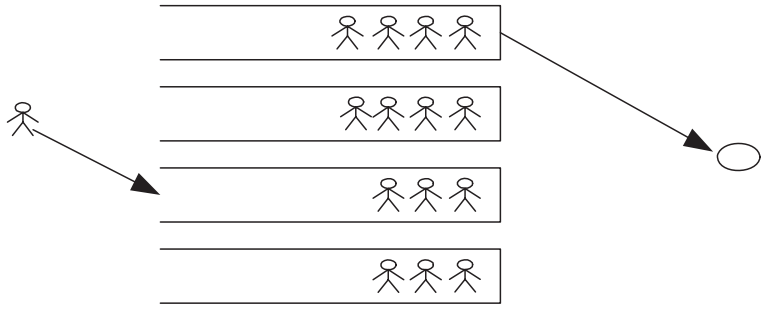


Fig. 2.42. A system with parallel queues

shown in Figure 2.42. Suppose that we operate the system as follows: a customer that arrives at the system joins the shortest queue (the queue with the least number of customers), and the server after completing the service of a customer always chooses a customer from the longest queue (the queue with the largest number of customers). By so doing, these parallel queues are kept in the most balance state, i.e., at any time the difference between the number of customers in the longest queue and the number of customers in the shortest queue is at most 1. If the service time of all the customers are all identical, then the longest queue service policy and the shortest queue dispatching policy are simply the round robin policy. Moreover, as the system always serves the longest queue, the customer with the longest waiting time is served. Thus, the system with parallel queues behaves as if it were a single FIFO queue with a shared buffer.

In Figure 2.43, we show the design of the L-to-1 multiplexer in the Knockout switch. The design consists of a shifter and L parallel buffers. We number the L inputs and the L parallel buffers from $0, 1, \ldots, L-1$. As in Figure 2.42, the multiplexer keeps two pointers: $O(t)$ for the longest queue at time t, and $I(t)$ for the shortest queue at time t. Within each time slot, a packet is removed from the longest queue. As all the packets have the same size,

$$O(t+1) = O(t) + 1 \mod L,$$

i.e., the pointer for the longest queue moves in a round robin fashion. Without loss of generality, one may consider packets arriving at the multiplexer in the order of their input numbers (even though they arrive at the same time slot). The shifter is an $L \times L$ crossbar switch fabric. The connection pattern of the shifter is set in the way that the

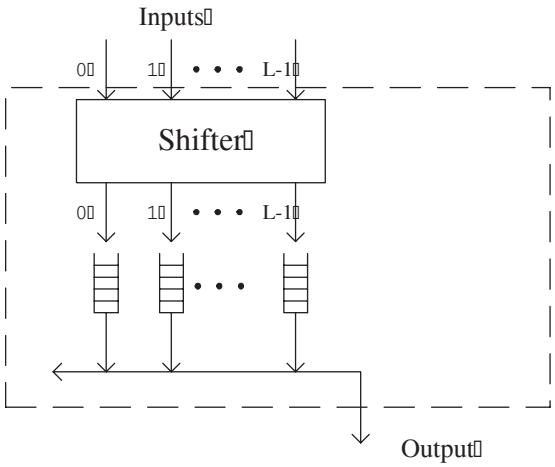


Fig. 2.43. The L-to-1 multiplexer in the Knockout switch

packet at input 0 is distributed to the shortest queue, the packet at input 1 is distributed to the second shortest queue, and so forth. To be precise, suppose that there are k packets arriving at the multiplexer at time t. The pointer for the shortest queue is updated as follows:

$$I(t+1) = I(t) + k \mod L.$$

The connection pattern is set to $P^{I(t)}$, where P is the $L \times L$ circular-shift matrix, i.e., $P_{i,j} = 1$ when $j = i+1 \bmod L$ and $P_{i,j} = 0$ otherwise.

Example 2.7.1. (Shifter) Consider the case that $L = 4$. The circular-shift matrix for this case is

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

As shown in Figure 2.44, suppose $I(t) = 0$ and there are three packet arrivals at time t. Then $P^{I(t)}$ is the identity matrix and these three packets are distributed to buffer 0, buffer 1, and buffer 2, respectively. Now $I(t+1)$ is updated to 3 and

$$P^3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

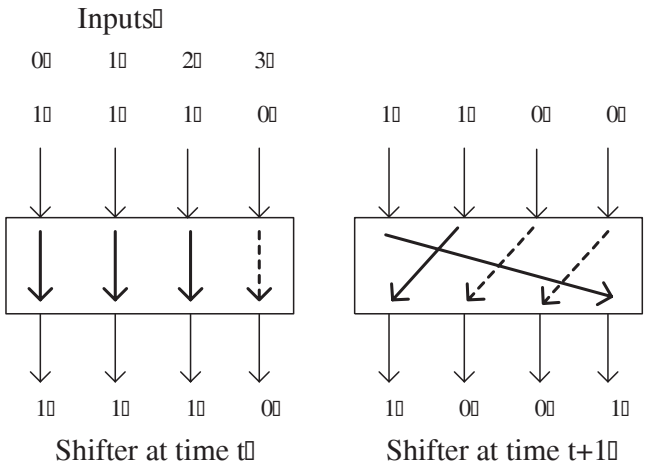


Fig. 2.44. An illustration of the shifter function

If there are two packet arrivals at time $t+1$, then these two packets are distributed to buffer 3 and buffer 0, respectively. Also,

$$I(t+2) = I(t+1) + 2 \mod 4 = 1.$$

2.7.3 Fast Knockout concentrator

In Section 2.5.6, we have already introduced a method to construct an N-to-L concentrator. The first stage of such a construction consists of two $\frac{N}{2} \times \frac{N}{2}$ sorters. The L largest elements of these two sorters are then selected to form a bitonic list with length $2L$. The L largest elements among the N inputs can then be obtained by feeding the bitonic list with length $2L$ to L 2×2 down sorters at the second stage.

In the case that $L << N/2$, using $\frac{N}{2} \times \frac{N}{2}$ sorters to find the sorted L largest elements may not be a good idea. Here we introduce a fast Knockout algorithm that finds the sorted L largest elements among $N/2$ inputs. Our presentation of the fast Knockout algorithm here follows the notions of state vectors used in [112, 111].

The objective of the fast Knockout algorithm is to find the sorted L largest elements among 2^k elements. Such a network element is called a 2^k-to-L concentrator/sorter. For this, one defines a j-element in a set as an element that is known to be smaller than at least j other elements in

that set. For a partially sorted set S, let x_j be the number of j-elements in S. Then S can be represented by a state vector $(x_0, x_1, \ldots, x_{L-1})$.

(i) Initially, every element is a 0-element. Thus, there are 2^k sets with the state vector $(1, 0, \ldots, 0)$.

(ii) (Fast Knockout stages) Let S_1 and S_2 be two disjoint sets with the same state vector $(x_0, x_1, \ldots, x_{L-1})$. Pair every j-element in S_1 with another j-element in S_2 and feed them to a 2×2 sorter. **After the comparison of a j-element pair, the winner remains as a j-element and the loser becomes a $2j+1$-element.** If $2j+1 \geq L$, this element is discarded as it cannot be one of the L largest elements. This leads to a new partially sorted set $S_1 \cup S_2$ with the following state vector

$$(x_0, x_0 + x_1, x_2, x_1 + x_3, x_4, x_2 + x_5, \ldots, x_{2i}, x_i + x_{2i+1}, \ldots).$$

For instance, after the first fast Knockout stage, there are 2^{k-1} sets with the state vector $(1, 1, 0, \ldots, 0)$.

(iii) (Knockout stages) After k fast Knockout stages, one obtains a single partially sorted set. Suppose that the state vector of this partially sorted set is $(x_0, x_1, \ldots, x_{L-1})$. Split the set into two disjoint sets S_1 and S_2 such that S_1 has the state vector

$$(\lfloor x_0/2 \rfloor, \lfloor x_1/2 \rfloor, \ldots \lfloor x_{L-1}/2 \rfloor)$$

and S_2 has the state vector

$$(\lceil x_0/2 \rceil, \lceil x_1/2 \rceil, \ldots, \lceil x_{L-1}/2 \rceil).$$

Note that $\lceil x \rceil$ is the ceiling function that returns an integer not smaller than x and $\lfloor x \rfloor$ is the floor function that returns an integer not larger than x. Pair every j-element in S_1 with another j-element in S_2 and feed them to a 2×2 sorter. **After the comparison of a j-element pair, the winner remains as a j-element and the loser becomes a $j+1$-element**. If $j+1 \geq L$, this element is discarded as it cannot be one of the L largest elements. This leads to a new partially sorted set $S_1 \cup S_2$ with the following state vector

$$\begin{aligned}(\lceil x_0/2 \rceil, \lfloor x_0/2 \rfloor + \lceil x_1/2 \rceil, \ldots, \lfloor x_{j-1}/2 \rfloor + \lceil x_j/2 \rceil,\\ \ldots, \lfloor x_{L-2}/2 \rfloor + \lceil x_{L-1}/2 \rceil).\end{aligned}$$

Repeat this step until the state vector becomes $(1, 1, \ldots, 1)$.

The difference between the fast Knockout stages and the Knockout stages is the transitive law. In a fast Knockout stage, one can apply the transitive law to infer that the loser in a j-element comparison becomes a $2j + 1$-element (since the two sets being compared are disjoint). However, in a Knockout stage, one can only infer that the loser in a j-element comparison becomes a $j + 1$-element. This is because there is no way to know whether the two sets that contain elements larger than these two j-elements are disjoint. In case that one would like to design an M-to-L concetrator/sorter with M not being a power of 2, one can always use the Knockout stages to achieve the objective as originally presented in the Knockout switches [170, 93].

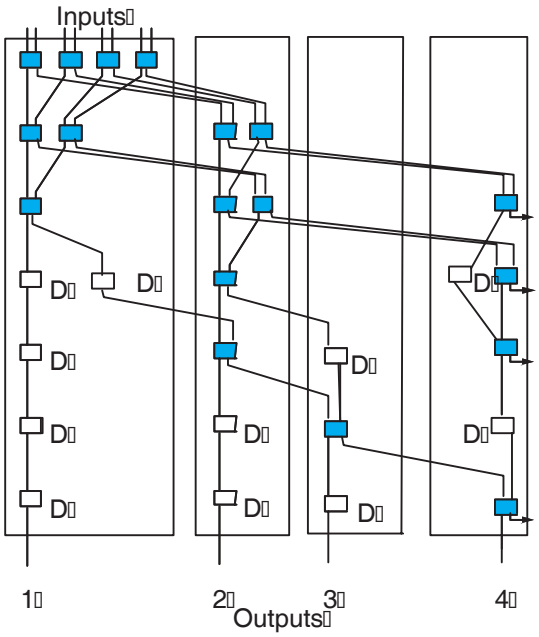


Fig. 2.45. The 8-to-4 fast Knockout concentrator/sorter

In Figure 2.45, we show the 8-to-4 fast Knockout concentrator/sorter. The first three stages are the fast Knockout stages and the last four stages are the Knockout stages. In Figure 2.45, each shaded box with 2 inputs and 2 outputs is a 2×2 sorter and each box with a single input and a single output is a delay element that synchronizes the timing for packets being compared. Initially, there are 8 disjoint sets with

the state vector $(1, 0, 0, 0)$. After the first stage, there are 4 disjoint sets with the state vector $(1, 1, 0, 0)$. After the second stage, there are 2 disjoint sets with the state vector $(1, 2, 0, 1)$. After the third stage, there is a single set with the state vector $(1, 3, 0, 3)$. Note that in each of the fast Knockout stages, one needs four 2×2 sorters.

Now we are in the Knockout stage. Split the set with the state vector $(1, 3, 0, 3)$ into two sets with the state vectors $(0, 1, 0, 1)$ and $(1, 2, 0, 2)$. For this stage, one needs two 2×2 sorters. After the fourth stage, we end up with the set with the state vector $(1, 2, 1, 2)$. Now split this set into two sets with the state vectors $(0, 1, 0, 1)$ and $(1, 1, 1, 1)$. For this stage, one still needs two 2×2 sorters. After the fifth stage, we end up with the set with the state vector $(1, 1, 2, 1)$. Split this set into two sets with the state vector $(0, 0, 1, 0)$ and $(1, 1, 1, 1)$. For this stage, one only needs one 2×2 sorter. After the sixth stage, we have the set with the state vector $(1, 1, 1, 2)$. Finally, split this set into two sets with the state vector $(0, 0, 0, 1)$ and $(1, 1, 1, 1)$. For this stage, one still needs one 2×2 sorter. After the seventh stage, we have reached the state vector $(1, 1, 1, 1)$.

2.8 Notes

The study of switching theory starts from circuit switching and then evolves into packet switching. The focus of circuit switching is to build *nonblocking* switches that can set up or tear down a connection in any manner. C. Clos (1953) proposed the three-stage network in [51] that sets up the first step to reduce the complexity of a crossbar switch fabric. A special case of the Clos network is the rearrangeable network that realizes all the sub-permutations. Unlike nonblocking switch, a rearrangeable network cannot set up a connection without affecting the existing connections. A. M. Duguid (1959) published the rearrangement theorem in [59]. Such a theorem was also prove by D. Slepian (1952) in an unpublished progress report [151]. The matrix representation of a network state was introduced by M. C. Paull (1962) in [135] that greatly eases the presentation of the Slepian-Duguid algorithm for the three-stage rearrangeable networks. V. E. Benes made the connecting network popular in his book [14] by recursively expanding the three-stage rearrangeable networks. Finding the routing paths in the three-stage rearrangeable networks (and Benes networks) by the framed Birkhoff-von Neumann decomposition was first introduced

by T. Inukai [85] (for time slot assignments in satellite switches). The most efficient algorithm for finding the routing paths in the three-stage rearrangeable networks was proposed by H. Y. Lee, F. K. Hwang and J. Capinelli in [106]. See also the book by F. K. Hwang (pp. 63-66, [84]) for the history for developing the routing algorithms. Our development for the two-stage construction of switch fabrics follows the book by S.-Y. R. Li [111], in particular the CU nonblocking switches. All this two-stage construction theory seems to be rooted from the work by K. E. Batcher for bitonic sorting networks [13]. The monotone and consecutive condition for banyan networks was due to A. Huang and S. Knauer [73]. The use of the Batcher-banyan network for output conflict resolution in the three phase switches was introduced by J. Y. Hui and E. Arthurs [77, 76].

Note that there are several issues in circuit switching that we did not address in this chapter:

(i) The search of the minimum complexity switch fabrics: it is known that the complexity of an $N \times N$ nonblocking switch is bounded below by $O(N \log N)$. By carefully expanding the three-stage nonblocking Clos network, it is known that the best complexity of such an expansion is $O(N(\log N)^{2.44})$ (see e.g., pp. 245-248, [111]). In Problem 20, we introduce the Cantor network [21] with the complexity $4N(\log_2 N)^2$. Using the concept of expanders [139, 119, 12, 64], it is known that there exists a nonblocking switch with $O(N \log N)$ complexity (see e.g., pp. 250-253, [111]). Though there exists a nonblocking switch with $O(N \log N)$ complexity, finding the routing paths of such a switch becomes extremely difficult. For sorting networks, it is shown in [5] that there exists a sorting network with $O(N \log N)$ complexity. The complexity of such a network is asymptotically smaller than that of the Batcher sorting network. As commented in [111], the leading constant of the $O(N \log N)$ complexity is extremely large and thus makes it infeasible for practical applications.

(ii) Wide sense nonblocking switches: Nonblocking switches allow a connection to set up or tear down in any manner. A wide sense nonblocking switch, a weaker version of a nonblocking switch, realizes all the sub-permutations by a specific routing algorithm of setting up connections (without affecting any existing connections). As such, the complexity of wide sense nonblocking switches should be between that of nonblocking switches and that of rearrangeable

networks. There have been a lot of works reported in the literature for wide sense nonblocking switches. See e.g., the book by V. E. Benes [14], the book by F. K. Hwang [84] and the book by S.-Y. R. Li [111].

(iii) Equivalence of multistage interconnection networks: as discussed in Section 2.5, there are several ways for recursive two-stage expansion of an $N \times N$ switch when N is a power of 2. As such, there are several ways to interconnect $\log_2 N$ stages of 2×2 switches (with $N/2$ switches in each stage) and this results in different names of multistage interconnection networks, such as banyan networks, omega networks, baseline networks, and shuffle exchange networks. All these interconnection networks, though appeared to be different, are in fact topologically equivalent (after rearrangements of the 2×2 switches). See e.g., [167, 168, 1, 16]. A recent algebraic approach for establishing the equivalence of these network is shown in the book by S.-Y. R. Li [111].

(iv) Multicasting and copy networks: in this chapter, we only consider switch fabrics that realize sub-permutations. This is also known as point-to-point connections. In the multicasting setting, an input is allowed to connect to multiple outputs at the same time. For this, one needs a copy network that forks a connection to multiple outputs (see e.g. [107]). See also the books [76, 84, 111] for more discussions on this.

(v) Traffic theory: traditionally, Markov processes are used for analyzing circuit switching systems. In the book by Benes [14], he also introduced a thermodynamic theory that established the connection between statistical mechanics and traffic theory for connecting networks. Such a theory is further extended in the book by J. Y. Hui [76] via the large deviation principles. Further developments along this line for circuit switching can be found in [66, 41]. Large deviation principles are also used in dimensioning the size of buffers and capacities in packet switching systems. A theory, known as the theory of effective bandwidths, was developed in the early 90's (see e.g., [94, 25, 166, 62] and the book [27] for more references). Readers interested in the large deviation principles may consult the following books: A. Shwartz and A. Weiss [150] for communication and computing, J. Bucklew [19] for engineering, R. Ellis [61] for statistical mechanics, and A. Dembo and O. Zeitouni [57] for mathematical theory.

With the advances of electronic memory in both the accessing speed and the density, packet switching systems become more cost effective than circuit switching systems. The simplest packet switch architecture is the shared memory switch. As addressed in this chapter, such a switch architecture does not scale and one has to resort to switches with parallel buffers. However, input-buffered switches with FIFO queue suffered from the head-of-line blocking problem and it was shown by M. J. Karol, M. G. Hluchyj, and S. P. Morgan [92] that such a switch architecture only has 58% throughput. To solve the head-of-line blocking problem, virtual output queueing is needed and this creates the matching problem between inputs and outputs. Parallel Iterative Matching (PIM) and Round-robin Matching (RRM) are two commonly used matching algorithms introduced in [6]. However, PIM needs random arbitration and that is expensive to build. On the other hand, RRM suffers from the pointer synchronization problem and may be trapped in a bad mode that yields very low throughput. To solve these two problems, N. Mckeown [121] proposed the SLIP algorithm. Though the SLIP algorithm performs very well for most of the traffic generated in [121], it is shown by C.-S. Chang, D.-S. Lee and Y.-S. Jou in [34] that there is still a bad mode in SLIP. Other popular matching algorithms include Dual Round Robin Matching (DRRM) by Y. Li, S. Panwar and H. J. Chao [113] (see Problem 8) and wave front arbitration by Y. Tamir and H. C. Chi [155] (see Problem 9).

N. McKeown, V. Anantharam and J. Walrand proved that 100% throughput can be achieved in an input-buffered switch by the maximum weighted matching in [123]. See also [124] for a more practical algorithm. The use of the Birkhoff-von Neumann decomposition for further providing rate guarantees in an input-buffered switch was introduced by C.-S. Chang, W.-J. Chen and H.-Y. Huang in [28]. See also [7, 100] for improvements. Framed Birkhoff-von Neumann decomposition for packet switching can also be found in [6, 108, 79, 110]. Other methods for providing rate guarantees in an input-buffered switch were reported in [152, 45, 102].

Combined Input Output Queuing (CIOQ) was first introduced by S.-T. Chuang, A. Goel, N. McKeown and B. Prabhakar in [50] for exact emulation of an output-buffered switch. See also [153] for an alternative algorithm of doing this.

Knockout switches were introduced by M. J. Karol and M. G. Hluchyj in [93] and Y. S. Yeh, M. G. Hluchyj, and A. S. Acampora in

[170]. The fast Knockout algorithm was introduced by S.-Y. R. Li and C.-M. Lau in [112]. Our development for the fast Knockout algorithm in this chapter follows that in [111].

For a survey of packet switch architectures before 90's, we refer to the survey paper by H. Ahmadi and W. E. Denzel [3]. For a survey of more recent development of practical switch architectures, we refer to the book by H. J. Chao, C. H. Lam and E. Oki [43].

Problems

1. Consider the Lindley equation in (2.1)

$$q(t+1) = (q(t) + a(t+1) - 1)^+.$$

Suppose that $q(0) = 0$. Show that

$$q(t) = \max_{0 \le s \le t} [A(t) - A(s) - (t-s)],$$

where $A(t) = \sum_{s=1}^{t} a(s)$ is the cumulative number of packets that arrive by time t (for a proof, see e.g. [27], Lemma 1.3.1).

2. Continue from the previous problem. Suppose that $\{a(t), t \ge 1\}$ is *stationary*, i.e., its joint distribution is invariant to time shift. Show that $\{q(t), t \ge 0\}$ is a sequence of stochastically increasing random variables, i.e.,

$$\mathsf{P}(q(t) \ge k) \le \mathsf{P}(q(t+1) \ge k)$$

for all k. As such, $\lim_{t\to\infty} \mathsf{P}(q(t) \ge k)$ exists.

3. (Loynes [116]) Continue from the previous problem. Suppose that $\{a(t), t \ge 1\}$ is *stationary* and *ergodic* with mean ρ, i.e.,

$$\lim_{t\to\infty} \frac{A(t)}{t} = \mathsf{E}a(1) = \rho, \quad a.s.$$

If $\rho < 1$, show that

$$\lim_{k\to\infty} \lim_{t\to\infty} \mathsf{P}(q(t) \ge k) = 0.$$

As such $q(t)$ converges to a steady state *random variable*.

4. (Pollaczek-Khinchin formula) Continue from the previous problem. Suppose that $\{a(t), t \geq 1\}$ is a sequence of independent and identically distributed random variables with mean $\rho < 1$. Let $G_a(z)$ be the moment generating function of $a(1)$, i.e.,

$$G_a(z) = \sum_{k=0}^{\infty} \mathsf{P}(a(1) = k) z^k.$$

From the previous problem, we know there exists a steady state random variable $q(\infty)$ for the Lindley equation, i.e.,

$$\mathsf{P}(q(\infty) = k) = \mathsf{P}((q(\infty) + a(1) - 1)^+ = k),$$

for all k. Let $G_q(z)$ be the moment generating function of $q(\infty)$, i.e.,

$$G_q(z) = \sum_{k=0}^{\infty} \mathsf{P}(q(\infty) = k) z^k.$$

Show that

$$G_q(z) = (1 - \rho)(z - 1)/(z - G_a(z)).$$

Use this moment generating function to show Proposition 2.1.5.

5. Instead of using the VOQ technique for the head-of-line blocking problem in input-buffered switches, one partial solution is to allow multiple head-of-line packets for each input port. This can be done either by looking into the first k packets in the single input queue (window policy) or by keeping k input queues at each input port. Use the analysis in Section 2.2.2 to show that the maximum throughput is $k + 1 - \sqrt{k^2 + 1}$ if there are k head-of-line packets at each input port (see e.g., [101, 157]). Hint: replace (2.29) by $\mathsf{E}(X(t)) = k$.
6. Continue from the previous problem. Suppose that every blocked head-of-line packet is randomly replaced by another packet in the queue. Show that the maximum throughput is $1 - e^{-k}$. In particular, if $k = 1$, then the maximum throughput is $1 - e^{-1} \approx 0.63$, which is higher than 0.58 in the original setting. Hint: $X(t)$ (the number of HOL packets that are destined for output port 1 at time t) converges to a Poisson random variable with mean k.
7. The maximum matching algorithm is the algorithm that finds the maximum number of matched links in a bipartite graph. Show by a counterexample that the maximum matching algorithm does not provide 100% throughput for non-uniform traffic.

8. A simpler matching algorithm than SLIP is called the Dual Round Robin Matching (DRRM) in [113]. As in SLIP, there is a pointer associated with each input and output. The DRRM algorithm consists of the following two steps:

   Step 1. *Request.* Each input sends a request to the output that has a non-empty VOQ and is closest to its pointer. The pointer at that input is incremented *clockwise* to one location beyond the requested output if and only if the request is granted in Step 2.

   Step 2. *Grant.* If an 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.

   As each input in DRRM only sends at most one request, there is no need to carry out the third step in SLIP. Show that DRRM achieves 100% throughput if all the VOQs have infinite numbers of packets. (Hint: it suffices to show that all the pointers will be desynchronized after a fixed number of time slots for any initial state of the pointers)

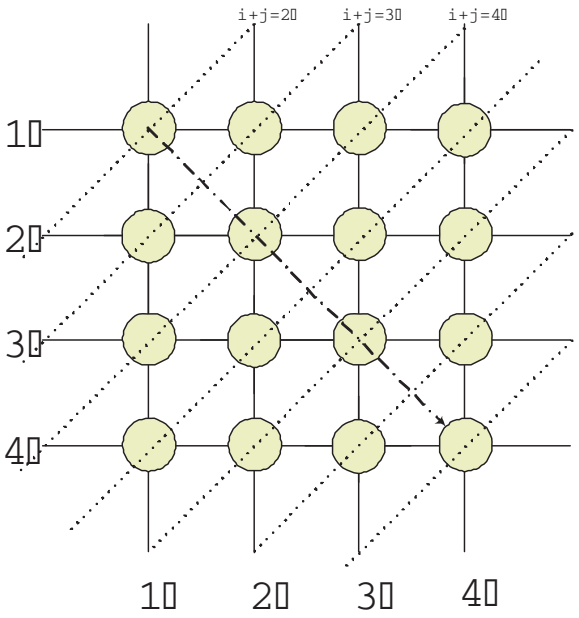


Fig. 2.46. Wave front arbitration for an 4×4 crossbar switch

9. (Wave front arbitration) Another way to find a maximal matching in an input-buffered switch is called the wave front arbitration (see e.g., [155]). Let $Z = (z_{i,j})$ be the $N \times N$ matrix with $z_{i,j} = 1$ if the j^{th} VOQ at the i^{th} input is nonempty and $z_{i,j} = 0$ otherwise. Then finding a matching is to find a sub-permutation matrix $P = (p_{i,j})$ such that $p_{i,j} \leq z_{i,j}$. The idea of the wave front arbitration is to find the sub-permutation matrix for the elements $p_{i,j}$ with identical $i + j$.

Step 1. Initially, set $x_{i,j} = y_{i,j} = 1$ for all i, j.

Step 2. For $i + j = 2, \ldots, 2N$ (see Figure 2.46), set $p_{i,j} = 1$ if $z_{i,j} = x_{i,j} = y_{i,j} = 1$ and $p_{i,j} = 0$ otherwise. If $p_{i,j} = 1$, set $x_{i+1,j} = 0$ and $y_{i,j+1} = 0$. If $x_{i,j} = 0$, set $x_{i+1,j} = 0$. If $y_{i,j} = 0$, set $y_{i,j+1} = 0$.

Note that once $x_{i,j}$ is set to 0, then $x_{k,j} = 0$ for all $k > i$. Similarly, once $y_{i,j}$ is set to 0, then $y_{i,k} = 0$ for all $k > j$. By so doing, the variable $x_{i,j}$ (resp. $y_{i,j}$) serves as an indicator whether there is a conflict in the same row (resp. column). Thus, if $p_{i,j}$ is set to 1, then the elements that are either in the same row or in the same column will not be chosen. As such, the algorithm finds a maximal matching.

(i) Show that each iteration in Step 2 in the wave front arbitration can be implemented in parallel.

(ii) Now define

$$i \oplus j = \begin{cases} i + j, & \text{if } i + j \leq N \\ i + j - N, & \text{if } i + j > N \end{cases}.$$

Replace every "+' in Step 2 by "$\oplus$". Show that Step 2 in this modified wave front arbitration can be completed by doing the N iterations from $i \oplus j = 1$ to N. Moreover, each iteration can also be implemented in parallel.

(iii) Let P_k, $k = 1, \ldots, N$, be the permutation matrix with its $(i, j)^{th}$ element being 1 if $i \oplus j = k$ and 0 otherwise. Show that the modified wave front arbitration is in fact a priority matching algorithm, starting from the matching P_1 to the matching P_N.

(iv) Give a method to achieve fairness among all the VOQs in the modified wave front arbitration (Hint: rotating the priorities in the N permutation matrices $P_1, \ldots, P_N$).

10. (Hall Theorem [71]) The vertices in a bipartite graph can be partitioned into two sets IN and OUT. Let A be a subset of the IN sets. Define the neighbor of the set A, denoted by $N(A)$, to be the set of vertices in the OUT set that is connected by an edge from a vertex in A. Denote by $|S|$ the number of elements in a set S. The Hall theorem says that there exists a perfect matching if and only if $|N(A)| \geq |A|$ for any subset A of the IN set. Use the Hall theorem to show the existence of a permutation in the first step of the Birkhoff decomposition (Algorithm 2), i.e., for a doubly stochastic matrix $\tilde{R}$, there exists a permutation $(i_1, i_2, \ldots, i_N)$ such that $\prod_{k=1}^{N} \tilde{r}_{k,i_k} > 0$. (Hint: map the doubly stochastic matrix $\tilde{R}$ to a bipartite graph by creating an edge from vertex i in the IN set to vertex j in the OUT set if $\tilde{r}_{i,j} > 0$. As the matrix $\tilde{R}$ is doubly stochastic,

$$|A| = \sum_{i \in A} \sum_{j=1}^{N} \tilde{r}_{i,j} = \sum_{i \in A} \sum_{j \in N(A)} \tilde{r}_{i,j}$$
$$= \sum_{j \in N(A)} \sum_{i \in A} \tilde{r}_{i,j}$$

and

$$\sum_{i \in A} \tilde{r}_{i,j} \leq 1.$$

From this, it follows that $|N(A)| \geq |A|$.)

11. One can trade off the number of permutation matrices in the Birkhoff-von Neumann decomposition with the throughput. Consider an $N \times N$ doubly substochastic matrix $R = (r_{i,j})$. Suppose that for some integer m

$$\sum_{i=1}^{N} r_{i,j} \leq 1 - \frac{1}{m}, \quad j = 1, 2, \ldots, N,$$

and

$$\sum_{j=1}^{N} r_{i,j} \leq 1 - \frac{1}{m}, \quad i = 1, 2, \ldots, N.$$

Let $\tilde{R} = (\tilde{r}_{i,j})$ with $\tilde{r}_{i,j} = \lceil mNr_{i,j} \rceil / mN$, where $\lceil x \rceil$ is the ceil function that returns the smallest integer that is not smaller than x.

(i) Show that $\tilde{R} \geq R$ and that $mN\tilde{R}$ is an integer-valued matrix.

(ii) Show that $\tilde{R}$ is still a doubly substochastic matrix, i.e.,

$$\sum_{i=1}^{N} \tilde{r}_{i,j} \le 1, \quad j = 1, 2, \ldots, N,$$

and

$$\sum_{j=1}^{N} \tilde{r}_{i,j} \le 1, \quad i = 1, 2, \ldots, N.$$

(iii) Use the framed Birkhoff-von Neumann decomposition to show that the number of matrices for $\tilde{R}$ is at most mN.

12. The number of permutation matrices in the Birkhoff-von Neumann decomposition is $O(N^2)$. This may not scale for a switch with a large number of input/output ports. A simpler, but less efficient, algorithm is proposed in [100].
(i) For an $N \times N$ doubly substochastic matrix $R = (r_{i,j})$, let $\phi_1 = r_{i_1,j_1}$ be the maximum element of the matrix R.
(ii) For $k = 2, \ldots, N$, find the maximum element r_{i_k,j_k} in the matrix that replaces every element in rows $i_1, \ldots, i_{k-1}$ and columns $j_1, j_2, \ldots, j_{k-1}$ by 0 in R.
(iii) Let P be the permutation matrix with $P_{i_k,j_k} = 1$, $k = 1, 2, \ldots, N$. Define the matrix

$$R_1 = (R - \phi_1 P_1)^+.$$

(iv) If R_1 is not a zero matrix, continue the decomposition from (i).
Show the following properties for this algorithm.
(i) The algorithm will be stopped by at most $2N - 1$ steps.
(ii) The algorithm generates a set of positive numbers ϕ_k and permutation matrices P_k, $k = 1, \ldots, K$, such that

$$R \le \sum_k \phi_k P_k.$$

(Note that $K \le 2N - 1$ from (i)).
(iii) $\phi_k \le 1/(1 + \lceil k/2 \rceil)$ for all k.
(iv) $\sum_k \phi_k \le 2 \log N + 1$.

13. Though the maximum weighted matching algorithm achieves 100% throughput for a single input-buffered switches, it is shown (see e.g., [8]) that it does not achieve 100% throughput in a network

of input-buffered switches. Show that if the rate for each flow in a network of input-buffered switches is known, then the Birkhoff-von Neumann decomposition can be used in every input-buffered switch to achieve 100% throughput.

14. Consider the doubly stochastic matrix

$$\tilde{R} = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.1 & 0.3 & 0.6 \\ 0.4 & 0.5 & 0.1 \end{bmatrix}.$$

Find a Birkhoff decomposition of $\tilde{R}$.

15. (Framed Birkhoff-von Neumann switch) Consider a 3×3 framed Birkhoff-von Neumann switch with frame size 10. At time t, the 10 sub-permutation matrices are as follows:

$$\tilde{P}_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \tilde{P}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \tilde{P}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

$$\tilde{P}_4 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \tilde{P}_5 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \tilde{P}_6 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

$$\tilde{P}_7 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tilde{P}_8 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tilde{P}_9 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\tilde{P}_{10} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

a) Find Paull's matrix to represent the 10 sub-permutation matrices.

b) Suppose that we would like to increase one unit of rate from input 3 to output 3. Use the Slepian-Duguid algorithm to find Paull's matrix to represent the 10 sub-permutation matrices after the rate has been increased.

16. In Figure 2.47, we show a three-stage 9×9 rearrangeable network. Consider the following permutation

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 4 & 5 & 9 & 7 & 8 & 6 & 1 & 2 & 3 \end{pmatrix}.$$

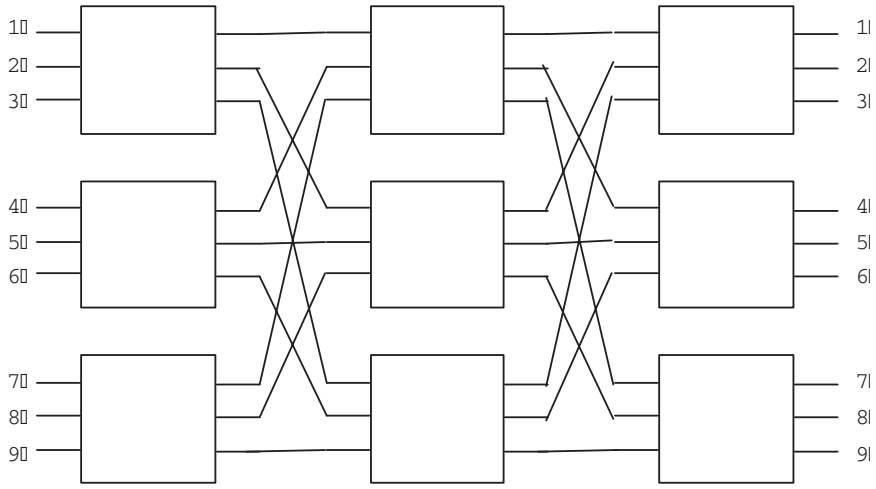


Fig. 2.47. A 9×9 switch

a) Use the Birkhoff-von Neumann decomposition to find all the connection patterns in the 9 3×3 switches that realizes π. (Hint: you need to decompose the frame matrix.)
b) Use the Lee-Hwang-Capinelli algorithm to find all the connection patterns in the 9 3×3 switches that realizes π. (Hint: you need to convert the frame matrix into a specification matrix.)

17. In Figure 2.48, we show the connection patterns of a three-stage 9×9 rearrangeable network for the following sub-permutation

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 4 & - & 9 & - & 8 & 6 & 1 & 2 & 3 \end{pmatrix}.$$

a) Name the three switches in the second stage from the top to the bottom: switch a, switch b and switch c. Write down the Paull matrix that represents the three connection patterns for the three switches in the second stage.
b) From the Paull matrix, use the Slepian-Duguid algorithm to find all the connection patterns in the 9 3×3 switches that realizes the following sub-permutation

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 4 & 7 & 9 & - & 8 & 6 & 1 & 2 & 3 \end{pmatrix}.$$

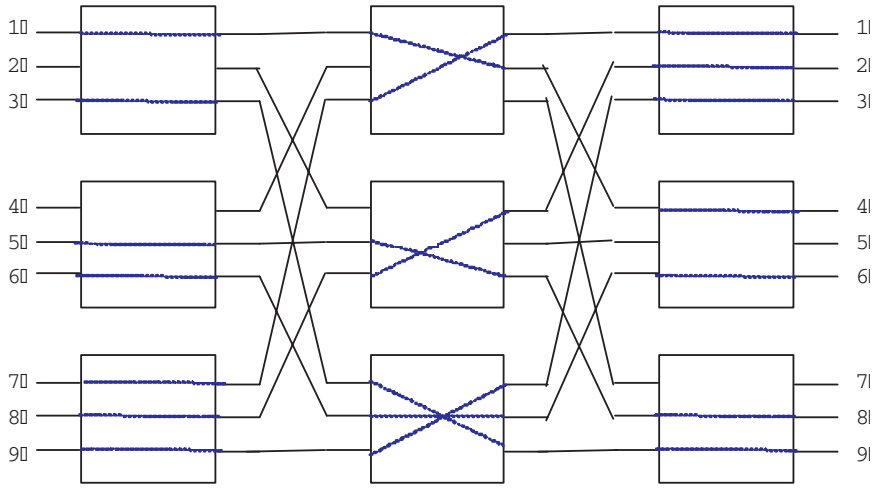


Fig. 2.48. The connection patterns in a 9×9 switch

18. For an $N \times N$ Benes network, show that the number of 2×2 switches is $N \log_2 N - \frac{N}{2}$ (Hint: let $N = 2^k$ and establish the recursive equation $B(k) = 2^k + 2B(k-1)$ with $B(1) = 1$).
19. As in Example 2.4.6, find a feasible set of connection patterns in the 8×8 Benes network for the following permutation matrix:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

20. It is known that an $N \times N$ Benes network is a rearrangeable network that realizes all the permutations. However, it is not a nonblocking network. To construct a nonblocking network, one may use parallel Benes networks as shown in Figure 2.49. Such a construction is known as the Cantor network [21]. In Figure 2.49, there are m $N \times N$ Benes networks. Each input is connected to one of the N inputs in every Benes network via a $1 \times m$ switch and each output is connected to one of the N outputs in every Benes network via an $m \times 1$ switch. In fact, a Cantor network is a special case of the

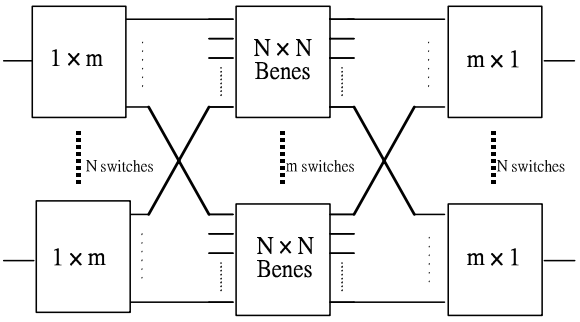


Fig. 2.49. The Cantor network

three stage Clos network with the switches in the middle stage replaced by the Benes networks.

(i) Let $Z(k)$ be the number of 2×2 switches at the k^{th} stage of the m Benes networks that are reachable by an input in the Cantor network. Clearly, $Z(1) = m$. Show that

$$Z(k) = 2Z(k-1) - 2^{k-2}.$$

(ii) Use the above recursive equation to show that

$$Z(\log_2 N) = \frac{1}{2}Nm - \frac{1}{4}(\log_2 N - 1)N.$$

(iii) Note that there are $Nm/2$ 2×2 switches at the $\log_2 N^{th}$ stage of the m Benes networks. As the Cantor network is symmetric, the number of 2×2 switches at the $\log_2 N^{th}$ stage that are reachable by an output is also $Z(\log_2 N)$. Use this to show that the Cantor network is nonblocking if $m > \log_2 N - 1$. (Hint: there must exist a 2×2 switch at the central stage that is reachable from both the input and the output if

$$Z(\log_2 N) + Z(\log_2 N) > \frac{Nm}{2}.)$$

(iv) From (iii), it follows that $m = \log_2 N$ suffices to ensure that the Cantor network is nonblocking. Show that the complexity of such a Cantor network is $O(N(\log_2 N)^2)$.

21. (Shuffle exchange network [73]) There are several networks that are topologically equivalent to banyan networks. One of them is the shuffle exchange network. As the banyan network, an $N \times N$

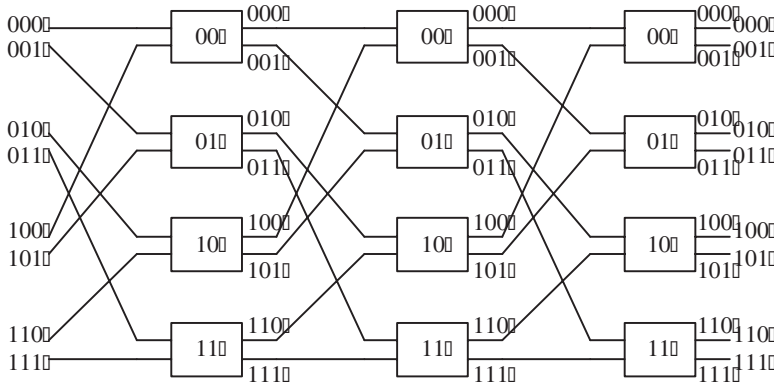


Fig. 2.50. An 8×8 shuffle exchange network

shuffle exchange network consists of $\log_2 N$ stages. Each stage consists of $N/2$ 2×2 switches. In each stage, the upper $N/2$ inputs are connected to the upper links of the $N/2$ 2×2 switches, and the lower $N/2$ inputs are connected to the lower links of the $N/2$ 2×2 switches. In Figure 2.50, we show an 8×8 shuffle exchange network. As the shuffle exchange network is topologically equivalent to the banyan network, it also has the self routing property. If we index each output by its binary representation, then a packet destined for a particular output takes the upper (resp. lower) output at the k^{th} stage if the k^{th} most significant bit of its binary representation is 0 (resp. 1). Number each switch at each stage from 0 to $N/2 - 1$ with the binary representation. Number the upper (lower) output link of a switch by appending a 0 (resp. 1) to the binary representation of the switch. Consider a packet from input i to output j. Let $n = \log_2 N$ and $i_1 i_2 \ldots i_n$ (resp. $j_1 j_2 \ldots j_n$) be the binary representation of i (resp. j). Show that the packet is routed to link $i_{k+1} \ldots i_n j_1 \ldots j_k$ at the output of the k^{th} stage, $k = 1, 2, \ldots, n$.

22. Continue from the previous problem. Consider another packet from input i' to output j'. Let $i'_1 i'_2 \ldots i'_n$ (resp. $j'_1 j'_2 \ldots j'_n$) be the binary representation of i' (resp. j'). Suppose that $i' > i$, $j' > j$, and $j' - j \geq i' - i$. Show that these two packets do not share the same link. As a result, as long as the monotone and consecutive condition is satisfied, there is no conflict in the shuffle exchange network. (Hint: prove by contradiction. Suppose both packets share the same output link at the k^{th} stage for some k. Then

$$i_{k+1} \dots i_n j_1 \dots j_k = i'_{k+1} \dots i'_n j'_1 \dots j'_k.$$

As $i' > i$, it follows that $i' - i \geq 2^{n-k}$. On the other hand, we have $j' - j < 2^{n-k}$. This then implies $i' - i > j' - j$ and we reach a contradiction.)

23. Merge-sort algorithm is a sequential algorithm that sorts a list into a monotonically increasing list. It uses a divide and conquer strategy as follows:
Algorithm Merge-sort
Input: An list $a = (a_1, a_2, \dots, a_N)$.
Procedure Sort:
Sort(a)=Merge(Sort($(a_1, \dots, a_{N/2})$),Sort($(a_{N/2+1}, \dots, a_N)$)).
Procedure Merge:
Merge($(a_1, \dots, a_m)$,$(b_1, \dots, b_k)$)
=(a_1, Merge($(a_2, \dots, a_m)$,$(b_1, \dots, b_k)$)) if $a_1 \leq b_1$,
=(b_1, Merge($(a_1, \dots, a_m)$,$(b_2, \dots, b_k)$)) if $a_1 > b_1$.
Show that the complexity of this algorithm is $O(N \log_2 N)$.
24. Recall that a bitonic list with length N is a list that is monotonically increasing to some k and then monotonically decreasing to N.
    a) (Unique cross over property) Consider a circular bitonic list $b = (b_1, b_2, \dots, b_N)$. Break this circular bitonic list into two equal halves $(b_1, \dots, b_{N/2})$ and $(b_{N/2+1}, \dots, b_N)$. Without loss of generality, assume that $b_1 > b_{N/2+1}$. Show that there exists a unique k such that $b_i > b_{N/2+i}$ for all $1 \leq i \leq k$ and $b_i < b_{N/2+i}$ for all $k < i \leq N/2$.
    b) (H-theorem) Let $b_i^M = \max(b_i, b_{N/2+i})$ for $i = 1, \dots, N/2$, and $b_i^m = \min(b_i, b_{N/2+i})$ for $i = 1, \dots, N/2$. Show that both $b^M = (b_1^M, \dots, b_{N/2}^M)$ and $b^m = (b_1^m, \dots, b_{N/2}^m)$ are circular bitonic lists. Moreover, the largest element in b^m is not larger than the smallest element in b^M.
25. (Batcher sorting theorem [13]) Use the results in the previous problem to show Theorem 2.5.5. (Hint: prove by induction. The $N/2$ 2×2 down sorters at the first stage breaks the input circular bitonic list into two equal halves, b^M and b^m. As both b^M and b^m are circular bitonic lists, they can be sorted by the two $\frac{N}{2} \times \frac{N}{2}$ bitonic down sorters at the second stage.)
26. (Outline of the proof for Theorem 2.5.3) To prove Theorem 2.5.3, one needs to show that given an $N \times N$ circular unimodal permutation π, the connection pattern realized (by the unique routing path

property) at every switch inside the X2 construction is a circular unimodal permutation. For this, one first dissects the permutation π into two parts: the increasing part and the decreasing part. For instance, for the permutation π in Example 2.5.2, the increasing part is

$$\begin{pmatrix} 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 1 & 3 & 5 & 7 & 9 & 13 & 16 & 19 & 20 \end{pmatrix}.$$

and the decreasing part is

$$\begin{pmatrix} 14 & 15 & 16 & 17 & 18 & 19 & 20 & 1 & 2 & 3 & 4 \\ 18 & 17 & 15 & 14 & 12 & 11 & 10 & 8 & 6 & 4 & 2 \end{pmatrix}.$$

Further partition the increasing part into consecutive segments $IS_1, IS_2, \ldots, IS_q$ so that IS_m (which may be null) contains the outputs in $\{(m-1)p+1, \ldots, mp\}$. For instance, in Example 2.5.2, one has $p = 4$, $q = 5$, and

$$IS_1 = \begin{pmatrix} 5 & 6 \\ 1 & 3 \end{pmatrix}, \quad IS_2 = \begin{pmatrix} 7 & 8 \\ 5 & 7 \end{pmatrix}, \quad IS_3 = \begin{pmatrix} 9 \\ 9 \end{pmatrix},$$

$$IS_4 = \begin{pmatrix} 10 & 11 \\ 13 & 16 \end{pmatrix}, \quad IS_5 = \begin{pmatrix} 12 & 13 \\ 19 & 20 \end{pmatrix}.$$

Similarly, partition the decreasing part into consecutive segments $DS_1, DS_2, \ldots, DS_q$ so that DS_m (which may be null) contains the outputs in $\{(m-1)p+1, \ldots, mp\}$. For instance, in Example 2.5.2, one has

$$DS_1 = \begin{pmatrix} 3 & 4 \\ 4 & 2 \end{pmatrix}, \quad DS_2 = \begin{pmatrix} 1 & 2 \\ 8 & 6 \end{pmatrix}, \quad DS_3 = \begin{pmatrix} 18 & 19 & 20 \\ 12 & 11 & 10 \end{pmatrix},$$

$$DS_4 = \begin{pmatrix} 16 & 17 \\ 15 & 14 \end{pmatrix}, \quad DS_5 = \begin{pmatrix} 14 & 15 \\ 18 & 17 \end{pmatrix}.$$

(i) Show that for all $m = 1, 2, \ldots, q$, the set $IS_m \cup IS_{m-1} \cup \ldots \cup IS_1 \cup DS_1 \cup \ldots \cup DS_{m-1} \cup DS_m$ is a cyclically consecutive segment on which the circular unimodal mapping π induces a circular unimodal mapping.

(ii) Show that the set $IS_m \cup IS_{m-1} \cup \ldots \cup IS_1 \cup DS_1 \cup \ldots \cup DS_{m-1} \cup DS_m$ represents m inputs on every switch at the first stage.

(iii) Show by induction on m that the set $IS_m \cup DS_m$ represents exactly one input of every switch at the first stage. This shows that there is no conflicting path inside the two-stage construction.

(iv) Note that a subset of the circular unimodal permutation π is also a circular unimodal mapping. Use this property to show that the connection pattern for every switch at the first stage is a circular unimodal permutation.

(v) Use the property derived in (i) to show (by induction on m) that the connection pattern for every switch at the second stage is a circular unimodal permutation.

27. An $N \times N$ switch is called a *compressor* if it realizes the set of subpermutations that satisfy the following two conditions: (i) there exists a circular shift such that the active outputs are consecutive, and (ii) the mapping between active inputs and outputs is monotonically increasing after the circular shift. Show that an $N \times N$ UC nonblocking switch is also a compressor.

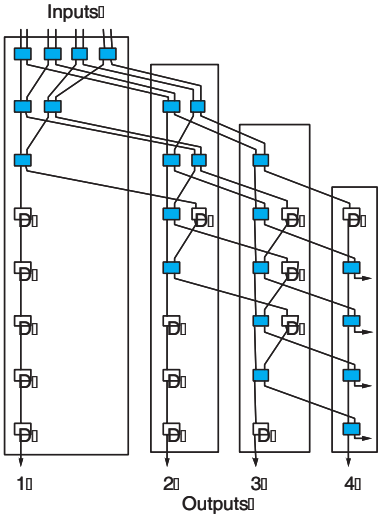


Fig. 2.51. The 8-to-4 Knockout concentrator/sorter

28. (Compressor theorem ([111], pp. 104)) Continue from the previous problem. Consider the two-stage construction in Figure 2.35, i.e., the 2X construction. If all the $p \times p$ switches at the first stage and

the $q \times q$ switches at the second stage are compressors, then the constructed $N \times N$ switch is also a compressor.

29. Continue from the previous problem. Show that the L-to-1 multiplexer in the Knockout switch in Figure 2.43. is indeed a $2X$ construction of a compressor ([111], pp. 105). (Hint: the $L \times L$ shifter is a compressor in space and the L parallel buffers served in a round robin fashion is a compressor in time.)
30. Continue from the previous problem. Show that the $L \times L$ shifter can be implemented by $\log_2 L$ stages of $L/2$ 2×2 switches. (Hint: use the 2X construction recursively for the compressor.)
31. Consider the L-to-1 multiplexer in the Knockout switch (see Figure 2.43). Suppose that $L = 5$ and the switch is empty at time 0. Furthermore, there are 3 arrivals at time 1, 4 arrivals at time 2, 2 arrivals at time 3, ... Find the connection pattern of the shifter at time 3. Note that the shifter is a 5×5 switch.
32. Design an 8-to-4 Knockout concentrator/sorter without using the fast Knockout stages (see Figure 2.51).

3. Load Balanced Birkhoff-von Neumann switches

As described in Chapter 2, the problem of shared memory switches (and shared medium switches) is the limitation of the memory access speed. To gain the needed speedup, input-buffered switches use parallel buffers at the inputs. As such, they are known to be more scalable than shared memory switches. However, synchronized parallel transmissions among parallel input buffers in every time slot require careful coordination to avoid conflicts. Thus, finding a scalable method (and architecture) for conflict resolution becomes the fundamental design problem of input-buffered switches.

Traditionally, conflict resolution is solved by finding a matching between inputs and outputs per time slot (see e.g., [92, 6, 155, 121, 123, 124, 56, 113]). Two steps are needed for finding a matching.

(i) Communication overhead: one has to gather the information of the buffers at the inputs.
(ii) Computation overhead: based on the gathered information, one then applies a certain algorithm to find a matching.

Most of the works in the literature pay more attention to reducing the computation overhead by finding scalable matching algorithms, e.g., wavefront arbitration in [155], PIM in [6], SLIP in [121], and DRRM in [113]. However, in our view, it is the communication overhead that makes matching per time slot difficult to scale. To see this, suppose that there are N inputs/outputs and each input implements N virtual output queues (VOQ). If we use a single bit to indicate whether a VOQ is empty, then we have to transmit N bits from each input (to a central arbiter or to an output) in every time slot. For instance, transmitting such N bit information in PIM and SLIP is implemented by an independent circuit that sends out parallel requests. Suppose that the packet size is chosen to be 64 bytes. Then building a switch with more than 512 inputs/outputs will have more communication overhead than transmitting the data itself.

To reduce the communication overhead, one approach is to gather the long term statistics of the VOQs, e.g., the average arrival rates, and then use such information to find a sequence of pre-determined connection patterns (see e.g., [6, 108, 79, 28, 29, 7]). Most of the works along this line are based on the well-known Birkhoff-von Neumann algorithm [17, 164] that decomposes a doubly substochastic matrix into a convex combination of (sub)permutation matrices. For an $N \times N$ switch, the computation complexity for the Birkhoff-von Neumann decomposition is $O(N^{4.5})$ and the number of permutation matrices produced by the decomposition is $O(N^2)$ (see e.g., [28, 29]). The need for storing the $O(N^2)$ number of permutation matrices in the Birkhoff-von Neumann switch makes it difficult to scale for a large N. Even though there are decomposition methods that reduce the number of permutation matrices (see e.g., [100]), they in general do not have good throughput. For instance, the throughput in [100] is $O(1/\log N)$ and it tends to 0 when N is large. Another problem of using long term statistics is that the switch does not adapt too well to traffic fluctuation.

It would be ideal if there is a switch architecture that yields good throughput without the need for gathering traffic information (no communication overhead) and computing connection patterns (no computation overhead). In this chapter, we will introduce some recent works on the load balanced Birkhoff-von Neumann switches (see e.g., [34, 35, 98, 40, 36]) that shed some light along this direction.

3.1 Load balanced Birkhoff-von Neumann switches: one-stage buffering

As described in Section 2.3, the Birkhoff-von Neumann switch requires the information of the arrival rate of each input-output pair in order to achieve 100% throughput. Another problem of such a switch is that 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. To cope with these problems, in this section we introduce the switch architecture proposed in [34].

The main idea in [34] is that the Birkhoff-von Neumann decomposition is easy if the incoming traffic is *uniform*.

Why don't we add something in front of the Birkhoff-von Neumann switch so that the traffic coming to the Birkhoff-von Neumann switch is uniform?

The switch architecture in [34] consists of two-stage switching fabrics and one-stage buffering. The first stage performs load balancing, while the second stage is a Birkhoff-von Neumann input-buffered switch that performs switching for load balanced traffic. Such a switch is called the load balanced Birkhoff-von Neumann switch in [34]. The switch has the following advantages:

(i) Scalability: the on-line complexity of the scheduling algorithm in the switch is $O(1)$.
(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.
(iii) 100% throughput: under a mild technical condition on the input traffic, the load balanced Birkhoff-von Neumann switch 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 load balanced Birkhoff-von Neumann switch is proven to converge to that of an output-buffered switch under heavy load. Also, by simulations, it is shown that load balancing is more effective than the conflict resolution algorithm, SLIP, in heavy load.
(v) Efficient buffer usage: when both the load balanced Birkhoff-von Neumann 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 load balanced Birkhoff-von Neumann switch is much smaller than that in an output-buffered switch when the buffer is large.

The main drawback of the switch is that FIFO might be violated for packets from the same input. One quick fix is to add a resequencing buffer at the output port. However, this increases the complexity of the hardware design. How to perform resequencing efficiently will be addressed in Section 3.3.

3.1.1 The switch architecture

The load balanced Birkhoff-von Neumann switch consists of two stages (see Figure 3.1). The first stage performs load balancing and the second stage performs switching. The second stage is a *framed* Birkhoff-von Neumann switch with the frame size being equal to the number of input/output ports. To be precise, suppose that the number of input/output ports is N. As the frame size is N, there are N permutation matrices, $P_1, P_2, \ldots, P_N$, that need to be defined. For this, we choose any N permutation matrices such that

$$P_1 + P_2 + \ldots + P_N = \underline{\mathbf{e}}, \tag{3.1}$$

where $\underline{\mathbf{e}}$ is an $N \times N$ matrix with all its elements being 1. By so doing, each input-output pair is assigned a time slot in every frame. Thus, the allocated rate for each input-output pair is exactly $\frac{1}{N}$. This implies that 100% throughput can be achieved if the input traffic to the second stage is *uniform*, which is exactly what we would like to do at the first stage.

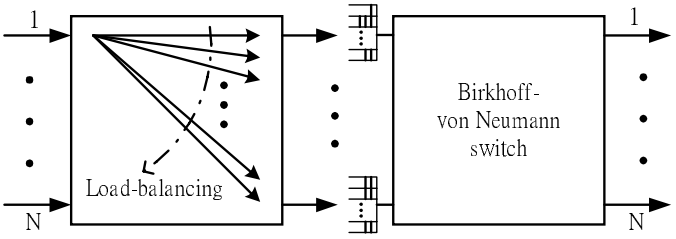


Fig. 3.1. The switch architecture

One way to find N permutation matrices that satisfy (3.1) is to use a one-cycle permutation matrix. An $N \times N$ permutation matrix P is called an one-cycle permutation matrix if for all i, N is the smallest integer such that $P_{i,i}^N = 1$. Note that for a permutation matrix P, there is a corresponding permutation $\pi : \{1, 2, \ldots, N\} \mapsto \{1, 2, \ldots, N\}$ with $\pi(i) = j$ if $P_{i,j} = 1$. Define $\pi^{(2)}$ to be the composite mapping $\pi \circ \pi$ and $\pi^{(k)} = \pi^{(k-1)} \circ \pi$, $k > 1$. A permutation matrix P is a one-cycle permutation matrix if the corresponding permutation is also a one-cycle permutation, i.e., N is the smallest integer such that $\pi^{(N)}(i) = i$ for all i.

For a one-cycle permutation matrix P, we may define $P_k = P^k$, $k = 1, 2, \ldots, N$ and the N matrices generated this way satisfy (3.1).

Example 3.1.1. (One-cycle permutation matrix) A typical one-cycle permutation matrix is the circular-shift matrix with $P_{i,j} = 1$ when $j = (i \mod N) + 1$, and $P_{i,j} = 0$ otherwise. If we use the circular-shift matrix in the second stage of a 4×4 switch, we then have

$$P_1 = P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad P_2 = P^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

$$P_3 = P^3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad P_4 = P^4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Clearly, we have

$$P_1 + P_2 + P_3 + P_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Note that the corresponding permutation for P is

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}.$$

Example 3.1.2. (Two-cycle permutation matrix) To provide more intuition on the one-cycle permutation matrix, consider the following permutation matrix

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

The corresponding permutation for P is

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 4 & 5 & 3 \end{pmatrix}.$$

It is clear that π can be decomposed as the following two cycles

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \quad \begin{pmatrix} 3 & 4 & 5 \\ 4 & 5 & 3 \end{pmatrix}.$$

Example 3.1.3. (Symmetric Time Division Multiplexing (TDM) switch) In this example, we introduce another method to generate N permutation matrices that satisfy (3.1). For $k = 1, \ldots, N$, let P_k be the permutation matrix with its $(i, j)^{th}$ element being 1 if

$$(i + j) \bmod N = (k + 1) \bmod N$$

and 0 otherwise. For instance, for $N = 4$, we have

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$P_3 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad P_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Clearly, we still have

$$P_1 + P_2 + P_3 + P_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Note that all the N permutation matrices generated this way are *symmetric*, i.e., the inverse permutations are themselves! Switch fabrics that use these N permutation matrices are called symmetric time division multiplexing switches and they will be addressed further in Section 3.2.2.

Now we show how load balancing is performed at the first stage. The first stage is an unbuffered crossbar switch. Packets arriving at the first stage at time t are switched instantly to the second stage according to the connection pattern set up at the crossbar switch. To

be precise, let $\underline{a}(t) = (a_{i,j}(t))$ be the $N \times N$ traffic matrix at time t, where $a_{i,j}(t)$ is the number of packet arriving at the i^{th} input port and destined to the j^{th} output port at time t. As there is at most one packet arriving at an input port per time slot, $a_{i,j}(t)$'s are indicator variables. Also, let $P_1(t)$ be the $N \times N$ permutation matrix assigned at time t at the first stage and $\underline{b}(t) = (b_{i,j}(t))$ be the $N \times N$ traffic matrix entering the second stage, where $b_{i,j}(t)$ is the number of packet arriving at the i^{th} input port of the second stage and destined to the j^{th} output port at time t. Then we have

$$\underline{b}(t) = P_1^T(t)\underline{a}(t), \tag{3.2}$$

where $P_1^T(t)$ is the transpose of $P_1(t)$.

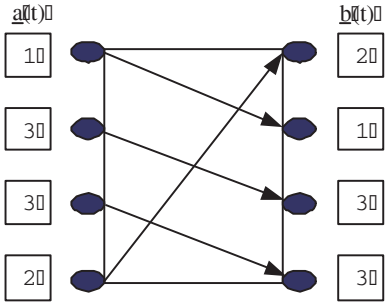


Fig. 3.2. Load balancing at the first stage

Example 3.1.4. As shown in Figure 3.2, suppose that at time t there is a packet arriving at input 1 and the packet is destined for output 1. At the same time, a packet destined for output 3 arrives at input 2, a packet destined for output 3 arrives at input 3, and a packet destined for output 2 arrives at input 4. Thus,

$$\underline{a}(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

The connection pattern at the first switch is the circular-shift matrix P in Example 3.1.1, i.e.,

$$P(t) = P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

As shown in Figure 3.2, a packet destined for output 2 (resp. 1,3,3) arriving input 1 (resp. 2,3,4) of the second stage. Thus,

$$\underline{b}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

It is easy to verify that (3.2) indeed holds.

To perform load balancing, we simply set up the permutation matrices $P_1(t)$ periodically as what we do in the second stage. Thus, the connection patterns at the first stage also have a framing structure with frame size N, and the N permutation matrices in each frame satisfy (3.1). Also, the connection patterns at the first stage are independent of the traffic coming to the switch.

Let $R = (r_{i,j})$ be the rate matrix of $\underline{a}(t)$. From the no overbooking conditions, we know that R is a doubly substochastic matrix, i.e., all the row sums and column sums of R are less than or equal to 1. As the N permutation matrices in each frame satisfy (3.1), we also have

$$\sum_{s=1}^{N} P_1^T(s) = \underline{\mathbf{e}}.$$

Thus, the long run average of $P_1^T(t)$ is the same as its mean and it is equal to $\frac{1}{N}\underline{\mathbf{e}}$, i.e.,

$$\lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} P_1^T(s) = \frac{1}{N} \sum_{s=1}^{N} P_1^T(s) = \mathsf{E} P_1^T(t) = \frac{1}{N}\underline{\mathbf{e}}, \quad a.s. \qquad (3.3)$$

Since $P_1(t)$ is independent of the arrivals,

$$\mathsf{E}[\underline{b}(t)] = \mathsf{E}[P_1^T(t)\underline{a}(t)] = \mathsf{E}[P_1^T(t)]\mathsf{E}[\underline{a}(t)] = \frac{1}{N}\underline{\mathbf{e}}\ R.$$

As R is a doubly substochastic matrix, we then have

$$\mathsf{E}\underline{b}(t) \leq \frac{1}{N}\underline{\mathbf{e}}. \tag{3.4}$$

Thus, the traffic entering the second stage is "uniform" in the sense that the rate between any input-output pair at the second stage is bounded above by $1/N$. As the allocated rate for any input-output pair at the second stage is exactly $1/N$, we then guarantee to have 100% throughput for the load balanced Birkhoff-von Neumann switch under the no overbooking conditions (the formal proof can be found in [34]).

It is worth pointing out that the inequality in (3.4) only requires the column sums of the rate matrix R are not greater than 1. This implies that the result for 100% throughput also holds for multicast traffic if fan-out splitting (packet duplication) is done at the buffer between the two stages. Further discussions along this line will be addressed in details in Section 3.3.

One of the main advantages of the two-stage load balanced Birkhoff-von Neumann switch is the reduction of complexity. In comparison with the original Birkhoff-von Neumann input-buffered switch, there is no need for rate estimation in the load balanced Birkhoff-von Neumann switch. As a result, there is also no need to perform the Birkhoff-von Neumann capacity decomposition. For the number of permutation matrices needed in the switch, the complexity is reduced from $O(N^2)$ to $O(N)$. Also, the on-line computational complexity for the scheduling algorithm is reduced from $O(\log N)$ to $O(1)$ as the scheduling policy is now simply periodic. With all the reduction of complexity, the load balanced Birkhoff-von Neumann switch still has good performance, including 100% throughput.

3.1.2 Ergodicity

We note that there is a catch in the intuitive argument in the previous section for the result of 100% throughput. The sequence $\{\underline{b}(t), t \geq 1\}$ may not be ergodic at all. Before we explain this, we first review the concept of ergodicity in the following example.

Example 3.1.5. (Permutation and ergodicity) As described in Section 2.1.2, a stochastic sequence $\{X(n), n \geq 1\}$ is *stationary* if its joint distribution is invariant with respect to time shift, i.e., for any time shift m and any k-dimensional joint distribution,

$$\begin{aligned}&\mathsf{P}(X(1) \le x_1, X(2) \le x_2, \ldots, X(k) \le x_k)\\ &= \mathsf{P}((X(1+m) \le x_1, X(2+m) \le x_2, \ldots, X(k+m) \le x_k).\end{aligned} \tag{3.5}$$

A stationary sequence $\{X(n), n \ge 1\}$ is *ergodic* if its long run average is the same as its ensemble average, i.e., for any function f with k variables

$$\begin{aligned}&\lim_{n\to\infty} \frac{1}{n} \sum_{m=0}^{n-1} f\Big(X(1+m), X(2+m), \ldots X(k+m)\Big)\\ &= \mathsf{E}\Big[f\Big(X(1), X(2), \ldots, X(k)\Big)\Big], \quad a.s.\end{aligned} \tag{3.6}$$

Consider an $N \times N$ permutation matrix P and its associated permutation π. Recall that $\pi(i) = j$ if $P_{i,j} = 1$. Let $X(1)$ be a uniform random variable over $\{1, 2, \ldots, N\}$. Define the stochastic sequence $\{X(n), n \ge 1\}$ recursively as follows:

$$X(n+1) = \pi(X(n)). \tag{3.7}$$

Let $\mu(n)$ be the N-vector that represents the probability mass function of $X(n)$. As $X(1)$ is uniform, we have for all $i = 1, 2, \ldots, N$,

$$\mathsf{P}(X(1) = i) = \frac{1}{N}.$$

Thus,

$$\mu(1) = (\frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N}). \tag{3.8}$$

In view of the recursive construction in (3.7), it is easy to see that

$$\mu(n+1) = \mu(n)P. \tag{3.9}$$

From (3.8), it then follows that for all n,

$$\mu(n) = (\frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N}), \tag{3.10}$$

and the stochastic sequence $\{X(n), n \ge 1\}$ is stationary.

If P is a one-cycle permutation matrix, it is easy to see that

$$\begin{aligned}&\lim_{n\to\infty} \frac{1}{n} \sum_{m=1}^{n} X(m) = \frac{1}{N} \sum_{m=1}^{N} X(m)\\ &= \frac{1}{N} \sum_{m=1}^{N} m = \frac{N+1}{2} = \mathsf{E}[X(1)].\end{aligned}$$

One can further verify that $\{X(n), n \geq 1\}$ is ergodic if P is a one-cycle permutation matrix.

On the other hand, suppose that P is the two-cycle permutation matrix in Example 3.1.2. Then

$$\lim_{n\to\infty} \frac{1}{n} \sum_{m=1}^{n} X(m) = \begin{cases} \frac{3}{2}, & \text{if } X(1) = 1 \text{ or } 2 \\ 4, & \text{if } X(1) = 3, 4, \text{ or } 5 \end{cases}.$$

The stochastic sequence $\{X(n), n \geq 1\}$ is not ergodic and it can be decomposed as two ergodic sequences.

Readers who are familiar with Markov chain may realize that the stochastic sequence $\{X(n), n \geq 1\}$ generated this way is in fact a Markov chain with the probability transition matrix P (see [129]). A Markov chain is ergodic if it is irreducible, i.e., it cannot be decomposed as two or more separate Markov chains.

To ensure that the sequence $\{\underline{b}(t), t \geq 1\}$ is ergodic, it is shown in [34] that one needs the sequence $\{\underline{a}(t), t \geq 1\}$ to be weakly mixing, a stronger technical condition than ergodicity. Both weak mixing and ergodicity are basically measures of how fast a stochastic sequence loses memory (see e.g., Petersen [137] and Nadkarni [127] for more details). In the following example, we show that if the weakly mixing condition is not satisfied, then 100% throughput cannot be guaranteed.

Example 3.1.6. (Weak mixing and ergodicity) To see the reason that we need the weak mixing condition, consider the case with $N = 2$. In this case, the only one-cycle permutation is

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Consider the periodic sequence $P(t) = P^{t'}$ with $t' = t \mod 2$. Let $P_1(t) = P(t + U_1)$ and $\underline{a}(t) = z(t)P(t + \tilde{U}_1)$, where U_1 and $\tilde{U}_1$ are two independent Bernoulli random variables with $\mathsf{P}(U_1 = 0) = \mathsf{P}(U_1 = 1) = \mathsf{P}(\tilde{U}_1 = 0) = \mathsf{P}(\tilde{U}_1 = 1) = 1/2$, and $\{z(t), t \geq 1\}$ is a sequence of i.i.d. Bernoulli random variables with $\mathsf{P}(z(1) = 1) = 0.9$ and $\mathsf{P}(z(1) = 0) = 0.1$. Thus, we have the doubly substochastic rate matrix

$$R = \mathsf{E}[\underline{a}(t)] = \mathsf{E}[z(t)]\mathsf{E}[P(t + \tilde{U}_1)] = \begin{bmatrix} 0.45 & 0.45 \\ 0.45 & 0.45 \end{bmatrix},$$

and the no overbooking conditions are satisfied.

Though both $\{P_1(t), t \geq 1\}$ and $\{\underline{a}(t), t \geq 1\}$ constructed this way are ergodic, the process $\{\underline{b}(t) = P_1^T(t)\underline{a}(t), t \geq 1\}$ is not *ergodic*. This can be easily seen from the fact that it can be decomposed as two ergodic sequences. When $U_1 \neq \tilde{U}_1$, we have $\underline{b}(t) = z(t)P$ for all t. On the other hand, when $U_1 = \tilde{U}_1$, we have $\underline{b}(t) = z(t)P^2$ for all t (note that P^2 is simply the identity matrix). For the case that $\underline{b}(t) = z(t)P$, the rate matrix entering the second stage is

$$\begin{bmatrix} 0 & 0.9 \\ 0.9 & 0 \end{bmatrix}.$$

For the other case, the rate matrix entering the second stage is

$$\begin{bmatrix} 0.9 & 0 \\ 0 & 0.9 \end{bmatrix}.$$

In either case, the buffer at each input port of the second stage goes to infinity as $t \to \infty$ when $\{\underline{b}(t), t \geq 1\}$ is fed into second stage.

The intuition behind this example is that load balancing at the first stage cannot be achieved if the input traffic is synchronized with the scheduled permutation matrices. To fix this problem, one can use a randomization approach [161, 126] and choose $P_1(t)$ randomly from $P_1, P_2, \ldots, P_N$ for every t. However, as pointed out in [122], it is not easy to implement *randomness* in high speed. Moreover, randomization increases variance and hence results in poorer performance than a deterministic round-robin approach (see e.g., [154]).

3.1.3 Uniform i.i.d. traffic model

To carry out the analysis for more specific performance measures, such as average queue length and average delay, we need to have a more specific model for the input. In this section, we consider a uniform i.i.d. traffic model. With probability ρ, a packet arrives at each input (of the first stage) for every time slot. This is independent of everything else. The destination of an arriving packet is chosen uniformly among the N output ports. This is also independent of everything else. Based on this traffic model, we make the following two observations:

(i) Load balancing at the first stage has no effect at all (as the traffic is already balanced). To be precise, $\{\underline{b}(t), t \geq 1\}$ has the same joint distribution as $\{\underline{a}(t), t \geq 1\}$. Moreover, $\{b_{i,j}(t), t \geq 1\}$ and $\{a_{i,j}(t), t \geq 1\}$ for all i and j are sequences of i.i.d. Bernoulli random variables with mean $\frac{\rho}{N}$.

(ii) Let $q_{i,j}(t)$ be the number of packets in the j^{th} VOQ of the i^{th} input of the second stage at time t. As the traffic is uniform, $q_{i,j}(t)$'s are all identically distributed.

Without loss of generality, let us look at the recursive equation for $q_{1,1}(t)$. Note from (i) that the arrival sequence to $q_{1,1}$ is simply a sequence of i.i.d. Bernoulli random variables with mean $\frac{\rho}{N}$. Let T be a time that input 1 of the second stage is connected to output 1. As such a connection happens every N time slots, $q_{1,1}$ is not served during $[T+1, T+N-1]$. We then have

$$q_{1,1}(T+s) = q_{1,1}(T) + \sum_{k=1}^{s} b_{1,1}(T+k), \quad s = 1, \ldots, N-1, \tag{3.11}$$

$$q_{1,1}(T+N) = (q_{1,1}(T) + \sum_{k=1}^{N} b_{1,1}(T+k) - 1)^+. \tag{3.12}$$

Note that (3.12) has the same form as the Lindley recursion in (2.1). As $b_{1,1}(t)$'s are i.i.d. Bernoulli random variables with mean ρ/N, one can easily compute that

$$\mathsf{E}[\sum_{k=1}^{N} b_{1,1}(T+k)] = \rho, \tag{3.13}$$

$$\mathrm{Var}[\sum_{k=1}^{N} b_{1,1}(T+k)] = \rho - \frac{\rho^2}{N}. \tag{3.14}$$

In the steady state, we have from Proposition 2.1.5 (with σ^2 in (3.14)) that

$$\mathsf{E}q_{1,1}(T) = \frac{N-1}{N} \frac{\rho^2}{2(1-\rho)}. \tag{3.15}$$

From (3.11), it follows that

$$\mathsf{E}q_{1,1}(T+s) = \mathsf{E}q_{1,1}(T) + s\frac{\rho}{N}, \quad s = 1, \ldots, N-1. \tag{3.16}$$

This then implies that in the steady state

$$\mathsf{E}q_{1,1}(\infty) = \frac{1}{N}\sum_{s=0}^{N-1}\mathsf{E}q_{1,1}(T+s) = \frac{N-1}{N}\frac{\rho}{2(1-\rho)}. \tag{3.17}$$

Let $\bar{d_1}$ be the average delay for a packet. As the average arrival rate to $q_{1,1}$ is $\frac{\rho}{N}$, we have from Little's formula that

$$\bar{d_1} = \frac{\mathsf{E}q_{1,1}(\infty)}{\frac{\rho}{N}} = \frac{N-1}{2(1-\rho)}. \tag{3.18}$$

To compare the performance with the corresponding output-buffered switch, let $q_1^o(t)$ be the number of packets at the buffer of the first output port at time t. As in (2.1), the corresponding Lindley equation is

$$q_1^o(t+1) = (q_1^o(t) + \sum_{i=1}^{N} a_{i,1}(t+1) - 1)^+. \tag{3.19}$$

Note that (3.19) and (3.12) are stochastically identical as both $\{b_{i,j}(t), t \geq 1\}$ and $\{a_{i,j}(t), t \geq 1\}$ are sequences of i.i.d. Bernoulli random variables with mean $\frac{\rho}{N}$. This then implies that

$$\mathsf{E}q_1^o(\infty) = \mathsf{E}q_{1,1}(T) = \frac{N-1}{N}\frac{\rho^2}{2(1-\rho)}.$$

Let $\bar{d_1^o}$ be the average delay for a packet in the corresponding output-buffered switch. As the arrival rate to an output port in the output-buffered switch is ρ, once again we have from Little's formula that

$$\bar{d_1^o} = \frac{\mathsf{E}q_1^o(\infty)}{\rho} = \frac{N-1}{N}\frac{\rho}{2(1-\rho)}. \tag{3.20}$$

This shows that

$$\frac{\bar{d_1^o}}{\bar{d_1}} = \frac{\rho}{N}, \tag{3.21}$$

and the performance of the load balanced Birkhoff-von Neumann switch is poor compared with that of an output-buffered switch. This is not surprising as load balancing has no effect at all for this traffic model.

3.1.4 Uniform bursty traffic model

In this section, we consider the following uniform bursty traffic model. Packets come as a burst of length N, which is exactly the same as

the number of input/output ports. Packets within the same burst are destined to the same output. For every N time slots, the probability that there is a burst arriving at a particular input port (of the first stage) is ρ, and the probability that there are no packet arrivals in these N slots is $1-\rho$. This is independent of everything else. The destination of the N packets within that burst is chosen uniformly among the N output ports. This is also independent of everything else. Based on this traffic model, we also make the following two observations:

(i) In contrast to the uniform i.i.d. traffic model in the previous section, load balancing achieves perfect burst reduction in this model (see Figure 3.3). The N packets within a burst are distributed *evenly* to the input ports at the second stage. In this model, $\{b_{i,j}(t), t \geq 1\}$ for all i and j are still sequences of i.i.d. Bernoulli random variables with mean $\frac{\rho}{N}$.
(ii) As the traffic is uniform, $q_{i,j}(t)$'s are still identically distributed.

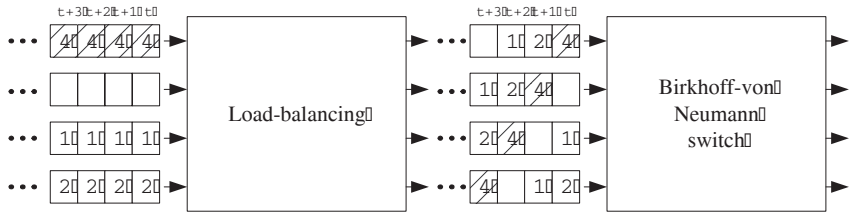


Fig. 3.3. Burst reduction in the uniform bursty traffic model

Without loss of generality, let us also look at the recursive equation for $q_{1,1}(t)$. As the arrival sequence to $q_{1,1}$ is still a sequence of i.i.d. Bernoulli random variables with mean $\frac{\rho}{N}$, the whole analysis is the same as that in the uniform i.i.d. traffic model, and we conclude that the average delay for a packet in this model, denoted by $\bar{d}_2$, is the same as that in the uniform i.i.d. traffic model, i.e.,

$$\bar{d}_2 = \frac{N-1}{2(1-\rho)}. \tag{3.22}$$

Now we do the performance analysis for the corresponding output-buffered switch. As packets come as a burst of length N, we have $\underline{a}(Nt+1) = \underline{a}(Nt+2) = \ldots = \underline{a}(Nt+N)$ for all t. Note from the Lindley recursion in (3.19) that for $s = 1, \ldots, N$,

$$q_1^o(Nt+s) = \max[q_1^o(Nt) + s\sum_{i=1}^{N} a_{i,1}(Nt+1) - s, 0]. \tag{3.23}$$

In particular, for $s = N$, we have

$$\frac{q_1^o(N(t+1))}{N} = \max[\frac{q_1^o(Nt)}{N} + \sum_{i=1}^{N} a_{i,1}(Nt+1) - 1, 0]. \tag{3.24}$$

This recursion is stochastically identical to that in (3.12). It then follows from (3.15) that (in the steady state)

$$\mathsf{E}q_1^o(Nt) = \frac{(N-1)\rho^2}{2(1-\rho)}. \tag{3.25}$$

Now we show that $\mathsf{E}q_1^o(Nt+s) = \mathsf{E}q_1^o(Nt)$ for $s = 1, \ldots, N-1$. To simplify the notation, let

$$Z = \sum_{i=1}^{N} a_{i,1}(Nt+1).$$

As packets come as a burst of length N, the random variable $q_1^o(Nt)$ only takes values on integer multiples of N. This implies that for $s = 1, \ldots, N$, $q_1^o(Nt+s) = q_1^o(Nt) + sZ - s$ if $q_1^o(Nt) > 0$ or $Z > 0$, and $q_1^o(Nt+s) = 0$ otherwise. Let $\mathbf{1}\{\mathcal{E}\}$ be the indicator random variable for an event $\mathcal{E}$. Then we can rewrite this as follows:

$$q_1^o(Nt+s) = (q_1^o(Nt) + sZ - s)(1 - \mathbf{1}\{q_1^o(Nt) = 0, Z = 0\}). \tag{3.26}$$

Taking expectations on both sides of (3.26) yields

$$\mathsf{E}q_1^o(Nt+s) = \mathsf{E}q_1^o(Nt) + s\mathsf{E}Z - s + s\mathsf{P}(q_1^o(Nt) = 0, Z = 0), \tag{3.27}$$

where we use the identity

$$\mathsf{E}[(q_1^o(Nt) + sZ)\mathbf{1}\{q_1^o(Nt) = 0, Z = 0\}] = 0.$$

When $s = N$, we have from $\mathsf{E}q_1^o(Nt+N) = \mathsf{E}q^o(Nt)$ and (3.27) that

$$\mathsf{E}Z = 1 - \mathsf{P}(q_1^o(Nt) = 0, Z = 0). \tag{3.28}$$

Replacing (3.28) in (3.27) yields $\mathsf{E}q_1^o(Nt+s) = \mathsf{E}q_1^o(Nt)$ for all $s = 1, \ldots, N-1$. This then implies that in the steady state

$$\mathsf{E}q_1^o(\infty) = \mathsf{E}q_1^o(Nt) = \frac{(N-1)\rho^2}{2(1-\rho)}. \tag{3.29}$$

Let $\bar{d}_2^o$ be the average delay for a packet in the corresponding output-buffered switch. Once again, we have from Little's formula that

$$\bar{d}_2^o = \frac{(N-1)\rho}{2(1-\rho)}. \tag{3.30}$$

For this traffic model, we have that

$$\frac{\bar{d}_2^o}{\bar{d}_2} = \rho. \tag{3.31}$$

This shows that the delay in this traffic model converges to that of an output-buffered switch when $\rho \to 1$.

The results for the average delay of these two traffic models are summarized in Table 3.1.

Delay	Output-buffered	Load balanced
I.i.d.	$\frac{N-1}{N}\frac{\rho}{2(1-\rho)}$	$\frac{(N-1)}{2(1-\rho)}$
Bursty	$\frac{(N-1)\rho}{2(1-\rho)}$	$\frac{N-1}{2(1-\rho)}$

Table 3.1. Average delay for output-buffered switches and load balanced Birkhoff-von Neumann switches

3.1.5 Simulation

In this section, we present various simulation results in [34] to support the observations and conclusions derived in the previous section. Since the real traffic is bursty (see e.g., [109]), it is reasonable to perform the simulations on bursty traffic. In all the simulations, the switch size is 16×16, i.e., $N = 16$. The uniform bursty traffic model in Section 3.1.4 is used in the first experiment. The simulation results for the average delay are shown in Figure 3.4 for the load balanced Birkhoff-von Neumann switch, the output-buffered switch, the Birkhoff-von Neumann switch, and the 4-SLIP, respectively. In the simulations for the Birkhoff-von Neumann switch, it is assumed that the arrival rates are known and no dynamic rate estimation and adjustment is performed. These simulation results are obtained with 99% confidence intervals. As expected, the simulation results of the output-buffered switch and the load balanced Birkhoff-von Neumann switch match perfectly with the theoretical results in (3.22) and (3.30). In comparison with the original Birkhoff-von Neumann switch, load balancing is very effective in reducing the average delay. In light load ($\rho \leq 0.4$), conflict resolution is very effective and the 4-SLIP performs much better than the load

balanced Birkhoff-von Neumann switches. However, as load increases, load balancing is much more effective than conflict resolution. From the simulations, the load balanced Birkhoff-von Neumann switches performs much better than the 4-SLIP in heavy load ($\rho \geq 0.7$). To verify this observation, the simulations in the second experiment are run with random burst length instead. As in the uniform bursty traffic model, packets come as a burst. However, the burst lengths are chosen independently according to the following (truncated) Pareto distribution:

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

where $c = (\sum_{i=1}^{10000} 1/i^{2.5})^{-1}$ is the normalization constant. In this experiment, the average burst length is 1.932, which is considerably smaller than 16, the fixed burst length in the first experiment. However, we still see the same effect in Figure 3.5. The intuition behind this is that the dominating effect on the average delay is the heavy tail of the burst length distribution (see e.g., [48, 89]). For a large burst, load balancing is quite effective in burst reduction and thus yields better performance. On the other hand, 4-SLIP does not perform well when the traffic is heavy and bursty since SLIP might get trapped in "bad modes" as described in Example 2.2.4.

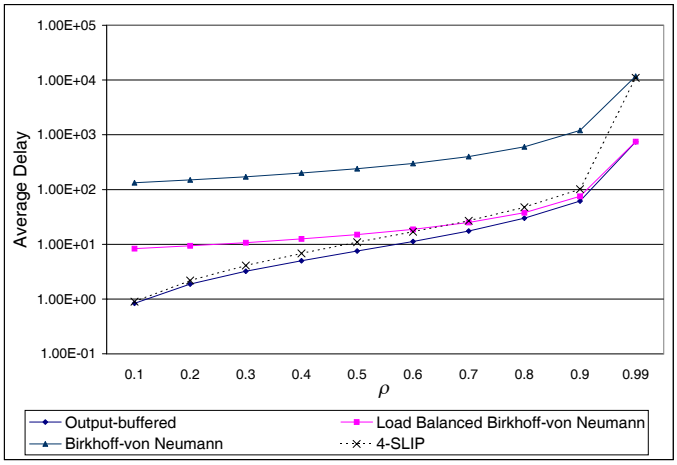


Fig. 3.4. Average delay under the uniform bursty traffic model

In the third experiment, we address the effect of buffer usage. The same amount of buffer is allocated to each port in the load balanced

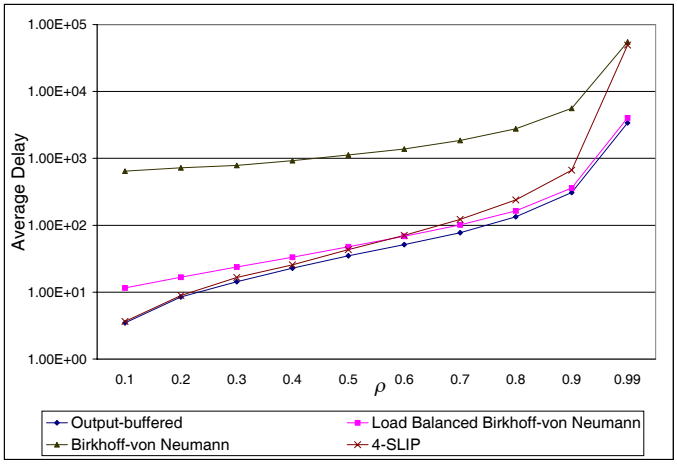


Fig. 3.5. Average delay under the uniform Pareto traffic model

Birkhoff-von Neumann switch and the output-buffered switch (without sharing with other ports). The simulations are run under the uniform bursty traffic model with the arrival rate $\rho = 0.8$ in both switches. The simulation results for packet loss probabilities are shown in Figure 3.6. This experiment shows that the load balanced Birkhoff-von Neumann switch has a much smaller packet loss probability than that in the corresponding output-buffered switch when the buffer is large. Further theoretical justification based on the theory of effective bandwidth can be found in [34].

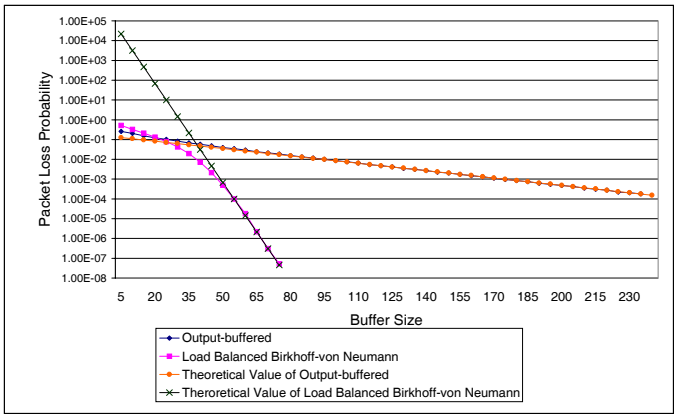


Fig. 3.6. Packet lost probability under uniform bursty traffic (switch size $N = 16$, arrival rate $\rho = 0.8$)

3.1.6 Further reduction of the requirement of the memory speed

The memories in the $N \times N$ load balanced Birkhoff-von Neumann switch need to perform a write operation (from an input port) and a read operation (to an output port) per time slot. Thus, the memory speed is only needed to be twice as large as the line speed. In this section, we introduce a refined architecture for the $N \times N$ load balanced Birkhoff-von Neumann switch such that the memory speed is $F/2$ times slower than the line speed. For this, we first consider an $FN \times FN$ load balanced Birkhoff-von Neumann switch in Figure 3.7. Among the FN input/output ports, only the first N input/output ports are active. The rest of input/output ports are idle, i.e., no packets are transmitted for the input/output ports from Port $N + 1$ to Port FN. As the connection patterns in both switch fabrics are periodic with period FN and there are only N active input/output ports, a VOQ buffer between the two switch fabrics only needs to read/write a packet every F time slots. Based on this, the memory speed for VOQs need not be comparable with the line speed and can be slowed down to $2/F$ of the line speed.

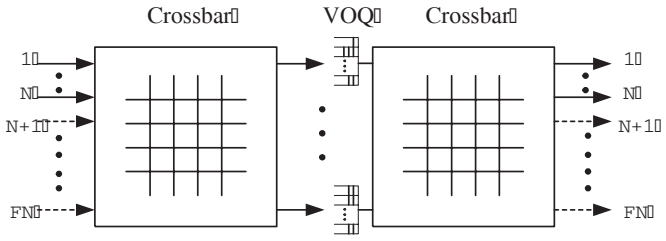


Fig. 3.7. An $FN \times FN$ load balanced Birkhoff-von Neumann switch with idle ports

We have illustrated how one uses an $FN \times FN$ load balanced Birkhoff-von Neumann switch for an $N \times N$ switch with memories running at $2/F$ of the line speed. The next step is to simplify the design of the two $FN \times FN$ crossbar switch fabrics. As there are only N active ports in both switch fabrics, the first switch fabric can be implemented by an $N \times FN$ switch fabric and the second switch fabric can be implemented by an $FN \times N$ switch fabric. One way to implement an $N \times FN$ switch fabric is the concatenation of an $N \times N$ switch fabric and N $1 \times F$ demultiplexers in Figure 3.8. In order to

distribute packets in a round-robin fashion to the FN VOQs, the connection patterns of the N demultiplexers are changed in a round-robin fashion every time slot. The connection patterns of the $N \times N$ switch fabric are changed in the same manner as the $N \times N$ load balanced Birkhoff-von Neumann switch except it is now changed every F time slots. Similarly, an $N \times FN$ switch fabric can be implemented by the concatenation of N $F \times 1$ multiplexers and an $N \times N$ switch fabric. A complete architecture of the refined $N \times N$ switch is shown in Figure 3.9.

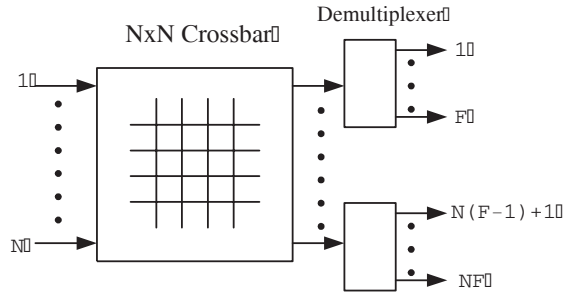


Fig. 3.8. An implementation of an $N \times FN$ switch fabric

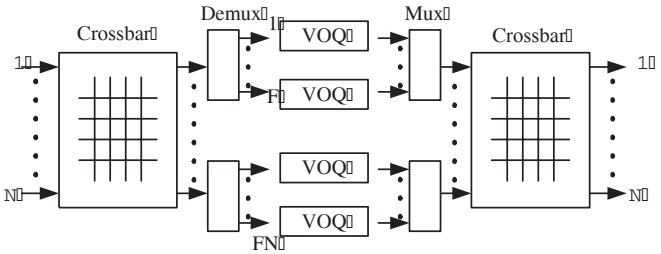


Fig. 3.9. A complete architecture of the refined $N \times N$ switch

3.2 Switch fabrics in the load-balanced Birkhoff-von Neumann switches

The load balanced Birkhoff-von Neumann switch requires two $N \times N$ crossbar switches. This may not be scalable for large N. In this section

we show that the crossbars in the load balanced Birkhoff-von Neumann switch can be implemented with $O(\log N)$ complexity.

The key observation is that we do not need to realize all the permutation matrices in the two crossbars. The connection patterns in the load balanced Birkhoff-von Neumann switch are periodic with period N. Thus, we only need to realize the N permutation matrices $P_1, P_2, \ldots, P_N$ such that $\sum_{i=1}^{N} P_i$ is an $N \times N$ matrix with all the elements being 1.

3.2.1 Construction by the banyan network

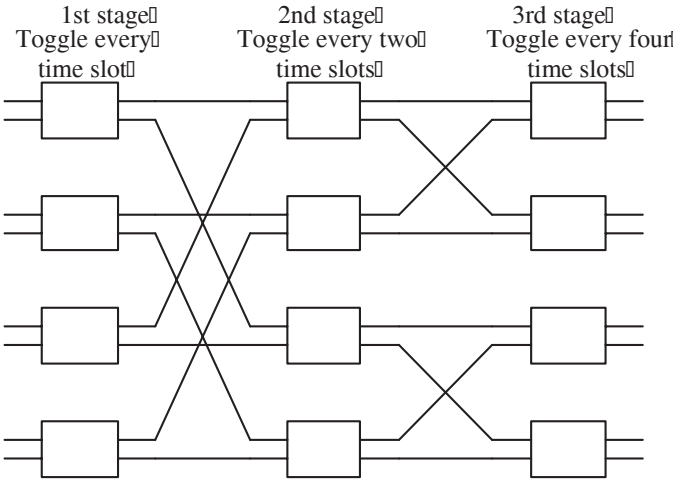


Fig. 3.10. An illustrating example for implementing the 8×8 crossbars in the load balanced Birkhoff-von Neumann switch

The easiest way to construct N permutation matrices $P_1, P_2, \ldots, P_N$ such that $\sum_{i=1}^{N} P_i$ is an $N \times N$ matrix with all the elements being 1 is to use the banyan network. In Figure 3.10, we illustrate how one implements an 8×8 crossbar in the load balanced Birkhoff-von Neumann switch by the banyan network with 2×2 switches. The banyan network is slightly different from the standard construction in Section 2.5.2. It does not have the shuffle exchange in the front of the standard construction. Note that there are only two connection patterns in a 2×2 switch. In Figure 3.10, we set the connection patterns at

the first stage to toggle every time slot, the connection patterns at the second stage to toggle every two time slots, and the connection patterns at the third stage to toggle every four time slots. By so doing, the connection patterns repeat themselves every 8 time slots and we have all the connection patterns needed for the load balanced Birkhoff-von Neumann switch. This can be easily extended to the case with $N = 2^n$ for some integer n. In the general case, the connection patterns at the n^{th} stage are set to toggle every 2^{n-1} time slots. As the number of 2×2 switches needed for the $N \times N$ banyan network is only $(N \log_2 N)/2$, we can build the crossbars in the load balanced Birkhoff-von Neumann switch with $O(\log N)$ complexity.

Another interesting observation is that the two-stage switching fabrics can be implemented by a single *optical* switch fabric with micromirrors (Keslassy and McKeown [98]). This is because light is *bi-directional* and the connection patterns at the two stages are identical. Such an implementation is called the folded version of the load balanced Birkhoff-von Neumann switch.

3.2.2 Recursive construction of the symmetric TDM switches

Among all the sequences of connection patterns that could be used in the load balanced Birkhoff-von Neumann switches, the following sequence of *symmetric* connection patterns is of particular importance. During the t^{th} time slot, input port i is connected to the output port j if

$$(i + j) \bmod N = (t + 1) \bmod N. \tag{3.32}$$

In particular, at $t = 1$, we have input port 1 connected to output port 1, input port 2 connected to output port N, ..., and input port N connected to output port 2. Clearly, such connection patterns are periodic with period N. Moreover, each input port is connected to each of the N output ports exactly once in every N time slot. Specifically, input port i is connected to output port 1 at time i, output port 2 at time $i + 1$, ..., output port N at time $i + N - 1$. Also, we note from (3.32) that such connection patterns are *symmetric*, i.e., input port i and output port j are connected if and only if input port j and output port i are connected. As such, we call a switch fabric that implements the connection patterns in (3.32) a *symmetric Time*

Division Multiplexing (TDM) switch (as introduced in Example 3.1.3). Note that one can solve j in (3.32) by the following function

$$j = h(i,t) = \Big((t-i) \bmod N\Big) + 1. \tag{3.33}$$

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

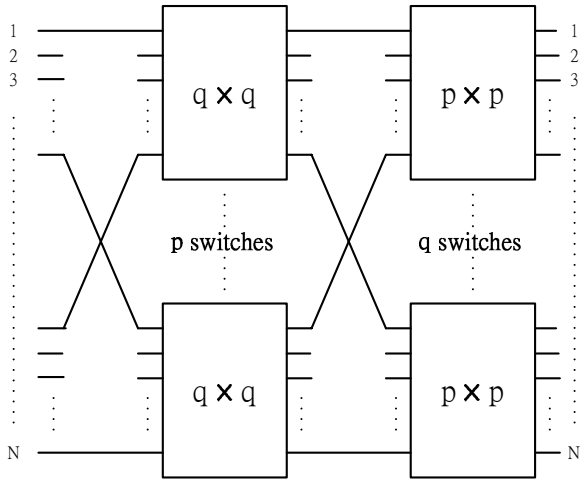


Fig. 3.11. A two-stage construction of an $N \times N$ symmetric TDM switch

In this section, we show that an $N \times N$ symmetric TDM switch can be easily constructed with $O(N \log N)$ complexity. To do this, we first show in Figure 3.11 a two-stage construction of an $N \times N$ symmetric TDM switch (with $N = pq$). As the X2 construction in Section 2.5.1, the first stage consists of p $q \times q$ symmetric TDM switches (indexed from $1, 2, \ldots, p$) and the second stage consists of q $p \times p$ symmetric TDM switches (indexed from $1, 2, \ldots, q$). These two stages of switches are connected by the perfect shuffle, i.e., the ℓ^{th} output of the k^{th} switch at the first stage is connected to the k^{th} input of the ℓ^{th} switch at the second stage. Also, index the N inputs and outputs from 1 to N. The N inputs of the $N \times N$ switch are connected to the inputs of the switches at the first stage by the perfect shuffle. To be precise, let

$$\ell(i) = \lfloor \frac{i-1}{p} \rfloor + 1, \tag{3.34}$$

and

$$k(i) = i - (\ell(i) - 1) * p. \tag{3.35}$$

Note that for $i = 1, 2, \ldots, N$, $\ell(i)$ is an integer between 1 and q and $k(i)$ is an integer between 1 and p. Then the i^{th} input of the $N \times N$ switch is connected to the $\ell(i)^{th}$ input of the $k(i)^{th}$ switch at the first stage. Also, we note that the j^{th} output of the $N \times N$ switch is the $k(j)^{th}$ output of the $\ell(j)^{th}$ switch at the second stage.

The symmetric TDM switches at these two stages are operated at different time scales. The connection patterns of the symmetric TDM switches at the second stage are changed every time slot. However, the connection patterns of the symmetric TDM switches at the first stage are changed every *frame* with each frame containing p time slots. To be specific, we define the m^{th} frame of the k^{th} switch at the first stage in Figure 3.11 to be the set of time slots $\{(m-1)p+k, (m-1)p+k+1, \ldots, mp+k-1\}$. Then every symmetric TDM switch at the first stage is operated according to its own frames. Note that the p symmetric TDM switches at the first stage do not change their connection patterns at the same time as the m^{th} frames of these switches contain different sets of time slots.

Lemma 3.2.1. *The two-stage construction in Figure 3.11 is an $N \times N$ symmetric TDM switch.*

Proof. In order for the $N \times N$ switch to be a symmetric TDM switch, we need to show that the i^{th} input port is connected to the j^{th} output at time t when

$$(i + j) \bmod N = (t + 1) \bmod N. \tag{3.36}$$

From the topology in Figure 3.11, we know there is a unique routing path from an input of the $N \times N$ switch to an output of the $N \times N$ switch. To be precise, the i^{th} input is connected to the $\ell(i)^{th}$ input of the $k(i)^{th}$ switch at the first stage. Also, the $\ell(j)^{th}$ output of the $k(i)^{th}$ switch at the first stage is connected to the $k(i)^{th}$ input of the $\ell(j)^{th}$ switch at the second stage. Note that the j^{th} output of the $N \times N$ switch is the $k(j)^{th}$ output of the $\ell(j)^{th}$ switch at the second stage. Thus, in order for the i^{th} input of the $N \times N$ switch to be connected to the j^{th} output of the $N \times N$ switch at time t, one must have

(i) the $\ell(i)^{th}$ input of the $k(i)^{th}$ switch at the first stage is connected to its $\ell(j)^{th}$ output at time t, and
(ii) the $k(i)^{th}$ input of the $\ell(j)^{th}$ switch at the second stage is connected to its $k(j)^{th}$ output at time t.

As the switches at the first stage are $q \times q$ symmetric TDM switches that change their connection patterns every frame, we have from (i) that t must be in the m^{th} frame of the $k(i)^{th}$ switch at the first stage, where m satisfies

$$(\ell(i) + \ell(j)) \bmod q = (m+1) \bmod q. \tag{3.37}$$

From (3.37), it follows that for some integer m_2

$$(\ell(i) - 1) + (\ell(j) - 1) = (m-1) + m_2 q. \tag{3.38}$$

Similarly, as the switches at the second stage are $p \times p$ symmetric TDM switches that change their connection patterns every time slot, we have from (ii) that

$$(k(i) + k(j)) \bmod p = (t+1) \bmod p. \tag{3.39}$$

Since t is in the m^{th} frame of the $k(i)^{th}$ switch, t is one of the p time slots $\{(m-1)p + k(i), (m-1)p + k(i) + 1, \ldots, mp + k(i) - 1\}$. Thus, we have from (3.39) that

$$t = (m-1)p + k(i) + k(j) - 1. \tag{3.40}$$

Note from (3.35), (3.38) and (3.40) that

$$\begin{aligned}
&(i+j) \bmod N \\
&= \Big((\ell(i) - 1)p + k(i) + (\ell(j) - 1)p + k(j)\Big) \bmod N \\
&= \Big((m-1)p + m_2 pq + k(i) + k(j)\Big) \bmod N \\
&= (t + 1 + m_2 N) \bmod N \\
&= (t+1) \bmod N.
\end{aligned} \tag{3.41}$$

■

Note that a 2×2 switch only has two connection patterns and it is a symmetric TDM switch if it alternates its two connection patterns every time slot. If N is a power of 2, then one can recursively expand the two-stage construction by 2×2 switches. The number of 2×2 switches needed for an $N \times N$ symmetric TDM switch is then $\frac{N}{2} \log_2 N$. This shows that one can build an $N \times N$ symmetric TDM switch with $O(N \log N)$ complexity. In Figure 3.12, we show an 8×8 symmetric TDM switch that uses the recursive construction. The eight connection patterns of each 2×2 switch are represented by a sequence of 8 characters in "b" and "x", where "b" denotes the bar connection

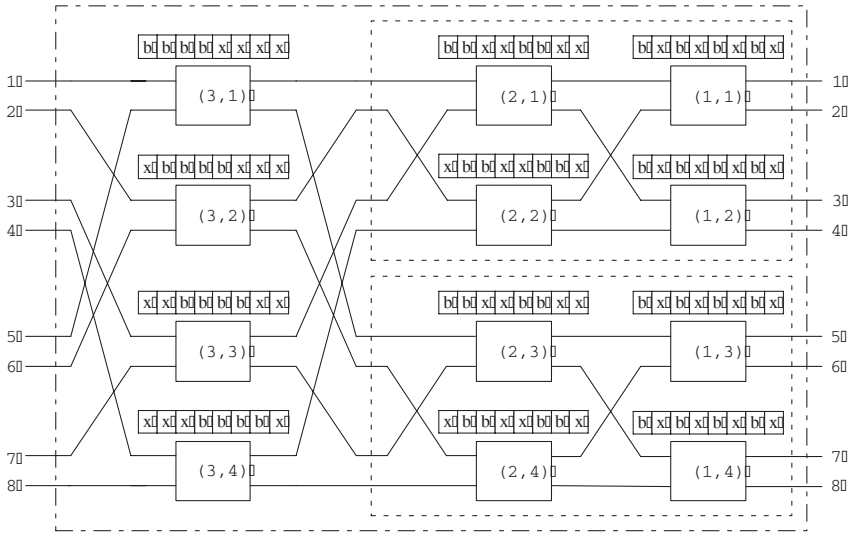


Fig. 3.12. An 8×8 symmetric TDM switch via 2×2 switches

and "x" denotes the cross connection of a 2×2 switch. To find out the connection patterns of the 2×2 switches in the general case, we index the stage from left to right by $1, 2, ..., \log_2 N$, and index the switch in each stage from top to bottom by $1, 2, ..., N/2$ as in Figure 3.12. Then the connection pattern of the m^{th} switch at the ℓ^{th} stage at time t is determined by the function $\psi(\ell, m, t)$:

$$\psi(\ell, m, t) = \lfloor \frac{(t - \phi(\ell, m)) \bmod 2^{\ell}}{2^{\ell-1}} \rfloor, \tag{3.42}$$

where

$$\phi(\ell, m) = ((m-1) \bmod 2^{\ell-1}) + 1. \tag{3.43}$$

We set the bar connection pattern if $\psi(\ell, m, t) = 0$, and the cross connection pattern if $\psi(\ell, m, t) = 1$.

3.3 Load balanced Birkhoff-von Neumann switches: multi-stage buffering

The main objective of this section is to show that one can build a load balanced Birkhoff-von Neumann switch that has comparable performance to the ideal output-buffered switch. This is done by addressing

the out-of-sequence problem in the load balanced Birkhoff-von Neumann switch with one-stage buffering. One quick fix for the out-of-sequence problem is to add a resequencing-and-output buffer after the second stage. However, as packets are distributed according to their *arrival times* at the first stage, there is no guarantee on the size of the resequencing-and-output buffer to prevent packet losses. For this, one needs to distributed packets in a more even manner.

First, we introduce the concept of *flows*. A (point-to-point) flow is a stream of packets that has a common input and a common output. In this section, we consider a more general traffic model with *multicasting* flows. A multicasting flow is a stream of packets that has one common input and a set of common outputs. Instead of distributing packets according to their arrival times, packets are now distributed according to their *flows*. This is done by adding a flow splitter and a load-balancing buffer in front of the first stage (see Figure 3.13). For an $N \times N$ switch, the load-balancing buffer at each input port of the first stage consists of N virtual output queues (VOQ) destined for the N output ports of that stage. Packets from the same *flow* are split in the round-robin fashion to the N virtual output queues and scheduled under the First Come First Served (FCFS) policy. By so doing, load balancing can be achieved for each flow as packets from the same flow are split almost evenly to the input ports of the second stage. More importantly, we will show that the delay and the buffer size of the load-balancing buffer can be bounded by constants that only depend on the size of the switch and the number of flows.

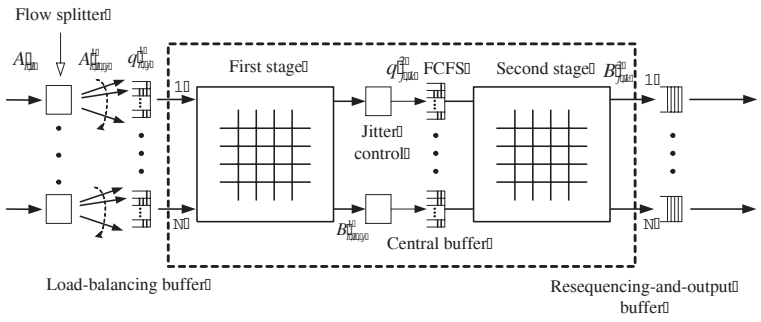


Fig. 3.13. The load balanced switch with multi-stage buffering under FCFS

Once we are able to bound the delay through the first stage, we can add a jitter control mechanism in the VOQs in front of the second stage (see Figure 3.13) to reconstruct the original traffic pattern. The jitter control mechanism delays every packet to its maximum delay. By so doing, every packet experiences the same delay before entering the buffer at an input port of the second stage. Thus, the traffic entering the VOQs (the central buffers) in front of the second stage is merely a time-shifted version of the original traffic. As in the load-balancing buffer, the scheduling policies in the VOQs in front of the second stage are FCFS, and the buffer size of these VOQs are assumed to be infinite so that no packets are lost inside the switch. Moreover, fan-out splitting (for multicasting flows) is done at these VOQs. i.e., packets with multiple outputs are duplicated at the central buffers. The intuition for this is that traffic load can be balanced at the central buffers.

The resequencing-and-output buffer after the second stage (see Figure 3.13) not only performs resequencing to keep packets in sequence, but also stores packets waiting for transmission from the output links. With jitter control at the central buffers, we will illustrate that the size of the resequencing-and-output buffer is also bounded. More importantly, the end-to-end delay through the multi-stage switch is bounded above by the sum of the delay from the corresponding FCFS output-buffered switch and a constant that only depends on the size of the switch and the number of multicasting flows supported by the switch.

The switch architecture in Figure 3.13 is called the load balanced Birkhoff-von Neumann switch with multi-stage buffering in [35]. We now summarize its key differences from the load balanced switch with one stage buffering in Section 3.1.

(i) Packets are now distributed according to their *flows*, not their *arrival times* in Section 3.1. For this, buffering is required at the first stage.
(ii) A jitter control mechanism is added in front of the central buffer to reconstruct the original traffic pattern.
(iii) Resequencing-and-output buffers are added after the second stage to ensure that packets depart in order.

Now we introduce some notations needed for our presentation. We consider an $N \times N$ switch with multicasting flows under the FCFS scheduling policy. Let L_i be the number of multicasting flows through the i^{th} input port. Denote by $A_{i,\ell}(t)$ the cumulative number of packet

arrivals by time t from the ℓ^{th} multicasting flow at the i^{th} input port, $i = 1, \ldots, N$, $\ell = 1, \ldots, L_i$. Also, let $S_{i,\ell}$ be the set of outputs of that flow, $S^*(k) = \{(i, \ell) : k \in S_{i,\ell}\}$ be the set of multicasting flows through the k^{th} output, and $M_k = |S^*(k)|$ be the number of multicasting flows through the k^{th} output port. Define $L_{\max} = \max_{1 \le i \le N} L_i$ as the maximum number of multicasting flow through an input port and $M_{\max} = \max_{1 \le k \le N} M_k$ as the maximum number of multicasting flow through an output port.

The key results of the load-balanced Birkhoff-von Neumann switch with multi-stage buffering are summarized in the following theorem. The proof of Theorem 3.3.1 will be shown in Section 3.3.2-Section 3.3.4.

Theorem 3.3.1. *Suppose that all the buffers are empty at time 0. Then the following results hold for FCFS scheme with the jitter control mechanism (described in Section 3.3.3).*

(i) The end-to-end delay for a packet through the load balanced Birkhoff-von Neumann switch with multi-stage buffering is bounded above by the sum of the delay through the corresponding FCFS output-buffered switch and $(N-1)L_{\max} + NM_{\max}$, where $L_{\max}$ (resp. $M_{\max}$) is the maximum number of flows at an input (resp. output) port.

(ii) The load-balancing buffer at an input port of the first stage is bounded above by $NL_{\max}$.

(iii) The delay through the load-balancing buffer at an input port of the first stage is bounded above by $(N-1)L_{\max}$.

(iv) The resequencing-and-output buffer at an output port of the second stage is bounded above by $NM_{\max}$.

3.3.1 The FCFS output-buffered switch

In order to compare the performance of the multi-stage switch with that of an output-buffered switch, we first establish some results for the FCFS output-buffered switch.

Now consider feeding these multicasting flows to an $N \times N$ output-buffered switch under the FCFS policy (see Figure 3.14). Assume that there is an infinite buffer at each output port and that all the buffers are empty at time 0. Let $A_k^o(t)$ be the cumulative number of arrivals at the k^{th} output buffer by time t, $q_k^o(t)$ be the number of packets at

the k^{th} output buffer at time t, and $B_k^o(t)$ be the cumulative number of departures at the k^{th} output buffer by time t. Note that

$$A_k^o(t) = \sum_{(i,\ell)\in S^*(k)} A_{i,\ell}(t) \tag{3.44}$$

is the superposition of the multicasting flows through the k^{th} output. From the Lindley equation in (2.1), it follows that

$$q_k^o(t+1) = (q_k^o(t) + a_k^o(t+1) - 1)^+, \tag{3.45}$$

where $a_k^o(t) = A_k^o(t) - A_k^o(t-1)$ is the number of arrivals at the k^{th} output buffer at time t. Since we start from an empty system, i.e., $q_k^o(0) = 0$, we have from Lemma 2.1.2 that

$$q_k^o(t) = \max_{0\le s\le t}[A_k^o(t) - A_k^o(s) - (t-s)]. \tag{3.46}$$

Moreover,

$$B_k^o(t) = A_k^o(t) - q_k^o(t) = \min_{0\le s\le t}[A_k^o(s) + (t-s)]. \tag{3.47}$$

Since there is at most one packet coming out from an input port per time slot, the number of packet arrivals to output k per time slot is bounded above by M_k, the number of flows through output k. Thus, under the FCFS scheduling policy, packets that arrive at the k^{th} buffer at time t will depart between $t + q_k^o(t) - M_k + 1$ and $t + q_k^o(t)$. Also, note from (3.47) that

$$B_k^o(t) - B_k^o(s) \le t - s, \quad \text{for all } s \le t, \tag{3.48}$$

as there is at most one packet coming out from an output port per time slot.

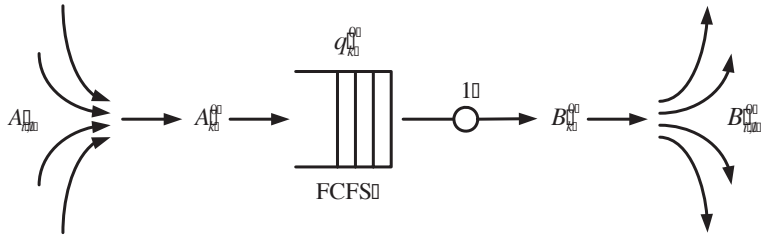


Fig. 3.14. The FCFS output-buffered switch with multicasting flows

3.3.2 The load-balancing buffer

In this section, we derive bounds for the load-balancing buffer in front of the first stage. Recall that the load-balancing buffer at each input port of the first stage consists of N virtual output queues (VOQ) destined for the N output ports of that stage. Packets from the same flow are split in the round-robin fashion to the N virtual output queues and scheduled under the FCFS policy. Without loss of generality, we may assume that the first packet of a flow is always assigned to the first VOQ. To be precise, let $A^1_{i,\ell,j}(t)$ be the cumulative number of $A_{i,\ell}$-flow packets that are split into the j^{th} VOQ at the i^{th} input port of the first stage by time t. Then

$$A^1_{i,\ell,j}(t) = \lceil \frac{A_{i,\ell}(t) - j + 1}{N} \rceil, \quad j = 1, \ldots, N, \tag{3.49}$$

and

$$A_{i,\ell}(t) = \sum_{j=1}^{N} A^1_{i,\ell,j}(t), \tag{3.50}$$

where $\lceil x \rceil$ (resp. $\lfloor x \rfloor$) is the ceiling (resp. floor) function of x. The ceiling function $\lceil x \rceil$ is the smallest integer that is not smaller than x, and the floor function $\lfloor x \rfloor$ is the largest integer that is not larger than x.

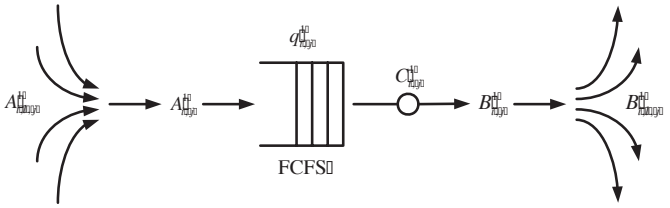


Fig. 3.15. The j^{th} VOQ at the i^{th} input port of the first stage

Now consider the j^{th} VOQ at the i^{th} input port of the first stage (see Figure 3.15). Let $A^1_{i,j}(t)$ be the cumulative number of arrivals by time t to this queue, $C^1_{i,j}(t)$ be its cumulative number of time slots assigned to this queue by time t, $q^1_{i,j}(t)$ be the number of packets queued at time t, and $B^1_{i,j}(t)$ be the cumulative number of departures by time t from this queue. Note that

$$A^1_{i,j}(t) = \sum_{\ell=1}^{L_i} A^1_{i,\ell,j}(t). \tag{3.51}$$

For such a queue, we have the following Lindley equation

$$q^1_{i,j}(t+1) = (q^1_{i,j}(t) + a^1_{i,j}(t+1) - c^1_{i,j}(t+1))^+, \tag{3.52}$$

where $a^1_{i,j}(t) = A^1_{i,j}(t) - A^1_{i,j}(t-1)$ is the number of arrivals to the queue at time t and $c^1_{i,j}(t) = C^1_{i,j}(t) - C^1_{i,j}(t-1)$ is the number of slot assigned to the queue at time t. Since we start from an empty queue, one has the following representation (cf. (3.46))

$$q^1_{i,j}(t) = \max_{0\le s\le t}[A^1_{i,j}(t) - A^1_{i,j}(s) - (C^1_{i,j}(t) - C^1_{i,j}(s))]. \tag{3.53}$$

Moreover,

$$B^1_{i,j}(t) = \min_{0\le s\le t}[A^1_{i,j}(s) + C^1_{i,j}(t) - C^1_{i,j}(s)]. \tag{3.54}$$

In Lemma 3.3.2, we show that both the queue length and the delay at the load-balancing buffer can be bounded by finite constants.

Lemma 3.3.2. *(i) The maximum queue length of the j^{th} VOQ at the i^{th} input port of the first stage is bounded above by L_i, i.e.,*

$$q^1_{i,j}(t) \le L_i, \quad \text{for all } t. \tag{3.55}$$

(ii) The maximum delay for a packet to depart the j^{th} VOQ at the i^{th} input port of the first stage is bounded above by $(N-1)L_i$.

For the proof of Lemma 3.3.2, we need to introduce the following properties for the ceiling function and the floor function. Its proof is left as an exercise for the readers.

Proposition 3.3.3. *(Properties of the ceiling and floor functions)*

(i) $\lceil a+b\rceil \le \lceil a\rceil + \lceil b\rceil \le \lceil a+b\rceil + 1$.
(ii) $\lfloor a+b\rfloor \ge \lfloor a\rfloor + \lfloor b\rfloor$.
(iii) $\lceil a\rceil \le \lfloor a\rfloor + 1$.
(iv) For an integer a, and $N > 0$,

$$\lceil \frac{a}{N} \rceil = \lfloor \frac{a+(N-1)}{N} \rfloor.$$

Proof. (Lemma 3.3.2) (i) Note from (3.51) and (3.49) that

$$A^1_{i,j}(t) - A^1_{i,j}(s) = \sum_{\ell=1}^{L_i} \Big(A^1_{i,\ell,j}(t) - A^1_{i,\ell,j}(s) \Big)$$
$$= \sum_{\ell=1}^{L_i} \Big(\lceil \frac{A_{i,\ell}(t) - j + 1}{N} \rceil - \lceil \frac{A_{i,\ell}(s) - j + 1}{N} \rceil \Big).$$

Applying the first inequality in Proposition 3.3.3 (i) yields

$$A^1_{i,j}(t) - A^1_{i,j}(s) \le \sum_{\ell=1}^{L_i} \lceil \frac{A_{i,\ell}(t) - A_{i,\ell}(s)}{N} \rceil. \tag{3.56}$$

Since the connection patterns at the first stage are periodic with period N for N permutation matrices satisfying (3.1), we have

$$C^1_{i,j}(t) - C^1_{i,j}(s) \ge \lfloor \frac{t-s}{N} \rfloor. \tag{3.57}$$

Observe that $\sum_{\ell=1}^{L_i} (A_{i,\ell}(t) - A_{i,\ell}(s))$ is the number of packet arrivals at the i^{th} input port during the interval of length $t - s$. As there is at most one packet arrival at an input port per time slot, we have

$$\sum_{\ell=1}^{L_i} (A_{i,\ell}(t) - A_{i,\ell}(s)) \le t - s. \tag{3.58}$$

In conjunction with (3.57),

$$C^1_{i,j}(t) - C^1_{i,j}(s) \ge \lfloor \frac{\sum_{\ell=1}^{L_i} (A_{i,\ell}(t) - A_{i,\ell}(s))}{N} \rfloor$$
$$\ge \sum_{\ell=1}^{L_i} \lfloor \frac{A_{i,\ell}(t) - A_{i,\ell}(s)}{N} \rfloor, \tag{3.59}$$

where we use Proposition 3.3.3 (ii) in the last inequality. From (3.53), (3.56), (3.59), and Proposition 3.3.3 (iii),

$$\begin{aligned} q^1_{i,j}(t) &\le \max_{0 \le s \le t} \Big[\sum_{\ell=1}^{L_i} \Big(\lceil \frac{A_{i,\ell}(t) - A_{i,\ell}(s)}{N} \rceil - \lfloor \frac{A_{i,\ell}(t) - A_{i,\ell}(s)}{N} \rfloor \Big) \Big] \\ &\le L_i. \end{aligned}$$

(ii) Since the scheduling policy at this queue is FCFS, it suffices to show that

$$B^1_{i,j}(t + (N-1)L_i) \ge A^1_{i,j}(t).$$

Note from (3.54) that

$$
\begin{aligned}
&B^1_{i,j}(t+(N-1)L_i)-A^1_{i,j}(t)\\
&= \min_{0\le s\le t+(N-1)L_i}[A^1_{i,j}(s)-A^1_{i,j}(t)+C^1_{i,j}(t+(N-1)L_i)-C^1_{i,j}(s)]\\
&= \min\Big[\min_{0\le s\le t}[C^1_{i,j}(t+(N-1)L_i)-C^1_{i,j}(s)-(A^1_{i,j}(t)-A^1_{i,j}(s))],\\
&\quad \min_{t+1\le s\le t+(N-1)L_i}[C^1_{i,j}(t+(N-1)L_i)-C^1_{i,j}(s)-(A^1_{i,j}(t)-A^1_{i,j}(s))]\Big].
\end{aligned}
$$

All the terms in the second minimum are clearly nonnegative as both $A^1_{i,j}(t)$ and $C^1_{i,j}(t)$ are non-decreasing in t. On the other hand, for $0 \le s \le t$, we have from (3.57), (3.58), (3.56) and Proposition 3.3.3 (ii), (iv) that

$$
\begin{aligned}
&C^1_{i,j}(t+(N-1)L_i)-C^1_{i,j}(s)-(A^1_{i,j}(t)-A^1_{i,j}(s))\\
&\ge \lfloor\frac{t+(N-1)L_i-s}{N}\rfloor-(A^1_{i,j}(t)-A^1_{i,j}(s))\\
&\ge \lfloor\frac{\sum_{\ell=1}^{L_i}(A_{i,\ell}(t)-A_{i,\ell}(s))+(N-1)L_i}{N}\rfloor-\sum_{\ell=1}^{L_i}\lceil\frac{A_{i,\ell}(t)-A_{i,\ell}(s)}{N}\rceil\\
&= \lfloor\frac{\sum_{\ell=1}^{L_i}(A_{i,\ell}(t)-A_{i,\ell}(s)+(N-1))}{N}\rfloor-\sum_{\ell=1}^{L_i}\lceil\frac{A_{i,\ell}(t)-A_{i,\ell}(s)}{N}\rceil\\
&\ge \sum_{\ell=1}^{L_i}\Big(\lfloor\frac{A_{i,\ell}(t)-A_{i,\ell}(s)+(N-1)}{N}\rfloor-\lceil\frac{A_{i,\ell}(t)-A_{i,\ell}(s)}{N}\rceil\Big)=0.
\end{aligned}
$$

■

3.3.3 The central buffer under FCFS

From Lemma 3.3.2 in the previous section, we know that the delay through the first stage is bounded above by $d_{1,\max}=(N-1)L_{\max}$. To reconstruct the original traffic pattern, one may add a jitter control mechanism in the VOQs in front of the second stage. As the delay through the load-balancing buffer is bounded above by $d_{1,\max}$, packets with delay less than $d_{1,\max}$ are delayed to $d_{1,\max}$. By so doing, every packet has the same delay before entering the buffer at an input port of the second stage. Thus, the traffic entering the VOQs in front of the second stage is merely a time-shifted version of the original traffic.

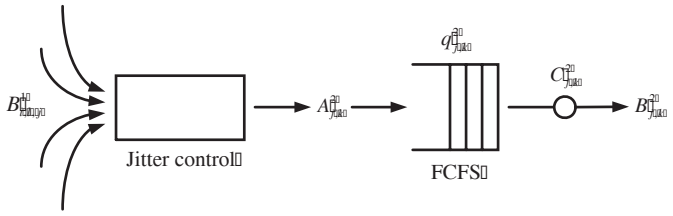


Fig. 3.16. The k^{th} VOQ at the j^{th} input port of the second stage under FCFS

As in the load-balancing buffer, the scheduling policies in the VOQs in front of the second stage are FCFS. Moreover, fan-out splitting (for multicasting flows) is done at these VOQs. Now consider the k^{th} VOQ at the j^{th} input port of the second stage (see Figure 3.16). Let $A^2_{j,k}(t)$ be the cumulative number of arrivals by time t to this queue, $C^2_{j,k}(t)$ be its cumulative number of time slots assigned to this queue by time t, $q^2_{j,k}(t)$ be the number of packets queued at time t, and $B^2_{j,k}(t)$ be the cumulative number of departures by time t from this queue. Since every packet has the same delay $d_{1,\max}$ through the first stage and fan-out splitting is done at this stage, we have

$$A^2_{j,k}(t) = \sum_{(i,\ell)\in S^*(k)} A^1_{i,\ell,j}(t - d_{1,\max}) \tag{3.60}$$

(Here we use the convention that $A^1_{i,\ell,j}(\tau) = 0$ for $\tau < 0$). Analogous to (3.53) and (3.54), one has

$$q^2_{j,k}(t) = \max_{0\le s\le t}[A^2_{j,k}(t) - A^2_{j,k}(s) - (C^2_{j,k}(t) - C^2_{j,k}(s))], \tag{3.61}$$

and

$$B^2_{j,k}(t) = \min_{0\le s\le t}[A^2_{j,k}(s) + C^2_{j,k}(t) - C^2_{j,k}(s)]. \tag{3.62}$$

Lemma 3.3.4. *For the FCFS scheme,*

(i) $q^2_{j,k}(t) \le \lceil \frac{q^o_k(t-d_{1,\max})}{N} \rceil + M_k$, *for all* t, *and*
(ii) $B^2_{j,k}(t + q^o_k(t - d_{1,\max}) + (N-1)M_k) \ge A^2_{j,k}(t)$, *for all* t.

Lemma 3.3.4 (i) provides an upper bound for the queue length in terms of the queue length of the corresponding output-buffered switch. As the scheduling policy is FCFS, Lemma 3.3.4 (ii) implies that a packet that arrives at the queue at time t will depart not later than $t + q^o_k(t - d_{1,\max}) + (N-1)M_{\max}$.

Proof. (i) Note from (3.60), (3.49) and the inequalities in Proposition 3.3.3 (i) that

$$\begin{aligned} & A^2_{j,k}(t) - A^2_{j,k}(s) \\ = & \sum_{(i,\ell)\in S^*(k)} (A^1_{i,\ell,j}(t-d_{1,\max}) - A^1_{i,\ell,j}(s-d_{1,\max})) \\ \le & \sum_{(i,\ell)\in S^*(k)} \lceil \frac{A_{i,\ell}(t-d_{1,\max}) - A_{i,\ell}(s-d_{1,\max})}{N} \rceil \qquad (3.63) \\ \le & \lceil \frac{\sum_{(i,\ell)\in S^*(k)} (A_{i,\ell}(t-d_{1,\max}) - A_{i,\ell}(s-d_{1,\max}))}{N} \rceil + M_k - 1. \qquad (3.64) \end{aligned}$$

Observe from (3.44) and (3.46) that

$$\begin{aligned} & \sum_{(i,\ell)\in S^*(k)} (A_{i,\ell}(t-d_{1,\max}) - A_{i,\ell}(s-d_{1,\max})) \\ = & A^o_k(t-d_{1,\max}) - A^o_k(s-d_{1,\max}) \\ \le & q^o_k(t-d_{1,\max}) + (t-s). \qquad (3.65) \end{aligned}$$

Thus,

$$A^2_{j,k}(t) - A^2_{j,k}(s) \le \lceil \frac{q^o_k(t-d_{1,\max}) + (t-s)}{N} \rceil + M_k - 1. \qquad (3.66)$$

Since the connection patterns at the second stage are also periodic with period N for N permutation matrices satisfying (3.1),

$$C^2_{j,k}(t) - C^2_{j,k}(s) \ge \lfloor \frac{t-s}{N} \rfloor. \qquad (3.67)$$

From (3.66), (3.67), (3.61) and Proposition 3.3.3(i) and (iii), it then follows that

$$q^2_{j,k}(t) \le \lceil \frac{q^o_k(t-d_{1,\max})}{N} \rceil + M_k.$$

for all t.

(ii) Let $d = q^o_k(t-d_{1,\max}) + (N-1)M_k$. Note from (3.62) that

$$\begin{aligned} & B^2_{j,k}(t+d) - A^2_{j,k}(t) \\ = & \min_{0\le s\le t+d} [A^2_{j,k}(s) - A^2_{j,k}(t) + C^2_{i,j}(t+d) - C^2_{i,j}(s)] \\ = & \min\Big[\min_{0\le s\le t} [C^2_{j,k}(t+d) - C^2_{j,k}(s) - (A^2_{j,k}(t) - A^2_{j,k}(s))], \\ & \min_{t+1\le s\le t+d} [C^2_{j,k}(t+d) - C^2_{j,k}(s) - (A^2_{j,k}(t) - A^2_{j,k}(s))]\Big] \end{aligned}$$

Clearly, all the terms in the second minimum are nonnegative as both $A^2_{j,k}(t)$ and $C^2_{j,k}(t)$ are nondecreasing in t. On the other hand, for $0 \le s \le t$, we have from (3.63) that

$$\begin{aligned} & A^2_{j,k}(t) - A^2_{j,k}(s) \\ & \le \sum_{(i,\ell)\in S^*(k)} \lceil \frac{A_{i,\ell}(t-d_{1,\max}) - A_{i,\ell}(s-d_{1,\max})}{N} \rceil. \end{aligned} \tag{3.68}$$

Also, it follows from (3.67), (3.65), and Proposition 3.3.3 (ii) that

$$\begin{aligned} & C^2_{j,k}(t+d) - C^2_{j,k}(s) \ge \lfloor \frac{t+d-s}{N} \rfloor \\ & = \lfloor \frac{q^o_k(t-d_{1,\max}) + (t-s) + (N-1)M_k}{N} \rfloor \\ & \ge \lfloor \frac{\sum_{(i,\ell)\in S^*(k)} (A_{i,\ell}(t-d_{1,\max}) - A_{i,\ell}(s-d_{1,\max})) + (N-1)M_k}{N} \rfloor \\ & = \lfloor \frac{\sum_{(i,\ell)\in S^*(k)} (A_{i,\ell}(t-d_{1,\max}) - A_{i,\ell}(s-d_{1,\max}) + (N-1))}{N} \rfloor \\ & \ge \sum_{(i,\ell)\in S^*(k)} \lfloor \frac{A^2_{j,k}(t-d_{1,\max}) - A^2_{j,k}(s-d_{1,\max}) + (N-1)}{N} \rfloor. \end{aligned} \tag{3.69}$$

As in the proof of Lemma 3.3.2, we then have from (3.68), (3.69) and Proposition 3.3.3 (iv) that

$$C^2_{j,k}(t+d) - C^2_{j,k}(s) - (A^2_{j,k}(t) - A^2_{j,k}(s)) \ge 0.$$

for $0 \le s \le t$. ■

3.3.4 The resequencing-and-output buffer

The resequencing-and-output buffer conceptually consists of two (virtual) buffers (see Figure 3.17): (i) the resequencing buffer and (ii) the output buffer. The objective of the resequencing buffer is to reorder 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 waiting for transmission from the output link.

Let $A_k^3(t)$ be the cumulative arrivals by time t to the k^{th} resequencing buffer, and $B_k^3(t)$ be its cumulative departures. Note that $B_k^3(t)$ is also the cumulative arrivals by time t to the k^{th} output buffer. Let $B_k^4(t)$ be the cumulative departures by time t from the k^{th} output buffer. Clearly, from the input-output relation of an output-buffered link (cf. (3.47)), we have

$$B_k^4(t) = \min_{0 \le s \le t} [B_k^3(s) + (t - s)]. \tag{3.70}$$

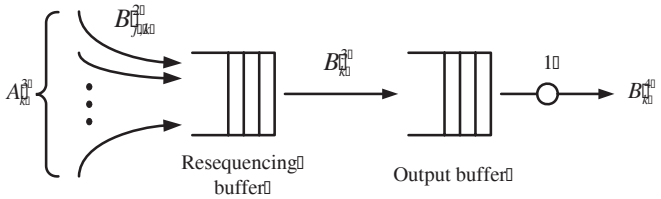


Fig. 3.17. The resequencing-and-output buffer

Lemma 3.3.5. *The following results hold for the FCFS scheme.*

(i) $A_k^3(t) \le B_k^o(t - d_{1,\max})$.
(ii) $B_k^3(t + d_{1,\max} + NM_k) \ge B_k^o(t)$.
(iii) $B_k^4(t) \ge B_k^o(t - d_{1,\max} - NM_k)$.
(iv) The number of packets queued at the k^{th} *resequencing-and-output buffer is bounded above by* NM_k*, i.e.,*

$$A_k^3(t) - B_k^4(t) \le NM_k, \quad \textit{for all } t. \tag{3.71}$$

Proof. (i) Note that

$$A_k^3(t) = \sum_{j=1}^{N} B_{j,k}^2(t).$$

From (3.62), it follows that

$$A_k^3(t) = \sum_{j=1}^{N} \min_{0 \le s \le t} [A_{j,k}^2(s) + C_{j,k}^2(t) - C_{j,k}^2(s)]$$
$$\le \min_{0 \le s \le t} [\sum_{j=1}^{N} (A_{j,k}^2(s) + C_{j,k}^2(t) - C_{j,k}^2(s))].$$

As the connection patterns at the second stage are periodic with period N for N permutation matrices satisfying (3.1),

$$\sum_{j=1}^{N} C_{j,k}^{2}(t) = t.$$

Thus,

$$A_k^3(t) \leq \min_{0\leq s\leq t}[\sum_{j=1}^{N} A_{j,k}^2(s) + t - s]. \tag{3.72}$$

Note from (3.60), (3.50), and (3.44) that for all s

$$\begin{aligned}
&\sum_{j=1}^{N} A_{j,k}^2(s) = \sum_{j=1}^{N} \sum_{(i,\ell)\in S^*(k)} A_{i,\ell,j}^1(s - d_{1,\max}) \\
&= \sum_{(i,\ell)\in S^*(k)} A_{i,\ell}(s - d_{1,\max}) \\
&= A_k^o(s - d_{1,\max}).
\end{aligned}$$

Using this and (3.47) in (3.72) yields

$$\begin{aligned}
&A_k^3(t) \leq \min_{0\leq s\leq t}[A_k^o(s - d_{1,\max}) + t - s] \\
&= \min_{d_{1,\max}\leq s\leq t}[A_k^o(s - d_{1,\max}) + t - s] \\
&= \min_{0\leq \tau\leq t-d_{1,\max}}[A_k^o(\tau) + t - d_{1,\max} - \tau] \\
&= B_k^o(t - d_{1,\max})
\end{aligned}$$

(ii) Consider a packet that is destined for the k^{th} output port. Without loss of generality, suppose that this packet arrives at time t and it is routed through the k^{th} VOQ at j^{th} input port of the second stage. From Lemma 3.3.2(ii), the packet leaves the first stage not later than $t + d_{1,\max}$. After the jitter control, it arrives at the j^{th} input port of the second stage exactly at $t + d_{1,\max}$. Since FCFS is used in the k^{th} VOQ at the j^{th} input port of the second stage, we have from Lemma 3.3.4(ii) that the packet leaves the second stage not later than $t+q_k^o(t)+(N-1)M_k + d_{1,\max}$. As the bound is independent of j, we conclude that every packet that arrives at the first stage at time t will depart from the second stage not later than $t + q_k^o(t) + (N-1)M_k + d_{1,\max}$. Note from (3.46) that $t + q_k^o(t)$ is non-decreasing in t. Thus, any other packets that are destined for the k^{th} output port and arrives before t leave the second stage not later than $t + q_k^o(t) + (N-1)M_k + d_{1,\max}$.

This implies the packet (that arrives at the first stage at time t and destined for the k^{th} output port) leaves the resequencing buffer not later than $t + q_k^o(t) + (N-1)M_k + d_{1,\max}$.

On the other hand, since the packet that arrives at time t departs from the corresponding output-buffered switch between $t+q_k^o(t)-M_k+1$ and $t+q_k^o(t)$, the departure time for a packet to leave the resequencing buffer is not later than the sum of that from the corresponding output-buffer switch and $d_{1,\max} + NM_k$. This shows that for all t

$$B_k^3(t + d_{1,\max} + NM_k) \geq B_k^o(t).$$

(iii) From (3.70) and (ii) of this lemma, it follows that

$$B_k^4(t) \geq \min_{0 \leq s \leq t} [B_k^o(s - d_{1,\max} - NM_k) + t - s].$$

Since $B_k^o(t) - B_k^o(s) \leq t - s$ in (3.48), the above minimum occurs at $s = t$. Thus,

$$B_k^4(t) \geq B_k^o(t - d_{1,\max} - NM_k).$$

(iv) From (i) and (iii) of this lemma and (3.48), it follows that

$$\begin{aligned} A_k^3(t) - B_k^4(t) &\leq B_k^o(t - d_{1,\max}) - B_k^o(t - d_{1,\max} - NM_k) \\ &\leq NM_k. \end{aligned}$$

■

Now we prove the main theorem of this section.

Proof. (Proof of Theorem 3.3.1) (i) This is a direct consequence of Lemma 3.3.5(iii).

(ii) Since there are N VOQ at each input port, the result then follows from Lemma 3.3.2(i).

(iii) This is also shown in Lemma 3.3.2(ii).

(iv) It is shown in Lemma 3.3.5(iv). ■

3.3.5 The EDF scheme

Instead of using a jitter control mechanism in the central buffers, there is an alternative approach, called the Earliest Deadline First (EDF) scheme, in [35]. In such a scheme (see Figure 3.18), every packet is

assigned a deadline that is the departure time from the corresponding FCFS output-buffered switch. Packets are scheduled according to their deadlines in the central buffers. As there is no need to implement the jitter control mechanism, average packet delay can be greatly reduced. However, as there is no jitter control, one might need a larger resequencing buffer than that in the FCFS scheme with jitter control. Since the first stage is the same as that in the FCFS scheme, the delay and the buffer size are still bounded by $(N-1)L_{\max}$ and $NL_{\max}$ respectively. In the following theorem, we summarize the results for the EDF scheme in [35] without giving the proof.

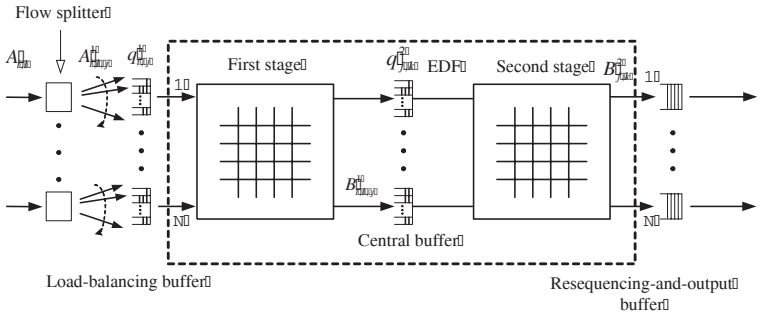


Fig. 3.18. The load balanced switch with multi-stage buffering under EDF

Theorem 3.3.6. *Suppose that all the buffers are empty at time 0. Then the following results hold for the EDF scheme.*

(i) The end-to-end delay for a packet through our switch with multi-stage buffering is bounded above by the sum of the delay through the corresponding FCFS output-buffered switch and $(N-1)(L_{\max}+M_{\max})$.

(ii) The resequencing-and-output buffer at an output port of the second stage is bounded above $(N-1)(L_{\max}+M_{\max})$.

Computing the departure times from the corresponding FCFS output-buffered switch needs global information of all the inputs. A simple way is to use the packet arrival times as deadlines. Then the EDF scheme based on arrival times yields the same departure order except those packets that arrive at same time. Since there are at most $M_{\max}$ packets that can arrive at the same time to an output port of the corresponding output-buffered switch, the end-to-end delay for a

packet through the multi-stage switch using arrival times as deadlines is bounded above by the sum of the delay through the corresponding FCFS output-buffered switch and $(N-1)L_{\max} + NM_{\max}$. Also, the resequencing-and-output buffer at an output port of the second stage in this case is bounded above $(N-1)L_{\max} + NM_{\max}$.

3.3.6 The Full Ordered Frames First scheme

Both the FCFS scheme and the EDF scheme yield deterministic bounds on the delay when comparing with an ideal output-buffered switch. If one does not require a deterministic delay bound to the ideal output buffer switch, the Full Ordered Frames First (FOFF) scheme proposed in [99] provides an easy way to achieve 100% throughput with in-sequence delivery of packets.

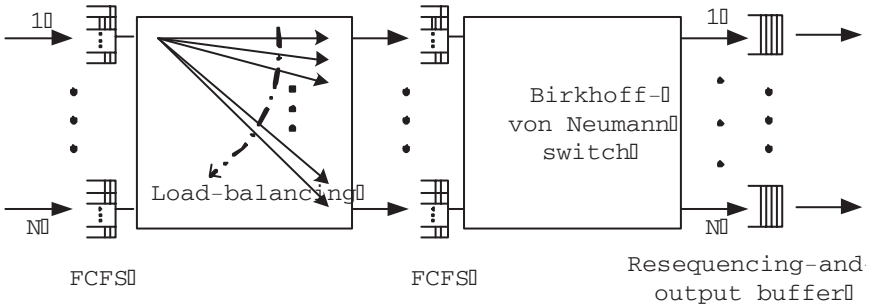


Fig. 3.19. The full ordered frames first scheme

The idea of the FOFF scheme is to add Virtual Output Queues (VOQs) at each input (see Figure 3.19) so that the traffic coming to the input can be converted into the bursty traffic with fixed burst length N (as described in Section 3.1.4). Specifically, the FOFF scheme works as follows:

(i) Each input maintains N FIFO queues, indexed from 1 to N. An arriving packet destined to the j^{th} output is placed in the j^{th} queue. For each queue, there is a pointer that keeps track of the central buffer that the next packet from this queue should be sent to.

(ii) Each input selects an FIFO queue to serve for the next N time slots. If there is a queue holding more than N packets, it will pick one of them (in the round-robin fashion) to serve for the next N

time slots. If there is no such queues, then it will pick a non-empty queue (in the round-robin fashion) to serve for the next N time slots. A head-of-line packet from the selected queue is distributed to the central buffer pointed by the pointer. The pointer is then advanced by one position (in the modulo N fashion).

Intuitively, one may view every N time slots as a frame. If there is a queue holding more than N packets, then the whole frame of packets can be distributed evenly to the central buffers. This is the ideal case. However, problems arise when no queue has more than N packets. In this case, a partial frame has to be sent. As such, bandwidth is wasted as there are empty time slots. Moreover, packets can be out-of-sequence because of this.

To bound the re-sequencing delay, consider a particular output, e.g., output 1. There are N VOQs at the central buffers destined to output 1. As there are at most $M_{\max}$ flows to every one of the N VOQs, the difference between the queue length of these N VOQs is bounded above by $M_{\max}$ (due to the partial frames from the $M_{\max}$ flows). Now suppose a particular packet is placed in one of the N VOQs at the central buffers at time t. According to the FOFF scheme, all the packets from the same flow must have been placed in these N VOQs before t. As it takes every N time slots to connect one of the N VOQs to output 1, the worst case re-sequencing delay is then bounded by N times of the difference of the queue length of the N VOQs, i.e., $NM_{\max}$.

The problem of the FOFF scheme, as pointed out by [44], is its large average packet delay due to the large frame size N. Instead of using framing, it is suggested in [44] that arbitration be done in every time slot at every input. For instance, the longest queue first (LQF) scheme in [44] selects the longest FIFO queue among all the queues that have the pointer pointing to the connected central buffer. It is shown in [44] by simulation that the LQF scheme yields almost the same average delay as the EDF scheme described in the previous section, and it performs much better than the FOFF scheme.

For the point-to-point flows, the maximum number of flows is N. This implies that the re-sequencing delay in the FCFS scheme, the EDF scheme, the FOFF scheme (and the variants in [44]) is $O(N^2)$. In the later development, we will show that there are other solutions that greatly reduce the re-sequencing delay.

3.4 Guaranteed rate services with the earliest eligible time first policy

The main objective of this section is to introduce a scheme for providing guaranteed rate services in an $N \times N$ load balanced Birkhoff-von Neumann switch with multicasting flows. This scheme originally proposed in [40] is based on the two-stage switch architecture in Section 3.3. As show in Figure 3.20, the switch architecture consists of two $N \times N$ crossbar switch fabrics and three stages of buffers. These three stages of buffers are the load-balancing buffers, the central buffers, and the output buffers. As described in Section 3.3, the connection patterns of both crossbar switch fabrics are generated from N permutation matrices that satisfy (3.1). As such, these connection patterns are periodic with period N and every input is connected to every output exactly once in every N time slots.

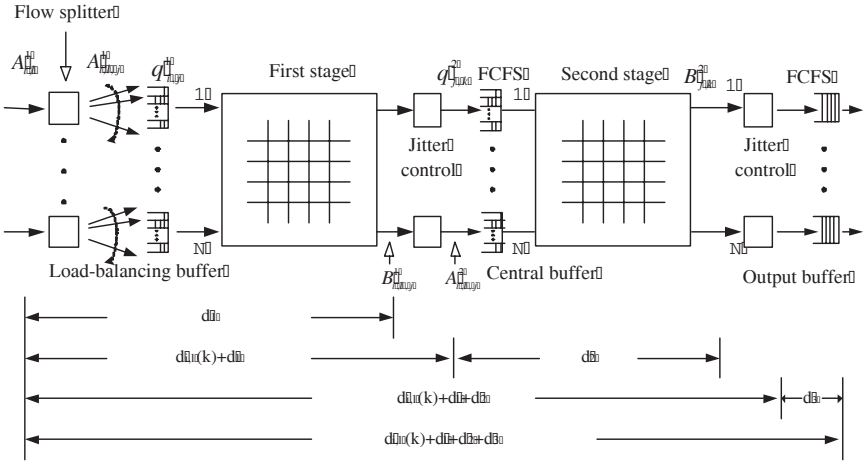


Fig. 3.20. A two-stage switch architecture with guaranteed rate services for multicasting flows

As in Section 3.3, the objective of the first stage is to perform load balancing. The load-balancing buffer at an input consists of N virtual output queues (VOQs). Packets in the j^{th} VOQ of the load-balancing buffer of the i^{th} input will be sent to the j^{th} central buffer. Suppose that there are L_i multicasting flows at the i^{th} input port, $i = 1, \ldots, N$. Packets that belong to the same multicasting flow are routed to the N VOQs in a round-robin fashion. Without loss of generality, one may

assume that the first packet in a flow is always routed to the first VOQ. To be precise, let $A_{i,\ell}(t)$ be the cumulative number of arrivals of the ℓ^{th} multicasting flow at the i^{th} input by time t, and $A^1_{i,\ell,j}(t)$ be the cumulative number of packets from that flow that are routed to the j^{th} VOQ at the i^{th} input by time t. Then

$$A^1_{i,\ell,j}(t) = \lceil \frac{A_{i,\ell}(t) - j + 1}{N} \rceil, \quad j = 1, \ldots, N. \tag{3.73}$$

One key result for such a load-balancing mechanism (shown in Theorem 3.3.1(iii) in Section 3.3) is that the maximum delay for a packet to depart from the first crossbar switch fabric is bounded above by a constant $d_1 = (N-1)L_{\max}$, where $L_{\max} = \max_{1 \le i \le N} L_i$ is the maximum number of flows supported at an input.

To provide guaranteed rate services, every packet of a (guaranteed rate) flow is assigned a *targeted departure time* that is the departure time from the corresponding FCFS work conserving link with capacity equal to the guaranteed rate of the flow. After leaving the first stage, a packet enters the jitter control stage in front of the central buffer. The time for a packet to leave the jitter control stage, called the *eligible time* of that packet, is set to be the sum of the targeted departure time and the maximum delay of the first stage (i.e., $(N-1)L_{\max}$). In the central buffer, packets are scheduled under the FCFS policy. We note that in implementation one may combine both the jitter control mechanism and the central buffer by using a single memory block. By time stamping every packet with its eligible time, the scheduling policy there is to schedule the *first eligible packet*. Such a policy is called the *earliest eligible time first policy*. Another point is that best effort service can be provided as background traffic. Flows from best effort service can be assigned to a low priority queue and they are only served when there are no packets from guaranteed rate services in the central buffer.

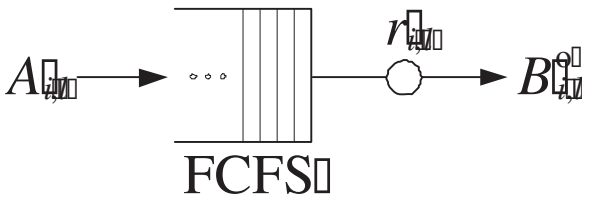


Fig. 3.21. The (fluid) work conserving link corresponding to the $A_{i,\ell}$-flow

To be precise, let $r_{i,\ell}$ be the guaranteed rate of the $A_{i,\ell}$-flow. Now consider feeding the $A_{i,\ell}$-flow to a fluid work conserving link with capacity $r_{i,\ell}$ (see Figure 3.21). Assume that the buffer in the fluid work conserving link is infinite and empty at time 0. Every packet brings in one unit of fluid to the fluid work conserving link. Let $\tilde{B}^o_{i,\ell}(t)$ be the cumulative number of fluid departures at the output by time t. Following the same argument in (3.47) (for a work conserving link with capacity 1), one knows that

$$\tilde{B}^o_{i,\ell}(t) = \min_{0\le s\le t}[A_{i,\ell}(s) + r_{i,\ell}(t-s)]. \tag{3.74}$$

Ideally, the cumulative number of *packet* departures for the guaranteed rate $A_{i,\ell}$-flow by time t should be the integer part of the cumulative number of *fluid* departures, i.e.,

$$B^o_{i,\ell}(t) = \lfloor \tilde{B}^o_{i,\ell}(t) \rfloor. \tag{3.75}$$

Let $d_{i,\ell}(k)$ be the targeted departure time of the k^{th} packet of the $A_{i,\ell}$-flow. If the k^{th} packet arrives at time t, then it can be found by the following inverse mapping (see e.g., [27], Lemma 2.3.20)

$$d_{i,\ell}(k) = \inf\Big[\tau : \tau \ge t, \ \min_{0\le u\le t-1}[A_{i,\ell}(u) + r_{i,\ell}(\tau - u)] \ge k\Big]. \tag{3.76}$$

The eligible time of the k^{th} packet of the $A_{i,\ell}$-flow at the central buffer is then set to be $d_{i,\ell}(k) + (N-1)L_{\max}$.

For the multicasting flows, fan-out splitting is also performed at the central buffer. Thus, a packet departing from the jitter control mechanism is duplicated and distributed to the VOQs corresponding to its destined outputs. By scheduling the first eligible packet in every VOQ, we can show that the maximum delay for a packet to depart the second crossbar switch fabric is bounded. The proof of Lemma 3.4.1 is shown in Section 3.4.1.

Lemma 3.4.1. *Let $S^*(k)$ be the set of flows through the k^{th} output, and $M_k = |S^*(k)|$ be the number of multicasting flows through the k^{th} output port. Define $M_{\max} = \max_{1\le k\le N} M_k$ as the maximum number of multicasting flow through an output port. Suppose that all the buffers are empty at time 0 and*

$$\sum_{(i,\ell)\in S^*(k)} r_{i,\ell} \le 1. \tag{3.77}$$

Then the time for a packet to depart the second crossbar switch fabric is bounded by the sum of its target departure time and $d_1 + d_2$, where $d_1 = (N-1)L_{\max}$ and $d_2 = NM_{\max}$.

After the second crossbar switch fabric, a packet is placed in another jitter control mechanism. As there is a maximum delay for every packet to depart the second crossbar switch fabric, the eligible time for a packet at this jitter control mechanism is set to be the sum of its target departure time and $(N-1)L_{\max} + NM_{\max}$. Once a packet becomes eligible, it is placed in the output buffer. The scheduling policy for the output buffer is also FCFS. As addressed before, one may combine the jitter control mechanism with the FCFS buffer by using the earliest eligible packet first policy. The following is the main theorem of this scheme. The proof of Theorem 3.4.2 is given in Section 3.4.2.

Theorem 3.4.2. *Suppose that all the buffers are empty at time 0 and that the rate condition in (3.77) hold. Then the following results hold.*

(i) Every packet of a guaranteed rate flow departs from the switch not later than the sum of its targeted departure time and $d_1 + d_2 + d_3$, where $d_1 = (N-1)L_{\max}$, $d_2 = NM_{\max}$ and $d_3 = M_{\max} - 1$.

(ii) The output buffer at an output port of the second stage is bounded by $d_2 + d_3 = (N+1)M_{\max} - 1$.

3.4.1 Maximum time to depart from the second switch fabric

In this section, we prove Lemma 3.4.1. For the proof of Lemma 3.4.1, we will need to use the following well-known properties for the ceiling and floor functions.

Proposition 3.4.3. *(i)* $\lceil a+b \rceil \le \lceil a \rceil + \lceil b \rceil \le \lceil a+b \rceil + 1$.
(ii) $\lfloor a+b \rfloor \ge \lfloor a \rfloor + \lfloor b \rfloor$.
(iii) $\lceil a \rceil \le \lfloor a \rfloor + 1$.
(iv) $\lceil a-b \rceil \ge \lfloor a \rfloor - \lfloor b \rfloor$.
(v) $\lceil \frac{\lceil a \rceil}{N} \rceil = \lceil \frac{a}{N} \rceil$ *for any positive integer* N.

Recall that $A^1_{i,\ell,j}(t)$ is the cumulative number of the $A_{i,\ell}$-flow packets that are split into the j^{th} VOQ at the i^{th} input port of the first stage by time t. Let $D_{i,\ell,j}(t)$ be the number of the $A^1_{i,\ell,j}$-flow packets that have targeted departure times not greater than t. Note that the first packet of a flow is always assigned to the first VOQ at the first stage. As the targeted departure times are simply the departure times from the ideal FCFS work conserving link with capacity $r_{i,\ell}$,

$$D_{i,\ell,j}(t) = \lceil \frac{B^o_{i,\ell}(t) - j + 1}{N} \rceil. \tag{3.78}$$

Moreover,

$$\sum_{j=1}^{N} D_{i,\ell,j}(t) = B^o_{i,\ell}(t). \tag{3.79}$$

Let $A^2_{i,\ell,j}(t)$ be the cumulative number of the $A_{i,\ell}$-flow packets at the j^{th} input port of the second stage by time t. We know that the maximum delay at the first stage is bounded by

$$d_1 = (N-1)L_{\max}. \tag{3.80}$$

As discussed in Section 3.4, a jitter control stage is added in front of the VOQs in the second stage (see Figure 3.22) and the eligible time of a packet is set to be the sum of its targeted departure time and the maximum delay d_1. Thus, we have from (3.78) that

$$\begin{aligned} A^2_{i,\ell,j}(t) &= D_{i,\ell,j}(t - d_1) \\ &= \lceil \frac{B^o_{i,\ell}(t - d_1) - j + 1}{N} \rceil. \end{aligned} \tag{3.81}$$

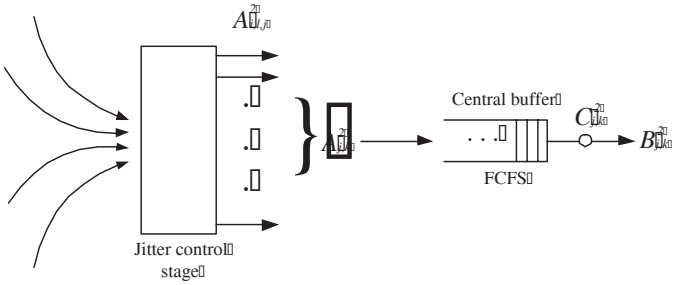


Fig. 3.22. The k^{th} VOQ at the j^{th} central buffer of the second stage

Now consider the k^{th} VOQ at the j^{th} central buffer of the second stage (see Figure 3.22). Denote by $A^2_{j,k}(t)$ (resp. $B^2_{j,k}(t)$) the cumulative number of arrivals (resp. departures) at that VOQ by time t. Then

$$\begin{aligned} A^2_{j,k}(t) &= \sum_{(i,\ell) \in S^*(k)} A^2_{i,\ell,j}(t) \\ &= \sum_{(i,\ell) \in S^*(k)} \lceil \frac{B^o_{i,\ell}(t - d_1) - j + 1}{N} \rceil. \end{aligned} \tag{3.82}$$

Now we show that the traffic coming into this VOQ is rate controlled. Note from (3.82) and Proposition 3.4.3(i) that

$$\begin{aligned} & A^2_{j,k}(t) - A^2_{j,k}(s) \\ &= \sum_{(i,\ell)\in S^*(k)} \lceil \frac{B^o_{i,\ell}(t-d_1) - j + 1}{N} \rceil - \lceil \frac{B^o_{i,\ell}(s-d_1) - j + 1}{N} \rceil \\ &\leq \sum_{(i,\ell)\in S^*(k)} \lceil \frac{B^o_{i,\ell}(t-d_1) - B^o_{i,\ell}(s-d_1)}{N} \rceil. \end{aligned} \tag{3.83}$$

From (3.75), Proposition 3.4.3(iv), and (3.74), it then follows that

$$\begin{aligned} & B^o_{i,\ell}(t-d_1) - B^o_{i,\ell}(s-d_1) \\ &= \lfloor \tilde{B}^o_{i,\ell}(t-d_1) \rfloor - \lfloor \tilde{B}^o_{i,\ell}(s-d_1) \rfloor \\ &\leq \lceil \tilde{B}^o_{i,\ell}(t-d_1) - \tilde{B}^o_{i,\ell}(s-d_1) \rceil \\ &= \lceil \min_{0\leq\tau\leq t-d_1} [A_{i,\ell}(\tau) + r_{i,\ell}(t-\tau)] \\ &\qquad - \min_{0\leq\tau\leq s-d_1} [A_{i,\ell}(\tau) + r_{i,\ell}(s-\tau)] \rceil \\ &\leq \lceil r_{i,\ell}(t-s) \rceil. \end{aligned} \tag{3.84}$$

Replacing this in (3.83) and using Proposition 3.4.3(v) yields

$$\begin{aligned} A^2_{j,k}(t) - A^2_{j,k}(s) &\leq \sum_{(i,\ell)\in S^*(k)} \lceil \frac{\lceil r_{i,\ell}(t-s) \rceil}{N} \rceil \\ &= \sum_{(i,\ell)\in S^*(k)} \lceil \frac{r_{i,\ell}(t-s)}{N} \rceil. \end{aligned} \tag{3.85}$$

Let $C^2_{j,k}(t)$ be the cumulative number of time slots assigned to this VOQ by time t. As the link at the second stage is a FCFS work conserving link, we have from Lemma 2.1.2 that

$$B^2_{j,k}(t) = \min_{0\leq s\leq t} [A^2_{j,k}(s) + C^2_{j,k}(t) - C^2_{j,k}(s)]. \tag{3.86}$$

Moreover, as the connection patterns at the second stage are periodic with period N for N permutation matrices satisfying (3.1),

$$C^2_{j,k}(t) - C^2_{j,k}(s) \geq \lfloor \frac{t-s}{N} \rfloor. \tag{3.87}$$

In order to prove that the maximum delay for a packet to depart the second crossbar switch fabric is bounded by the sum of its target departure time and $(N-1)L_{\max} + NM_{\max}$, it suffices to show that the

maximum delay incurred at the VOQ is bounded above by $NM_{\max}$, i.e.,

$$B^2_{j,k}(t + NM_{\max}) - A^2_{j,k}(t) \geq 0.$$

Let $d_2 = NM_{\max}$. Note from (3.86) that

$$\begin{aligned}
&B^2_{j,k}(t + d_2) - A^2_{j,k}(t) \\
&= \min_{0 \leq s \leq t+d_2} [A^2_{j,k}(s) - A^2_{j,k}(t) + C^2_{j,k}(t + d_2) - C^2_{j,k}(s)] \\
&= \min \Big[\min_{0 \leq s \leq t} [A^2_{j,k}(s) - A^2_{j,k}(t) + C^2_{j,k}(t + d_2) - C^2_{j,k}(s)], \\
&\quad \min_{t+1 \leq s \leq t+d_2} [A^2_{j,k}(s) - A^2_{j,k}(t) + C^2_{j,k}(t + d_2) - C^2_{j,k}(s)] \Big].
\end{aligned} \tag{3.88}$$

All the terms in the second minimum are clearly nonnegative as both $A^2_{j,k}(t)$ and $C^2_{j,k}(t)$ are non-decreasing in t. On the other hand, for $0 \leq s \leq t$, we have from (3.87) and (3.77) that

$$\begin{aligned}
&C^2_{j,k}(t + d_2) - C^2_{j,k}(s) \geq \lfloor \frac{t - s + d_2}{N} \rfloor \\
&\geq \lfloor \frac{(\sum_{(i,\ell) \in S^*(k)} r_{i,\ell}(t - s)) + NM_k}{N} \rfloor \\
&= \lfloor \frac{\sum_{(i,\ell) \in S^*(k)} (r_{i,\ell}(t - s) + N)}{N} \rfloor.
\end{aligned}$$

Using (3.85) and Proposition 3.4.3(ii) ,(iii) yields

$$\begin{aligned}
&C^2_{j,k}(t + d_2) - C^2_{j,k}(s) - (A^2_{j,k}(t) - A^2_{j,k}(s)) \\
&\geq \sum_{(i,\ell) \in S^*(k)} \lfloor \frac{r_{i,\ell}(t - s) + N}{N} \rfloor - \sum_{(i,\ell) \in S^*(k)} \lceil \frac{r_{i,\ell}(t - s)}{N} \rceil \\
&= \sum_{(i,\ell) \in S^*(k)} (\lfloor \frac{r_{i,\ell}(t - s)}{N} \rfloor + 1) - \sum_{(i,\ell) \in S^*(k)} \lceil \frac{r_{i,\ell}(t - s)}{N} \rceil \\
&\geq 0.
\end{aligned}$$

3.4.2 Maximum time to depart from the whole switch

In this section, we prove Theorem 3.4.2.

(i) Let $B^3_k(t)$ be the cumulative departures by time t to the jitter control mechanism at the k^{th} output port. Note that $B^3_k(t)$ is also the cumulative arrivals by time t to the k^{th} output buffer. Denote by $B^3_{i,\ell}(t)$ the cumulative arrivals of the $A_{i,\ell}$-flow by the time t to the output buffer. Thus, we have

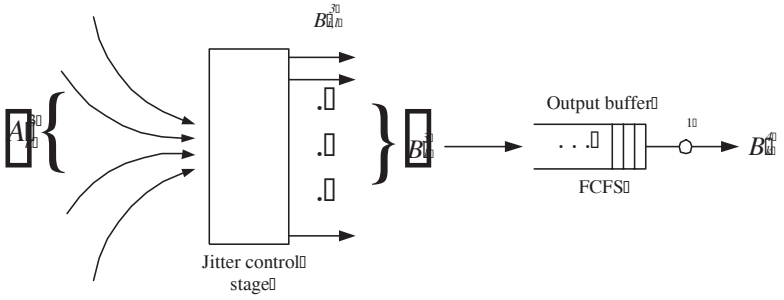


Fig. 3.23. The output buffer

$$B_k^3(t) = \sum_{(i,\ell)\in S^*(k)} B_{i,\ell}^3(t). \tag{3.89}$$

Let $B_k^4(t)$ be the cumulative departures by time t from the k^{th} output buffer. From the representation for Lindley's equation in Lemma 2.1.2, it follows that

$$B_k^4(t) = \min_{0\le s\le t}[B_k^3(s) + (t-s)]. \tag{3.90}$$

As shown in Lemma 3.4.1, the maximum delay for a packet to depart the second crossbar switch fabric is bounded by the sum of its target departure time and $(N-1)L_{\max} + NM_{\max}$. The eligible time for an $A_{i,\ell}$-flow packet is set to be the sum of its target departure time and $(N-1)L_{\max} + NM_{\max}$. Thus, we have

$$B_{i,\ell}^3(t) = B_{i,\ell}^o(t - d_1 - d_2), \tag{3.91}$$

where $d_1 = (N-1)L_{\max}$ and $d_2 = NM_{\max}$. To show that every packet of a guaranteed rate flow departs from the switch not later than the sum of its targeted departure time and $(N-1)L_{\max}+(N+1)M_{\max}-1$, it suffices to show that there is a bounded delay d_3 at the output buffer, where $d_3 = M_{\max} - 1$. As the proof for Lemma 3.4.1, we only need to verify that

$$B_k^4(t+d_3) - B_k^3(t) \ge 0. \tag{3.92}$$

Note from (3.90) that

$$\begin{aligned} &B_k^4(t+d_3) - B_k^3(t) \\ &= \min_{0\le s\le t+d_3}[B_k^3(s) - B_k^3(t) + (t+d_3-s)] \\ &= \min\Big[\min_{0\le s\le t}[B_k^3(s) - B_k^3(t) + (t+d_3-s)], \end{aligned}$$

$$\min_{t+1\le s\le t+d_3}\big[B_k^3(s)-B_k^3(t)+(t+d_3-s)\big]. \tag{3.93}$$

As in the proof of Lemma 3.4.1 in Section 3.4.1, all the terms in the second minimum are clearly nonnegative. Thus, we only need to verify the case for $0 \le s \le t$. Using the inequality in (3.84), one can show from (3.89) and (3.91) that

$$\begin{aligned} & B_k^3(t) - B_k^3(s) \\ &= \sum_{(i,\ell)\in S^*(k)} \Big(B_{i,\ell}^o(t-d_1-d_2) - B_{i,\ell}^o(s-d_1-d_2)\Big) \\ &\le \sum_{(i,\ell)\in S^*(k)} \lceil r_{i,\ell}(t-s)\rceil. \end{aligned} \tag{3.94}$$

Applying Proposition 3.4.3(i) and the assumption in (3.77) yields

$$\begin{aligned} & B_k^3(t) - B_k^3(s) \\ &\le \lceil \sum_{(i,\ell)\in S^*(k)} r_{i,\ell}(t-s)\rceil + M_k - 1 \\ &\le (t-s) + M_{\max} - 1 \end{aligned} \tag{3.95}$$

Thus, all the terms in the first minimum are also nonnegative.

(ii) Let $A_k^3(t)$ be the cumulative arrivals by time t to the jitter control mechanism at the k^{th} output port. Note that there is already a jitter control mechanism in front of the center buffer. Thus, a packet cannot arrive at the jitter control mechanism at an output port before its eligible time set by the jitter control mechanism in front of the center buffer. Since the eligible time at the first jitter control mechanism is the sum of its targeted departure time and $(N-1)L_{\max}$, we then have

$$A_k^3(t) \le \sum_{(i,\ell)\in S^*(k)} B_{i,\ell}^o(t-d_1), \tag{3.96}$$

where $d_1 = (N-1)L_{\max}$. Using (3.96), (3.90), (3.91) and (3.95), the number of packet stored at the k^{th} output buffer at time t is then bounded by

$$\begin{aligned} & A_k^3(t) - B_k^4(t) \\ &\le \sum_{(i,\ell)\in S^*(k)} B_{i,\ell}^o(t-d_1) \\ &\qquad - \min_{0\le s\le t}[\sum_{(i,\ell)\in S^*(k)} B_{i,\ell}^o(s-d_1-d_2) + (t-s)] \end{aligned}$$

$$
\begin{aligned}
&= \max_{0\le s\le t}[\sum_{(i,\ell)\in S^*(k)} (B^o_{i,\ell}(t-d_1) - B^o_{i,\ell}(s-d_1-d_2)) - (t-s)] \\
&\le \max_{0\le s\le t}[(t-s+d_2) + M_{\max} - 1 - (t-s)] \\
&= d_2 + M_{\max} - 1.
\end{aligned}
$$

3.5 Frame based schemes

The drawback of the previous schemes in Section 3.3 and Section 3.4 is its hardware implementation complexity of their scheduling policies. The objective of this section is to propose a much simpler scheme that does not require implementing complicated scheduling. The idea (as in Keslassy and McKeown [98]) is to use a framed structure so that resequencing is not needed. The architecture of the scheme is shown in Figure 3.24.

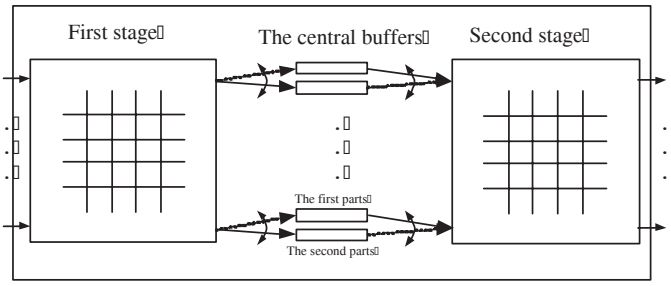


Fig. 3.24. The architecture for the frame based scheme

To ease our presentation, we shall describe the scheme for fixed size packets and point-to-point flows. Extensions to variable length packets and multicasting flows will be left to the readers for an exercise. As in the two-stage switch in the previous section, there are two $N \times N$ crossbar switch fabrics and buffers between these two crossbar switch fabrics. In this scheme, time slots are grouped into fixed size frames. Each frame has F time slots. Thus, the m^{th} time frame is from time slot $(m-1)F+1$ to time slot mF (see Figure 3.25).

Unlike the schemes in Section 3.3 and Section 3.4, both switches now change their connection patterns according to time frames. Specifically, both switches are symmetric TDM switches operated in the time

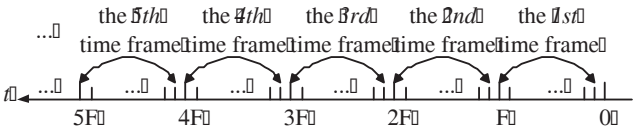


Fig. 3.25. The time frame structure

scale of frames, i.e., during the m^{th} time frame, the i^{th} input port is connected to the $h(i,m)^{th}$ output port of these two crossbar switch fabrics, where

$$h(i,m) = \Big((m-i) \bmod N\Big) + 1. \tag{3.97}$$

As such, all the packets that arrive at the i^{th} input port during the m^{th} frame are all routed to the $h(i,m)^{th}$ output port. As the connection pattern is *symmetric*, during the m^{th} frame the k^{th} output port is also connected to the $h(k,m)^{th}$ input port for these two switch fabrics.

There are N central buffers between these two switch fabrics, indexed from 1 to N. Each central buffer consists of two alternating memory blocks. The buffer size of each memory block is NF, which is divided into N bins, each with buffer size of F. To ease the presentation for the operation of these central buffers, we introduce the concept of superframes. The p^{th} superframe of the i^{th} input port of the both stages is defined to be the set of time slots in the N time frames, starting from the $((p-1)N+i)^{th}$ frame to the $(pN+i-1)^{th}$ frame. Note that the j^{th} time frame in the p^{th} superframe of the i^{th} input port (of both stages) is the $((p-1)N+i+j-1)^{th}$ frame. Since $h(i,(p-1)N+i+j-1) = j$, it follows that *during the j^{th} time frame in the p^{th} superframe of the i^{th} input port, the i^{th} input port is always connected to the j^{th} output port*. Moreover, the j^{th} time frame in the p^{th} superframe of the i^{th} input port is also the i^{th} frame in the p^{th} superframe of the j^{th} input port.

Consider a particular packet that arrives at the i^{th} input port of the first stage during the j^{th} time frame in the p^{th} superframe of the i^{th} input port. As just described, the i^{th} input is connected to the j^{th} output during that frame and the packet is thus sent to the j^{th} central buffer without delay. As there are two alternating memory blocks in the j^{th} central buffer, the packet is sent to the second (resp. first) memory block if p is odd (resp. even). If, furthermore, the packet is destined for the k^{th} output port, it will be placed in the k^{th} bin of that memory block. As each bin only has the buffer size F, one might

wonder whether there is enough buffer space for such an assignment. We will show in Theorem 3.5.2 that under certain traffic assumptions there are no packet overflows for such an assignment. For the time being, reader might simply consider that overflowed packets are lost.

Without loss of generality, let us assume that p is *odd* and the packet is placed in the k^{th} bin of the *second* memory block of the j^{th} central buffer. During the k^{th} time frame in the $(p+1)^{th}$ superframe of the j^{th} input port of the second stage, the j^{th} input port of the second stage is connected to the k^{th} output of the second stage. As each frame has F time slots and each bin can hold at most F packets, during that frame all the packets in the k^{th} bin of the second memory block of the j^{th} central buffer are transmitted to the k^{th} output of the second stage.

Example 3.5.1. We illustrate this scheme by a 4×4 switch fabric. In Figure 3.26, we show the operation for the first stage. We denote by $I(i,m)$ the set of packets that arrive at the i^{th} input port of the first stage during the m^{th} time frame, and $I_s(i,p)$ the set of packets that arrive the i^{th} input port of the first stage during the p^{th} superframe of the i^{th} input port. Note that $I(1,1)$, $I(2,2)$, $I(3,3)$ and $I(4,4)$ are all routed to the second memory block of the *first* central buffer. Each of the four frames is the *first* frame in the superframe of its input. Upon the arrival of each packet in these four frames, it is placed immediately in the bin that corresponds to its destined output. At the end of the first superframe of the first input (i.e., the end of the 4^{th} frame), all the packets in the bins of the second memory block of the first central buffer are well packed and ready to be transmitted to the second stage. Similarly, $I(1,2)$, $I(2,3)$, $I(3,4)$ and $I(4,5)$ are all routed to the second memory block of the *second* central buffer, $I(1,3)$, $I(2,4)$, $I(3,5)$ and $I(4,6)$ are all routed to the second memory block of the *third* central buffer, and $I(1,4)$, $I(2,5)$, $I(3,6)$ and $I(4,7)$ are all routed to the second memory block of the *fourth* central buffer.

In Figure 3.27, we illustrate the operation for the second stage. We denote by $O(j,m)$ the set of packets that depart from the j^{th} *input* port of the second stage during the m^{th} time frame, and $O_s(j,p)$ the set of packets that depart from the j^{th} *input* port of the second stage during the p^{th} superframe of the j^{th} input port. Now consider the four bins at the second memory block of the first central buffer. Since they are ready at the end of the first superframe of the first input, packets

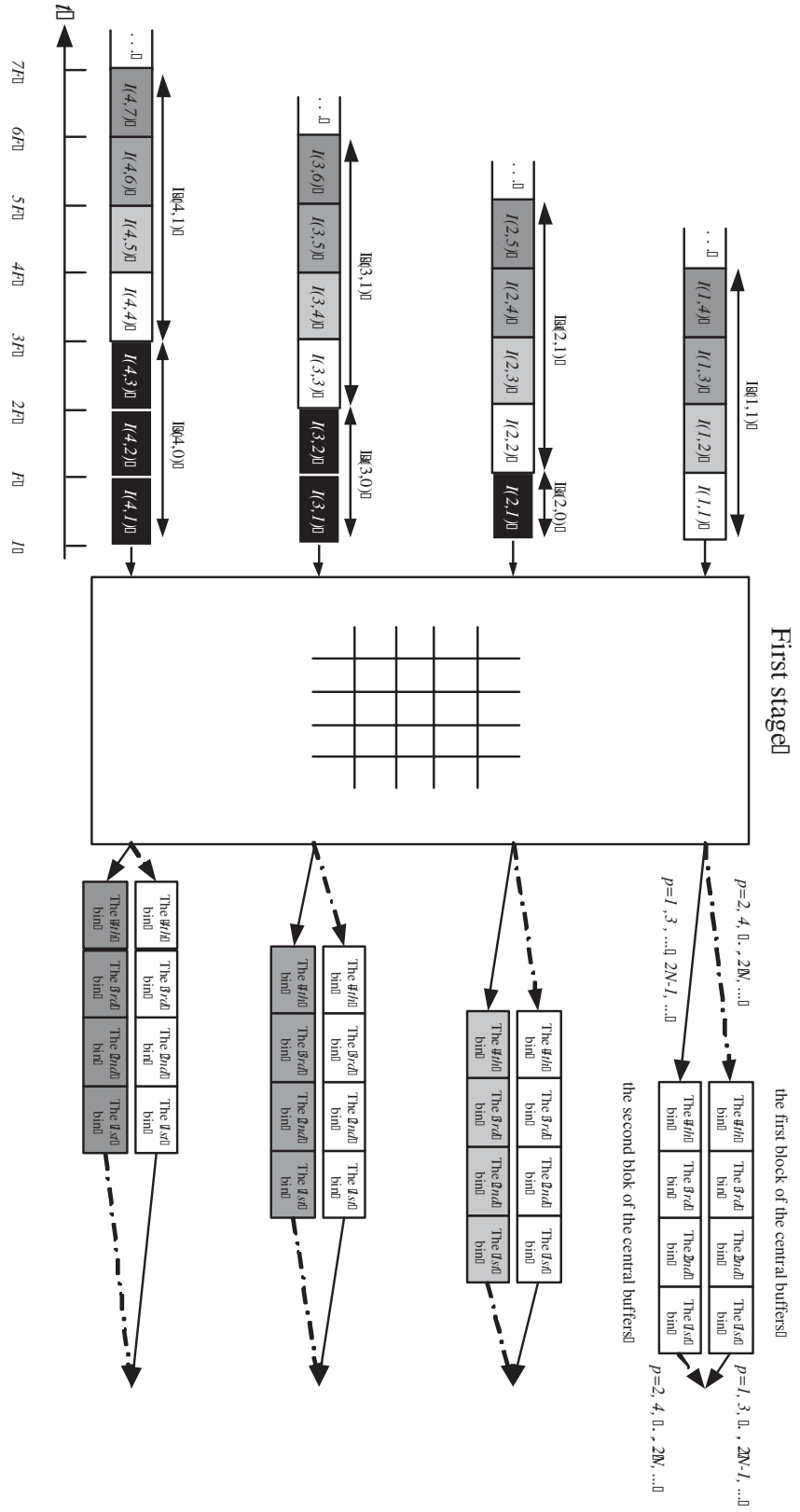
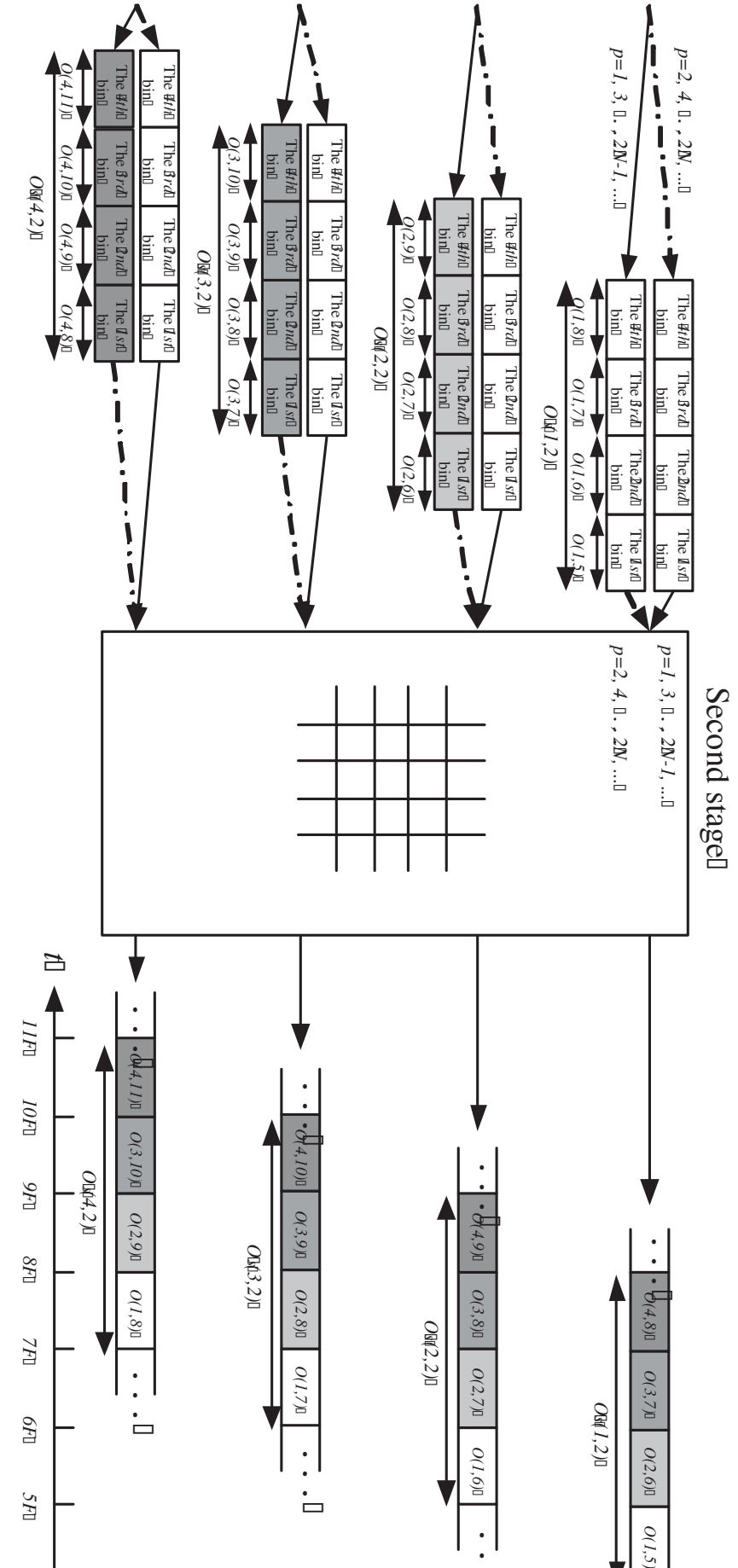


Fig. 3.26. The first stage of a 4×4 switch fabric


Second stage
p=1, 3, . . . , 2N-1, ...
p=2, 4, . . . , 2N, ...
The 4th bin
The 3rd bin
The 2nd bin
The 1st bin
O(1,8)
O(1,7)
O(1,6)
O(1,5)
O(1,2)
O(2,9)
O(2,8)
O(2,7)
O(2,6)
O(2,2)
O(3,10)
O(3,9)
O(3,8)
O(3,7)
O(3,2)
O(4,11)
O(4,10)
O(4,9)
O(4,8)
O(4,2)
t
11T
10T
9T
8T
7T
6T
5T

in the *first* bin are routed to the *first* output during the *first* frame of the second superframe of the first input, i.e., the 5^{th} frame. Similarly, packets in the *second* bin are routed to the *second* output during the *second* frame of the second superframe of the first input, i.e., the 6^{th} frame, packets in the *third* bin are routed to the *third* output during the *third* frame of the second superframe of the first input, i.e., the 7^{th} frame, and packets in the *fourth* bin are routed to the *fourth* output during the *fourth* frame of the second superframe of the first input, i.e., the 8^{th} frame. In other words, $O(1,5)$ contains the packets in the first bin, $O(1,6)$ contains the packets in the second bin, $O(1,7)$ contains the packets in the third bin, and $O(1,8)$ contains the packets in the fourth bin.

The four bins in the second memory block of the *second* central buffer are ready at the end of the 5^{th} frame. These four bins are routed to the first output during the 6^{th} frame, the second output during the 7^{th} frame, the third output during the 8^{th} frame and the fourth output during the 9^{th} frame. The operation for the other two central buffers are done in a similar manner as shown in Figure 3.27.

3.5.1 Regulated inputs

In this section, we show that if the input traffic is regulated to satisfy certain technical conditions, then it is possible to guarantee that no packets are lost inside the switch.

(A1) Let $A_{i,k}$-flow be the flow of packets that arrive at input port i and are destined for output port k. Suppose that the number of packets from the $A_{i,k}$-flow during a frame is bounded above by $M_{i,k}$, $i = 1, 2, \ldots, N$ and $k = 1, 2, \ldots, N$.
(A2) $\sum_{k=1}^{N} M_{i,k} \leq F$, for $i = 1, 2 \ldots, N$.
(A3) $\sum_{i=1}^{N} M_{i,k} \leq F$, for $k = 1, 2, \cdots, N$.
(A4) All the buffers are empty at the beginning.

Note that (A2) and (A3) are equivalent to the "no overbooking" conditions in (2.27) and (2.28).

Theorem 3.5.2. *Assume that (A1)-(A4) hold. A packet that arrives at the i^{th} input and is destined to the k^{th} output during the j^{th} time frame in the p^{th} superframe of the i^{th} input of the first stage (i.e.,*

the $((p-1)N+i+j-1)^{th}$ *time frame) will depart during the* k^{th} *time frame in the* $(p+1)^{th}$ *superframe of the* j^{th} *input of the second stage (i.e., the* $(pN+j+k-1)^{th}$ *time frame), for* $i=1,2,\cdots,N$ *and* $k=1,2,\cdots,N$.

There are several consequences of Theorem 3.5.2.

(i) Even though the central buffer is finite, no packets are lost inside the switch.

(ii) Packets of the same flow (the same i and k) depart in the FCFS order. This is trivial for packets of the same flow that arrive within the same frame. For packets of the same flow that arrive in different frames, one can see from Theorem 3.5.2 that the departure time of a packet is increasing in both j and p.

(iii) From Theorem 3.5.2, the maximum delay for all arrivals from the i^{th} input port to the k^{th} output port through the switch fabric is bounded by

$$\begin{aligned}&(pN+j+k-1)F\\&\quad -((p-1)N+i+j-1)F+F\\&=(N+k-i+1)F.\end{aligned}\tag{3.98}$$

Thus, the maximum delay for all arrivals from the i^{th} input port through the switch fabric is bounded by $(2N-i+1)F$, which in turn is bounded above by $2NF$.

(iv) In comparison with the framed Birkhoff-von Neumann switch in Section 2.3.4, the framed Birkhoff-von Neumann switch needs to know the rate matrix and carries out the Birkhoff-von Neumann decomposition. Incremental updates in the framed Birkhoff-von Neumann switch can then be carried out by the Slepian-Duguid algorithm. None of the above is needed for the framed based scheme discussed in this section. The drawback of the framed based scheme discussed in this section is the large packet delay (depending on the frame size F and the switch size N).

Proof. (Theorem 3.5.2)

From (A1), the number of packets of the $A_{i,k}$-flow that arrive during the j^{th} time frame in the p^{th} superframe of the i^{th} input port of the first stage (i.e., the $((p-1)N+i+j-1)^{th}$ time frame) is bounded by $M_{i,k}$. Without loss of generality, assume that p is odd. The total number of packets that are placed in the k^{th} bin of the second memory block of

the j^{th} central buffer during the p^{th} superframe of the j^{th} input port of the second stage is not greater than $\sum_{i=1}^{N} M_{i,k}$. From (A3), we know that

$$\sum_{i=1}^{N} M_{i,k} \leq F.$$

Thus, if the k^{th} bin of the second memory block of the j^{th} buffer is empty at the beginning of the p^{th} superframe of the j^{th} input port of the second stage, then all of the packets that arrive during this superframe can be placed in that bin without causing buffer overflow. During the k^{th} time frame in the $(p+1)^{th}$ superframe of the j^{th} input port of the second stage (i.e., the $(pN + j + k - 1)^{th}$ time frame), all of packets in that bin are routed to the k^{th} output port of the second stage. As a result, the k^{th} bin of the second memory block of the j^{th} buffer is *empty* again at the beginning of the $(p + 2)^{th}$ superframe of the j^{th} input port of the second stage! By induction, all packets of the $A_{i,k}$-flow in the j^{th} time frame of the p^{th} superframe of the i^{th} input port of the first stage (i.e., the $((p-1)N+i+j-1)^{th}$ time frame) will depart during the k^{th} time frame in the $(p+1)^{th}$ superframe of the j^{th} input of the second stage (i.e., the $(pN + j + k - 1)^{th}$ time frame), for $k = 1, 2, \cdots, N$ and $i = 1, 2, \cdots, N$.

The argument for the case that p is even is similar.

■

3.5.2 Input-buffered switches with head-of-line blocking

In Section 3.5.1, traffic is regulated in the way so that no packets are lost inside the switch. In this section, we provide a scheme without traffic regulation.

The idea is that an input and its corresponding output are usually built on the same line card and the symmetric connection patterns in the two symmetric TDM switches set up a feedback path from the central buffers to an input/output port. As such, once a packet is lost in a central buffer, the central buffer can notify the input port that sends the packet and the packet can be retransmitted in the next frame.

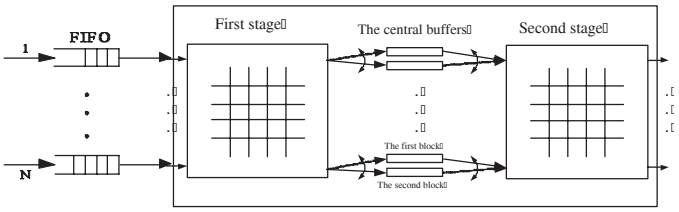


Fig. 3.28. The architecture for the frame based scheme with head-of-line blocking

A scheme described above requires adding input buffers at input ports to hold packets that require retransmitting (see Figure 3.28). During each frame, (at most) F packets can be transmitted from an input port to a central buffer. Some of these packets may be placed in the central buffer successfully and some of them may be dropped due to buffer overflows. Those dropped packets requires retransmitting in the next frame and might cause the head-of-line blocking problem as described in Section 2.2.2.

To find the maximum throughput for such an input-buffered scheme, we follow the analysis in Section 2.2.2. Assume that there are infinite number of packets at each input port and each packet selects its output *independently* and *uniformly*. Let $q(m)$ be the number of head-of-line (HOL) packets that are destined for output port 1 during the m^{th} frame. As there is at most F packets that can be transmitted to output port 1, we then have the following (modified) Lindley equation:

$$q(m+1) = (q(m) - F)^+ + a(m). \tag{3.99}$$

where $a(m)$ is the number of packets that become HOL packets and choose output port 1 as their destination during the m^{th} frame. As in Section 2.2.2, the sequence $\{a(m), m \geq 1\}$ can be viewed as a sequence of independent Poisson random variables with mean ρF, where ρ is the maximum throughput that we would like to find.

As long as $\rho < 1$, we know from Theorem 2.1.4 that $q(m)$ converges in distribution to a unique steady state random variable $q(\infty)$ that satisfies

$$q(\infty) =_{st} (q(\infty) - F)^+ + a(1), \tag{3.100}$$

where $=_{st}$ denotes the stochastic identity, i.e., $X =_{st} Y$ if the two random variables X and Y have the same distribution. The random

variable $a(1)$ is a Poisson random variable with mean ρF and it is independent of $q(\infty)$.

On the other hand, as the total number of head-of-line packets in a frame is NF, we have from symmetry that the expected number of head-of-line packets destined for output port 1 in a frame is F, i.e.,

$$\mathsf{E}q(\infty) = F. \tag{3.101}$$

Unfortunately, unlike (2.32), there is no closed form expression for $\mathsf{E}q(\infty)$ in (3.100) (see Problem 10 and Problem 11) for a numerical procedure to compute the maximum throughput). In the following lemma, we show a lower bound for the maximum throughput.

Lemma 3.5.3. *The maximum throughput ρ that satisfies (3.100) and (3.101) has the following lower bound:*

$$\rho \geq 1 - \frac{2}{\sqrt{8F+1}+1}. \tag{3.102}$$

Proof. Let $Z = q(\infty) - F$ and $Y = a(1) - F$. From (3.100), it follows that

$$Z =_{st} Z^+ + Y, \tag{3.103}$$

and Y is independent of Z^+. Note that $Z = Z^+ - Z^-$ (with $Z^- = \max(0, -Z)$). Taking expectation on both sides of (3.103) yields

$$\mathsf{E}Z^- = -\mathsf{E}Y. \tag{3.104}$$

Also, note that $Z^2 = (Z^+)^2 + (Z^-)^2$ as $Z^+Z^- = 0$. Thus, squaring both sides of (3.103) and taking expectation yields

$$\mathsf{E}(Z^-)^2 = 2\mathsf{E}Z^+\mathsf{E}Y + \mathsf{E}Y^2. \tag{3.105}$$

As the variance of a random variable is nonnegative, we then have from (3.105) and (3.104) that

$$\begin{aligned} 0 &\leq \mathsf{E}(Z^-)^2 - (\mathsf{E}Z^-)^2 \\ &= 2\mathsf{E}Z^+\mathsf{E}Y + \mathsf{E}Y^2 - (\mathsf{E}Z^-)^2 \\ &= 2\mathsf{E}Z^+\mathsf{E}Y + \mathsf{E}Y^2 - (\mathsf{E}Y)^2 \end{aligned}$$

Thus,

$$\mathsf{E}Z^+ \leq -\frac{\mathsf{E}Y^2 - (\mathsf{E}Y)^2}{2\mathsf{E}Y}. \tag{3.106}$$

As $a(1)$ is a Poisson random variable with mean ρF, we have $\mathsf{E}Y = \rho F - F$ and $\mathsf{E}Y^2 - (\mathsf{E}Y)^2 = \rho F$. From (3.103) and (3.106), it then follows

$$\begin{aligned}\mathsf{E}q(\infty) &\leq -\frac{\mathsf{E}Y^2 - (\mathsf{E}Y)^2}{2\mathsf{E}Y} + \mathsf{E}Y + F \\ &\leq \rho F + \frac{\rho}{2(1-\rho)}. \end{aligned} \tag{3.107}$$

Since $\mathsf{E}q(\infty) = F$ in (3.101), it then follows from (3.107) that

$$2F\rho^2 - (1+4F)\rho + 2F \leq 0. \tag{3.108}$$

Solving (3.108) yields the lower bound in (3.102). ■

In addition to the lower bound in Lemma 3.5.3, we provide an approximation for the maximum throughput when F is large. The idea is to assume that the distribution of $Z(\infty) = (q(\infty) - F)/\sqrt{F}$ is approximately normal with zero mean and variance σ^2. We can rewrite (3.100) into

$$Z(\infty) =_{st} (Z(\infty))^+ + \frac{a(1) - F}{\sqrt{F}}. \tag{3.109}$$

Since $Z(\infty)$ is assumed to be normal with zero mean and variance σ^2, we have

$$E[(Z(\infty))^+] = \frac{\sigma}{\sqrt{2\pi}}.$$

Taking expectation on both sides of (3.109) yields

$$0 = \frac{\sigma}{\sqrt{2\pi}} + (\rho - 1)\sqrt{F}. \tag{3.110}$$

Also, note that

$$E[((Z(\infty))^+)^2] = \frac{\sigma^2}{2}.$$

Squaring both sides of (3.109) and taking expectation yields

$$\sigma^2 = \frac{\sigma^2}{2} + (\rho + F(1-\rho)^2) + 2 \cdot \sqrt{\frac{F}{2\pi}}(\rho - 1)\sigma, \tag{3.111}$$

where we use the fact that $a(1)$ is a Poisson random variance with mean ρF and it is independent of $Z(\infty)$. Solving (3.110) and (3.111) for ρ, we have

$$\rho = \frac{2(\pi+1)F + 1 - \sqrt{4(\pi+1)F + 1}}{2(\pi+1)F} \tag{3.112}$$

$$\sigma = (1-\rho)\sqrt{2\pi F}. \tag{3.113}$$

In Table 3.2, we show that both the lower bound in (3.102) and the approximation in (3.112) are quite accurate when comparing with simulations for large F. In our simulations, the size of the switch N is chosen to be 128.

F	Eq. (3.112)	Eq.(3.102)	simulation (N=128)
1	0.614735	0.500000	0.59
3	0.753705	0.666667	0.74
5	0.803072	0.729844	0.79
7	0.830724	0.766077	0.82
10	0.856217	0.800000	0.85
30	0.914221	0.878965	0.91
60	0.938543	0.912785	0.94
90	0.949528	0.928190	0.95

Table 3.2. Comparison of bounds and approximations with simulation results

3.6 Mailbox switches

Instead of using the frame based schemes to solve the re-sequencing problem, in this section we introduce another alternative, the mailbox switches in [36]. The mailbox switch has the same architecture as the load balanced Birkhoff-von Neumann switch. Instead of using an arbitrary set of periodic connection patterns generated by a one-cycle permutation matrix, the key idea in the mailbox switch is to use the *symmetric TDM* switches. As an input and its corresponding output are usually built on the same line card, the symmetric connection patterns set up a feedback path from the central buffers to an input/output port. **Since everything inside the switch is pre-determined and periodic, the scheduled packet departure times can then be fed back to inputs to compute the waiting time for the next packet so that packets can depart in sequence**. Thus, the communication overhead incurred by this is the transmission of the information of the packet departure time, which is a constant independent of the size of the switch. On the other hand, the computation overhead incurred by this is the computation of the waiting time, which also requires only a constant number of operations. As the frame based scheme in Section 3.5.2, simplicity comes at the cost of throughput. The throughput of the mailbox switch is no

longer 100%. Both the theoretical models and simulations show that it can achieve more than 75% throughput. Theoretical results also show that a special case of the mailbox switch reduces to the classical head-of-line blocking switch in Section 2.2.2 that yields 58% throughput. By allowing limited resequencing delay, a modified version of the mailbox switch can achieve more than 95% throughput.

3.6.1 Generic mailbox switch

As in the load balanced Birkhoff-von-Neumann switch, the $N \times N$ mailbox switch consists of two $N \times N$ crossbar switch fabrics and buffers between the two crossbar switch fabrics. The buffers between the two switch fabrics are called mailboxes. There are N mailboxes, indexed from 1 to N. Each mailbox contains N bins (indexed from 1 to N), and each bin contains F cells (indexed from 1 to F). Each cell can store exactly one packet. Cells in the i^{th} bin of a mailbox are used for storing packets that are destined for the i^{th} output port of the second switch. In addition to these, a First In First Out (FIFO) queue is added in front of each input port of the first stage.

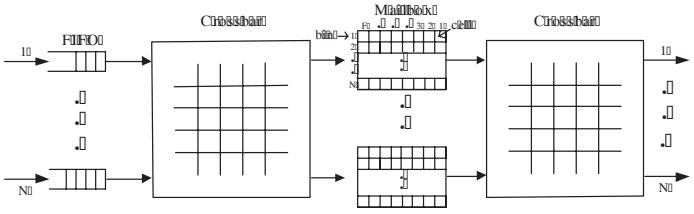


Fig. 3.29. The switch architecture of the mailbox switch

Both the switch fabrics are *symmetric Time Division Multiplexing (TDM)* switches in Section 3.2.2.

Thus, during the t^{th} time slot the i^{th} input port is connected to the $h(i,t)^{th}$ output port of these two crossbar switch fabrics, where

$$h(i,t) = \Big((t-i) \bmod N\Big) + 1. \tag{3.114}$$

As input port i of the first switch and output port i of the second switch are on the same line card, the symmetric property then enables us to establish a bi-directional communication link between a line card and a mailbox. As we will see later, such a property plays an important role in keeping packets in sequence.

As the connection patterns in the mailbox switch is a special case of the load-balanced Birkhoff-von Neumann switch with one-stage buffering [34], one might expect that it also approaches 100% throughput if we use the FIFO policy for each bin and increase the bin size F to ∞. However, we also suffer from the out-of-sequence problem by doing this. Packets that have the same input port at the first switch and the same output port at the second switch may be routed to different mailboxes and depart in a sequence that is different from the sequence of their arrivals at the input port of the first switch.

To solve the out-of-sequence problem, one may add a resequencing buffer and adapt a more careful load balancing mechanism as in the load balanced Birkhoff-von Neumann switch with multi-stage buffering [35]. However, such an approach requires complicated scheduling and jitter control in order to have a bounded resequencing delay. Here we take a much simpler approach.

The idea is that we do know the packet departure time once it is placed in a mailbox as the connection patterns are deterministic and periodic. Also, as 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 information of packet departure times can be fed back to the inputs so that packets can be scheduled in the order of their arrivals.

To be specific, define flow (i, j) as the sequence of packets that arrives at the i^{th} input port of the first switch and are destined for the j^{th} output port of the second switch. Let $V_{i,j}(t)$ be the number of time slots that a packet of flow (i, j) has to wait once it becomes the head-of-line (HOL) packet at the FIFO queue of the i^{th} input port of the first stage at time t. Following the terminology in queueing theory, we call $V_{i,j}(t)$ the virtual waiting time of flow (i, j). Now we describe how the mailbox switch works to keep packets of the same flow in sequence.

(iA) **Retrieving mails**: at time t, the i^{th} output port of the second switch is connected to the $h(i, t)^{th}$ mailbox. The packet in the first cell of the i^{th} bin is transmitted to the i^{th} output port. Packets in cells $2, 3, \ldots, F$ of the i^{th} bin are moved forward to cells 1,2,. . . , $F - 1$.

(iiA) **Sending mails**: suppose that the HOL packet of the i^{th} input port of the first switch is from flow (i, j). Note that the i^{th} input port of the first switch is also connected to the $h(i,t)^{th}$ mailbox. In order to keep packets in sequence, this HOL packet is placed in the first empty cell of the j^{th} bin of the $h(i,t)^{th}$ mailbox such that it will depart not 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 queue.

(iiiA) **Updating virtual waiting times**: all the flows that do not send mails (packets) at time t decrease their virtual waiting time by 1. This includes flows that have blocked transmissions. To update the virtual waiting time for flow (i, j), suppose that the HOL packet is placed in the f^{th} cell of the j^{th} 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 j^{th} output port of the second switch at $t + ((j - i - 1) \bmod N) + 1$. Thus, the departure time for this packet is simply $t + (f - 1)N + ((j - i - 1) \bmod N) + 1$. As such, the number of time slots that has to be waited at $t+1$ for flow (i, j) is $(f-1)N+((j-i-1) \bmod N)$ and we have

$$V_{i,j}(t + 1) = (f - 1)N + ((j - i - 1) \bmod N). \tag{3.115}$$

3.6.2 Mailbox switch with cell indexes

In view of (3.115), there is a simple way to represent the virtual waiting time of a flow. The virtual waiting time $V_{i,j}(t + 1)$ can be written as a sum of two components: $(f - 1)N$ and $((j - i - 1) \bmod N)$. The first term is only a function of the cell index f and the second term is a number between 0 and $N - 1$. This leads to a much easier way to implement the mailbox switch. Define $f_{i,j}(t)$ to be the smallest index of the cell such that the HOL packet will not depart earlier than $t+V_{i,j}(t)$ if the HOL packet is placed in that cell. To simplify our representation, we call $f_{i,j}(t)$ the cell index of $V_{i,j}(t)$. Now we modify the second and the third phase as follows:

(iiB) **Sending mails**: suppose that the HOL packet of the i^{th} input port of the first switch is from flow (i, j). This HOL packet is sent to the $h(i,t)^{th}$ mailbox along with $f_{i,j}(t)$. This packet is then placed in the first empty cell of the j^{th} bin with the cell index not smaller than $f_{i,j}(t)$. If successful, the index of that cell, say

f, is transmitted to the i^{th} output port of the second switch. If no such empty cell can be found, an error message, say $f = 0$, is transmitted to the i^{th} output port of the second switch to indicate a HOL blocking.

(iiiB) **Updating virtual waiting times**: in addition to the cell index of the virtual waiting time $f_{i,j}(t)$, we also keep a counter $g_{i,j}(t)$ for flow (i, j). If flow (i, j) has a successful transmission of a packet at time t, then $f_{i,j}(t + 1)$ is set by the index f returned by the mailbox at time t and $g_{i,j}(t+1)$ is reset to N. On the other hand, if flow (i, j) does not have a successful transmission of a packet at time t, then $f_{i,j}(t + 1) = f_{i,j}(t)$ and $g_{i,j}(t + 1) = g_{i,j}(t) - 1$. If $g_{i,j}(t+1)$ is reduced to 0, then we reset $g_{i,j}(t+1)$ back to N. When this happens and $f_{i,j}(t + 1) > 1$, we decrease $f_{i,j}(t + 1)$ by 1.

In view of (3.115) and (iiiB), the virtual waiting time $V_{i,j}(t)$ can be represented by $f_{i,j}(t)$ and $g_{i,j}(t)$ as follows:

$$V_{i,j}(t) = \max[(f_{i,j}(t) - 1)N + ((j - i - 1) \bmod N) - (N - g_{i,j}(t)), 0]. \tag{3.116}$$

The main advantage of the scheme that uses cell indexes is that there is no need to transmit the whole information of the virtual waiting times. Instead, only cell indexes are transmitted. This greatly reduces the communication overhead needed in the mailbox switch. Also, it is easier to place a HOL packet in a mailbox by using the cell index of its virtual waiting time.

3.6.3 Mailbox switch with a limited number of forward tries

Note that the mailbox switch resolves conflict implicitly over *time* and *space*. First, packets are distributed evenly to the N mailboxes via the symmetric TDM switch at the first stage. Intuitively, one may view this as conflict resolution over space. Once a packet is transmitted to a mailbox, the mailbox switch has to find an empty cell with its cell index not smaller than the cell index of the virtual waiting time of the packet. As cells in the same bin are ordered in the FIFO manner, this can be viewed as conflict resolution over time. In the search for an empty cell to place the packet, there might be several tries until an empty cell is found. For each unsuccessful try, it may be viewed as a "collision," and each collision leads to back off N time slots for the packet departure time. Such a backoff 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 time slots. If there are many collisions, the increase of the virtual waiting time will be large and eventually packets will be distributed over time *sparsely*. This will result in low throughput and large delay. To avoid such an event, it might be better to block the packet by putting a limit on the amount of virtual waiting time that can be increased for each placement. This leads to the following modified scheme.

(iiC) **Sending mails**: let δ be the maximum increment of the cell index of the virtual waiting time. We only search for an empty cell from the cell $f_{i,j}(t)$ to the cell $\min[f_{i,j}(t) + \delta, F]$. If successful, the index of that cell, say f, is transmitted to the i^{th} output port of the second switch. If no such empty cell can be found, an error message, say $f = 0$, is transmitted to the i^{th} output port of the second switch to indicate a HOL blocking.

3.6.4 Mailbox switch with limited numbers of forward and backward tries

To perform conflict resolution more efficiently over *time*, we may also search for an empty cell with a limited number of *backward* tries. By so doing, packets in the mailbox switch might be out of sequence. But resequencing delay is bounded.

(iiD) **Sending mails**: let δ_b be the maximum number of backward tries. We only search for an empty cell from the cell $\max[f_{i,j}(t) - \delta_b, 1]$ to the cell $\min[f_{i,j}(t) + \delta, F]$. If successful, the index of that cell, say f, is transmitted to the i^{th} output port of the second switch. If no such empty cell can be found, an error message, say $f = 0$, is transmitted to the i^{th} output port of the second switch to indicate a HOL blocking.

(iiiD) **Updating virtual waiting times**: the case without a successful transmission of a packet is the same as (iiiB). For the case with a successful transmission of a packet, we have to deal with the following two subcases. If the returned index f is not smaller than $f_{i,j}(t)$, then it is the same as before. That is, we set $f_{i,j}(t+1) = f$ and reset $g_{i,j}(t+1)$ to N. On the other hand, if the returned index f is smaller than $f_{i,j}(t)$, then the packet being placed will depart

earlier than its previous one. As such, it is treated in the same way as the case without a successful transmission of a packet.

Note that the resequencing delay in the scheme with backward tries is bounded by $N\delta_b$ time slots.

3.6.5 Exact analysis for the throughput with $\delta = 0$

To further explain the mailbox switches, we consider the well-known uniform i.i.d. traffic model described in Section 3.1.3. We assume that both the bin size F and the buffers for the FIFO queues at the input ports of the first switch are infinite. Also, we do not allow backward tries, i.e., $\delta_b = 0$.

In this section, we consider the mailbox switch with $\delta = 0$. Since $\delta = 0$, the cell index of the virtual waiting time will never be increased. As such, there is no need to keep track of the virtual waiting times at all! Moreover, even though we assume that $F = \infty$ in our model, only the *first* cell in every bin is used. As such, it can be implemented with $F = 1$. Since $F = 1$, there is no need to transmit and feedback the cell index of the virtual waiting time. However, we still need to feedback a single bit information to indicate whether a HOL packet is successfully placed, i.e., $f = 0$ for a HOL blocking and $f = 1$ for a successful placement.

Our objective of this section is to show that this special case of the mailbox switch with $\delta = 0$ yields the same throughput as the classical HOL blocking switch in Section 2.2.2, i.e., it achieves 58% throughput. In fact, the mailbox switch with $\delta = 0$ can be viewed as a HOL blocking switch with distributed and pipelined conflict resolution.

As the traffic is uniform, we only need to consider a particular output port of the second switch, say, the first output. At time t, it is connected to the $h(1,t)^{th}$ mailbox, and this mailbox is also connected to the first input port of the first switch. If the first bin of the $h(1,t)^{th}$ mailbox is occupied at the beginning of the t^{th} time slot, then the packet is retrieved by the first output port and the first bin becomes empty at time t. In any event, we know that the first bin of the $h(1,t)^{th}$ mailbox is empty at time t.

Let $Y_i(t) = 1$ if the HOL packet of the FIFO queue of the i^{th} input port is destined for the first output port of the second switch at time t, and $Y_i(t) = 0$ otherwise. Let

$$q(t) = \sum_{i=1}^{N} Y_i(t+i-1). \tag{3.117}$$

As the $h(1,t)^{th}$ mailbox is connected to the first input port at time t, it will be connected to i^{th} input port at time $t+i-1$. Thus, $q(t)$ is the total number of HOL packets that can be placed in the first bin of the $h(1,t)^{th}$ mailbox from t to $t+N-1$. If $q(t) \geq 1$, then there is exactly one HOL packet that will be placed in the first bin of the $h(1,t)^{th}$ mailbox as the bin is empty at time t. Those blocked HOL packets remain the HOL packets and they can be placed in the first bin of the $h(1,t+1)^{th}$ mailbox from $t+1$ to $t+N$. Thus, we have

$$q(t+1) = (q(t)-1)^+ + a(t), \tag{3.118}$$

where $a(t)$ is the number of packets that becomes the HOL packets and can be placed in the first bin of the $h(1,t+1)^{th}$ mailbox from $t+1$ to $t+N$. Once we have the recursive equation in (3.118), we can follow the standard argument to show that the maximum throughput is $2-\sqrt{2}$ (see Section 2.2.2).

3.6.6 Exact analysis for the throughput with $\delta = \infty$

In this section, we consider the mailbox switch with $\delta = \infty$. Since $\delta = \infty$, there is no head-of-line blocking at the FIFO queues. As such, there is no need to buffer packets at the input ports of the first switch.

Our objective of this section is to show that the mailbox switch with $\delta = \infty$ achieves 67.5% throughput under the uniform i.i.d. traffic model. To see this, consider a particular flow, say flow (i,j). Let $W(n)$ be the virtual waiting time seen by the n^{th} packet of flow (i,j) (upon its arrival). Let $T(n)$ be the number of time slots between the arrivals of the n^{th} and $n+1^{th}$ packets of this flow. Note that the virtual waiting time of a flow is reduced by 1 for every time slot if the flow does not have a successful transmission. Let $S(n)$ be the increment of the virtual waiting time after the n^{th} packet is placed in a cell. Then we have the following Lindley recursion:

$$W(n+1) = (W(n) + S(n) - T(n))^+. \tag{3.119}$$

In order for the virtual waiting times to be stable, we need to have the rate condition (cf. Theorem 2.1.4)

$$\mathsf{E}[S(n)] < \mathsf{E}[T(n)]. \tag{3.120}$$

From the uniform i.i.d. traffic model, it follows that $T(n)$ is a geometric random variable (r.v.) with parameter ρ_a/N, i.e.,

$$\mathsf{P}(T(n) = k) = (1 - \frac{\rho_a}{N})^{k-1}\frac{\rho_a}{N}, \quad k = 1, 2, \ldots \tag{3.121}$$

Thus,

$$\mathsf{E}[T(n)] = \frac{N}{\rho_a}. \tag{3.122}$$

To find $\mathsf{E}[S(n)]$, note that the increment of the virtual waiting time consists of two factors: (i) the increment of the cell index and (ii) the increment of the counter (after being reset to N). The increment of $S(n)$ due to the second factor is simply $(T(n) - 1) \bmod N$. On the other hand, the increment of the cell index is the number of collisions encountered when the n^{th} packet of flow(i, j) is placed in the mailbox. Let $B(n)$ be the number of collisions encountered when the n^{th} packet of flow(i, j) is placed in the mailbox. Then, we have

$$\mathsf{E}[S(n)] = \mathsf{E}[B(n)] \cdot N + \mathsf{E}[\tilde{T}(n)], \tag{3.123}$$

where

$$\tilde{T}(n) = (T(n) - 1) \bmod N.$$

Recall that $T(n)$ is a geometric r.v. with parameter ρ_a/N, Thus, for $k = 0, 1, 2, \ldots N - 1$,

$$\mathsf{P}(\tilde{T}(n) = k) = \frac{(1 - \frac{\rho_a}{N})^k \cdot (\frac{\rho_a}{N})}{1 - (1 - \frac{\rho_a}{N})^N}. \tag{3.124}$$

When N is large, this implies that

$$\mathsf{E}[\tilde{T}(n)] \approx (\frac{1 - e^{-\rho_a} - \rho_a e^{-\rho_a}}{\rho_a(1 - e^{-\rho_a})})N. \tag{3.125}$$

To find $\mathsf{E}[B(n)]$, we first find the expected number of "collisions" that occurs in a time slot at an output port. From symmetry, this is equivalent to finding the expected number of "collisions" in a cell of a particular mailbox. Let $\tilde{q}(k)$ be the number of packets that are ever tried to be placed in the k^{th} cell for that particular mailbox. Also, let $\tilde{a}(k)$ be the number of packets that are placed to the k^{th} cell as their first trial. As the Poisson assumption used in the case with $\delta = 0$, we may assume that $\tilde{a}(k)$ is a Poisson random variable. As the k^{th} cell can only hold one packet and the rest of packets have to try the $k+1^{th}$ cell, we then have the following Lindley recursion:

$$\tilde{q}(k + 1) = (\tilde{q}(k) - 1)^+ + \tilde{a}(k + 1). \tag{3.126}$$

Note that $\mathsf{P}(\tilde{q}(k) > 0)$ is the probability that the k^{th} cell is occupied and hence it equal to the throughout ρ_d. Using a similar argument to that in Proposition 2.1.5, we have

$$\mathsf{P}(\tilde{q}(k) > 0) = \mathsf{E}(\tilde{a}(k)) = \rho_d \tag{3.127}$$

and

$$\mathsf{E}[\tilde{q}(k)] = \frac{2\rho_d - \rho_d^2}{2(1-\rho_d)}. \tag{3.128}$$

Since the first packet can be placed in the k^{th} cell successfully, the number of collisions in the k^{th} cell is $(\tilde{q}(k)-1)^+$. Thus, the expected number of "collisions" in a time slot is

$$\mathsf{E}[(\tilde{q}(k)-1)^+] = \mathsf{E}[\tilde{q}(k)] - \rho_d = \frac{\rho_d^2}{2(1-\rho_d)}. \tag{3.129}$$

As $\mathsf{E}[B(n)]$ is the expected number of "collisions" when a packet is placed in a cell, it can be computed by the following limit

$$\mathsf{E}[B(n)] = \lim_{t\to\infty} \frac{N_c(t)}{N_p(t)}, \tag{3.130}$$

where $N_c(t)$ is the cumulative number of "collisions" by time t at an output port and $N_p(t)$ is the cumulative number of departures by time t at an output port. If the system is ergodic, i.e., the ensemble average is the same as its time average, then we have from (3.129) and (3.130) that

$$\begin{aligned} \mathsf{E}[B(n)] &= \lim_{t\to\infty} \frac{N_c(t)}{N_p(t)} = \frac{\lim_{t\to\infty} \frac{N_c(t)}{t}}{\lim_{t\to\infty} \frac{N_p(t)}{t}} \\ &= \frac{\frac{\rho_d^2}{2(1-\rho_d)}}{\rho_d} \\ &= \frac{\rho_d}{2(1-\rho_d)}. \end{aligned} \tag{3.131}$$

From (3.123), (3.125), and (3.131), we have

$$\mathsf{E}[S(n)] = \frac{\rho_d}{2(1-\rho_d)} \cdot N + \left(\frac{1 - e^{-\rho_a} - \rho_a e^{-\rho_a}}{\rho_a(1-e^{-\rho_a})}\right) \cdot N. \tag{3.132}$$

When the system is stable, we have $\rho_d = \rho_a$. It then follows from (3.122) and (3.132) that the inequality in (3.120) can be rewritten as

$$\frac{\rho_a}{2(1-\rho_a)} + \left(\frac{1 - e^{-\rho_a} - \rho_a e^{-\rho_a}}{\rho_a(1-e^{-\rho_a})}\right) < \frac{1}{\rho_a}. \tag{3.133}$$

The maximum stable throughput can be found to be 0.6748 when the above inequality becomes an equality.

One interesting phenomenon is that when one increases the arrival rate ρ_a beyond the maximum stable throughput 0.6748, the system becomes unstable and the expected virtual waiting time $W(n)$ is increased to ∞ as n goes to ∞. However, the throughput ρ_d is also increased with respect to the arrival rate ρ_a. To see this, note that for an unstable system, we have

$$W(n+1) = W(n) + S(n) - T(n) \tag{3.134}$$

for large n. Thus, the expected interdeparture time between the n^{th} packet and the $n+1^{th}$ packet is simply $\mathsf{E}[S(n)]$. As the throughput of a particular flow ρ_d/N is simply the inverse of the expected interdeparture time between two packets of that flow, we then have from (3.132) that

$$\frac{N}{\rho_d} = \mathsf{E}[S(n)] = \frac{\rho_d}{2(1-\rho_d)} \cdot N + (\frac{1 - e^{-\rho_a} - \rho_a e^{-\rho_a}}{\rho_a(1-e^{-\rho_a})}) \cdot N. \tag{3.135}$$

To find the maximum unstable throughput, we solve the above equation by setting the maximum arrival rate $\rho_a = 1$. This yields the maximum unstable throughput 0.6786. Even though the difference between the maximum stable throughput and the maximum unstable throughput is very small, the existence of two types of throughput in the mailbox switch is quite interesting. Both the maximum stable throughput and the maximum unstable throughput are found to be quite close to our simulation in Figure 3.32 for $N = 100$.

3.6.7 Approximation for the throughput with $0 < \delta < \infty$

As described in Section 3.6.5, the key factor that limits the throughput for $\delta = 0$ is the head-of-line (HOL) blocking problem at the input buffers. On the other hand, as shown in Section 3.6.6, the key factor that limits the throughput for $\delta = \infty$ is the stability of virtual waiting times. It is expected that the throughput for the mailbox switch with $0 < \delta < \infty$ is limited by both the head-of-line blocking problem and the stability problem of virtual waiting times. Unlike the cases with $\delta = 0$ and $\delta = \infty$, exact analysis for the finite case with $0 < \delta < \infty$ is much more difficult. Instead, our objective is to find a simple approximation formula for the maximum throughput of the mailbox switch with $0 < \delta < \infty$.

First, let us consider a FIFO queue, say the i^{th} queue, at the input port of the first switch. In order to have a stable queue, we need to make sure that the arrival rate to the queue is smaller than the service rate of the queue. From the uniform i.i.d. traffic model described in Section 3.1.3, the arrival rate to the queue is simply ρ_a. To compute the service rate, consider a HOL packet of the queue at time t. Suppose the HOL packet is destined for the j^{th} output port of the second switch. The HOL packet is blocked only if there is no empty cell among the cells $f_{i,j}(t), f_{i,j}(t)+1, \ldots, f_{i,j}(t)+\delta$. Let ρ_d be the throughput of the mailbox switch. As a packet eventually leaves the mailbox switch once it is placed in a cell, the throughput ρ_d is also the probability that a cell is occupied. To simplify our analysis, we make the following assumption on the independence of cell occupancy:

(A5) Every cell is occupied *independently* with probability ρ_d. This is independent of everything else.

From (A5), the probability that the HOL packet is blocked is $\rho_d^{\delta+1}$. Thus, the service rate is $1-\rho_d^{\delta+1}$. This leads to the following condition for the FIFO queue to be stable:

$$\rho_a < 1 - \rho_d^{\delta+1}. \tag{3.136}$$

Now we consider the cell index of the virtual waiting time for a particular flow, say flow (i,j). In order for $f_{i,j}(t)$ to be stable, we need to make sure that the increase rate of $f_{i,j}(t)$ is smaller than the decrease rate of $f_{i,j}(t)$. To compute the increase rate, note that $f_{i,j}(t)$ is increased by k for some $0 \le k \le \delta$ if the following three conditions hold: (i) the HOL packet at the i^{th} input port of the first switch is a packet from flow (i,j), (ii) the cells in the j^{th} bin with the indexes $f_{i,j}(t), f_{i,j}(t)+1, \ldots, f_{i,j}(t)+k-1$ are occupied, and (iii) the cell with the index $f_{i,j}(t)+k$ is empty. From the uniform i.i.d. traffic model, the probability that the HOL packet at the i^{th} input port of the first switch is a packet from flow (i,j) is simply ρ_a/N. As everything is assumed to independent in (A5), the probability that $f_{i,j}(t)$ is increased by k is

$$\frac{\rho_a}{N} \cdot \rho_d^k \cdot (1-\rho_d).$$

Thus, the increase rate of $f_{i,j}(t)$ is

$$\sum_{k=0}^{\delta} k \cdot \frac{\rho_a}{N} \cdot \rho_d^k \cdot (1-\rho_d)$$

$$= \frac{\rho_a}{N} \frac{\delta \rho_d^{\delta+2} - (\delta+1)\rho_d^{\delta+1} + \rho_d}{1-\rho_d}. \tag{3.137}$$

To compute the decrease rate, note that $f_{i,j}(t)$ is decreased by 1 if the following two conditions hold: (i) there is no successful transmission of a packet from flow (i,j), and (ii) the counter $g_{i,j}(t) = 1$. The event that there is no successful transmission of a packet from flow (i,j) can be decomposed as the union of the two disjoint events: the HOL packet at the i^{th} input port of the first switch is *not* a packet from flow (i,j) or the HOL packet at the i^{th} input port of the first switch is a *blocked* packet from flow (i,j). Thus, the probability that there is no successful transmission of a packet from flow (i,j) is

$$1 - \frac{\rho_a}{N} + \frac{\rho_a}{N} \cdot \rho_d^{\delta+1}.$$

To compute the probability that $g_{i,j}(t) = 1$, we make the following assumption.

(A6) The counter $g_{i,j}(t)$ is uniformly distributed over $\{1, 2, \ldots, N\}$.

As such, the probability that $g_{i,j}(t) = 1$ is simply $1/N$. Thus, the decrease rate of $f_{i,j}(t)$ is

$$(1 - \frac{\rho_a}{N} + \frac{\rho_a}{N}\rho_d^{\delta+1})\frac{1}{N}. \tag{3.138}$$

Using (3.137) and (3.138) and letting $N \to \infty$, we have the following condition for the $f_{i,j}(t)$ to be stable:

$$\rho_a \frac{\delta \rho_d^{\delta+2} - (\delta+1)\rho_d^{\delta+1} + \rho_d}{1-\rho_d} < 1. \tag{3.139}$$

As the throughput ρ_d cannot be larger than the arrival rate ρ_a, it follows from (3.136) and (3.139) that the throughput ρ_d is limited by the following two inequalities:

$$\rho_d + \rho_d^{\delta+1} < 1, \tag{3.140}$$

$$\rho_d \frac{\delta \rho_d^{\delta+2} - (\delta+1)\rho_d^{\delta+1} + \rho_d}{1-\rho_d} < 1. \tag{3.141}$$

In Figure 3.30, we use the bound obtained by (3.140) and (3.141) to plot the maximum throughput as a function of δ. For $\delta < 5$, the inequality in (3.140) sets the limit on the maximum throughput. On the other hand, for $\delta \geq 5$, the inequality in (3.141) sets the limit on the maximum throughput. From these, it is interesting to see that the

curve is peaked when $\delta = 4$ and that gives the maximum throughput of 0.755. The intuition behind this is that if we set δ too small, it is quite likely that the HOL blocking will become a problem. On the other hand, if we set δ too large, then packets will be distributed over time *sparsely* and that also results in a low throughput. We also note that when $\delta \to \infty$, the mailbox switch has the maximum throughput 0.618, which is higher than $2 - \sqrt{2} \approx 0.58$ for the case with $\delta = 0$.

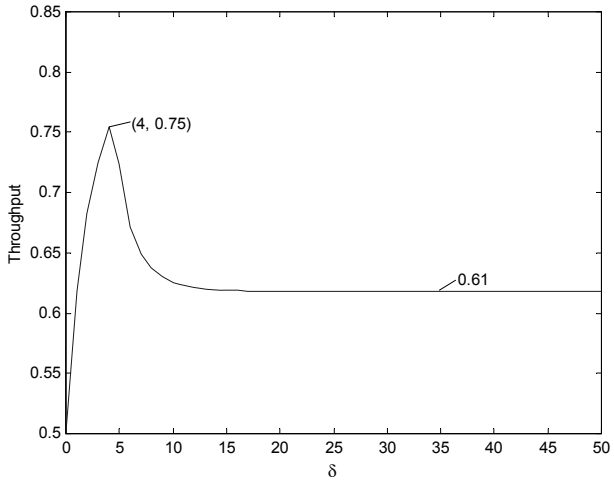


Fig. 3.30. The maximum throughput as a function of δ

3.6.8 Simulation Study

In this section, we perform various simulations to verify our theoretical results in the previous section. In all our simulations, we consider 100×100 mailbox switches, i.e., $N = 100$. Our first experiment is to find the maximum throughput of the mailbox switch. To achieve this, the arrival rate of each input port is set to 1, i.e., a packet arrives at each input port in every time slot. In Figure 3.31, we plot the simulation results (along with the theoretical results in Section 3.6.7) for the maximum throughput as a function of δ under the uniform i.i.d. traffic.

Note that the curve from the simulation results in Figure 3.31 is similar to that from the theoretical results. Both curves show that the

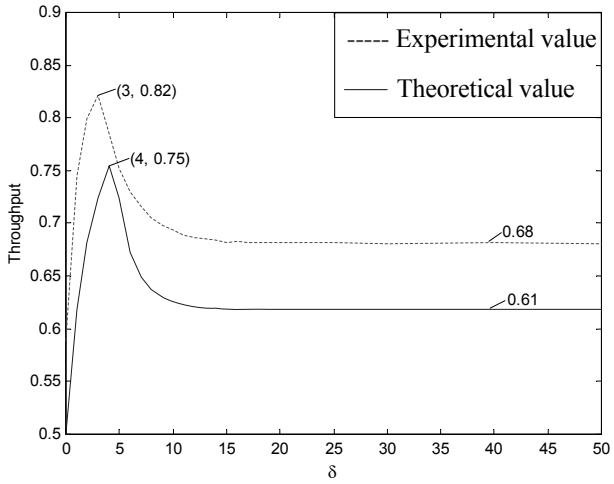


Fig. 3.31. The maximum throughput as a function of δ

throughput can be increased by increasing δ at the beginning, and it then starts to decrease if δ is increased further. As explained in our theoretical model, this is because the throughput is limited by the HOL blocking at the FIFO queues of the first switch when δ is small. On the other hand, when δ is large, the throughput is limited by the stability of the virtual waiting times. Thus, the throughput model based on the stability of the FIFO queues and the virtual waiting times seems to be valid (at least qualitatively).

We also note that for the case $\delta = 0$ the simulation result shows the maximum throughput is 0.58 as predicted by our theoretical model in Section 3.6.5. On the other hand, for the case $\delta \to \infty$, the simulation results show that the mailbox switch has the maximum throughput 0.68, which is quite close to 0.6786 predicted by our theoretical model in Section 3.6.6. But it is higher than 0.61 predicted by the theoretical model in Section 3.6.7. The main reason behind this is that the independence assumption for cell occupancy in (A5) in Section 3.6.7 is an over simplified assumption. In fact, we expect that nonempty cells are more likely to be clustered together as we always search for the first empty cell. As such, packets destined for the same output are more well packed and the increase rate of the cell indexes of the virtual waiting times is not as large as predicted in (3.137).

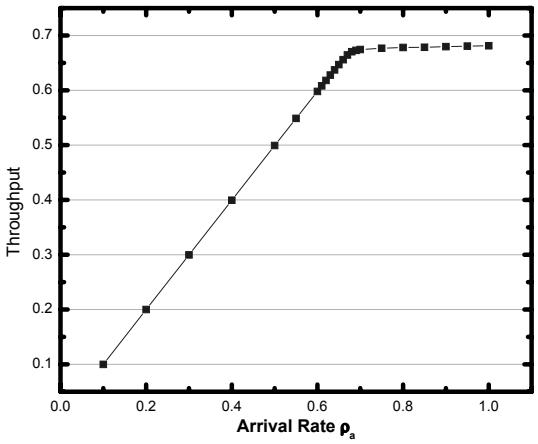


Fig. 3.32. Throughput as a function of the arrival rate ρ_a

In our second experiment, we measure the throughput by increasing the arrival rate ρ_a. For this experiment, we choose $\delta = 50$. In Figure 3.32, we plot the throughput as a function of the arrival rate ρ_a. Note that the throughput of the mailbox switch increases linearly as a function of the arrival rate ρ_a until it reaches its maximum stable throughput near 0.67. From that point on, the throughout is increased at a much slower rate to it maximum (unstable) throughput near 0.68. This shows that the mailbox switch does not have the undesired catastrophic behavior in some random conflict resolution algorithms such as ALOHA and CSMA (see e.g., [128]), where the throughput decreases as the load is increased further.

In this experiment, we also measure the normalized average increment of the virtual waiting time when a packet is placed successfully in a mailbox. The normalized average increment of the virtual waiting time is obtained by the ratio of the average increment of the virtual waiting time to the number of input ports N. In Figure 3.33, we plot the normalized average increment of the virtual waiting time as a function of the arrival rate and compare it with the theoretical model in (3.132). Specifically, we plot the theoretical curve as a function of the arrival rate ρ_a by

$$\frac{\rho_d}{2(1-\rho_d)} + \left(\frac{1 - e^{-\rho_a} - \rho_a e^{-\rho_a}}{\rho_a(1 - e^{-\rho_a})}\right), \tag{3.142}$$

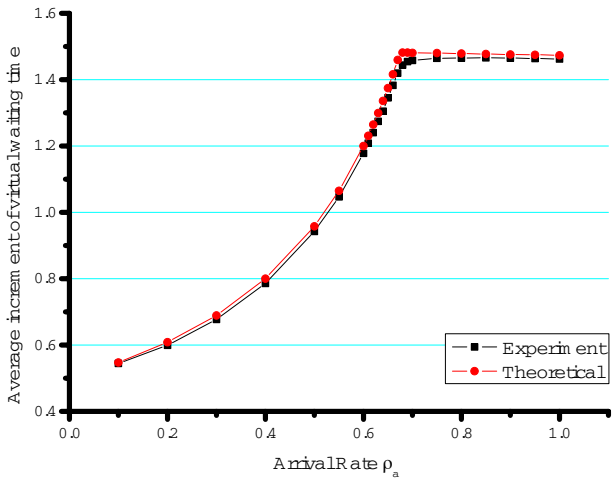


Fig. 3.33. Normalized average increment of the virtual waiting time as a function of the arrival rate ρ_a

where $\rho_d = \rho_a$ if ρ_a is smaller than the maximum stable throughput 0.6748, and ρ_d is obtained from (3.135) otherwise. As shown in Figure 3.33, the simulation result is quite close to that obtained from our theoretical model.

Note from Figure 3.33 that the curve from simulation can also be broken into two different segments near the maximum throughput 0.68: the stable segment with $\rho_a < 0.68$ and the overloaded segment with $\rho_a > 0.68$. In the stable segment, the product of the arrival rate and the normalized average increment of the virtual waiting time is less than 1. As the virtual waiting time is decreased by 1 per time slot, the virtual waiting time remains finite in the stable segment. To verify this, we plot in Figure 3.34 a sample path of the cell index of the virtual waiting time for flow $(1, 50)$ when $\rho_a = 0.55$. In Figure 3.34, the cell index of the virtual waiting time exhibits the typical behavior of a stable queue, i.e., the virtual waiting time returns to zero recurrently. Note that when the virtual waiting time is zero, its cell index is one. Thus, on average every HOL packet only needs to wait for a finite amount of time after being placed in a mailbox. As such, every HOL packet departs from the mailbox switch within a finite average delay. In this case, the throughput is the same as the arrival rate (since the

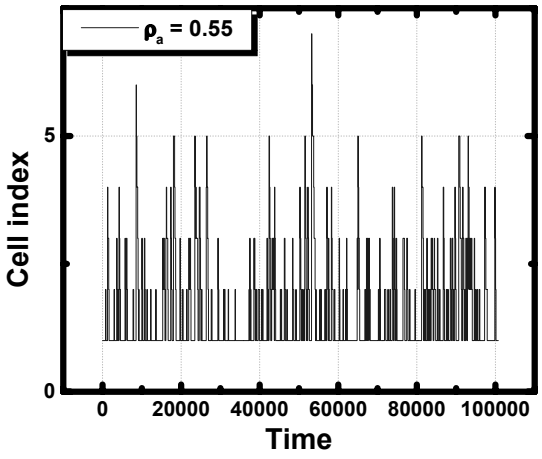


Fig. 3.34. A sample path of the cell index of the virtual waiting time of flow $(1, 50)$ when arrival rate is 0.55

HOL blocking is not a constraint when $\delta = 50$). This is consistent with the throughput plot in Figure 3.32.

On the other hand, the virtual waiting time in the overloaded segment will go to ∞ eventually. In Figure 3.35, we plot a sample path of the cell index of the virtual waiting time for flow $(1, 50)$ when $\rho_a = 0.69$. Note that the cell index of the virtual waiting time in this figure increases almost linearly to ∞ with respect to time. This is also consistent with the well known behavior of a unstable queue. As ρ_a exceeds the maximum throughput of the mailbox switch, it is intuitive to see that late arrivals of HOL packets have to wait much longer than early arrivals. As such, the virtual waiting time increases with respect to time.

To further explore the behavior of the mailbox switch, we plot packet delay as a function of the arrival rate for various numbers of forward trials δ. For every curve in the Figure 3.36, we observe that packet delay increases rapidly to ∞ as the arrival rate approaches its maximum throughput. This phenomenon provides further support for the throughput predicted by our theoretical models.

Moreover, as shown in Figure 3.36, in order to obtain the best packet delay, it seems that one should use the least δ that has the maximum throughput larger than the arrival rate ρ_a. For instance,

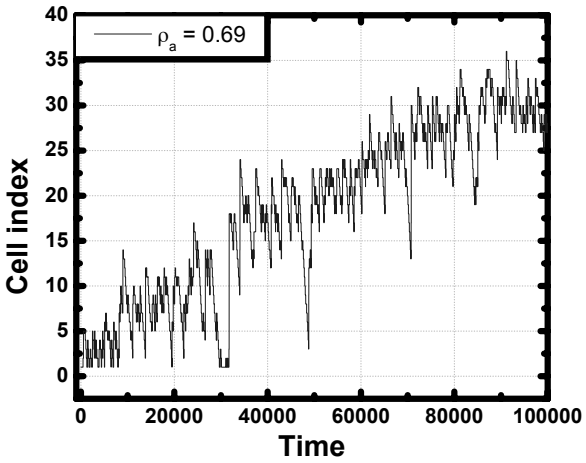


Fig. 3.35. A sample path of the cell index of the virtual waiting time of flow $(1, 50)$ when the arrival rate is 0.69

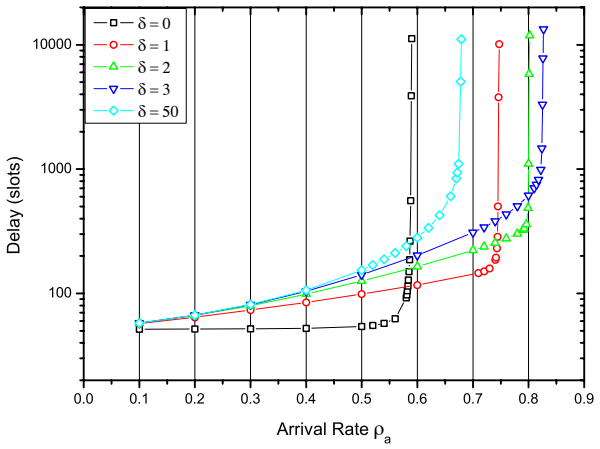


Fig. 3.36. Packet delay as a function of the arrival rate for various numbers of forward tries δ.

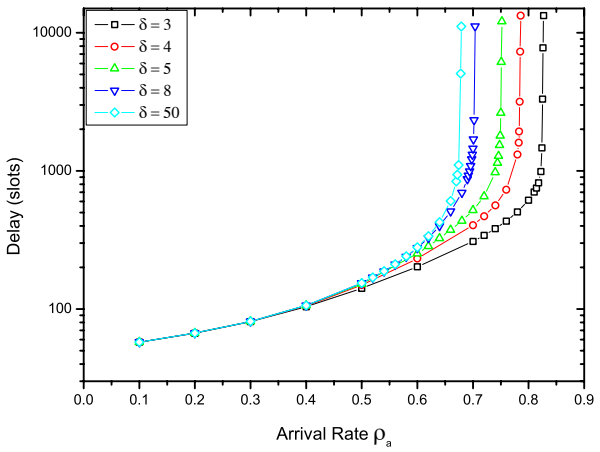


Fig. 3.37. Packet delay as a function of the arrival rate for forward trials not less than 3, i.e., $\delta \geq 3$.

when the arrival rate ρ_a is smaller than 0.58, the case with $\delta = 0$ has the best performance in terms of packet delay. In this case, the average packet delay is around $N/2 = 50$, which is the average number of time slots needed for a bin in a mailbox to be connected to its output. However, when the arrival rate ρ_a is between 0.58 and 0.74, the case with $\delta = 1$ is the best choice. As shown in Figure 3.31, the maximum throughput is achieved when $\delta = 3$. It is interesting to see in Figure 3.37 that the case with $\delta = 3$ is better than any other cases with $\delta > 3$ in terms of packet delay for the whole range of arrival rates.

In the third experiment, we consider the mailbox switch with limited numbers of forward and backward tries. As discussed in Section 3.6.4, there are two parameters for such a mailbox switch: δ and δ_b. The search for an empty cell for flow (i, j) is started from the cell with the index $\max[1, f_{i,j}(t)-\delta_b]$ to the cell with the index $\min[F, f_{i,j}(t)+\delta]$. The resequencing delay for such a mailbox switch is bounded by $N\delta_b$ slots. In Figure 3.38 we plot the throughput as a function of δ_b for $\delta = 5, 6$, and 7. From Figure 3.38, we note that the mailbox switch can achieve more than 95% throughput with small δ and δ_b. The throughput is an increasing function of δ_b as placing a packet in a cell with the index smaller than the cell index of its virtual waiting time does not result in the increase of its virtual waiting time. Another interesting

observation is that increasing δ does increase the throughput when δ_b is large. In the case that $\delta_b = 0$, we have known from Figure 3.31 that the throughput decreases as δ increases when $\delta \geq 4$. This is because a large δ tends to increase a large amount of the virtual waiting time when backward tries are not allowed ($\delta_b = 0$). However, this is not the case when δ_b is large. Even though a large δ tends to increase a large amount of the virtual waiting time, a large δ_b allows packets to be repacked in the cells that are "wasted" by a large increase of the virtual waiting time. Thus, the constraint is shifted from the stability of the virtual waiting time to the HOL blocking of FIFO queues. As a large δ tends to have a small probability of HOL blocking, this explains why the mailbox switch with a large δ has better throughput than that with a small δ when δ_b is large.

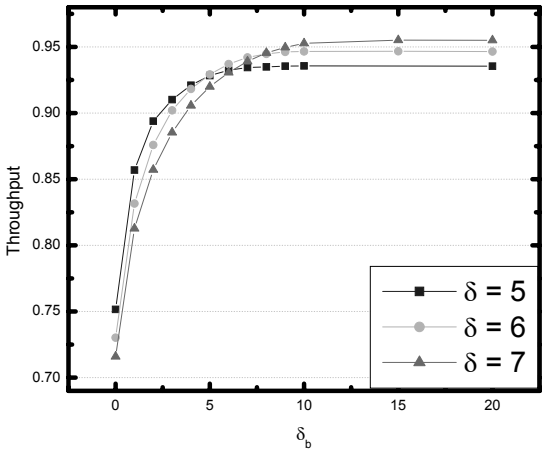


Fig. 3.38. The maximum throughput as a function of δ_b

3.7 Finite central buffers

From the simulation results in the previous section, we know that one can achieve higher throughput by allowing backward tries in the mailbox switch. If we set the number of backward tries δ_b to be the bin size F, then the entire bin has to be searched (from the first cell to

the last cell) for each placement of a packet. As such, there is no need to keep track of the cell indexes and the communication overhead can be greatly reduced to the single bit that indicates whether a packet is successfully placed in its bin, i.e., $f = 0$ for a failure and $f \neq 0$ for a success.

In this section, we will consider the special case of the mailbox switch with $\delta_b = F$. In this case, the maximum resequencing delay is then bounded by NF time slots. Clearly, there is a tradeoff between the throughput and the maximum resequencing delay. If one increases the bin size F, the throughput is increased. However, the maximum resequencing delay is also increased. As such, the complexity of resequencing buffer is also increased. On the other hand, if we choose a very small F, then it is quite likely that packets will be blocked at the central buffer. This leads to very low throughput for the switch. **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.**

3.7.1 Sizing the central buffers

The objective of the section is to determine the bin size F for the switch. As the exact analysis for the finite buffer case is much more difficult than the infinite buffer case in Section 3.1, one has to resort to computer simulations. In all the simulations, we set $N = 100$. In the first experiment, each simulation is run for 200000 time slots under the uniform i.i.d. traffic model described in Section 3.1.3.

To find the maximum throughput, 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 3.39, we plot the (measured) maximum throughput as a function of F (the bin size of the central buffer). As shown in Figure 3.39, 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 3.40. As shown in Figure 3.40, 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 3.40, it also shows that the case $F = 15$ has a reasonably good delay-throughput performance.

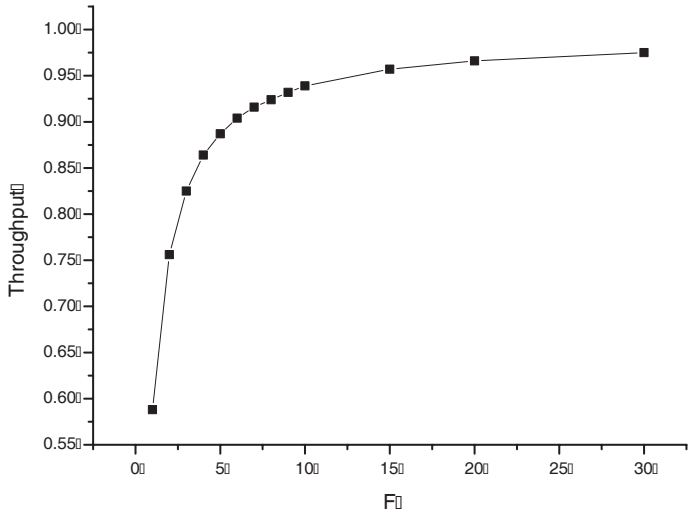


Fig. 3.39. The maximum throughput as a function of F under the uniform i.i.d. traffic model

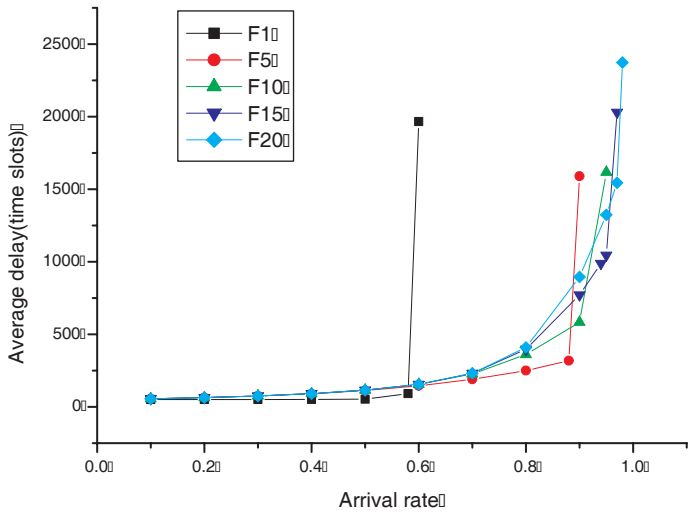


Fig. 3.40. The average delay for various F under the uniform i.i.d. traffic model

The first experiment suggests that $F = 15$ might be a good choice. To verify this, we set $F = 15$ and measure the throughput for various arrival rates under the uniform i.i.d traffic model and the uniform Pareto traffic model in Section 3.1.5.

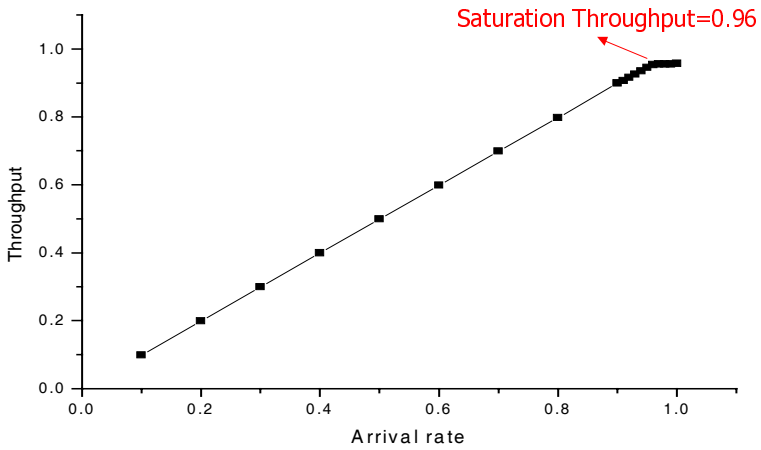


Fig. 3.41. Throughput as a function of the arrival rate for the uniform i.i.d traffic model with $F = 15$

In Figure 3.41 and Figure 3.42, we plot the throughput as a function of the arrival rate for the uniform i.i.d. traffic model and the uniform Pareto traffic model. These figures show that the throughput increases with the arrival rate 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 is increased further. The variation of the maximum throughput in Figure 3.42 is due to the measurement inaccuracy of the simulation for the bursty Pareto traffic. In Figure 3.41, it shows that the maximum throughput is 96% for the uniform i.i.d. model. But for the uniform Pareto traffic model, the maximum throughput is down to 70% as shown in Figure 3.42.

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

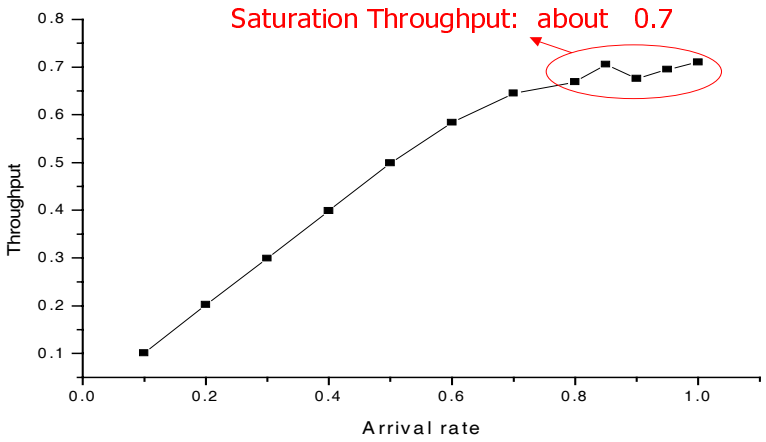


Fig. 3.42. Throughput as a function of the arrival rate for the uniform Pareto traffic model with $F = 15$

that a bin is full is governed by the power law, i.e., $1/F^{\alpha}$ for some $\alpha > 0$ (see e.g., [130, 89] 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., [27] 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. 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 address in the next section.

3.7.2 Round-robin policy for multiple VOQs at input buffers

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

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 require 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 this experiment, we replace the single FIFO queue (used in Section 3.7.1) 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 for two different traffic models: the uniform Pareto traffic model and the uniform i.i.d. traffic model.

In Figure 3.43, we plot the throughput as a function of the arrival rate 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 3.44, 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., R. Nelson [128]). 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 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.

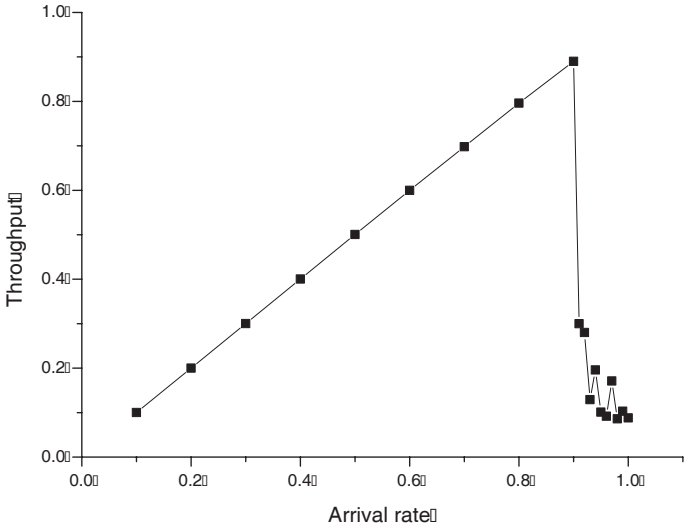


Fig. 3.43. The throughput of the switch with $m = 10$ and the RR service policy for the uniform Pareto traffic model with $F = 15$

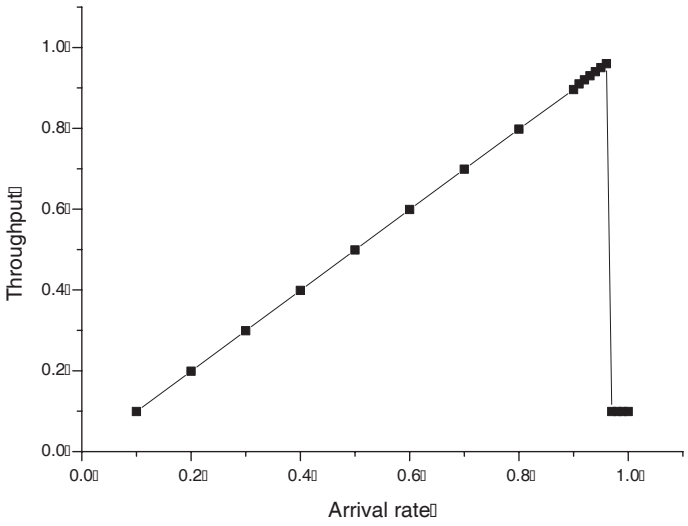


Fig. 3.44. The throughput of the switch with $m = 10$ and the RR service policy for the uniform i.i.d. traffic model with $F = 15$

3.7.3 Non-ergodic mode

To understand the non-ergodic mode in our switch, we note that **when the offered load exceeds the maximum (stable) throughput, all the VOQs start to grow. As we choose $m = 10$ and $N = 100$ in our experiments, the number of output ports is an integer multiple of the number of VOQs. Moreover, as 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 3.45, we illustrate concept of non-ergodic modes via a state transition diagram of a Markov chain. A Markov chain is a discrete-time stochastic process that its future only depends on its most recent state. A non-ergodic mode in a Markov chain 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.

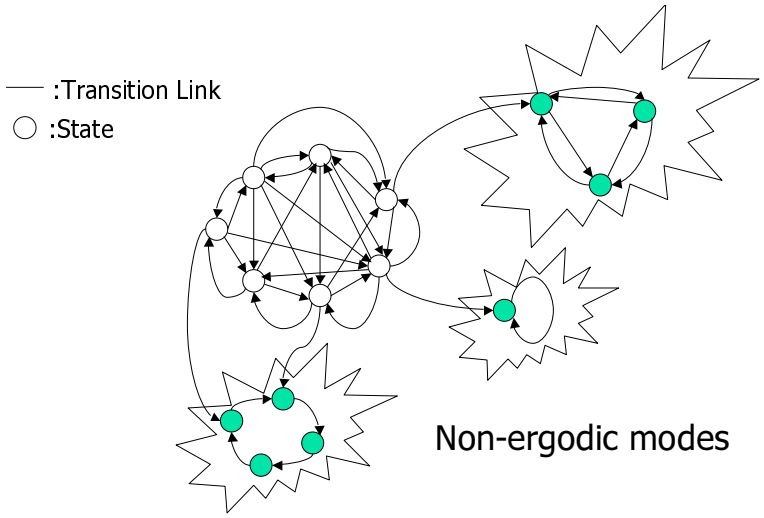


Fig. 3.45. The non-ergodic modes in a state transition diagram

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., $m = N = 100$.

3.7.4 The effect of randomness for the non-ergodic mode

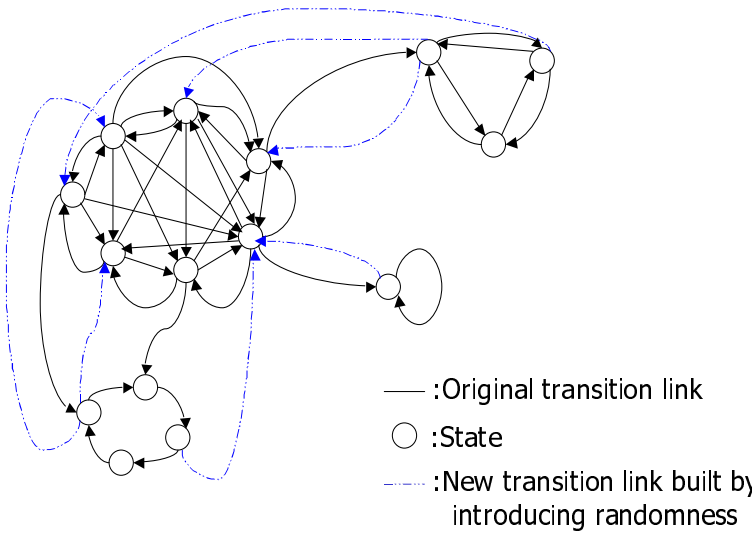


Fig. 3.46. Jumping out of a non-ergodic mode by introducing randomness

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 3.46, 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 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 3.47 and Figure 3.48, it shows that the throughput in heavy load has a great deal of improvement after modifying the RR service policy. In Figure 3.49 and Figure 3.50, we enlarge the heavy load segments (for the arrival rate from 0.80 to 1.00) in Figure 3.47 and Figure 3.48. 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% throughput 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

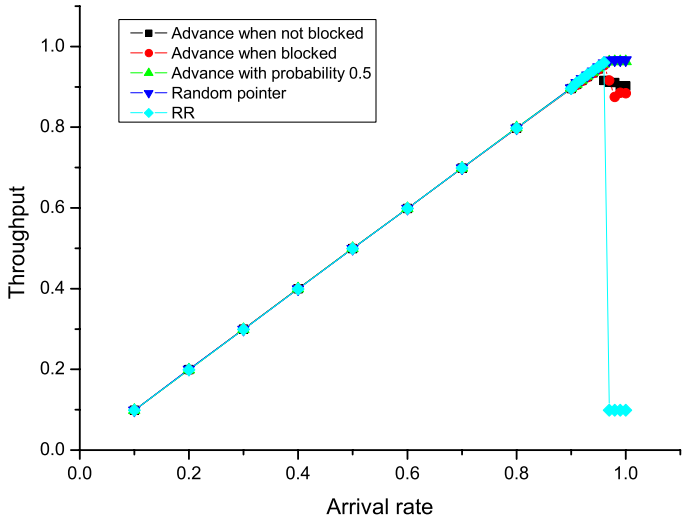


Fig. 3.47. The throughput of the switch with $m = 10$ under various service policies for the uniform i.i.d. traffic model

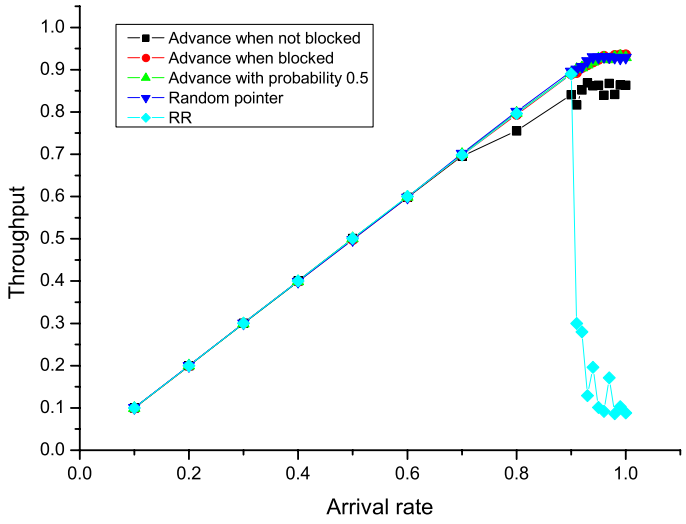


Fig. 3.48. The throughput of the switch with $m = 10$ under various service policies for the uniform Pareto traffic model

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) in Section 2.2.4. 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.

As addressed in Section 2.2.4, McKeown [121] observed 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 problem, he proposed using SLIP (see Section 2.2.5 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 SLIP 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 Section 2.2.5, there is a deterministic traffic model that also leads SLIP to a non-ergodic mode.

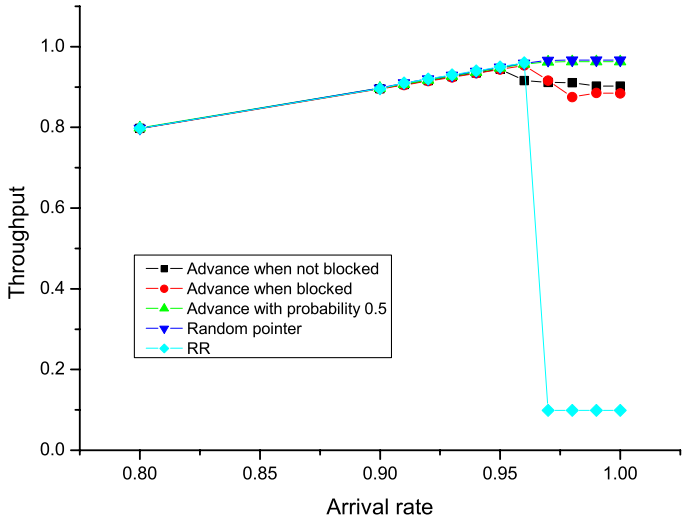


Fig. 3.49. Enlargement of Figure 3.47 under heavy load

3.8 Notes

The idea of using load balancing for parallel communications was introduced by L. G. Valiant [161]. In Valiant's scheme, packets are sent *randomly* to an intermediate node for load balancing. Such a scheme is known as "randomization." It was also used by D. Mitra and R. A. Cieslak [126] by randomly spreading packets over multistage banyan type of networks. Such a randomization scheme was also depicted in the book by J. Y. Hui [76].

Unlike "randomization," load balancing in the load balanced Birkhoff-von Neumann switches is done in a more "deterministic" man-

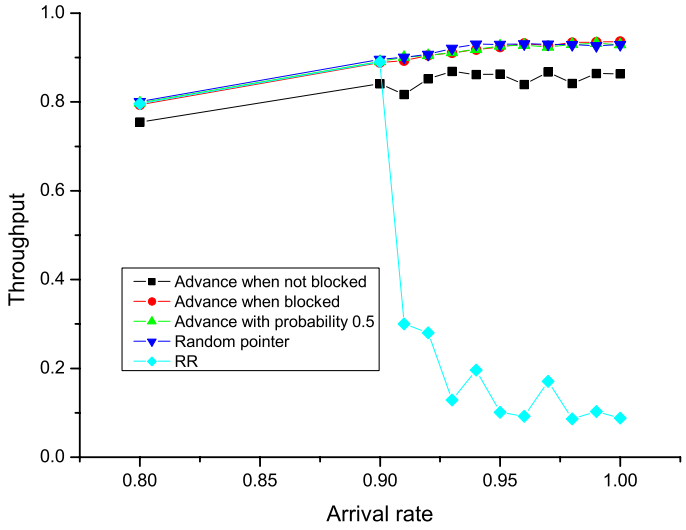


Fig. 3.50. Enlargement of Figure 3.48 under heavy load

ner. The one-stage buffering scheme in Section 3.1 was introduced by C.-S. Chang, D.-S. Lee and Y.-S. Jou [34], and the multi-stage buffering scheme in Section 3.3 was introduced by C.-S. Chang, D.-S. Lee and C.-M. Lien [35]. The key differences between the randomization schemes and the load balanced Birkhoff-von Neumann switches can be summarized as follows:

(i) Hardware complexity: as PIM and SLIP, one uses a random arbitration scheme and the other uses a deterministic round-robin arbitration scheme. In terms of hardware design, SLIP is much easier than PIM. Similarly, the load balanced Birkhoff-von Neumann switches that use deterministic connection patterns are much easier to design than a randomization scheme.

(ii) Deterministic performance bounds: there are deterministic delay bounds for the load balanced Birkhoff-von Neumann switches in comparison with the corresponding ideal output-buffered switches. This is not possible for a randomization scheme.

(iii) Non-ergodic modes: due to their deterministic nature, there exist non-ergodic modes for certain load balanced Birkhoff-von Neumann switches with finite buffers. As SLIP and RRM, when

trapped in a non-ergodic mode, the performance could be considerably worse than its normal behavior. A randomization scheme does not have such a problem.

(iv) Internal blocking: in the randomization scheme in [126], every packet, upon its arrival at the first stage, randomly selects an output port at the first stage. The packet is then routed through the banyan network at the first stage via the self routing property of the banyan network. The problem of the randomization scheme is internal blocking. There might be two or more packets that share a common internal link in the banyan network. As a result, packets might be lost inside the banyan network, and this leads to throughput degradation. The load balanced Birkhoff-von Neumann switches do not have such a problem.

The Full Ordered Frames First (FOFF) scheme in Section 3.3.6 was proposed by the research team at Stanford University [99]. It provides an easy way to achieve 100% throughput and the correction of out-of-sequence packets (see also Problems 19 and 20 for the application flow-based routing scheme and the uniform frame spreading scheme). In [99], they also proposed an optical implementation of the FOFF scheme for a 100 tearbits/sec optical router. A variant of the FOFF scheme was reported by I. Keslassy and N. McKeown [98] to show a deterministic delay bound in comparison with the ideal output-buffered switch. It was further shown by I. Keslassy, S.-T. Chuang and N. McKeown in [97] that it is possible to add an arbitrary number of linecards in a load-balanced switch. An interesting theoretical result was obtained by I. Keslassy, C.-S. Chang, N. McKeown, D.-S. Lee [96]. They showed that there exists a unique optimal load balancing scheme that achieves the best universal throughput (see Problems 14-18). The load balanced Birkhoff-von Neumann switch, though not optimal, is asymptotically optimal when the size of the switch is increased to ∞. The problem of the FOFF scheme, as pointed out by [44], is its large average packet delay due to the large frame size. An improved scheme, called the byte-focal switch, was proposed by H. J. Chao, J. Song, N. S. Artan, G. Hu, and S. Jiang [44]. J.-J. Jaramillo, F. Milan, and R. Srikant also proposed another alternative by using padded frames (see Problem 21).

Guaranteed rate services in the load balanced Birkhoff-von Neumann switches in Section 3.4 was proposed by C.-S. Chang, D.-S. Lee, and C.-Y. Yue [40]. The analytic results for input-buffered switches

with head-of-line blocking in Section 3.5.2 was taken from C.-S. Chang, D.-S. Lee, and C.-L. Yu [39]. The mailbox switches in Section 3.6 was proposed by C.-S. Chang, D.-S. Lee, and Y.-J. Shih [36]. The existence of non-ergodic modes in the load balanced Birkhoff-von Neumann switches with finite central buffers in Section 3.7 was discovered by C.-Y. Tu, C.-S. Chang, D.-S. Lee, and C.-T. Chiu [159].

Problems

1. (Non-uniform i.i.d. traffic model) Consider a non-uniform i.i.d. traffic model to an $N \times N$ switch. With probability ρ_i, a packet arrives at the i^{th} input for every time slot. This is independent of everything else. With probability $p_{i,j}$, the j^{th} output is chosen as the destination of an arriving packet at the i^{th} input. This is also independent of everything else. Note that $p_{i,j} \geq 0$ and $\sum_{j=1}^{N} p_{i,j} = 1$ for all i.
   a) Suppose that $N = 2$. For an output-buffered switch under the above non-uniform traffic model, show that the average queue length at the first output is $\rho_1\rho_2 p_{11}p_{21}/(1 - \rho_1 p_{11} - \rho_2 p_{21})$.
   b) Suppose that $N = 2$. For an output-buffered switch under the above non-uniform traffic model, show that the average packet delay through the first output is
   $$\frac{\rho_1\rho_2 p_{11}p_{21}}{(1 - \rho_1 p_{11} - \rho_2 p_{21})(\rho_1 p_{11} + \rho_2 p_{21})}.$$
   c) Suppose that $N = 2$. Consider a load-balanced Birkhoff-von Neumann switch with one stage buffering in Section 3.1. Assume that the connection patterns of both switch fabrics are symmetric TDM switches. Show that the average queue length under the above non-uniform traffic model at the first VOQ of the first central buffer (i.e., $q_{1,1}$) is $\frac{1}{2}\rho_2 p_{21} + \rho_1\rho_2 p_{11}p_{21}/(1 - \rho_1 p_{11} - \rho_2 p_{21})$.
2. (Non-uniform bursty traffic model) Consider a non-uniform bursty traffic model to an $N \times N$ switch. As the uniform bursty traffic model, packets come as a burst of length N. Packets within the same bursty are destined to the same output. For every N time slots, the probability that there is a burst arriving at the i^{th} input is ρ_i. This is independent of everything else. With probability $p_{i,j}$, the j^{th} output is chosen as the destination of an arriving burst

to the i^{th} input. This is also independent of everything else. Note that $p_{i,j} \geq 0$ and $\sum_{j=1}^{N} p_{i,j} = 1$ for all i. Suppose that $N = 2$. Consider a load-balanced Birkhoff-von Neumann switch with one stage buffering in Section 3.1. Assume that the connection patterns of both switch fabrics are symmetric TDM switches. Show that the average queue length under the above non-uniform traffic model at the first VOQ of the first central buffer (i.e., $q_{1,1}$) is $\frac{1}{2}\rho_2 p_{21} + \rho_1\rho_2 p_{11} p_{21}/(1 - \rho_1 p_{11} - \rho_2 p_{21})$.

3. Using the two-stage construction to design a 16×16 symmetric TDM switch with 8 4×4 symmetric TDM switches.
4. In Figure 3.51, we show a 16×16 symmetric TDM switch via 2×2 switches. Use "b" to denote the bar connection and "x" to denote the cross connection of a 2×2 switch.
   a) Find the connection patterns of switch (3,6) for $t = 1, 2, \ldots, 16$.
   b) Find the connection patterns of switch (4,3) for $t = 1, 2, \ldots, 16$.
5. Prove Proposition 3.3.3.
6. Prove Theorem 3.3.6 (see [35]).
7. Show that the difference between the queue length of these N VOQs at the central buffers in FOFF scheme is bounded above by $M_{\max}$.
8. Show how the frame based scheme in Section 3.5.1 can be used for variable length packets.
9. Identify the technical conditions needed to ensure that no packets are lost inside the frame based scheme in Section 3.5.1 for multicasting flows.
10. (Generalized Pollaczek-Khinchin formula [39]) Consider the following Lindley equation:

$$q(\infty) =_{st} (q(\infty) - F)^+ + a(1).$$

Let $\mathrm{i} = \sqrt{-1}$,

$$p_i = \mathsf{P}(q(\infty) = i), \quad i = 0, 1, \ldots$$

and

$$P(z) = \sum_{i=0}^{\infty} p_i z^i = \mathsf{E}[z^{q(\infty)}]$$

be the generating function of $q(\infty)$. Also, let $A(z) = \mathsf{E}[z^{a(1)}]$ be the generating function of $a(1)$,

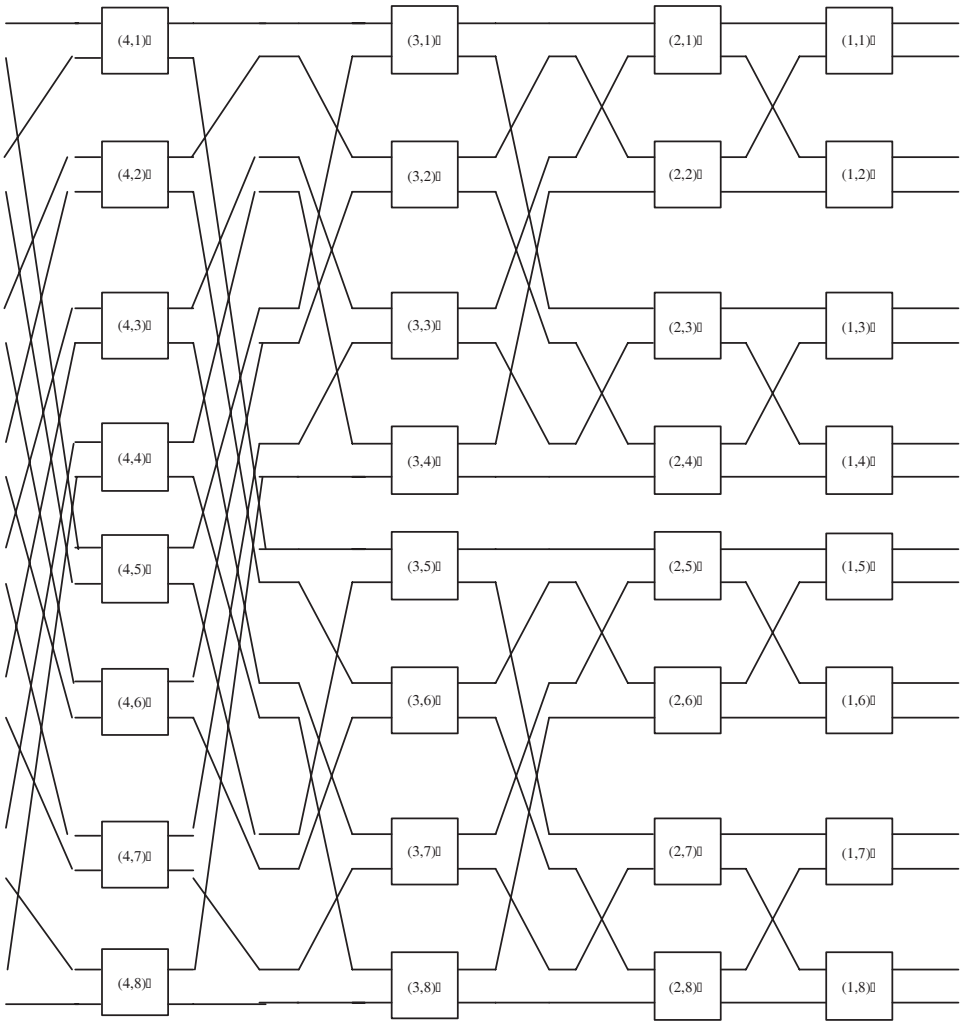


Fig. 3.51. A 16×16 symmetric TDM switch

$$\rho = \frac{\mathsf{E}[a(1)]}{F}$$

be its normalized mean, and

$$\sigma^2 = \frac{\mathsf{E}[(a(1))^2] - (\mathsf{E}[a(1)])^2}{F^2}$$

be its normalized variance. Suppose that $a(1)$ is independent of $q(\infty)$ and $\rho < 1$. Show that

$$P(z) = \frac{(z-1)G(z)}{H(z)},$$

where

$$H(z) = \frac{z^F}{A(z)} - 1,$$

$$G(z) = F(1-\rho) \prod_{j=1}^{F-1} \frac{(z-\alpha_j)}{(1-\alpha_j)},$$

and α_j is the unique root within the unit circle of the following equation:

$$z - e^{\mathrm{i}2\pi j/F}(A(z))^{1/F} = 0,$$

for $j = 0, 1, \ldots, F-1$. Moreover,

$$\begin{aligned}\mathsf{E}[q(\infty)] &= \frac{\sigma^2 F - F\rho^2 - \rho + 2F\rho - F + 1}{2(1-\rho)} \\ &\quad + \sum_{j=1}^{F-1} \frac{1}{1-\alpha_j}.\end{aligned}$$

11. To solve (3.100), note that

$$A(z) = e^{\rho F(z-1)}$$

for Poisson arrivals. Use the generalized Pollaczek-Khinchin formula to show that

$$P(z) = \frac{e^{\rho F(z-1)}(z-1)G(z)}{z^F - e^{\rho F(z-1)}}, \tag{3.143}$$

where

$$G(z) = F(1-\rho) \prod_{j=1}^{F-1} \frac{(z-\alpha_j)}{(1-\alpha_j)}, \tag{3.144}$$

and α_j is the unique root within the unit circle of the following equation:

$$z - e^{\mathrm{i}2\pi j/F} e^{\rho(z-1)} = 0, \tag{3.145}$$

for $j = 0, 1, \ldots, F-1$. Use (3.143) to show that

$$\mathsf{E}[q(\infty)] = \frac{1 - F(1-\rho)^2}{2(1-\rho)} + \sum_{j=1}^{F-1} \frac{1}{1-\alpha_j}, \tag{3.146}$$

12. Continue from the previous problem. Suppose that $\mathsf{E}q(\infty) = F$. Show that the maximum throughput ρ has the following upper and lower bounds:

$$1 - \frac{1}{\sqrt{F+1}+1} \le \rho \le 1 - \frac{2}{\sqrt{F^2+6F+1}+F+1}. \tag{3.147}$$

13. In the simple design of a mailbox switch with finite central buffers in Section 3.7, one still needs to design a re-sequencing buffer. Show that the re-sequencing buffer can be bounded above by NF, where N is the number of input/output ports and F is the size of central buffers. Also, show that the delay in the re-sequencing buffer can be bounded above by $2NF$ (see e.g., [159]).
14. (Optimal load-balancing [96]) Consider a network of N nodes with fixed interconnections. Let $C_{i,j}$ be the link capacity (measured in bits for a unit of time) from node i to node j and $C = (C_{i,j})$ be the $N \times N$ capacity matrix. Assume that every node is capable of transmitting and receiving at most one bit per unit of time, i.e.,

$$\sum_{i=1}^{N} C_{i,j} \le 1, \quad j = 1, 2, \ldots, N, \tag{3.148}$$

and

$$\sum_{j=1}^{N} C_{i,j} \le 1, \quad i = 1, 2, \ldots, N. \tag{3.149}$$

Let $T_{i,j}$ be the traffic demand (measured in bits per unit of time) from node i to node j and $T = (T_{i,j})$ be the $N \times N$ traffic matrix for the network. A traffic matrix is *feasible* if for all i and j one can find a set of routing paths $R(i,j)$ such that

$$\sum_{p \in R(i,j)} T^p = T_{i,j}, \tag{3.150}$$

and

$$\sum_{p:(i\to j)\in p} T^p \le C_{i,j}, \tag{3.151}$$

where T^p is the traffic carried by the routing path p. The condition in (3.150) says that the total traffic carried by all the routing paths from node i to node j should meet the traffic demand. On the other hand, (3.151) means that the total traffic carried by the routing paths that use the link from node i to node j should not exceed the link capacity. Show that for the case that $C_{i,j} = 1/N$ for all i and j, any traffic matrix T that satisfies

$$\sum_{i=1}^{N} T_{i,j} \le \frac{1}{2}, \quad j = 1, 2, \ldots, N, \tag{3.152}$$

and

$$\sum_{j=1}^{N} T_{i,j} \le \frac{1}{2}, \quad i = 1, 2, \ldots, N, \tag{3.153}$$

is feasible. (Hint: this corresponds to the load balancing Birkhoff-von Neumann switch. Use the two hop routes $i \to k \to j$ with each route carrying $T_{i,j}/N$.)

15. Continue from the previous problem. Suppose that two traffic matrices $\hat{T}$ and $\tilde{T}$ are feasible for a fixed interconnection network with the capacity matrix C satisfying (3.148) and (3.149). Show that any traffic matrix T that is a convex combination of these two traffic matrices, i.e., for some $0 \le \alpha \le 1$

$$T = \alpha\hat{T} + (1-\alpha)\tilde{T},$$

is also feasible. (Hint: use the routing paths for $\hat{T}$ with the traffic being reduced by a factor α. Also, use the routing paths for $\tilde{T}$ with the traffic being reduced by $1-\alpha$.)

16. Continue from the previous problem. A fixed interconnection network with the capacity matrix C satisfying (3.148) and (3.149) is said to achieve a universal throughput θ if it is feasible for any traffic matrix T that satisfies

$$\sum_{i=1}^{N} T_{i,j} \le \theta, \quad j = 1, 2, \ldots, N, \tag{3.154}$$

and

$$\sum_{j=1}^{N} T_{i,j} \leq \theta, \quad i = 1, 2, \ldots, N. \tag{3.155}$$

Use the result in the previous problem and the Birkhoff-von Neumann decomposition to show a network that achieves a universal throughput θ if it is feasible for any traffic matrix $T = \theta P$, where P is a permutation matrix.

17. A fixed interconnection network is called a *ring* if $C_{i,j} = 1$ for $j = (i+1) \bmod N$, and 0 otherwise. Show that a ring only achieves a universal throughput $1/N$. (Hint: consider the case that the permutation matrix P is the identity matrix.)
18. A fixed interconnection network is called the *biased mesh* if $C_{i,j} = 2/(2N-1)$ for $j \neq i$, and $C_{i,j} = 1/(2N-1)$ for $j = i$. Show that a biased mesh achieves a universal throughput $N/(2N-1)$. In fact, it is further shown in [96] that $N/(2N-1)$ is the maximum universal throughput that can be achieved by any interconnection network satisfying (3.148) and (3.149). Moreover, the biased mesh is the unique interconnection network that achieves the maximum universal throughput. (Hint: consider a permutation matrix P. If $P_{i,i} = 1$, then use the direct route $i \to i$ with the traffic $1/(2N-1)$, and the two hop routes $i \to k \to i$ $(k \neq i)$ with the traffic $1/(2N-1)$. If $P_{i,j} = 1$ for some $j \neq i$, then use the direct route $i \to j$ with the traffic $2/(2N-1)$ and the two hop routes $i \to k \to j$ $(k \neq i, j)$ with the traffic $1/(2N-1)$.)

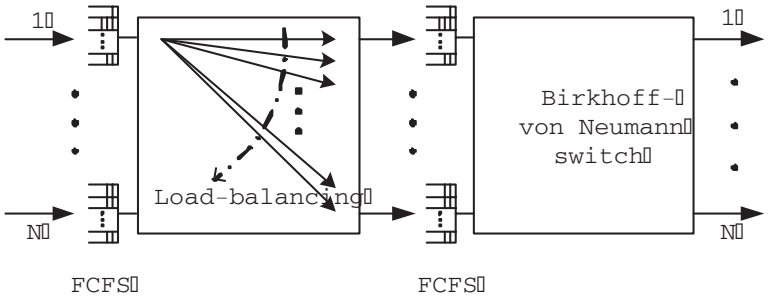


Fig. 3.52. The application flow-based routing scheme

19. (Application flow-based routing [95]) In the setting of load-balanced Birkhoff von Neumann switches, a flow is a sequence of packets that have the same input port and the same output port. In fact, a

flow might consist of many *application flows*, e.g., TCP flows, and one only needs to make sure that the packets in each application flow are in sequence. To achieve this, one can assign the packets of the same application flow to go through the same route (central buffer). For this, one may add Virtual Output Queues (VOQs) at each input port of the load-balanced Birkhoff von Neumann switch with one-stage buffering (see Figure 3.52). Packet of the same application flow are routed to the same VOQ at each input port and thus routed to the same central buffer. By so doing, packets of the same application flow depart in sequence. If there are many application flows in each flow, then load-balancing might be achieved by randomly assigning each application flow to one of the VOQs at an input port. Show by a counterexample that the application flow-based routing scheme cannot achieve 100% throughput (Hint: each flow only consists of an application flow.)

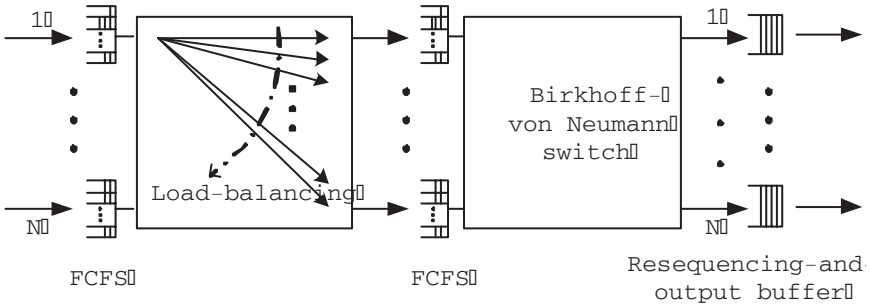


Fig. 3.53. The uniform frame spreading scheme

20. (Uniform frame spreading scheme [95]) It is shown in Section 3.1.4 that load-balancing is perfect if the incoming traffic is the uniform bursty traffic. Moreover, resequencing is limited to the packets in the same burst, i.e., N packets for an $N \times N$ switch. The idea of the uniform frame spreading (UFS) scheme is to convert the incoming traffic into the uniform bursty traffic. The architecture of the UFS scheme (shown in Figure 3.19) is almost the same as the application flow-based scheme in Figure 3.53 except a resequencing-and-output buffer is added at each output port. In the UFS scheme, every N time slots is grouped into a frame. At the beginning of a frame, each input checks whether there is a full frame (a VOQ contains more than or equal to N packets). If there is one, the longest

VOQ is selected and N packets from that VOQ are sent to the N central buffers. Otherwise, nothing is transmitted during that frame. Show that there are at most N^2 packets at each input and thus the UFS scheme achieves 100% throughput. (Hint: let $q(m)$ be the total number of packets at a particular input at the end of the m^{th} frame and $a(m+1)$ be the number of arrivals during the $m+1^{th}$ frame. Write down the governing equation for the UFS scheme and prove the bound by induction.)

21. (Padded frames [88]) Note that the full ordered frame first (FOFF) scheme in Section 3.3.6 is in fact an improvement based on the UFS scheme. In the FOFF scheme, partial frames are transmitted when there does not have a full frame. Another alternative is the padded frame scheme in [88]. The padded frame scheme, as the UFS scheme, operates in frames. If there is a full frame, the longest VOQ is selected and N packets from that VOQ are sent to the N central buffers. Otherwise, the longest VOQ is selected and the partial frame of that VOQ is padded with fictitious packets to form a padded frame with N packets. The padded frame is sent only if the total number of padded frames in the central buffers does not exceed a threshold T. By so doing, the average packet delay can be reduced in light traffic. Clearly, when T is 0, it reduces to the UFS scheme. Show that the padded frame scheme still achieves 100% throughput as long as T is finite.

4. Quasi-circuit switching and quasi-circuit switches

There are two fundamental approaches towards building core networks: circuit switching and packet switching. In circuit-switched networks, resources, including bandwidth and buffers, are reserved along a path for the duration of communication. As such, quality of service (QoS) is easily guaranteed. However, as resources are reserved for dedicated use, resources are not used efficiently in circuit-switched networks. On the other hand, as there is no resource reservation in packet-switched networks, resources could be used more efficiently in packet-switched networks. However, it is much more difficult to provide QoS in packet-switched networks.

It would be nice to have the advantages from both packet switching and circuit switching. For this, we propose the concept of *quasi-circuit switching*. Quasi-circuit switching is a generalization and an abstraction of the stop-and-go queueing proposed by Golestani [67, 68]. As in [67, 68], time in a quasi-circuit switched network is partitioned into frames. Flows (such as virtual circuits in ATM [72] and traffic trunks in MPLS [10]) entering a quasi-circuit switched network are rate controlled so that the number of bits of each flow in every frame is always bounded.

The key components in a quasi-circuit switched network are *quasi-circuit switches*. As long as the total rate at every link of a quasi-circuit switch does not exceed its link capacity, a quasi-circuit switch has the following property:

(P1) For any two packets with the same input and output links, the difference of their arriving frames is the same as the difference of their departing frames. If such a difference is (resp. bounded by) d, then the switch is called a quasi-circuit switch with exact (resp. maximum) frame delay d.

A quasi-circuit switch is order-preserving if it further satisfies the following property:

(P2) Packets that arrive during the same frame and have the same input and output links depart in the First Come First Serve (FCFS) order.

These two properties can then be propagated through a quasi-circuit switched network to provide QoS guarantees. As such, one can build a quasi-circuit switch by a network of quasi-circuit switches.

In this chapter, we will rehash several well-known switch architectures to quasi-circuit switches, including the shared memory switches, and the crosspoint buffered switches. Unfortunately, these known switch architectures are either too simple to scale or too complicate to build. For scalable quasi-circuit architectures, we propose several multi-stage switches. The first one is the frame based scheme in Section 3.5.1. In such an architecture, the memory access speed is only required to match the link speed. Moreover, it is on-line complexity is only $O(1)$.

To reduce the maximum frame delay, we propose the Clos quasi-circuit switches (with speedup). Unlike the classical Clos three-stage network, routing paths can be easily determined in the Clos quasi-circuit switches. By recursively expanding the Clos quasi-circuit switches, we derive the Benes quasi-circuit switches in which the routing path of a packet can be easily determined by the binary representation of the output of that packet. Moreover, the frame delay in the Benes quasi-circuit switch is substantially smaller than that in the load balanced Birkhoff-von Neumann quasi-circuit switch.

To increase the link utilization, one may allow packets to be dropped inside a quasi-circuit switch once the link capacity is exceeded. By so doing, one can then apply the Chernoff bound to infer the statistical QoS via measuring the average link utilization. As such, there is no need for centralized control as required in circuit switching and complicated packet scheduling as required in packet switching.

4.1 Quasi-circuit switching

4.1.1 Definitions and basic properties

In this section, we introduce the concept of quasi-switching. Quasi-switching is a generalization of the stop-and-go queueing in [67]. We will follow some terminologies used in [67]. As in [67], time is partitioned into frames. In this chapter, we only consider a single frame

size. Let T be the universal frame time. Everything will be normalized with respect to the frame time T.

Definition 4.1.1. *A link is said to be with capacity c if it is capable of transmitting c bits in a frame.*

Definition 4.1.2. *A flow is a sequence of packets with a common source and a common destination. A flow in a link is said to be (r, T)-smooth if the total number of bits (of the packets) of that flow in every frame in that link is bounded above by r bits. The quantity r is called the (peak) rate of that flow.*

It is not necessary to use *bits* as a measure for capacity. If packets are of the same size, one may describe the capacity of a link by the number of packets that it can transmit within a frame. Also, one can further partition a frame into time slots so that a fixed size packet can be transmitted within a time slot. By so doing, the (peak) rate of a flow can also be characterized by the (maximum) number of time slots that it occupies within a frame.

A switch that has M input links and N output links is called an $M \times N$ switch. In this chapter, we assume that all the links in a switch are *synchronized* so that all the frames at the input/output links start at the same time. In other words, there is no phase mismatch between any two links of a switch (the case with phase mismatch can be easily compensated by adding a delay constant as in [67]).

Consider an $M \times N$ switch. Let c_i^I, $i = 1, \ldots, M$, be the capacity of the i^{th} input link and c_k^O, $k = 1, \ldots, N$, be the capacity of the k^{th} output link. As there are M inputs and N outputs, there are MN local flows in an $M \times N$ switch. Let $A_{i,k}$ be the local flow for the packets that arrive at the i^{th} input and are destined to the k^{th} output of the $M \times N$ switch.

Definition 4.1.3. *The inputs of an $M \times N$ switch is said to satisfy the no overbooking conditions if (A1), (A2) and (A3) listed below hold.*

(A1) For all $i = 1, \ldots, M$, $k = 1, \ldots, N$, the local flow $A_{i,k}$ is $(r_{i,k}, T)$-smooth when it arrives at the i^{th} input of the switch.

(A2) The total rate coming out from an input link does not exceed its capacity, i.e.,

$$\sum_{k=1}^{N} r_{i,k} \le c_i^I, \quad i = 1, \ldots, M. \tag{4.1}$$

(A3) The total rate going to an output link does not exceed its capacity, i.e.,

$$\sum_{i=1}^{M} r_{i,k} \le c_k^O, \quad k = 1, \ldots, N. \tag{4.2}$$

An $M \times N$ switch is called a quasi-circuit switch *with maximum frame delay d if it has the following property when its inputs satisfy the no overbooking conditions:*

(P1) For any two packets of the same local flow, the difference of their arriving frames is the same as the difference of their departing frames. Moreover, such a difference is bounded above by d. To be precise, let m_1 and m_2 (resp. $\tilde{m}_1$ and $\tilde{m}_2$) be the arriving frame and the departing frame of the first (resp. second) packet. Then

$$m_2 - m_1 = \tilde{m}_2 - \tilde{m}_1 \le d. \tag{4.3}$$

We will also call a quasi-circuit switch with exact *frame delay d if the inequality in (4.3) is replaced by an equality. Also, a quasi-circuit switch is order-preserving if it further satisfies the following property:*

(P2) Packets of the same local flow that arrive during the same frame depart in the FCFS order.

To gain more intuition of quasi-circuit switching, consider the case that the frame time T is infinity. In this case, (P1) is irrelevant and quasi-circuit switching is reduced to the usual packet switching that only guarantees the order of packets if (P2) is satisfied. To see the connection between circuit switching and quasi-circuit switching, one may further partition a frame into a group of fixed size time slots. For circuit switching, assume that every packet can be transmitted in a time slot. Then circuit switching guarantees that for any two packets of the same local flow, the difference of their arriving *time slots* is the same as the difference of their departing *time slots*. This is much stronger than (P1). To summarize, quasi-circuit switching is between packet switching and circuit switching. In circuit switching, traffic is completely isolated by slot assignment. On the other hand, traffic is completely mixed in packet switching. Quasi-circuit switching uses frames to isolate traffic. As such, it can viewed as circuit switching at the time scale of frames. Thus, it can still provide some guarantees of quality of services (QoS) at the frame level. At the same time, quasi-circuit switching allows packets to be multiplexed within a frame.

Thus, it can achieve statistical multiplexing gain at the time scale of time slots. This is equivalent to the statistical line grouping in the circuit switching context (see e.g., [111]) and is known as the duration limited statistical multiplexing in [68].

There are several direct consequences from (P1) and (P2).

Proposition 4.1.4. *Consider an $M \times N$ quasi-circuit switch with maximum frame delay d. Suppose that its inputs satisfy the no overbooking conditions.*

(P3) The maximum packet delay is bounded by $(d+1)T$, where T is the universal frame time.
(P4) No packets are lost inside the switch.
(P5) The local flow $A_{i,k}$ is $(r_{i,k}, T)$-smooth when it departs from the k^{th} output link of the switch.
(P6) If the quasi-circuit switch is order-preserving, then packets of the same local flow depart in the FCFS order.

Proof. The proofs for (P3) and (P4) are rather straightforward and thus omitted.

Now we prove (P5). As we assume that the local flow $A_{i,k}$ is $(r_{i,k}, T)$-smooth when it arrives at the i^{th} input of the switch, there are at most $r_{i,k}$ bits (of the packets) of that local flow in every frame. From (4.3), we also know that packets of the same local flow depart during the same frame if and only if they arrive during the same frame. Thus, there are at most $r_{i,k}$ bits (of the packets) of the local flow $A_{i,k}$ in every frame of the k^{th} output link. This then implies the local flow $A_{i,k}$ is also $(r_{i,k}, T)$-smooth when it departs from the k^{th} output link of the switch.

To show (P6), we consider two packets of the same local flow. Let m_1 and m_2 (resp. $\tilde{m}_1$ and $\tilde{m}_2$) be the arriving frame and the departing frame of the first (resp. second) packet. If $m_1 = \tilde{m}_1$, we know from (P2) that these two packets depart in the FCFS order. On the other hand, if $m_1 < \tilde{m}_1$, then we have from (P1) that $m_2 < \tilde{m}_2$. Thus, these two packets departs in the FCFS order. ■

In the following examples, we provide several methods for constructing quasi-circuit switches.

Example 4.1.5. (Stop-and-go queueing in shared memory switches) The simplest way to build a quasi-circuit switch is to use the *shared memory* architecture in Section 2.1.1. Consider an $N \times N$ switch with an identical link capacity c. For a shared memory switch architecture (see Figure 2.1), packets that arrive during the same frame from all input links are read into a common shared memory. In the next frame, a central controller then writes all those packets (in the FCFS order for order-preserving) to the destined output links (according to an address lookup table). This is known as the stop-and-go queueing in [67].

During a frame time, the shared memory has to read all the packets from the current frame and write all the packets from the previous frame. Thus, its memory access speed must be at least $2Nc$. As such, the shared memory architecture is not *scalable* if the number of input/output links is large.

Note that under the no overbooking conditions, packets that arrive at the same frame will all depart in the next frame. Thus, the frame delay in the shared memory switch is exactly one frame time. Also, no packets are lost inside the switch if the buffer is larger than Nc bits.

Example 4.1.6. (Crosspoint buffered switches) One way to solve the memory access speed problem is to use the crosspoint buffered switch in Section 2.6.1. As shown in Figure 2.36, a crosspoint buffered switch consists of a crossbar switch fabric and separate buffers at the crosspoints. For an $N \times N$ switch, the number of separate buffers is N^2. Each buffer at a crosspoint stores packets from one input to one output. Call the $(i,j)^{th}$ buffer for the buffer at the crosspoint of the i^{th} input and the j^{th} output. To build an $N \times N$ quasi-circuit switch with capacity c, we allocate each buffer with buffer size larger than c. As the shared memory architecture, there are two phases of memory access. During the first phase of a frame, the $(i,j)^{th}$ buffers, $j = 1, 2, \ldots, N$, are connected to the i^{th} input link, $i = 1, 2, \ldots, N$. An arriving packet from the i^{th} input link and destined for the j^{th} output link is then placed in the $(i,j)^{th}$ buffer. During the second phase of a frame, the $(i,j)^{th}$ buffers, $i = 1, 2, \ldots, N$, are connected to the j^{th} output link, $j = 1, 2, \ldots, N$. Packets that are stored in the $(i,j)^{th}$ buffers, $i = 1, 2, \ldots, N$, are then read out sequentially to the j^{th} output link. Clearly, under the no overbooking conditions, no packets are

lost inside the switch and the frame delay is exactly one frame time. Moreover, the memory access speed for this architecture is only required to be twice of the link capacity. However, as pointed out in [3], the architecture based on crosspoint buffers does not scale for a large number of input/output links because of the square growth. It is only good either for small switches, or as basic architecture for the building blocks of modular multi-stage switches.

Example 4.1.7. (Load balanced Birkhoff-von Neumann quasi-circuit switches) Both switch architectures in Example 4.1.5 and Example 4.1.6 do not scale when the number of input/output ports is large. Consider the frame based scheme for the load balanced Birkhoff-von Neumann switch in Section 3.5. In such an architecture, (see Figure 3.24), there are two crossbar switch fabrics and buffers between these two crossbar switch fabrics. To build an $N \times N$ quasi-circuit switch with capacity c by this architecture, one first partitions time into fixed size frames. Both the $N \times N$ crossbar switches in Figure 3.24 have identical connection patterns for all time and change their connection patterns simultaneously at the beginning of every frame. The connection patterns are set up corresponding to a circular-shift matrix. As such, every input is connected to every output exactly once in every N frames.

There are N central buffers between these two switch fabrics. Each central buffer consists of two alternating memory blocks as the double-queue structure in [67]. The buffer size of each memory block is Nc bits, which is divided into N bins (indexed from 1 to N), each with buffer size of c bits. The j^{th} bin in every memory block is used for buffering packets that are destined for the j^{th} output, $j = 1, 2, \ldots, N$.

When a packet arrives at an input, it is immediately transferred to one of the central buffers that is connected to that input. According to the destination of the packet, the packet is placed in the bin that corresponds to the destination of that packet. When the bin is connected to the output, packets in that bin are all transferred to that output. It is shown in Theorem 3.5.2 of Section 3.5.1 that under the no overbook condition, the maximum packet delay is bounded by $2N - 1$ frame times. As such, the switch architecture described in Section 3.5 is a *quasi-circuit switch* with maximum frame delay $2N - 1$. The key feature of this architecture is its simplicity. Its on-line complexity is

$O(1)$ and it is independent of the number of input/output links. Such complexity is much lower than input-buffered packet switches. However, its frame delay is $O(N)$. In Section 4.1.2 and Section 4.2.1 below, we will show how to reduce the frame delay by building networks of quasi-circuit switches.

4.1.2 Networks of quasi-circuit switches

In this section, we consider a network interconnected by quasi-circuit switches. We assume that *the propagation delay to traverse through a link* is compensated so that all the frames in the input links of a quasi-circuit switch start at the same time. For the ease of the presentation, we assume that the propagation delay to traverse through a link is a positive integer multiple of Δ for some $\Delta > 0$. The constant Δ has nothing to do with the frame time T. It is only used as an induction step for the proof of Theorem 4.1.8 below.

In Section 4.1.1, only local flows are considered from input links to output links in a switch. In addition to local flows, we consider end-to-end flows that traverse through a series of links.

Suppose that there are I links in the network, indexed from 1 to I, and J end-to-end flows, indexed from 1 to J. Let c_i, $i = 1, \ldots, I$, be the capacity of the i^{th} link. Suppose that the j^{th} end-to-end flow is (r_j, T)-smooth when it arrives at the network. Let $D = (D_{i,j})$ be the $I \times J$ routing matrix, i.e., $D_{i,j} = 1$ if the j^{th} flow traverses through link i and $D_{i,j} = 0$ otherwise. Also, let $\boldsymbol{r}$ be the $J \times 1$ column vector with its j^{th} element being r_j, and $\boldsymbol{c}$ be the $I \times 1$ column vector with its i^{th} element being c_i. Note that the total rate at link i does not exceed its capacity if

$$\sum_{j=1}^{J} D_{i,j} r_j \leq c_i. \tag{4.4}$$

Thus, the condition that the total rate at every link does not exceed its capacity can be written in the following matrix form:

$$D\boldsymbol{r} \leq \boldsymbol{c}. \tag{4.5}$$

To summarize, we make the following assumptions:

(A1′) The j^{th} end-to-end flow is (r_j, T)-smooth when it arrives at the network (the first link of its route), $j = 1, \ldots, J$.

(A2′) The total rate at every link does not exceed its capacity, i.e., the inequality in (4.5) holds.
(A3′) The propagation delay to traverse a link is a positive integer multiple of Δ for some $\Delta > 0$.

Theorem 4.1.8. *Assume that (A1′)-(A3′) hold. Then the network of quasi-circuit switches has the following properties.*

(P1′) For any two packets of the same end-to-end flow, the difference of their arriving frames at the network is the same as the difference of their departing frames at every link traversed by the flow.
(P2′) If every quasi-circuit switch is order-preserving, then packets of the same end-to-end flow that arrive at the network during the same frame depart in the FCFS order at every link traversed by the flow.
(P3′) The maximum delay for packets of an end-to-end flow is bounded by the sum of the propagation delay of each link and the maximum delay of each quasi-circuit switch along its route.
(P4′) No packets are lost inside the network.
(P5′) For $j = 1, \ldots, J$, the j^{th} end-to-end flow is (r_j, T)-smooth at every link traversed by the flow.
(P6′) If every quasi-circuit switch is order-preserving, then packets of the same end-to-end flow depart the network in the FCFS order.

Proof. The key is to prove (P5′) by induction on time under (A1′)-(A3′). At time 0, the traffic of end-to-end flows only arrive at the network (as we assume the propagation delay of a link is positive in (A3′)). Thus, the induction hypothesis (P5′) holds trivially from (A1′).

Now suppose the induction hypothesis hold up to time $n\Delta$. From the induction hypothesis and (A2′), it follows that the no overbooking conditions in (A1)-(A3) in Definition 4.1.3 are satisfied for every quasi-circuit switch in the network. Using the same argument in the proof of (P5) in Proposition 4.1.4, one can easily show that the induction hypothesis (P5′) holds for every frame up to time $(n + 1)\Delta$ (as we assume the propagation delay of a link is positive in (A3′)).

That (P1′)-(P4′) and (P6′) hold follows trivially from (P1)-(P4) and (P6) in Definition 4.1.3 and Proposition 4.1.4. ■

Define the maximum jitter of an end-to-end flow as the difference between its maximum packet delay and its minimum delay through the network.

Corollary 4.1.9. *Under the conditions in Theorem 4.1.8, the maximum jitter of an end-to-end flow is bounded above by* $2T$.

Proof. Note from (P1′) that for any two packets of the same end-to-end flow, the difference of their arriving frames at the network is the same as the difference of their departing frames from the network. This then implies that the jitter is bounded by $2T$. ■

4.2 Recursive construction of quasi-circuit switches

4.2.1 Clos quasi-circuit switches

Theorem 4.1.8 suggests a way to construct a large quasi-circuit switch via a network of smaller quasi-circuit switches. **The idea is to use the three-stage Clos network architecture in Section 2.4.1.** To do this, we further assume in this section that *packets are of the same size*. Under such an assumption, each frame can be further partitioned into a group of time slots so that a packet can be transmitted within a time slot. Moreover, both the link capacity and the (peak) rate of a flow can be represented by the number of packets in a frame.

As shown in Figure 4.1, there are three stages of quasi-circuit switches. The first stage consists of q $p \times p$ quasi-circuit switches with exact frame delay d_1. Every input (resp. output) link of these switches has capacity c_1 (resp. c_2). That is, the maximum number of packets that can be transmitted in a frame in an input (resp. output) link of the switches at the first stage is c_1 (resp. c_2). Every one of the p outputs from a switch at the first stage is connected to an input of the p $q \times q$ quasi-circuit switches with exact frame delay d_2 at the second stage. Every one of the q outputs from a switch at the second stage is connected to an input of the q $p \times p$ quasi-circuit switches with exact frame delay d_3 at the third stage. Every input (resp. output) link of the switches at the third stage has capacity c_2 (resp. c_1). In this three-stage switch, there are pq inputs and pq outputs.

Number the switches at the first stage from 1 to p. Do the same for the switches at the third stage. As such, we can use the $(i, \ell)^{th}$ input, $i = 1, 2, \ldots, q$, $\ell = 1, 2, \ldots, p$, to denote the ℓ^{th} input of the i^{th} switch at the first stage. Similarly, we use the $(j, \ell)^{th}$ output, $j = 1, 2, \ldots, q$, $\ell = 1, 2, \ldots, p$, to denote the ℓ^{th} output of the j^{th} switch at

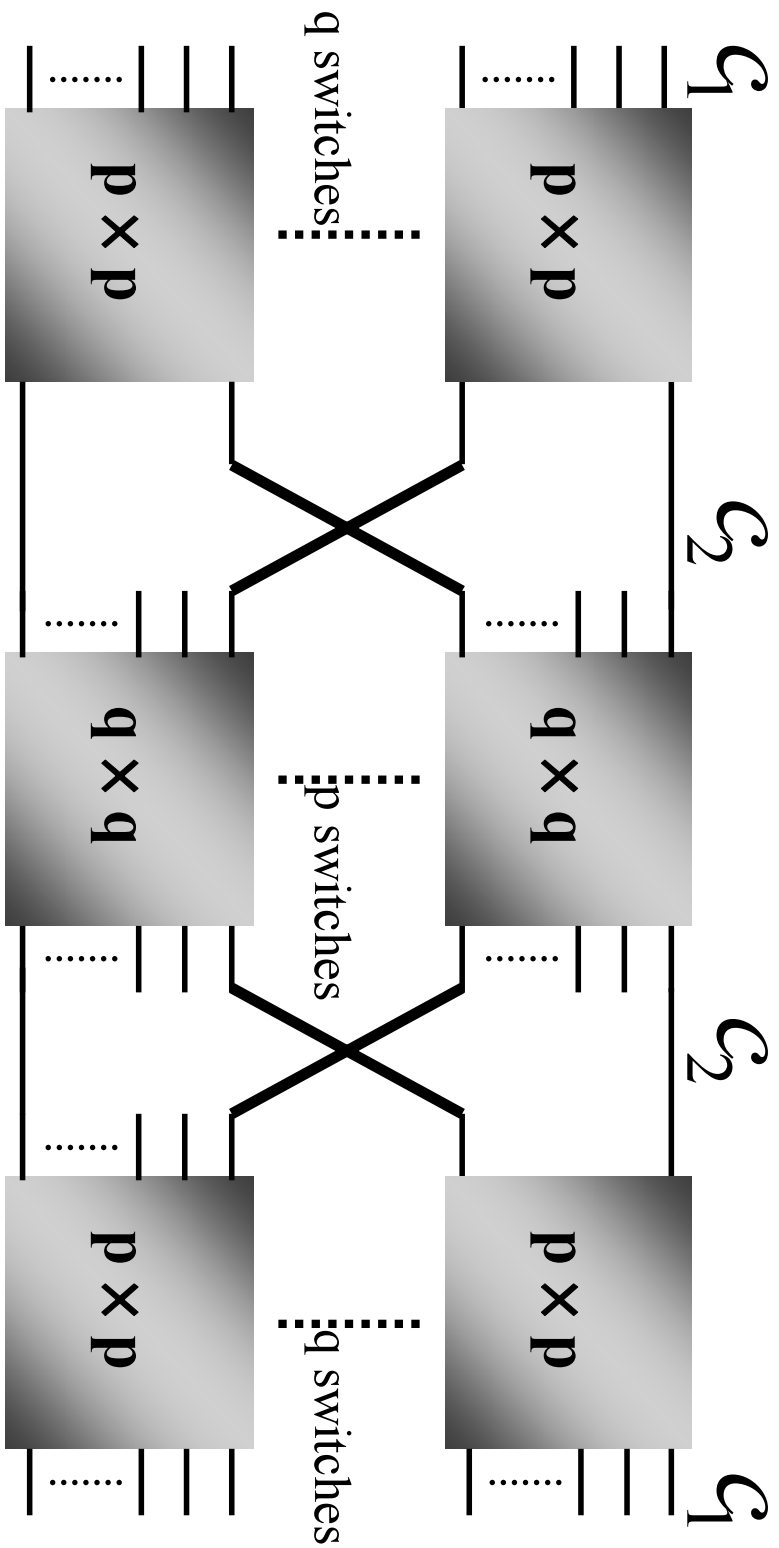


Fig. 4.1. A three-stage Clos quasi-circuit switch

the third stage. The flow from the $(i,\ell_1)^{th}$ input to the $(j,\ell_2)^{th}$ output is denoted by the $(i,\ell_1) \to (j,\ell_2)$ flow. Let $r_{(i,\ell_1),(j,\ell_2)}$ be (peak) rate of the $(i,\ell_1) \to (j,\ell_2)$ flow, i.e., the maximum number of packets of the $(i,\ell_1) \to (j,\ell_2)$ flow in a frame.

The key problem of the three-stage Clos network is *routing*. Traditionally, finding the routing path in such a network requires running sophisticated algorithms that may not be scalable. For instance, in the case without speedup, i.e., $c_2 = c_1$, finding the routing path for every packet relies on the well-known Birkhoff-von Neumann decomposition (see Proposition 2.3.1 and Proposition 2.3.3), and such an algorithm does not scale for a switch with a large number of input/output ports. The nice thing of the quasi-circuit switches is that routing can be solved *easily* at the cost of speedup, i.e., $c_2/c_1 > 1$. When $c_1 >> pq$, the cost of speedup is relatively small.

The idea is to use the round-robin routing policy for load balancing at the first stage. We consider the following two round-robin splitting policies.

Round-robin splitting:

(R1) Round-robin splitting for every flow at every input link: the k^{th} packet of the $(i,\ell_1) \to (j,\ell_2)$ flow in a frame is routed to the $(k \bmod p) + 1^{th}$ output at the first stage.

(R2) Round-robin splitting for every *aggregated* flow from all the input links at a switch of the first stage: consider the aggregated $i \to j$ flow from multiplexing all the $(i,\ell_1) \to (j,\ell_2)$ flows, $\ell_1, \ell_2 = 1,\ldots,p$. The k^{th} packet of the aggregated $i \to j$ flow in a frame is routed to the $(k \bmod p) + 1^{th}$ output at the first stage.

Note that there is a unique path for every packet to its destination from the second stage onward. For both round-robin routing policies, a packet destined to the $(j,\ell_2)^{th}$ output is routed to the j^{th} output at the second stage and the ℓ_2^{th} output at the third stage.

Clearly, the first scheme is much easier to do than the second one as it only needs the information for every flow at an input link. Thus, the first scheme in (R1) can be carried out *distributively at every input link*. However, for the second scheme in (R2), one needs a centralized control at every switch of the first stage as it has to gather the information of all the flows coming into that switch. A suitable architecture for the first scheme is the crosspoint buffered switch architecture in Example 4.1.6. On the other hand, the shared memory architecture in Example 4.1.5 is more suitable for the second scheme as every packet in the

shared memory architecture is processed sequentially. As we shall see in the following theorem, the needed speedup for the first scheme is substantially larger than that of the second scheme.

Theorem 4.2.1. *(i) Suppose that the three-stage Clos network in Figure 4.1 is operated under the round-robin splitting in (R1). If*

$$c_2 \geq c_1 + p^2 q, \tag{4.6}$$

then the network in Figure 4.1 is a $pq \times pq$ quasi-circuit switch with exact frame delay $d_1 + d_2 + d_3$.

(ii) Suppose that the three-stage Clos network in Figure 4.1 is operated under the round-robin splitting in (R2). If

$$c_2 \geq c_1 + q - 1, \tag{4.7}$$

then the network in Figure 4.1 is a $pq \times pq$ quasi-circuit switch with exact frame delay $d_1 + d_2 + d_3$.

To ensure that the quasi-circuit switch is order-preserving, one may add a resequencing element after each output link from the third stage. The resequencing element only needs to reorder packets in the same frame and causes an additional frame delay.

Proof. (i) For this network of quasi-circuit switches, the $(i, \ell_1) \to (j, \ell_2)$ flow is now split into p sub-flows as there are p routing paths from the $(i, \ell_1)^{th}$ input to the $(j, \ell_2)^{th}$ output (with each of them traversing through a central switch). As the routing policy is round-robin, the (peak) rate of each sub-flow is then bounded above by $\lceil \frac{r_{(i,\ell_1),(j,\ell_2)}}{p} \rceil$. Since all the quasi-circuit switches at the same stage have the same frame delay, it follows from (P1) that packets of these p sub-flows in the m^{th} frame from the $(i, \ell_1)^{th}$ input link will arrive at the $(j, \ell_2)^{th}$ output link in the $m + d_1 + d_2 + d_3{}^{th}$ frame if every link inside the network is not overbooked. To see that the no overbooking conditions are satisfied, we consider a particular output link at the i^{th} switch of the first stage. Let M be the total number of packets that can be routed to that link in a frame. Note that there are $p^2 q$ sub-flows traversing through that link and thus

$$\begin{aligned} M &\leq \sum_{\ell_1=1}^{p} \sum_{j=1}^{q} \sum_{\ell_2=1}^{p} \lceil \frac{r_{(i,\ell_1),(j,\ell_2)}}{p} \rceil \\ &\leq p^2 q + \sum_{\ell_1=1}^{p} \sum_{j=1}^{q} \sum_{\ell_2=1}^{p} \frac{r_{(i,\ell_1),(j,\ell_2)}}{p}. \end{aligned} \tag{4.8}$$

From the no overbooking conditions for every input link of the first stage switches, we have

$$\sum_{j=1}^{q}\sum_{\ell_2=1}^{p} r_{(i,\ell_1),(j,\ell_2)} \le c_1. \tag{4.9}$$

Using (4.9) and the assumption in (4.6), we then have

$$M \le p^2 q + c_1 \le c_2. \tag{4.10}$$

The argument for the output links at the second switch is similar.

(ii) The proof is quite similar to that in (i). It suffices to show that the no overbooking conditions are satisfied for every link inside the network. Let $r_{i,j} = \sum_{\ell_1=1}^{p}\sum_{\ell_2=1}^{p} r_{(i,\ell_1),(j,\ell_2)}$ be the rate of the $i \to j$ aggregated flow. Consider a particular output link at the i^{th} switch of the first stage. Let M be the total number of packets that can be routed to that link in a frame. As the routing policy is round-robin for the aggregated flows, we then have

$$M \le \sum_{j=1}^{q} \lceil \frac{r_{i,j}}{p} \rceil \le q - 1 + \lceil \sum_{j=1}^{q} \frac{r_{i,j}}{p} \rceil. \tag{4.11}$$

From the no overbooking condition in (4.9), we have

$$\sum_{j=1}^{q} r_{i,j} \le p c_1. \tag{4.12}$$

Using (4.12) and the assumption in (4.7), we then have

$$M \le q - 1 + c_1 \le c_2. \tag{4.13}$$

The argument for the output links at the second switch is similar.

■

In particular, if we choose $p = q = \sqrt{N}$ and $c_1 = N^{\frac{3}{2}}$ for the round-robin splitting scheme in (R1), then $c_2 = 2N^{\frac{3}{2}} = 2c_1$. Thus, for this case, a speedup factor of 2 is enough to build an $N \times N$ quasi-circuit switch by the three-stage architecture. By choosing a larger frame time T, one can further increase c_1 so that the speedup factor can be further reduced. For instance, if $c_1 = kN^{\frac{3}{2}}$, then $c_2/c_1 = 1 + \frac{1}{k}$. However, reducing the needed speedup is at the cost of delay as now the frame time is increased by k times.

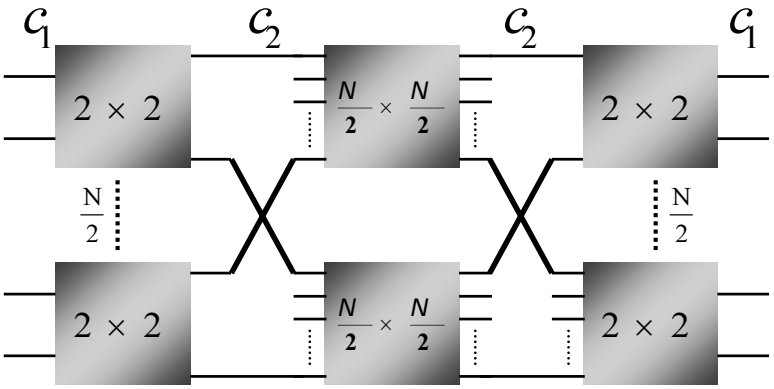


Fig. 4.2. Recursive construction of the Benes quasi-circuit switch

4.2.2 Benes quasi-circuit switches

Another interesting case of the Clos networks is when the total number of input/output links N is a power of 2. In this case, one can construct a Benes network by using 2×2 quasi-circuit switches. In Figure 4.2, one first constructs a three-stage Clos network with 2×2 quasi-circuit switches at the first stage and the last stage. Every input link at the first stage and every output link at the third stage has capacity c_1. The second stage consists of two $\frac{N}{2} \times \frac{N}{2}$ quasi-circuit switches with link capacity c_2. As N is a power of 2, each of the two $\frac{N}{2} \times \frac{N}{2}$ quasi-circuit switches in Figure 4.2 can be further implemented by a three-stage Clos network with 2×2 quasi-circuit switches at the first stage and the last stage, and two $\frac{N}{4} \times \frac{N}{4}$ quasi-circuit switches with link capacity c_3 at the second stage. One can expand the $\frac{N}{4} \times \frac{N}{4}$ switches recursively and we obtain a multi-stage switch with all 2×2 quasi-circuit switches. In Figure 4.3, we show an 8×8 Benes quasi-circuit switch. For an $N \times N$ Benes network, it is well-known that the number of stages of is $2 \log_2 N - 1$ and the number of 2×2 quasi-circuit switches is $N \log_2 N - \frac{N}{2}$.

Let $n = \log_2 N$. For the round-robin splitting scheme in (R1), one can easily verify from the recursive equation that one needs

$$c_{j+1} = c_j + \frac{4N}{2^j}, \quad j = 1, 2, \ldots, n-1. \tag{4.14}$$

As such, the link capacity of the most inner 2×2 quasi-circuit switch is

$$c_n = c_1 + 4N - 8. \tag{4.15}$$

On the other hand, for the round-robin splitting scheme in (R2), one needs that

$$c_{j+1} = c_j + \frac{N}{2^j} - 1, \quad j = 1, 2, \ldots, n-1. \tag{4.16}$$

As such, the link capacity of the most inner 2×2 quasi-circuit switch is

$$c_n = c_1 + N - \log_2 N - 1. \tag{4.17}$$

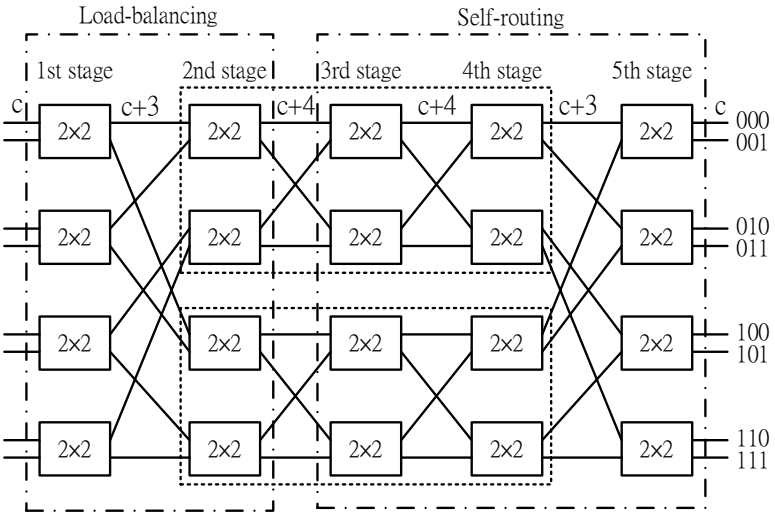


Fig. 4.3. An 8×8 Benes quasi-circuit switch using the second round-robin scheme

Unlike the classical Benes network, finding the routing path in the Benes quasi-circuit switch is very easy. Index the output links at the last stage from $0, 1, 2, \ldots, N-1$. Consider a packet that is destined for output j (at the last stage). Let $b_1 b_2 \ldots b_n$ be the n-bit binary representation of j, e.g., 100 for $j = 4$ in Figure 4.3. This binary representation is added in the packet header. As in the three-stage Clos quasi-circuit switch, the routing policy used in the Benes quasi-circuit switch consists of two parts: self-routing and load-balancing.

(i) Self-routing: when the packet arrives at stage $n - 1 + j$, $j = 1, 2, \ldots, n$, the packet is routed to the upper link if $b_j = 0$ and the lower link if $b_j = 1$.

(ii) Load-balancing: when the packet arrives at stage j, $j = 1, 2, \ldots, n-1$, and it is the k^{th} packet in that frame that has the same

$b_1 b_2 \ldots b_{n-j+1}$'s in its packet header, then the packet is routed to the upper link if k is even, and the lower link if k is odd. This can be implemented by keeping a 2^{n-j+1}-bit vector at each input link of stage j. Let $\ell = b_1 2^0 + b_2 2^1 + \ldots + b_{n-j+1} 2^{n-j}$. Then the packet is routed to the upper link if ℓ^{th} bit of that bit vector is 0 and the lower link otherwise. Moreover, the ℓ^{th} bit of that bit vector is toggled after the packet is routed.

Note that for the round-robin splitting scheme in (R1), load-balancing is performed (and implemented) for the flows at each input link. On the other hand, for the round-robin splitting scheme in (R2), load-balancing is performed for the aggregated flows of all the input links.

Example 4.2.2. (Benes quasi-circuit switch) In this example, we argue that one can build a 256×256 Benes quasi-circuit switch with OC-192 input/output links using today's technology for memory. Suppose that we choose the universal frame time $T = 125\mu s$. Furthermore, the frame is partitioned into time slots, each is capable of transmitting 64 bytes. For an OC-192 link, its link capacity is 64 times of that of an OC-3 link (149.760 Mbits/sec [72]). Each OC-192 link is roughly 10 Gbits/sec and the overall capacity of this switch is then roughly 2.56 Tbits/sec.

One can easily calculate that the capacity of an OC-192 link in this setting, denoted by c_1, is 2340 time slots/frame. Suppose that we use the shared memory architecture in Example 4.1.5 to build each of the 2×2 quasi-circuit switch. As such, we may use the round robin splitting scheme in (R2). From (4.17), the needed speedup is $1 + \frac{N - \log_2 N - 1}{c_1} \approx 1.11$ (with $N = 256$). The needed memory access speed is around $10 + 10 + 10 * 1.11 + 10 * 1.11 = 42.2$ Gbits/sec. This is within the limit of the memory access speed in today's technology.

Note that there are $2 \log_2 N - 1$ stages in the Benes network and each stage causes exactly one frame delay. Adding the additional one frame delay for resequencing after the last stage, the maximum delay is $2(\log_2 N)T = 2ms$ (if the input/output links satisfy the no over-booking conditions). Note that the maximum delay in the load balanced Birkhoff-von Neumann switch architecture in Example 4.1.7 is $(2N - 1)T = 63.875ms$, which is substantially larger than that in the corresponding Benes quasi-circuit switch.

4.3 Lossy quasi-circuit switches

In Section 4.1.2, we have developed a scalable QoS architecture based on quasi-circuit switches. Such an architecture ensures that every packet is delivered within a maximum delay. As such, no packets are lost inside the network. However, the (deterministic) no overbooking condition in (A2′) might lead to low utilization. To increase the utilization, we may relax the (deterministic) no overbooking condition in (A2′) and allow some links in the network to be overbooked. When a link is overbooked in a frame, packets are dropped. By controlling the probability that a frame in a link is overbooked, we can still retain QoS while keeping a very low packet loss probability.

In the following, we define the concept of *lossy* quasi-circuit switches.

Definition 4.3.1. *The inputs of an $M \times N$ switch is said to satisfy the no overbooking conditions with* losses *if (A1) in Definition 4.1.3, (A2L) and (A3L) below hold.*

(A2L) The total number of bits coming out from an input link in a frame does not exceed its capacity.

(A3L) Packets are dropped only when the total number of bits going to an output link in a frame exceeds its capacity.

An $M \times N$ switch is called a lossy *quasi-circuit switch with maximum frame delay d if (P1) in Definition 4.1.3 hold for all packets that are not dropped by the switch when its inputs satisfy the no overbooking conditions with* losses.

Note that (A2L) is weaker than (A2) in Definition 4.1.3. The assumption in (A2) ensures that the total (peak) rate does not exceed the capacity, while (A2L) only assumes that the *actual* rate does not exceed the capacity. In (A3L), we do not specify how packets are dropped. The key point of achieving good quality of service is to minimize the probability that there is a link exceeding its capacity in a frame. Following the same proof in Theorem 4.1.8, it is easy to see that Theorem 4.1.8 still holds for a network of lossy quasi-circuit switches except (P4′).

4.3.1 Statistical multiplexing gain in high speed switching

To see the statistical multiplexing gain in high speed switching, consider the network model in Section 4.1.2 except that the link capacities are now increased n times, i.e., the capacity of the i^{th} link is

nc_i, $i = 1, \ldots, I$. As the capacities are increased, more flows can be admitted to the network. We call an end-to-end flow a class j flow, $j = 1, \ldots, J$, if the flow is (r_j, T)-smooth and it follows the same route as the j^{th} flow in Section 4.1.2. In view of the no overbooking conditions in (4.4), the network now can admit n flows for each class without exceeding the link capacity at each link.

Suppose that we have further information about the class j flow.

(A4′) The number of bits in a frame in a class j flow is a random variable with mean $\bar{r}_j < r_j$ (when the flow arrives at the network). This is independent of any other flow.

The following theorem illustrates how statistical multiplexing gain can be achieved.

Theorem 4.3.2. *Suppose that there are $n\alpha_j$ class j flows admitted to the network for some $\alpha_j \geq 1$, $j = 1, \ldots, J$. If*

$$\sum_{j=1}^{J} D_{i,j}\bar{r}_j\alpha_j < c_i, \tag{4.18}$$

then the probability that packets are dropped at the i^{th} link in a frame is bounded above by

$$e^{-n\Lambda_i^*(c_i)}, \tag{4.19}$$

where

$$\Lambda_i^*(c_i) = \sup_{\theta \geq 0}[\theta c_i - \sum_{j=1}^{J} D_{i,j}\alpha_j\Lambda_j(\theta)], \tag{4.20}$$

and

$$\Lambda_j(\theta) = \log\Big(\frac{\bar{r}_j}{r_j}e^{r_j\theta} + (1 - \frac{\bar{r}_j}{r_j})\Big), \quad j = 1, \ldots, J. \tag{4.21}$$

Note that $\Lambda_i^*(\cdot)$ is known as the (one-sided) Legendre transform (see e.g., Rockafellar [143], Avriel [9]) of the sum of $\Lambda_j(\theta)$'s. As such, $\Lambda_i^*(\cdot) > 0$ under the condition in (4.18). When n is large, the probability that there exists a link exceeding its capacity in a frame can be made very small.

Proof. Consider a particular frame at link i. Let $\tilde{R}_{j,\ell}$ be the random variable that represents the number of bits of the ℓ^{th} class j flow in that frame, $\ell = 1, \ldots, n\alpha_j$, $j = 1, \ldots, J$. Furthermore, let $R_{j,\ell}$ be the

number of bits generated by the flow in that frame when the flow enters the network. Due to possible packet loss in the quasi-circuit switches along the route, $\tilde{R}_{j,\ell} \leq R_{j,\ell}$. The event that packets are dropped at the i^{th} link in that frame is the same as the event that the i^{th} link exceeds its capacity in that frame, i.e., $\{\sum_{j=1}^{J} D_{i,j} \sum_{\ell=1}^{n\alpha_j} \tilde{R}_{j,\ell} > nc_i\}$. From the Chernoff bound (see Problem 5), it follows that for all $\theta \geq 0$

$$\begin{aligned}
&\mathsf{P}\Big(\sum_{j=1}^{J} D_{i,j} \sum_{\ell=1}^{n\alpha_j} \tilde{R}_{j,\ell} > nc_i\Big) \\
&\leq \mathsf{P}\Big(\sum_{j=1}^{J} D_{i,j} \sum_{\ell=1}^{n\alpha_j} R_{j,\ell} > nc_i\Big) \\
&\leq e^{-\theta nc_i} \mathsf{E} \exp\Big(\theta \sum_{j=1}^{J} D_{i,j} \sum_{\ell=1}^{n\alpha_j} R_{j,\ell}\Big). \qquad (4.22)
\end{aligned}$$

From (A4′) and the assumption that the j^{th} end-to-end flow is (r_j, T)-smooth, we know that $R_{j,\ell}$ is a random variable that has the mean $\bar{r}_j$ and the bounded support $[0, r_j]$. It is well-known (see Problem 15) that the moment generating function of such a random variable is bounded above by the Bernoulli random variable R with $\mathsf{P}(R = r_j) = \frac{\bar{r}_j}{r_j}$ and $\mathsf{P}(R = 0) = 1 - \frac{\bar{r}_j}{r_j}$. Thus,

$$\mathsf{E} e^{\theta R_{j,\ell}} \leq e^{\Lambda_j(\theta)},$$

where $\Lambda_j(\theta)$ is defined in (4.21). As we assume independence in (A4′), we have for all $\theta \geq 0$

$$\begin{aligned}
&\mathsf{E} \exp\Big(\theta \sum_{j=1}^{J} D_{i,j} \sum_{\ell=1}^{n\alpha_j} R_{j,\ell}\Big) = \prod_{\{j: D_{i,j}=1\}} \prod_{\ell=1}^{n\alpha_j} \mathsf{E} e^{\theta R_{j,\ell}} \\
&\leq \prod_{\{j: D_{i,j}=1\}} (e^{\Lambda_j(\theta)})^{n\alpha_j} = \exp\Big(n \sum_{j=1}^{J} D_{i,j} \alpha_j \Lambda_j(\theta)\Big). \qquad (4.23)
\end{aligned}$$

Using (4.23) in (4.22), we have

$$\begin{aligned}
&\mathsf{P}\Big(\sum_{j=1}^{J} D_{i,j} \sum_{\ell=1}^{n\alpha_j} \tilde{R}_{j,\ell} > nc_i\Big) \leq e^{-\theta nc_i} \exp\Big(n \sum_{j=1}^{J} D_{i,j} \alpha_j \Lambda_j(\theta)\Big) \\
&= \exp\Big(-n[\theta c_i - \sum_{j=1}^{J} D_{i,j} \alpha_j \Lambda_j(\theta)]\Big) \qquad (4.24)
\end{aligned}$$

Choosing the best bound in the exponent yields

$$\begin{aligned}
&\mathsf{P}\Big(\sum_{j=1}^{J} D_{i,j} \sum_{\ell=1}^{n\alpha_j} \tilde{R}_{j,\ell} > nc_i\Big) \\
&\leq \exp\Big(-n \sup_{\theta\geq 0}[\theta c_i - \sum_{j=1}^{J} D_{i,j}\alpha_j \Lambda_j(\theta)]\Big) \\
&= \exp(-n\Lambda_i^*(c_i)), \qquad (4.25)
\end{aligned}$$

where $\Lambda_i^*(c_i)$ is defined in (4.20). ■

The vector $\alpha = (\alpha_1, \ldots, \alpha_J)$ can be viewed as the statistical multiplexing gain. We further illustrate the statistic multiplexing gain in the following example.

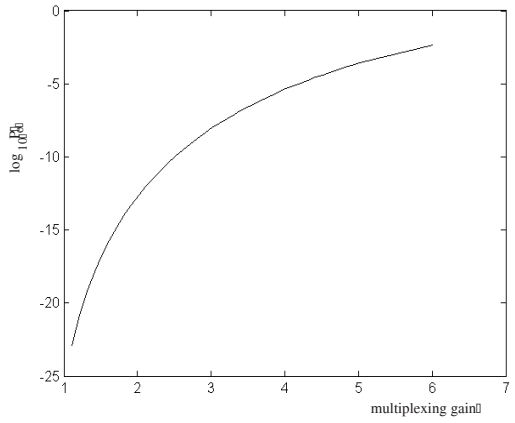


Fig. 4.4. $\log_{10} P_o$ as a function of multiplexing gain

Example 4.3.3. (Multiplexing gain) Consider the case with a single link. For an OC-3 link, its link rate is 149.760 Mbits/sec (155.520Mbits/sec with overhead [72]). Suppose we choose the frame time T to be 125μs. Then the capacity of an OC-3 link is 18720 bits/frame (2340 bytes/frame). The maximum packet size for an Ethernet packet is 1500 bytes. Thus, a flow of Ethernet packets is (r, T)-smooth with $r = 1500 \times 8$ =12000 bits and $T = 125\mu s$. To ensure the no overbooking conditions, one can admit at most one flow of Ethernet packets in an OC-3 link no matter how its average rate might be.

To see the multiplexing gain, suppose that the average rate for a flow of Ethernet packets is 8 Mbits/sec, or equivalently $\bar{r} = 1000$ bits in a frame. For an OC-48 link (16 times of the rate of OC-3), its link capacity is 292320 bits/frame. As a flow of Ethernet packets is (r, T)-smooth with $r = 12000$ bits and $T = 125\mu s$, only $n = \lfloor 292320/12000 \rfloor = 16$ flows can be admitted to the link under the (deterministic) no overbooking conditions. Let P_o be the upper bound in (4.19) for the probability that packets are dropped in a frame. With $c = 292320/n$, in Figure 4.4 we plot $\log_{10} \mathsf{P}_o$ as a function of the multiplexing gain α for an OC-48 link. As shown in Figure 4.4, one can achieve 2.7 times of multiplexing gain while keeping the probability that packets are dropped in a frame smaller than 10^{-9}.

4.3.2 Inferring QoS via the average link utilization

We have shown in Section 4.1.2 and Section 4.3.1 how one achieves QoS via appropriate admission control. To implement an admission control mechanism, it is usually required for a router to gather the routing information (the routing matrix) and the detailed statistics of each flow (the average rate). All these additional tasks might slow down a router and cause performance degradation.

Instead of using admission control, in this section we take a much simpler approach to infer the quality of service. We admit all the flows to the network (considered in Section 4.3.1). However, we assume that flows come and go so that we can have the flow level multiplexing. To be specific, assume that class j flows arrive according to a Poisson process with rate $n\lambda_j$. The duration of a class j flow is a random variable with mean $1/\mu_j$ and this is independent of everything else. Then it follows from the classical theory for $M/G/\infty$ queues (see Problem 12) that the number of class j flows in any frame is a Poisson random variable with mean $n\lambda_j/\mu_j$. Moreover, these random variables are independent of each other.

Let

$$r_{\max} = \max_{1 \le j \le J} r_j \tag{4.26}$$

be the maximum peak rate of the flows entering the network and

$$\rho_i = \frac{\sum_{j=1}^{J} D_{i,j} \bar{r}_j \frac{\lambda_j}{\mu_j}}{c_i} \tag{4.27}$$

be the average utilization of link i. In the following theorem, we show how one infers the quality of service via the average link utilization.

Theorem 4.3.4. *For the model without admission control, if*

$$\rho_i < 1, \tag{4.28}$$

then the probability that packets are dropped at the i^{th} link in a frame is bounded above by $\exp(-\Gamma(c_i, \rho_i, r_{\max}))$*, where*

$$\Gamma(c_i, \rho_i, r_{\max}) = \frac{c_i}{r_{\max}}(\rho_i - 1 - \log \rho_i). \tag{4.29}$$

Proof. As in the proof of Theorem 4.3.2, consider a particular frame at link i. Let N_j be the number of class j flows in that frame, and $R_{j,\ell}$ be the random variable that represents the number of bits of the ℓ^{th} class j flow in that frame, $\ell = 1, \ldots, N_j$, $j = 1, \ldots, J$. The event that packets are dropped at the i^{th} link in that frame is the same as the event that the i^{th} link exceeds its capacity in that frame, i.e., $\{\sum_{j=1}^{J} D_{i,j} \sum_{\ell=1}^{N_j} R_{j,\ell} > nc_i\}$. Using the Chernoff bound and the fact that N_j's are independent Poisson random variables with mean $n\lambda_j/\mu_j$, it follows from a similar argument to the proof of Theorem 4.3.2 that for all $\theta \geq 0$

$$\begin{aligned} &\log \mathsf{P}\Big(\sum_{j=1}^{J} D_{i,j} \sum_{\ell=1}^{N_j} R_{j,\ell} > nc_i\Big) \\ &\leq -n \sup_{\theta \geq 0}[\theta c_i - \sum_{j=1}^{J} D_{i,j} \frac{\lambda_j}{\mu_j}(e^{\Lambda_j(\theta)} - 1)], \end{aligned} \tag{4.30}$$

where $\Lambda_j(\theta)$'s are defined in (4.21).

Since flow j is (r_j, T)-smooth, it is also $(r_{\max}, T)$-smooth. Replacing $\Lambda_j(\theta)$ in (4.30) with

$$\log\Big(\frac{\bar{r}_j}{r_{\max}} e^{r_{\max}\theta} + (1 - \frac{\bar{r}_j}{r_{\max}})\Big)$$

yields

$$\begin{aligned} &\log \mathsf{P}\Big(\sum_{j=1}^{J} D_{i,j} \sum_{\ell=1}^{N_j} R_{j,\ell} > nc_i\Big) \\ &\leq -\frac{nc_i}{r_{\max}}(\rho_i - 1 - \log \rho_i) \\ &= -n\Gamma(c_i, \rho_i, r_{\max}). \end{aligned} \tag{4.31}$$

As $\log x < x - 1$ for all $x \neq 1$, we have $\Gamma(c_i, \rho_i, r_{\max})$ for $\rho_i < 1$. Thus, under the condition in (4.28), this probability can be made very small if the link capacity nc_i is much larger than $r_{\max}$. ■

In view of Theorem 4.3.4, one only needs to know three things to infer the quality of a link: the average utilization ρ_i, the link capacity nc_i and the maximum rate of a flow $r_{\max}$. There is no need to know the routing matrix D and the detailed statistics of each flow (the flow arrival rate $n\lambda_j$, the average duration of a flow $1/\mu_j$, and the average rate $\bar{r}_j$, $j = 1, \ldots, J$). Note that the information for the average utilization of a link can be collected *locally* by measuring the average number of bits used in a frame. As such, the network can be managed in a distributed manner and one might not need to implement an admission control mechanism to ensure the condition in (4.28). In the following example, we illustrate how one infers the statistical QoS without admission control.

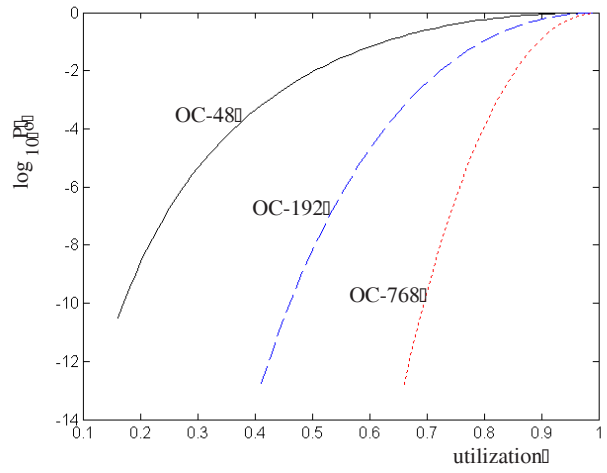


Fig. 4.5. $\log_{10} P_o$ as a function of the average link utilization

Example 4.3.5. (Statistical QoS) As in Example 4.3.3, consider the single link case with the frame time $T = 125\mu s$. Suppose that every flow is a flow of Ethernet packets and there is at most one packet in a frame for each flow. Thus, each flow is (r, T)-smooth with $r = 12000$

bits. Note that the capacity of an OC-48 link is 292320 bits/frame. The capacity of an OC-192 link is four times larger than the capacity of an OC-48 link. Similarly, the capacity of an OC-768 link is 16 times larger than the capacity of an OC-48 link. Let P_o be the bound in Theorem 4.3.4 for the probability that packets are dropped in a frame in the link. In Figure 4.5, we plot $\log_{10} P_o$ as a function of the average link utilization for the OC-48, OC-192 and OC-768 links. Suppose that we would like to operate the link so that $\mathsf{P}_o \leq 10^{-9}$. Then for an OC-48 link, the average utilization cannot exceed 19%. For an OC-192 link, the average utilization can be raised to 48%. For an OC-768 link, the average utilization can be even raised to 70%. Now suppose we lower the requirement to $\mathsf{P}_o \leq 10^{-6}$. Then the average utilization for an OC-48 (resp. OC-192,OC-768) link is roughly 27% (resp. 59%,75%).

4.4 Notes

In this chapter we developed the theory of quasi-circuit switching and the associated methods of building quasi-circuit switches. Most of the material of this chapter was adopted from the unpublished manuscript by the authors [33]. In comparison with packet switching and circuit switching, quasi-circuit switching has the following nice features:

(i) Quasi-circuit switching can be viewed as circuit switching at the time scale of frames. As such, QoS can still be provided at the frame level.

(ii) Quasi-circuit switching allows packets to be multiplexed within a frame. As such, it can achieve statistical multiplexing gain within frames.

(iii) Quasi-circuit switches are much more scalable than packet switches and circuit switches in the context of high speed switching. In particular, the load balanced Birkhoff-von Neumann quasi-circuit switch has the on-line complexity of $O(1)$. Such complexity is much lower than input-buffered packet switches. Moreover, finding routing paths in the Benes network of quasi-circuit switches is much easier than the corresponding Benes network of circuit switches.

(iv) By allowing packets to be dropped in a quasi-circuit switch, statistical QoS can be easily inferred by measuring the average link utilization. There is no need for centralized control as required in

circuit switching and complicated packet scheduling as required in packet switching.

Problems

1. Show how to build an $N \times N$ quasi-circuit switch with the input-buffered switch architecture. (Hint: use the Birkhoff-von Neumann decomposition.)
2. Verify the recursive equations in (4.14) and (4.16).

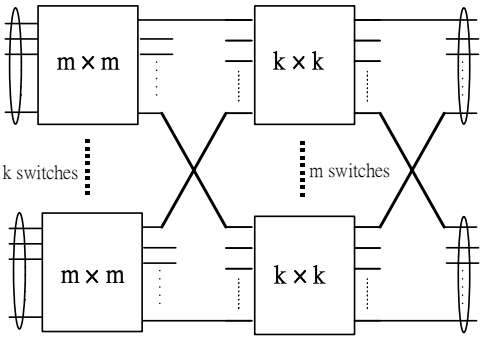


Fig. 4.6. A 2X construction for $k \times k$ quasi-circuit switch (in space)

3. (Line grouping) In Figure 4.6, it is a two-stage switch using the 2X construction. Instead of defining a frame as a period of time, one may define a frame as a group of lines (this is known as line grouping in circuit switching). In Figure 4.6, one may view the m inputs in every $m \times m$ switch at the first stage as a frame (or a group). Similarly, one can also do the same for the outputs as shown in Figure 4.6. Use the Birkhoff-von Neumann decomposition (in the proof of Theorem 2.4.2) to show that such a 2X construction can be operated as a $k \times k$ quasi-circuit switch (in space).
4. (Markov inequality) Consider a nonnegative random variable X. Show that for all $x > 0$,

$$\mathsf{P}(X \geq x) \leq \frac{\mathsf{E}[X]}{x}.$$

5. (Chernoff bound) Consider a random variable X. Use the Markov inequality to show that for all $\theta > 0$,

$$\mathsf{P}(X \geq x) = \mathsf{P}(e^{\theta X} \geq e^{\theta x}) \leq e^{-\theta x}\mathsf{E}[e^{\theta X}].$$

6. (Doubly stochastic Poisson process) A stochastic process $\{N(t), t \geq 0\}$ is a counting process if $N(t)$ is integer-valued and it is nondecreasing in t with $N(0) = 0$. To understand the meaning of a counting process $\{N(t), t \geq 0\}$, one may simply view $N(t)$ as the cumulative number of arrivals by time t. A counting process $\{N(t), t \geq 0\}$ is a doubly stochastic Poisson process with the intensity process $\{\lambda(t), t \geq 0\}$ if the event that there is an arrival in $[t, t + \Delta t]$ is with probability $\lambda(t)\Delta t$ for very small Δt, i.e.,

$$\mathsf{P}(N(t + \Delta t) - N(t) = 1) \approx \lambda(t)\Delta t,$$

and this event is independent of the past. On the other hand, the event that there is no arrival in $[t, t + \Delta t]$ is with probability $1 - \lambda(t)\Delta t$ for very small Δt, i.e.,

$$\mathsf{P}(N(t + \Delta t) - N(t) = 0) \approx 1 - \lambda(t)\Delta t,$$

and this event is also independent of the past. A doubly stochastic Poisson process is called a Poisson process with rate λ if its intensity process $\lambda(t) = \lambda$ for all t. For a Poisson process with rate λ, show that

$$\begin{aligned}
&\mathsf{P}(N(t + \Delta t) = k) \\
&= \mathsf{P}(N(t) = k)\mathsf{P}(N(t + \Delta t) - N(t) = 0) \\
&\qquad + \mathsf{P}(N(t) = k - 1)\mathsf{P}(N(t + \Delta t) - N(t) = 1) \\
&= \mathsf{P}(N(t) = k)(1 - \lambda\Delta t) + \mathsf{P}(N(t) = k - 1)\lambda\Delta t. \quad (4.32)
\end{aligned}$$

Let $p_k(t) = \mathsf{P}(N(t) = k)$. Use (4.32) to derive the following differential equation:

$$\frac{dp_k(t)}{dt} = \lambda(p_{k-1}(t) - p_k(t)), \tag{4.33}$$

with the initial conditions $p_0(0) = 1$ and $p_k(0) = 0$ for $k > 0$. Verify that

$$p_k(t) = \frac{e^{-\lambda t}(\lambda t)^k}{k!}$$

is the solution of the above differential equation. Thus, $N(t)$ is a Poisson random variable with parameter λt.

7. (Superposition of Poisson processes) Continue from the previous problem. Suppose that $\{N_1(t), t \geq 0\}$ is a doubly stochastic Poisson process with the intensity process $\{\lambda_1(t), t \geq 0\}$ and $\{N_2(t), t \geq 0\}$ is a doubly stochastic Poisson process with the intensity process $\{\lambda_2(t), t \geq 0\}$. If $\{N_1(t), t \geq 0\}$ and $\{N_2(t), t \geq 0\}$ are independent, show that $\{N_1(t)+N_2(t), t \geq 0\}$ is also a doubly stochastic Poisson process with the intensity process $\{\lambda_1(t) + \lambda_2(t), t \geq 0\}$.
8. (Random splitting of a Poisson process) Suppose that $\{N(t), t \geq 0\}$ is a doubly stochastic Poisson process with the intensity process $\{\lambda(t), t \geq 0\}$. Construct two counting processes $\{N_1(t), t \geq 0\}$ and $\{N_2(t), t \geq 0\}$ by randomly (and independently) selecting an arrival from $\{N(t), t \geq 0\}$ to $\{N_1(t), t \geq 0\}$ with probability p. Those arrivals not selected by $\{N_1(t), t \geq 0\}$ form $\{N_2(t), t \geq 0\}$. Show that $\{N_1(t), t \geq 0\}$ is a doubly stochastic Poisson process with the intensity process $\{p\lambda(t), t \geq 0\}$ and $\{N_2(t), t \geq 0\}$ is a doubly stochastic Poisson process with the intensity process $\{(1-p)\lambda(t), t \geq 0\}$. (In fact, it can be further shown that $\{N_1(t), t \geq 0\}$ and $\{N_2(t), t \geq 0\}$ are independent.)
9. (Shift of a Poisson process) Suppose that $\{N_1(t), t \geq 0\}$ is a doubly stochastic Poisson process with the intensity process $\{\lambda_1(t), t \geq 0\}$. Construct another counting process $\{N_2(t), t \geq 0\}$ by delaying every arrival from $\{N_1(t), t \geq 0\}$ by z. Show that $\{N_2(t), t \geq 0\}$ is a doubly stochastic Poisson process with the intensity process $\{\lambda_1(t-z), t \geq 0\}$.
10. (Random shift of a Poisson process) Suppose that $\{N_1(t), t \geq 0\}$ is a doubly stochastic Poisson process with the intensity process $\{\lambda_1(t), t \geq 0\}$. Construct another counting processes $\{N_2(t), t \geq 0\}$ by randomly (and independently) delaying an arrival from $\{N_1(t), t \geq 0\}$ according to a density function $f(z)$. Use the results in Problems 7-9 to show that $\{N_2(t), t \geq 0\}$ is a doubly stochastic Poisson process with the intensity process $\{\int_0^\infty \lambda_1(t-z)f(z)dz, t \geq 0\}$. In particular, if $\{N_1(t), t \geq 0\}$ is a Poisson process with rate λ, then $\{N_2(t), t \geq 0\}$ is also a Poisson process with rate λ (in the "steady state" as $t \to \infty$).
11. An $M/D/\infty$ queue is a queueing system with the arrival process being a Poisson process. Every arrival to the queueing system is delayed by the same constant. Suppose that the arrival process is a Poisson process with rate λ and the constant delay is z. Show that (in steady state) the number of customers in the $M/D/\infty$ queue is

a Poisson random variable λz (Hint: given a time t, those arrivals are still in the $M/D/\infty$ queue at time t if and only if they arrive within $[t-z,t]$).

12. An $M/G/\infty$ queue is a queueing system with the arrival process being a Poisson process. Every arrival to the queueing system is randomly delayed according to a general distribution. Suppose that the arrival process is a Poisson process with rate λ and the density function of the delay is $f(z)$. Let $1/\mu = \int_0^\infty zf(z)$ be the mean delay. Show that (in steady state) the number of customers in the $M/G/\infty$ queue is a Poisson random variable λ/μ (Hint: use the random splitting property in Problem 8 to decompose the arrival process into several independent Poisson processes such that each of them has a constant delay. Then use Problem 11 and the superposition property in Problem 7 to show the desired result.)
13. (Departure property) Continue from the previous problem. Use the result in Problem 10 to show that the departure process from an $M/G/\infty$ queue is also a Poisson process (in the steady state).
14. (A network of $M/G/\infty$ queues) In a network of $M/G/\infty$ queues, customers of the same class arrive to the network according to a Poisson process, and they are randomly delayed at every queue according to a general distribution. Using the departure property in Problem 13, the superposition property in Problem 7 and the random splitting property in Problem 8 to show that the arrival process to every queue in a network of $M/G/\infty$ queues is also a Poisson process. Moreover, every $M/G/\infty$ can be analyzed in isolation once the overall arrival rate to that queue is known. (see [129] for more details).
15. (Variability ordering) A nonnegative random variable X is more variable than another nonnegative random variable Y, denoted by $X \geq_v Y$, if for all $c \geq 0$,

$$\int_c^\infty \mathsf{P}(X \geq x)dx \geq \int_c^\infty \mathsf{P}(Y \geq x)dx.$$

It is known (see e.g., [145, 154]) that if $X \geq_v Y$ and $\mathsf{E}[X] = \mathsf{E}[Y]$, then $\mathsf{E}[f(X)] \geq \mathsf{E}[f(Y)]$ for all convex function f. Use this to show that the moment generating function of a random variable with bounded support $[0,1]$ and mean p is bounded above by the moment generating function of a Bernoulli random variable with parameter p (see also [63, 18] for an alternative proof).

5. Optical packet switches

5.1 Optical memory cells and SDL elements

We have discussed several approaches for solving the conflicts in high speed packet switches with electronic memories. The key problem of extending these approaches to optical switches is the lack of inexpensive optical random access memory. In Figure 5.1, we show a simple architecture for building a random access memory with size N. As shown in Figure 5.1, the first stage is a $1 \rightarrow N$ demultiplexer ($1 \times N$ crossbar) and the third stage is an $N \rightarrow 1$ multiplexer ($N \times 1$ crossbar). The second stage consists of N memory cells. To perform a write operation, the demultiplexer at the first stage selects the right memory cell to store the information. Similarly, to perform a read operation, the multiplexer at the third stage selects the right memory cell to read out the information.

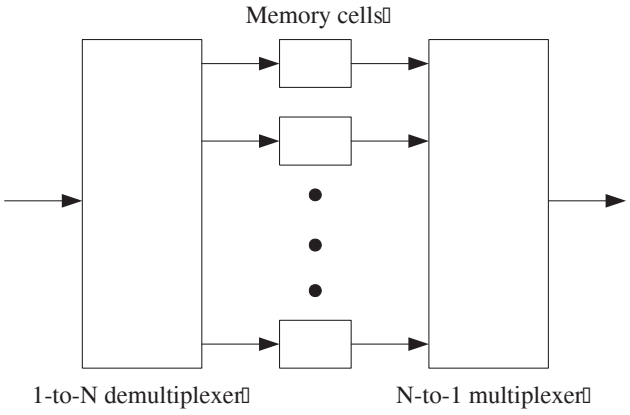


Fig. 5.1. A simple architecture for random access memory

A memory cell in electronic memory can be easily implemented by a few transistors that store electrical charges. As such, the size of electronic random access memory can be very large, e.g., 512Mbits. Thus, the cost of using electronic random access memory is usually assumed to be independent of the size of memory. Such an assumption is called the *uniform* cost assumption in the literature. However, it is much more difficult to store photons. One way to implement a memory cell for optical memory is to use a 2×2 optical crossbar switch and a fiber delay line as shown in Figure 5.2. To write the information to the memory cell, set the 2×2 crossbar switch to the "cross" state so that photons can be directed to the fiber delay line. Once the write operation is completed, the crossbar switch is then set to the "bar" state so that the photons directed into the fiber delay line keep circulating through the fiber delay line. To read out the information from the memory cell, set the crossbar switch to the "cross" state so that the photons in the fiber delay line can be directed to the output link. Unlike transistors, the cost of a 2×2 optical crossbar switch is high in today's technology. As the number of 2×2 switches needed in the simple architecture in Figure 5.1 is proportional to the number of memory cells, it is very costly to build optical random access memory by using such an architecture.

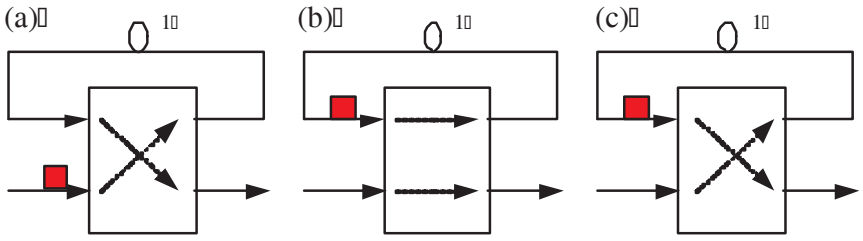


Fig. 5.2. An optical memory cell: (a) writing information (b) circulating information (c) reading information

A network element that is built by optical crossbar switches and fiber delay lines as described in Figure 5.2 is called a **Switched Delay Line (SDL) element** in this chapter. An $N \times N$ crossbar switch, as discussed in Chapter 2, is a network element that has N input links and N output links. Moreover, it is capable of realizing all the $N!$ permutations that connects the N input links to the N output links. To give a more precise definition of SDL elements, we need to provide

a formal definition of fiber delay lines. For this, we consider fixed size packets over optical links in this chapter (unless otherwise specified). Assume that time is slotted and synchronized so that a packet can be transmitted within a time slot. Since there is at most one packet within a time slot, we may use indicator variables to represent the state of a link. A link is in state 1 at time t (for some $t = 0, 1, 2, \ldots$) if there is a packet in the link at time t, and it is in state 0 at time t otherwise.

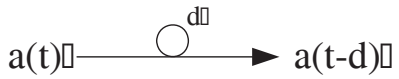


Fig. 5.3. An optical delay line with delay d

Definition 5.1.1. (Delay line) *An (optical) delay line in Figure 5.3 is a network element that has one input link and one output link. In Figure 5.3, the delay is d. Let $a(t)$ be the state of the input link. Then the state of the output link is $a(t - d)$.*

Note that at the end of the t^{th} time slot, the packets that arrive at time $t, t-1, \ldots, t-(d-1)$, are stored in the optical delay line with delay d. As such, the state of a delay line with delay d at time t can be characterized by the d-vector

$$(a(t), a(t-1), \ldots, a(t-d+1)).$$

As optical crossbar switches do not have memory, the state of an SDL element can be represented by the concatenation of the states of the delay lines in that SDL element. But such a representation in general results in a huge state space for an SDL element with a lot of delay lines. As we shall see later in this chapter, one may be able to use state aggregation for certain SDL elements to reduce the size of the state space.

Recall that a Switched Delay Line (SDL) element is a network element that is built by optical crossbar switches and fiber delay lines.

Definition 5.1.2. (Scaled SDL element) *A scaled SDL element is said to be with scaling factor m if the delay in every delay line is m times of that in the original (unscaled) SDL element.*

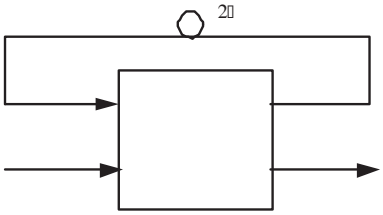


Fig. 5.4. An optical memory cell with scaling factor 2

In Figure 5.4, we show an optical memory cell with scaling factor 2. Note that the length of the delay line in Figure 5.4 is twice of that in the original optical memory cell in Figure 5.2. As such, every packet directed into the delay line takes two units of time to circulate back to the 2×2 optical crossbar switch. One of the most important properties of SDL elements is the following time interleaving property.

Proposition 5.1.3. (Time interleaving property) *A scaled SDL element with scaling factor m can be operated as time interleaving of m SDL elements.*

To understand the intuition of the time interleaving property, consider the memory cell with scaling factor 2 in Figure 5.4. To operate the scaled optical memory cell with scaling factor 2, one first partitions time into even and odd numbered time slots. For the even numbered time slots, one can set the connection patterns of the 2×2 optical crossbar switch in the scaled SDL element according to the read/write operation for one memory cell. Similarly, for the odd numbered time slots, one can set the connection patterns of the 2×2 optical crossbar switch in the scaled SDL element according to the read/write operation for another memory cell. By so doing, the scaled optical memory cell with scaling factor 2 can be operated as time interleaving two optical memory cells.

Proof. (Proof of Proposition 5.1.3) It suffices to illustrate this for the case with $m = 2$. As explained in the case for the scaled optical memory cell with scaling factor 2, we partition time into even and odd numbered time slots. To perform time interleaving of two SDL elements, we then operate these two SDL elements *alternatively* between even numbered time slots and odd numbered time slots. As such, the states of each of the two SDL elements are changed every two time slots (and remain unchanged when the SDL element is not operated).

In short, each of the time interleaved SDL elements is operated at the clock rate that is one half of that in the original SDL element. In view of this, each of the time interleaved SDL elements can then be implemented by the original SDL element by doubling the delay in each delay line and changing the state in each switch every two time slots. Clearly, each of the time interleaved SDL elements can then be implemented by a scaled SDL element with scaling factor 2.

Instead of using *two* scaled SDL element with scaling factor 2 for time interleaving of two SDL elements, now we show that we only need *one*. To see this, call the two time interleaved SDL elements *SDL element A* and *SDL element B*, and the scaled SDL element (with scaling factor 2) *SDL element C*. As described in the last paragraph, note that the states of the switches in SDL elements A and B are changed every two time slots. Thus, for the even numbered time slots, we can set the connection patterns of the optical crossbar switches in SDL element C according to the connection patterns of the optical crossbar switches in SDL element A. Similarly, for the odd numbered time slots, we can set the connection patterns of the optical crossbar switches in SDL element C according to the connection patterns of the optical crossbar switches in SDL element B. By so doing, we can operate a scaled SDL element with scaling factor 2 as time interleaving of two 2-to-1 SDL elements. ■

In the following sections, we shall discuss SDL elements that implement various types of optical memories.

5.2 Time slot interchange

5.2.1 Optical time slot interchange by serial/parallel conversion

Definition 5.2.1. (Time slot interchange) *A time slot interchange (see Figure 5.5) is a network element that has one input link and one output link. A time slot interchange is said to have frame N if for a frame of N slots, the network element is capable of interchanging the position of these N time slots according to any specific permutation of time slots. To ease our presentation, a time slot interchange with frame N is denoted by an $N \times N$ time slot interchange (TSI).*

Fig. 5.5. A 4×4 time slot interchange

A time slot interchange has the same function as a crossbar switch. The difference is that a time slot interchange rearranges packets in time and a crossbar switch rearranges packets in space. As such, a time slot interchange is known as a *time* switch, while a crossbar switch is known as a *space* switch.

As a shared memory switch in Section 2.1, a time slot interchange can be easily implemented by using random access memory. One first writes the packets in the time slots of a frame into the memory and then reads out the packets according to the desired order. In the following, we discuss how one implements a time slot interchange with optical crossbar switches and fiber delay lines.

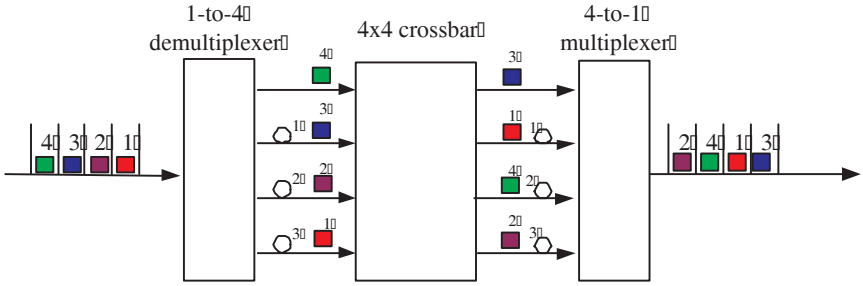


Fig. 5.6. A simple implementation of an optical time slot interchange

One easy way to implement an optical time slot interchange is to use serial/parallel conversion with optical crossbar switches. We illustrate this by an optical 4×4 time slot interchange in Figure 5.6. We first use the 1-to-4 demultiplexer and the fiber delay lines to perform serial-to-parallel conversion. During the i^{th} time slot, the 1-to-4 demultiplexer is connected to the fiber delay line with delay $4 - i$, $i = 1, 2, 3, 4$. By so doing, at the beginning of the fourth time slot, all the packets in

these four time slots appear at the four input links of the 4×4 crossbar switch. The crossbar switch then sets up the connection pattern according to the desired permutation matrix. Finally, we use the fiber delay lines and the 4-to-1 multiplexer to perform the parallel-to-serial conversion. During the $3 + i^{th}$ time slot, the output of the multiplexer is connected to the fiber delay line with delay $i-1$, $i = 1, 2, 3, 4$. At the end of the seventh time slot, the order of the four time slots are now interchanged according to the desired permutation. Note that there is delay of three time slots in doing the time slot interchange. Such delay is due to the serial-to-parallel conversion.

5.2.2 Optical Clos time slot interchange

As discussed in Section 2.4.1, one can build a larger switch fabric by using the three-stage Clos network. In this section, we demonstrate the idea of using the time interleaving property in Proposition 5.1.3 and the three-stage Clos network to form a larger time slot interchange.

To further illustrate the time interleaving property in Proposition 5.1.3, we consider the scaled 4×4 time slot interchange with scaling factor 2 in Figure 5.7. Note that the delay in each fiber delay line in Figure 5.7 is exactly 2 times of that in the corresponding fiber delay line in Figure 5.6. The demultiplexer and the multiplexer in Figure 5.7 change their connection patterns in every two time slots. For the odd numbered time slots (slots 1,3,5,7), it is a 4×4 time slot interchange. Similarly, for the even numbered time slots (slots 2,4,6,8), it is also a 4×4 time slot interchange. Thus, such an SDL element can be viewed as time interleaving of two 4×4 time slot interchanges.

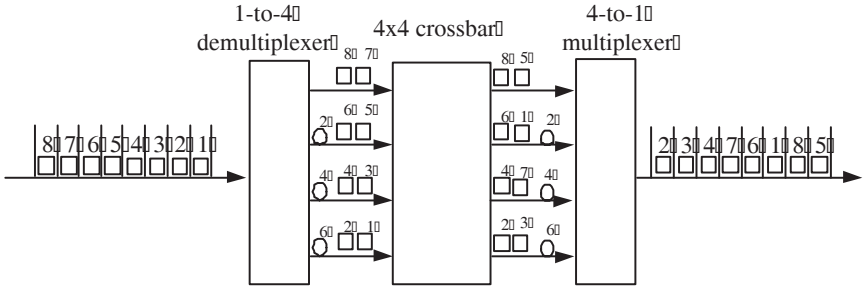


Fig. 5.7. Time interleaving of two time slot interchanges

Now consider the three-stage rearrangeable network in Section 2.4.2. As shown in Figure 5.8(a), the first stage and the third stage consist of k $m \times m$ crossbar switches, and the second stage consists of m $k \times k$ crossbar switches. Crossbar switches between the first two stages are connected by the perfect shuffle exchange, i.e., the m output links of every crossbar switch at the first stage are connected to exactly one input link of the m crossbar switches at the second stage. Similarly, crossbar switches between the last two stages are connected by the perfect shuffle exchange. It is shown in Theorem 2.4.2 that the three-stage network in Figure 5.8 realizes all the $N \times N$ sub-permutation matrices, where $N = m \times k$. Moreover, via the Birkhoff-von Neumann decomposition one can find non-conflicting paths from all inputs to all outputs as long as the connection pattern is a sub-permutation matrix.

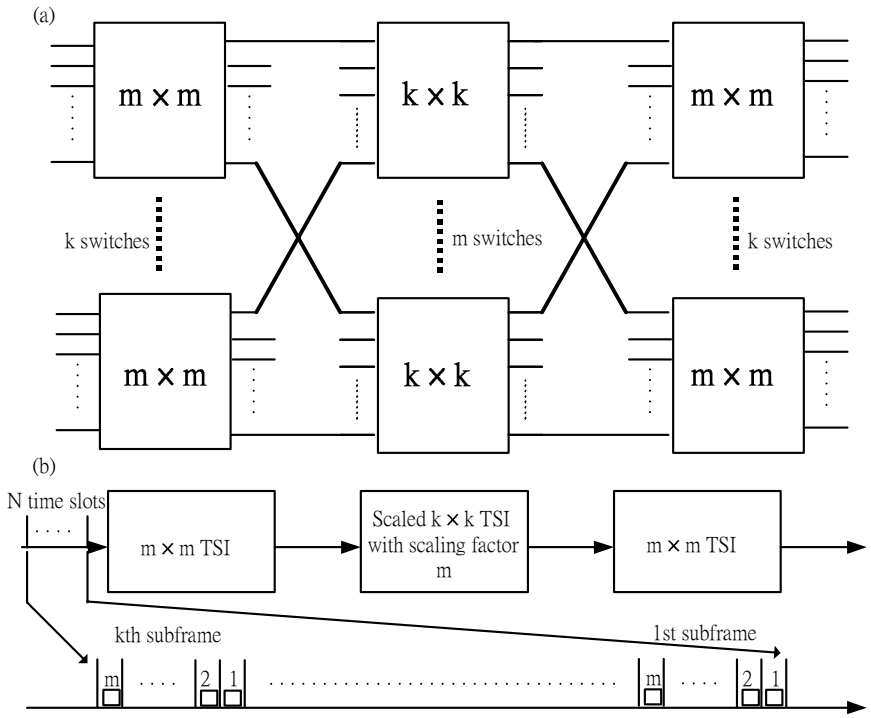


Fig. 5.8. The mapping of a three-stage rearrangeable network (shown in part (a)) and a three-stage time slot interchange (shown in part (b))

In Figure 5.8(b), we consider a three-stage time slot interchange that is a concatenation of an $m \times m$ time slot interchange, a scaled

$k \times k$ time slot interchange with scaling factor m, and an $m \times m$ time slot interchange. Via mapping such a three-stage construction to the three-stage rearrangeable network, we will show that such a construction is indeed an $N \times N$ time slot interchange with $N = m \times k$. To see this, we number the switches at the first stage in the rearrangeable network from 1 to k and do the same for the switches at the third stage (as in the proof of Theorem 2.4.2). As such, we can use the $(i, \ell)^{th}$ input, $i = 1, 2, \ldots, k$, $\ell = 1, 2, \ldots, m$, to denote the ℓ^{th} input/output of the i^{th} switch at the first and the third stages. Similarly, we can partition the N time slots in the three-stage time slot interchange into k subframes, each with m time slots. As such, we can use the $(i, \ell)^{th}$ time slot, $i = 1, 2, \ldots, k$, $\ell = 1, 2, \ldots, m$, to denote the ℓ^{th} time slot of the i^{th} subframe. By so doing, we can map the $(i, \ell)^{th}$ time slot in the three-stage time slot interchange to the $(i, \ell)^{th}$ input in the three-stage rearrangeable network. Now by the time interleaving property in Proposition 5.1.3, the scaled $k \times k$ time slot interchange with scaling factor m can be operated as time interleaving of m $k \times k$ time slot interchanges, i.e., the set of time slots $\{(i, \ell), i = 1, 2, \ldots, m\}$ from the output of the time slot interchange at the first stage are fed to the ℓ^{th} time interleaved $k \times k$ time slot interchange in the second stage. This corresponds to the perfect shuffle that is used to connected the switches between the first stage and the second stage in the three-stage rearrangeable network. Following the same argument, one can see that the concatenation of another $m \times m$ time slot interchange at the third stage corresponds to the perfect shuffle that connects the k $m \times m$ switches at the third stage of the rearrangeable network.

5.2.3 Optical Benes time slot interchange

As described in the previous section, if N is a power of 2, one can construct an $N \times N$ time slot interchange via a concatenation of a 2×2 time slot interchange, a scaled $N/2 \times N/2$ time slot interchange with scaling factor 2, and a 2×2 time slot interchange. Then one can recursively expand the scaled $N/2 \times N/2$ time slot interchange with scaling factor 2 via a concatenation of a scaled 2×2 time slot interchange with scaling factor 2, a scaled $N/4 \times N/4$ time slot interchange with scaling factor 4, and a scaled 2×2 time slot interchange with scaling factor 2. As in Section 4.2.2, eventually we can build an $N \times N$ time slot interchange by $2 \log_2 N - 1$ scaled 2×2 time slot interchanges with various scaling factors. Such a network is called the Benes time

slot interchange as it can be mapped to a Benes network. In Figure 5.9, we show an 8×8 Benes time slot interchange.

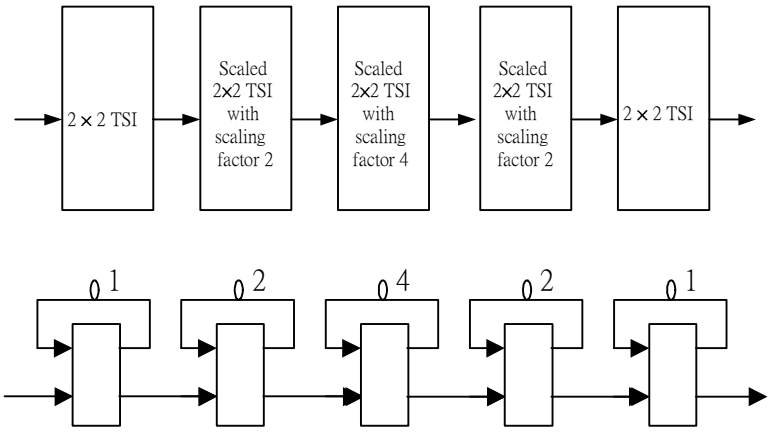


Fig. 5.9. An 8×8 Benes time slot interchange and its implementation with optical memory cells

Instead of using the serial/parallel conversion in Figure 5.6 to build a 2×2 time slot interchange, one can use a memory cell as shown in Figure 5.10. At $t = 1$, the first packet is written into the memory cell by setting the 2×2 switch to the "cross" state. At $t = 2$, the switch is set to the "bar" state. As such, the first packet keeps circulating in the memory cell, while the second packet departs from the switch. At $t = 3$, the switch is then set back to the "cross" state and the first packet departs in this time slot. By so doing, the second packet departs before the first packet and their order is reversed. To keep the same departing order, we simply set the switch to the "cross" state at $t = 2$. It is easy to see that now the first packet departs at $t = 2$ and the second packet departs at $t = 3$. Thus, the memory cell in Figure 5.10 can be used as a 2×2 time slot interchange. Note that there is one time slot delay for this time slot interchange.

At the first glance, it takes three time slots to interchange the two packets in two consecutive time slots in Figure 5.10. We note that one can do *pipelining* so that the time slot interchange in Figure 5.10 can be operated for multiple frames. In Figure 5.11, we show this for consecutive 8 time slots (4 frames). At odd numbered time slots (e.g., $t = 1, 3, 5, 7, 9$), the 2×2 switch is always set to the "cross" state so that packets arriving at odd numbered time slots are written into

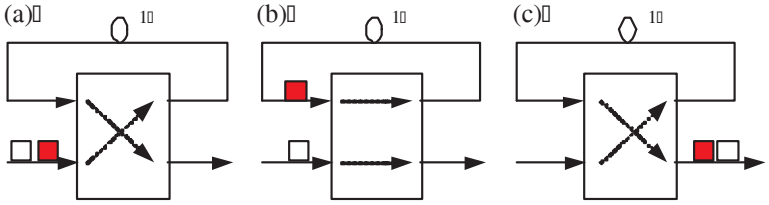


Fig. 5.10. Using a memory cell as a 2×2 time slot interchange

the memory cell. At an even numbered time slot, the switch can be set to the "bar" state to interchange the order of the packet and the packet arriving immediately before it. On the other hand, the order is preserved if the 2×2 switch is set to the "cross" state in an even numbered time slot. For instance, at $t = 2$ and $t = 6$, the switch is set to the "bar" state. As such, the order of packet 1 and packet 2 and the order of packet 5 and packet 6 are reversed. At $t = 4$ and $t = 8$, the switch is set to the "cross" state, and the order of packet 3 and packet 4 and the order of packet 7 and packet 8 are preserved. To summarize, there are four things that one needs to keep in mind for pipelining a memory cell.

(i) Odd numbered time slots are always set to the "cross" state.
(ii) The "bar" state at an even numbered time slot reverses the order of two adjacent packets, and the "cross" state at an even numbered time slot preserves the order of two adjacent packets.
(iii) Except for the first time slot, there is always a packet stored in the memory cell.
(iv) There is one slot delay for the time slot interchange.

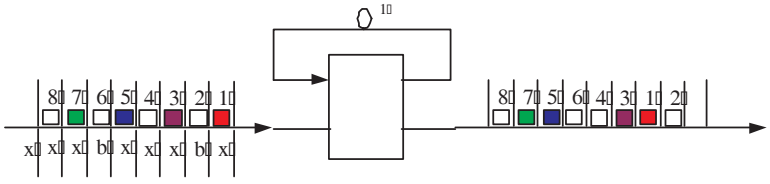


Fig. 5.11. Pipelining a memory cell

Example 5.2.2. (Connection patterns in a Benes time slot interchange) In Figure 5.12, we show on the right an SDL implemen-

tation for an 8×8 Benes time slot interchange via a concatenation of memory cells. Clearly, the memory cell at the 1^{st} stage is a 2×2 time slot interchange, the memory cell at the 2^{nd} stage is a scaled 2×2 time slot interchange with scaling factor 2, the memory cell at the 3^{rd} stage is a scaled 2×2 time slot interchange with scaling factor 4, the memory cell at the 4^{th} stage is a scaled 2×2 time slot interchange with scaling factor 2, and the memory cell at the 5^{th} stage is a 2×2 time slot interchange. As such, it can be operated as an 8×8 time slot interchange.

As in Example 2.4.6, suppose that we would like to reverse the order of 8 consecutive time slots, i.e., from the order of $1, 2, \ldots, 8$ to the order of $8, 7, \ldots, 1$. Via mapping the 8×8 time slot interchange into an 8×8 Benes network, it is equivalent to realizing the following permutation matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

As shown in Figure 2.23, a feasible realization for an 8×8 Benes network is to set all the switches at the first stage and the second stage to the "bar" state and all the switches at the third stage, the fourth stage, and the fifth stage to the "cross" state.

Now we map these connection patterns to the switches in the memory cells. As stated before, all the odd numbered time slots ($t = 1, 3, 5, 7, 9, \ldots$) are set to the "cross" state for the switch at the first stage. Since the four switches at the first stage of the Benes network are set to the "bar" state, the even numbered time slots ($t = 2, 4, 6, 8, \ldots$) are set to the "cross" state. By so doing, the packets entering the first stage from $t = 1$ to 8 are simply delayed one time slot and they enter the second stage with the same order.

Note that the second stage is time interleaving of two 2×2 time slot interchanges. Call these two time slot interchanges, TSI A and TSI B. As there is one unit delay from the first stage, the "odd" numbered time slots for TSI A are $t = 2, 6, 10, \ldots$ and the "even" numbered time

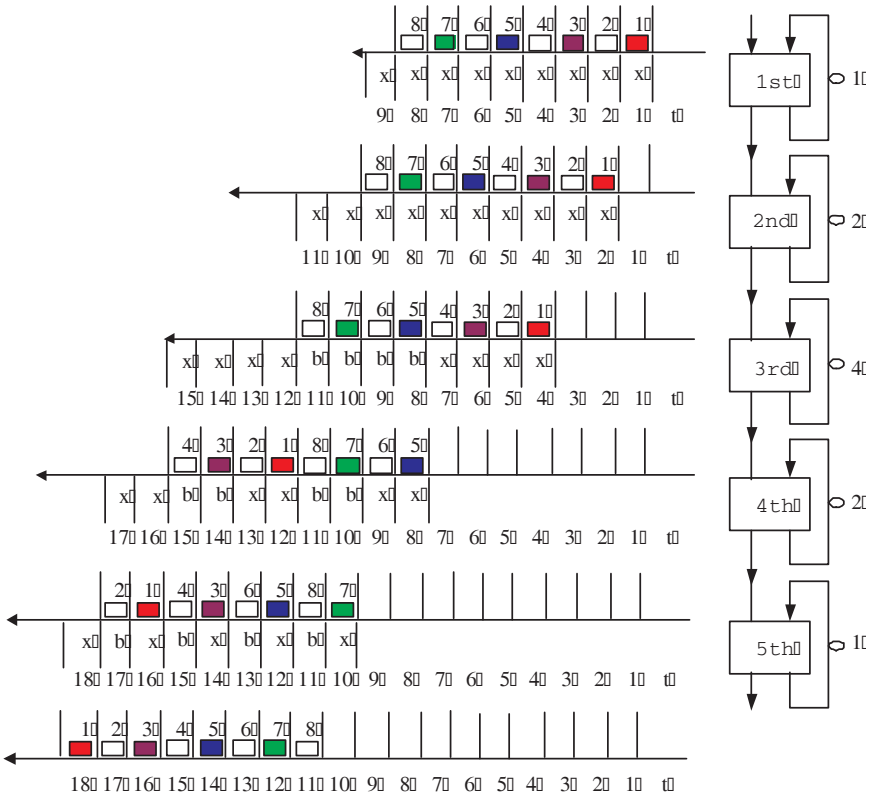


Fig. 5.12. Connection patterns in the Benes time slot interchange for Example 5.2.2

slots for TSI A are $t = 4, 8, \ldots$. Similarly, the "odd" numbered time slots for TSI B are $t = 3, 7, 11, \ldots$ and its "even" numbered time slots are $t = 5, 9, \ldots$. Since the four switches at the 2^{nd} stage of the Benes network are also set to the "bar" state, all the time slots are set to the "cross" state in the switch at the 2^{nd} memory cell. As such, all the packets are delayed by another two time slots and they depart in the same order as they arrive.

Now the delay is three time slots before entering the 3^{rd} stage. As the 3^{rd} stage is time interleaving of four 2×2 time slot interchanges. Call these four time slot interchanges, TSI C, TSI D, TSI E and TSI F. Now the "odd" numbered time slots for TSI C are $t = 4, 12, 20, \ldots$ and its "even" numbered time slots are $t = 8, 16, \ldots$. Similarly, the "odd" numbered time slots for TSI D are $t = 5, 13, 21, \ldots$ and its "even" numbered time slots are $t = 9, 17, \ldots$. The "odd" numbered time slots for TSI E are $t = 6, 14, 22, \ldots$ and its "even" numbered time slots are $t = 10, 18, \ldots$. The "odd" numbered time slots for TSI F are $t = 7, 15, 23, \ldots$ and its "even" numbered time slots are $t = 11, 19, \ldots$. Since the four switches at the 3^{rd} stage of the Benes network are set to the "cross" state, all the "even" number time slots in TSIs C,D,E,F, i.e., $t = 8, 9, 10, 11$, are set to the "bar" state and the rest of time slots are set to the "cross" state in the switch at the 3^{rd} memory cell. After being delayed by 7 time slots, now the 8 packets entering the 4^{th} stage in the order of 5,6,7,8,1,2,3,4.

As described at the second stage, the 4^{th} stage is time interleaving of two 2×2 time slot interchanges. Call these two time slot interchanges, TSI G and TSI H. As there is 7 unit delay from the first stage, the "odd" numbered time slots for TSI G are $t = 8, 12, 16, \ldots$ and the "even" numbered time slots for TSI G are $t = 10, 14, \ldots$. Similarly, the "odd" numbered time slots for TSI H are $t = 9, 13, 17, \ldots$ and the "even" numbered time slots for TSI H are $t = 11, 15, \ldots$. Since the four switches at the 4^{th} stage of the Benes network are set to the "cross" state, all the "even" number time slots in TSIs G and H, i.e., $t = 10, 11, 14, 15$, are set to the "bar" state and the rest of time slots are set to the "cross" state in the switch at the 4^{th} memory cell. After being delayed by 9 time slots, now the 8 packets entering the 5^{th} stage in the order of 7,8,5,6,3,4,1,2.

As the first stage, the 5^{th} stage is a 2×2 time slot interchange. With the 9 time slot delay in mind, we know that the "odd" numbered time slots are $t = 10, 12, 14, 16, 18, \ldots$ and the "even" numbered

time slots are $t = 11, 13, 15, 17, \ldots$. Since the four switches at the 5^{th} stage of the Benes network are set to the "cross" state, all the "even" numbered time slots in this time slot interchange are set to the "bar" state and the rest of time slots are set to the "cross" state. It is ready to check that the packets now depart from the 5^{th} stage in the order of $8, 7, 6, 5, 4, 3, 2, 1$ after 10 time slot delay. Note that the total delay 10 is also the sum of the delay in all the fiber delay lines in this 5 stage construction.

To summarize, one can build an $N \times N$ SDL time slot interchange with $2\log_2 N - 1$ 2×2 optical crossbars. The delay for such a time slot interchange is $\frac{3}{2}N - 2$, which is exactly the sum of the delay in its fiber delay lines.

5.3 2-to-1 buffered multiplexers with switched delay lines

Our main objective of this section is to show how one uses switched delay lines (SDL) to construct a 2-to-1 buffered multiplexer. The main idea is to redistribute packets through delay lines with different delays so that packets competing for the same output link can be resolved over time and space. In Figure 5.13, we illustrate this idea via a simple example. There are two input links that are multiplexed into an output link. Suppose that two packets arrive at both input links. This causes a conflict for the output link. To resolve such a conflict, we can first put these two packets through a 2×2 switch and distribute one packet to a zero delay transmission line (the upper link) and the other to a fiber delay line with one unit of delay (the lower link). By so doing, these two packets can be multiplexed into the same link sequentially. In fact, as we will show later, the construction in Figure 5.13 is indeed a 2-to-1 multiplexer with one unit of buffer.

In the following, we provide a formal definition of a 2-to-1 multiplexer with buffer B.

Definition 5.3.1. (Multiplexer) *A 2-to-1 multiplexer with buffer B has two input links and two output links (see Figure 5.14). One output link is for departing packets and the other is for lost packets. As shown in Figure 5.14, let $a_1(t)$ (resp. $a_0(t)$) be the state of the dotted (resp.*

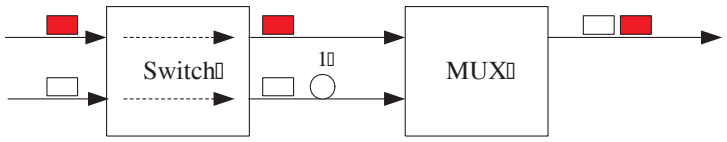


Fig. 5.13. An illustrating example of using switched delay lines for conflict resolution

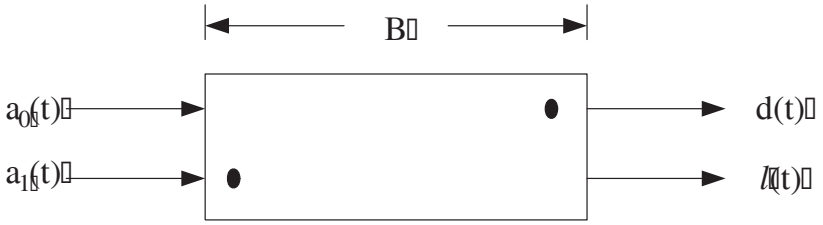


Fig. 5.14. A 2-to-1 multiplexer with buffer B

undotted) input link, $d(t)$ (resp. $\ell(t)$) be state of the output link for departing (resp. lost) packets, and $q(t)$ be the number of packets queued at the multiplexer at time t (at the end of the t^{th} time slot). Then the 2-to-1 multiplexer with buffer B satisfies the following four properties:

(P1) Flow conservation: arriving packets from the two input links are either stored in the buffer or transmitted through the two output links, i.e.,

$$q(t) = q(t-1) + a_0(t) + a_1(t) - d(t) - \ell(t). \tag{5.1}$$

(P2) Non-idling: there is always a departing packet if there are packets in the buffer or there are arriving packets, i.e.,

$$d(t) = \begin{cases} 0 & \text{if } q(t-1) = a_0(t) = a_1(t) = 0 \\ 1 & \text{otherwise} \end{cases}. \tag{5.2}$$

(P3) Maximum buffer usage: arriving packets are lost only when buffer is full, i.e.,

$$\ell(t) = \begin{cases} 1 & \text{if } q(t-1) = B \text{ and } a_0(t) = a_1(t) = 1 \\ 0 & \text{otherwise} \end{cases}. \tag{5.3}$$

(P4) FIFO with prioritized inputs: packets depart in the first in first out (FIFO) order. Moreover, if each input has an arriving packet, the packet from the dotted input is put in the multiplexer first. In other word, the virtual delay for the dotted input $a_1(t)$ is $q(t-1)$, the number of packets that is stored in the multiplexer at $t-1$. The

virtual delay for the undotted input $a_0(t)$ is then $q(t-1)+a_1(t)$ (if that packet is not lost).

Note from (P1) and (P2) that

$$q(t) = (q(t-1) + a_0(t) + a_1(t) - 1)^+ - \ell(t),$$

where $x^+ = \max(0, x)$. In conjunction with (P3), we have the following Lindley equation

$$q(t) = \min[(q(t-1) + a_0(t) + a_1(t) - 1)^+, B]. \tag{5.4}$$

Thus, a 2-to-1 multiplexer with buffer B is simply a discrete-time FIFO queue with buffer B and two inputs in the queueing context.

As such, the state of a 2-to-1 multiplexer with buffer B is simply $q(t)$, the number of packets stored in the multiplexer at the end of the t^{th} time slot. This is crucial in simplifying the complexity of our design (and later analysis) as a large number of binary states in optical delay lines now can be summarized by a single number.

In addition to the state aggregation property described above, we also need the time interleaving property in Proposition 5.1.3. For this, we define the scaled multiplexers below.

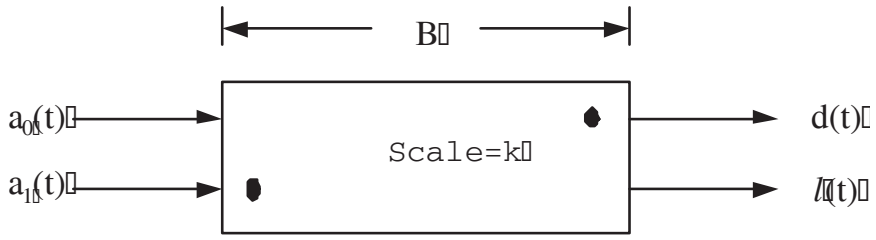


Fig. 5.15. A scaled 2-to-1 multiplexer with buffer B and scaling factor k

Definition 5.3.2. (SDL multiplexer) *A 2-to-1 multiplexer is called a 2-to-1* SDL *multiplexer if the multiplexer is built with crossbar switches and delay lines (in Definition 5.1.1). A 2-to-1* SDL *multiplexer is with scaling factor k (see Figure 5.15) if the delay in every delay line is k times of that in the original (unscaled) 2-to-1* SDL *multiplexer.*

As described in Proposition 5.1.3, a scaled 2-to-1 SDL multiplexer is with scaling factor k can be operated as time interleaving of k 2-to-1 SDL multiplexers.

5.3.1 Prioritized concentrator

One key element for building a 2-to-1 SDL multiplexer is the prioritized concentrator defined below.

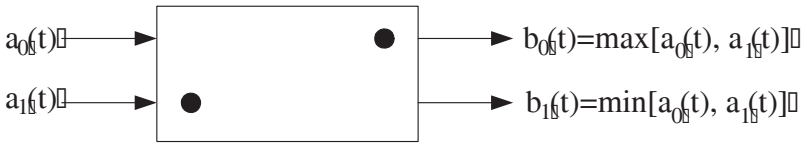


Fig. 5.16. A prioritized concentrator

Definition 5.3.3. (Concentrator) *A prioritized concentrator in Figure 5.16 is a* 2×2 *switch with its connection patterns depending on its two inputs. Let* $a_1(t)$ *(resp.* $a_0(t)$*) be the state of the dotted (resp. undotted) input, and* $b_0(t)$ *(resp.* $b_1(t)$*) be the state of the dotted (resp. undotted) output. The switch is set to the cross state at time* t *if* $a_1(t) = 1$*, i.e.,* there is a packet arrival at the dotted input at time t. *Otherwise, the switch is set to the bar state.*

From the operating rule of a prioritized concentrator, if there is a packet at the dotted input at time t, this packet is transmitted to the dotted output. If there is another packet at the undotted input, then the packet at the undotted input is transmitted to the undotted output. When there is no packet at the dotted input and there is a packet at the undotted input, the packet at the undotted input is transmitted to the dotted output. Such an operation rule ensures that

$$b_0(t) = \max[a_0(t), a_1(t)], \tag{5.5}$$

and

$$b_1(t) = \min[a_0(t), a_1(t)]. \tag{5.6}$$

Equations (5.5) and (5.6) are the governing equations for a prioritized concentrator.

By letting $d(t) = b_0(t)$ and $\ell(t) = b_1(t)$, it is easy to see that a prioritized concentrator in Definition 5.3.3 is indeed a 2-to-1 multiplexer with buffer 0. In this case, it is stateless as $q(t) = 0$ for all t.

We note that a prioritized concentrator is called a track changer in J. T. Tsai [158] for 2-to-1 multiplexers. As its main objective is to perform traffic concentration (this will become clear in the general case of

n-to-1 multiplexers), we prefer using the name prioritized concentrator as the name reflects its functional objective.

5.3.2 Recursive construction

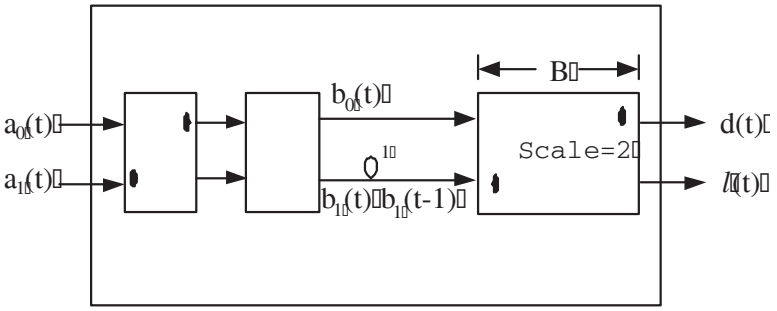


Fig. 5.17. Recursive construction of a 2-to-1 multiplexer with buffer $2B + 1$

In this section, we show how one constructs a 2-to-1 multiplexer with a large buffer by time interleaving two 2-to-1 multiplexers with small buffers. In Figure 5.17, we consider a network element with two inputs and two outputs. It is a concatenation of a prioritized concentrator, a 2×2 switch and a scaled 2-to-1 multiplexer with buffer B and scaling factor 2. The two outputs of the 2×2 switch are connected to two delay lines with delay 0 and 1, respectively. We will show in Theorem 5.3.4 that such a construction can be operated as a 2-to-1 multiplexer with buffer $2B + 1$.

As described in Proposition 5.1.3, a scaled 2-to-1 multiplexer with buffer B and scaling factor 2 can be operated as time interleaving of two 2-to-1 multiplexers with buffer B (see Figure 5.18). To be specific, we partition time into *even* and *odd* numbered time slots. For the even numbered time slots, the two outputs of the delay lines (after the 2×2 switch) and the two outputs of the network element are connected to the two inputs and the two outputs of one multiplexer, respectively. On the other hand, for the odd numbered time slots, the two outputs of the delay lines and the two outputs of the network element are connected to the two inputs and the two outputs of the other multiplexer. The state of each multiplexer, i.e., the number of packets stored in the multiplexer, remains unchanged when the multiplexer is not connected. Thus, every multiplexer changes its state every two time slots.

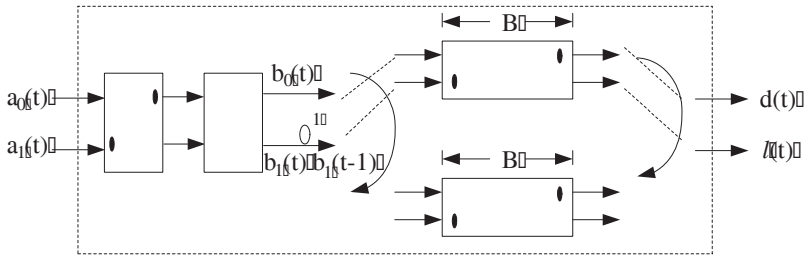


Fig. 5.18. A 2-to-1 multiplexer with buffer $2B + 1$

For the construction in Figure 5.18, we needs to specify the connection patterns of the 2×2 switch. Define the total number of packets stored in the network element as the sum of the number of packets stored in each multiplexer and the number of packet stored in the delay line with delay 1. **The 2×2 switch is set to the cross state if there is an odd number of packets stored in the network element (at the end of the previous time slot), and is set to the bar state otherwise.**

Theorem 5.3.4. *If the network element in Figure 5.17 is started from an empty system, then it is a 2-to-1 multiplexer with buffer* $2B + 1$.

The intuition of Theorem 5.3.4 is the same as that for the construction of the multiplexer in Section 2.7.2. In Figure 5.18, there are two time interleaved multiplexers with buffer B. This is equivalent to have two parallel FIFO queues with buffer B. To make these two parallel queues behave as if it is a single FIFO queue, we need to operate these two queues under the join-the-shortest-queue policy and the serve-the-longest-queue policy (see Figure 2.42). As these two queues are served periodically with period 2, the join-the-shortest-queue policy and the serve-the-longest-queue policy are simply the round-robin policy. To implement the round-robin policy, these two parallel queues need to be kept in the most balanced state, i.e., the difference between these two queues is at most one. The connection patterns of the 2×2 switch in Figure 5.18 specified by the total number of packets in the system are exactly doing that. The formal proof of Theorem 5.3.4 is given in Section 5.3.3.

Example 5.3.5. Since a prioritized concentrator is a 2-to-1 multiplexer with buffer 0, it follows from Theorem 5.3.4 that the network

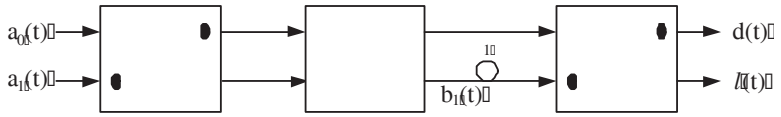


Fig. 5.19. A 2-to-1 multiplexer with buffer 1

element in Figure 5.19 is a 2-to-1 multiplexer with buffer 1. At time t, the 2×2 switch is set to the cross state if there is a packet stored in the network element (i.e., $b_1(t-1) = 1$) and is set to the bar state if the network element is empty (i.e., $b_1(t-1) = 0$).

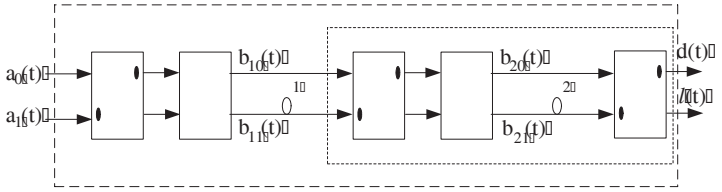


Fig. 5.20. A 2-to-1 multiplexer with $B = 3$.

Example 5.3.6. In Figure 5.20, we illustrate how one applies Theorem 5.3.4 to construct a 2-to-1 multiplexer with buffer 3 from the 2-to-1 multiplexer with buffer 1 in Figure 5.19. Note that the inner block in Figure 5.20 is exactly the same as the 2-to-1 multiplexer in Figure 5.19 except that the delay in the delay line is doubled from 1 to 2. As such, it is a scaled 2-to-1 SDL multiplexer with buffer 1 and scaling factor 2. As a direct consequence of Theorem 5.3.4, the network element in Figure 5.20 is indeed a 2-to-1 multiplexer with buffer 3. As shown in Figure 5.19, let $b_{11}(t)$ (resp. $b_{21}(t)$) be the state of the lower output of the first (resp. second) 2×2 switch. Also, note that the state of the network element at time $t-1$ is

$$(b_{11}(t-1), b_{21}(t-2), b_{21}(t-1)).$$

Recall that the state of the switches at time t are based on the state of the network element at time $t-1$. Note that the total number of packets stored in the network element at time $t-1$ is $b_{11}(t-1)+b_{21}(t-1)+b_{21}(t-2)$. Thus, we have from the operation rule in Theorem 5.3.4

that the first switch is set to the cross state at time t if $b_{11}(t-1) + b_{21}(t-1) + b_{21}(t-2)$ is an odd number and the bar state otherwise. On the other hand, we have from the operation rule in Example 5.3.5 that the second switch is set to the cross state at time t if $b_{21}(t-2) = 1$ and the bar state otherwise.

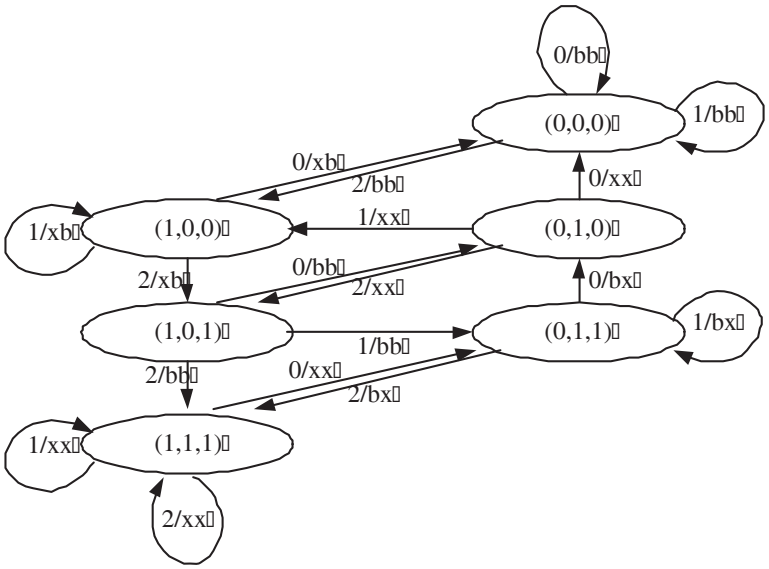


Fig. 5.21. The state transition diagram for $B = 3$.

In Figure 5.21, we draw the state transition diagram (from $(b_{11}(t-1), b_{21}(t-2), b_{21}(t-1))$ to $(b_{11}(t), b_{21}(t-1), b_{21}(t))$) for this 2-to-1 multiplexer with buffer 3. We use the notation *(number of arrivals)/(state of the first switch)(state of the second switch)* for each transition. For instance, the notation $2/xb$ denotes two arrivals, the cross state of the first switch and the bar state of the second switch.

To gain more intuition on this multiplexer, we consider the case that $a_0(t) = a_1(t) = 1$ for $t = 1, 2, 3, 4$ and $a_0(t) = a_1(t) = 0$ for $t = 5, 6, 7$. As the multiplexer is started from an empty system, the state is at $(0, 0, 0)$ for $t = 0$. Thus, at $t = 1$, both 2×2 switches are set to the bar state. As there are two arrivals at $t = 1$, the arrival from $a_1(1)$ departs at $t = 1$ and the arrival from $a_0(1)$ is routed to the delay line with delay 1 after the first switch. This implies that $b_{11}(1) = 1$. At the end of the 1^{st} time slot, the state is at $(1, 0, 0)$.

At $t = 2$, the first 2×2 switch is set to the cross state and the second 2×2 switch is set to the bar state. The packet that arrives from $a_0(1)$ and is stored in the delay line with delay 1 departs at $t = 2$. As there are two arrivals at $t = 2$, the arrival from $a_1(2)$ is routed to the delay line with delay 1 after the first switch and the arrival from $a_0(2)$ is routed to the delay line with delay 2 after the second switch. This implies that $b_{11}(2) = 1$ and $b_{21}(2) = 1$. At the end of the 2^{nd} time slot, the state is at $(1, 0, 1)$.

At $t = 3$, both 2×2 switches are set to the bar state. The packet that arrives from $a_1(2)$ and is stored in the delay line with delay 1 departs at $t = 3$. As there are two arrivals at $t = 3$, the arrival from $a_1(3)$ is routed to the delay line with delay 2 after the second switch and the arrival from $a_0(3)$ is routed to the delay line with delay 1 after the first switch. This implies that $b_{11}(3) = 1$ and $b_{21}(3) = 1$. At the end of the 3^{nd} time slot, the state is at $(1, 1, 1)$.

At $t = 4$, both 2×2 switches are set to the cross state. The packet that arrives from $a_0(2)$ and is stored in the delay line with delay 2 departs at $t = 4$. The packet that arrives from $a_0(3)$ and is stored at the delay line with delay 1 after the first switch is routed to the delay line with delay 2 after the second switch. This implies that $b_{21}(4) = 1$. As there are two arrivals at $t = 4$, the arrival from $a_1(4)$ is routed to the delay line with delay 1 after the first switch and the arrival from $a_0(4)$ is routed to the output link for loss packets (i.e., $\ell(4) = 1$). This implies that $b_{11}(4) = 1$. At the end of the 4^{th} time slot, the state is at $(1, 1, 1)$.

At $t = 5$, both 2×2 switches are set to the cross state. The packet that arrives from $a_1(3)$ and is stored in the delay line with delay 2 departs at $t = 5$. The packet that arrives from $a_1(4)$ and is stored at the delay line with delay 1 after the first switch is routed to the delay line with delay 2 after the second switch. This implies that $b_{21}(5) = 1$. As there are no arrivals at $t = 5$, we have that $b_{11}(5) = 0$. At the end of the 5^{th} time slot, the state is at $(0, 1, 1)$.

At $t = 6$, the first 2×2 switch is set to the bar state and the second 2×2 switch is set to the cross state. The packet that arrives from $a_0(3)$ and is stored in the delay line with delay 2 departs at $t = 6$. As there are no arrivals at $t = 6$, we have that $b_{11}(6) = 0$ and $b_{21}(6) = 0$. At the end of the 6^{th} time slot, the state is at $(0, 1, 0)$.

At $t = 7$, both 2×2 switches are set to the cross state. The packet that arrives from $a_1(4)$ and is stored in the delay line with delay 2

departs at $t = 7$. As there are no arrivals at $t = 6$, we have that $b_{11}(7) = 0$ and $b_{21}(7) = 0$. At the end of the 7^{th} time slot, the state is back to $(0, 0, 0)$.

Note from this state transition diagram that $(0, 0, 1)$ and $(1, 1, 0)$ are not reachable from the empty state $(0, 0, 0)$. Moreover, all these states satisfies the following inequality

$$b_{21}(t) \le b_{11}(t) + b_{21}(t-1) \le b_{21}(t) + 1.$$

Such an inequality will play an important role in the proof of Theorem 5.3.4.

Now one can use Theorem 5.3.4 and the time interleaving property in Proposition 5.1.3 to recursively construct a 2-to-1 multiplexer with buffer $2^k - 1$. The m^{th} switch, $m = 1, 2, \ldots, k$, is set to the cross state at time t if $\sum_{i=m}^{k} \sum_{j=1}^{2^{i-m}} b_{i1}(t - 2^{m-1}j)$ is an odd number and the bar state otherwise, where $b_{m1}(t)$ is the state of the lower output of the m^{th} switch. One can further combine the operation of the prioritized concentrator and the 2×2 switch at each stage by a 2×2 switch (see Figure 5.22). The combined switch is to set to the bar state if both the prioritized concentrator and the original 2×2 switch are set to the same state, and it is set to the cross state otherwise. For such a multiplexer, all its switch patterns are completely determined by the states of the multiplexer.

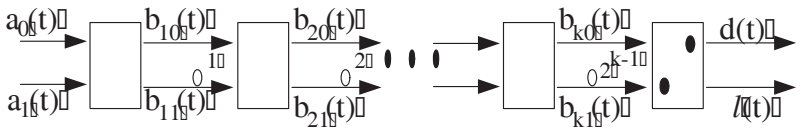


Fig. 5.22. A self-routing 2-to-1 multiplexer with $B = 2^k - 1$.

We note that the 2-to-1 multiplexer in Figure 5.22 can be a self-routing multiplexer. Since the delay of a 2-to-1 buffered multiplexer is governed by the Lindley equation in (5.4), the delay of a packet is in fact known upon its arrival (as stated in (P4)). Moreover, any packet, leaving from either the departure link or the loss link, experiences a delay between 0 and $2^k - 1$. Note that there is a unique path through the 2-to-1 multiplexer in Figure 5.22 for every delay d with $0 \le d \le 2^k - 1$,

and this path corresponds to the binary representation of d. As such, the binary representation of the delay of a packet can be used for self-routing a packet through the multiplexer. To be precise, suppose an arriving packet is with delay $d = \sum_{j=1}^{k} r_j 2^{j-1}$, where r_j is either 0 or 1. If r_j is 0, then this packet is routed to the upper output link of the j^{th} 2×2 switch. On the other hand, it is routed to the lower output link of of the j^{th} 2×2 switch if r_j is 1.

5.3.3 Inductive proof of Theorem 5.3.4

In this section, we prove Theorem 5.3.4.

As shown in Figure 5.18, let $a_1(t)$ (resp. $a_0(t)$) be the state of the dotted (resp. undotted) input of the prioritized concentrator at time t, $b_1(t)$ (resp. $b_0(t)$) be the state of the lower (resp. upper) output of the 2×2 switch at time t, and $d(t)$ (resp. $\ell(t)$) be the state of the link for departing (resp. lost) packets at time t. Also, let $q_1(t-1)$ be the number of packet stored in the 2-to-1 multiplexer at time $t-1$ that is going to be *connected* to the 2×2 switch at time t and $q_2(t-1)$ be the number of packet stored in the other 2-to-1 multiplexer at time $t-1$. As there is one unit delay line after the 2×2 switch in Figure 5.18, the state of the network element at time $t-1$ is the 3-vector $(b_1(t-1), q_1(t-1), q_2(t-1))$.

Note that $q_1(\cdot)$ represents the multiplexer that will be connected next time slot and $q_2(\cdot)$ represents the multiplexer that will stay unchanged next time slot. This state representation is different from the representation that partitions time into even numbered time slots and odd numbered time slots. The reason of using this state representation is that it is easier to write down the governing equations as described below.

Now we write down the governing equations for the network element. As the 2×2 switch is connected to the two multiplexers alternatively, we have from (5.1)-(5.3) that

$$q_2(t) = q_1(t-1) + b_1(t-1) + b_0(t) - d(t) - \ell(t), \tag{5.7}$$

$$q_1(t) = q_2(t-1), \tag{5.8}$$

$$d(t) = \begin{cases} 0 & \text{if } q_1(t-1) = b_1(t-1) = b_0(t) = 0 \\ 1 & \text{otherwise} \end{cases}, \tag{5.9}$$

$$\ell(t) = \begin{cases} 1 & \text{if } q_1(t-1) = B \text{ and } b_1(t-1) = b_0(t) = 1 \\ 0 & \text{otherwise} \end{cases}.$$

(5.10)

Let

$$q(t-1) = q_1(t-1) + q_2(t-1) + b_1(t-1) \tag{5.11}$$

be the total number of packets stored in the network element at the end of the $t-1^{th}$ time slot. According to the operation rule of the 2×2 switch, the switch is set to the cross state if $q(t-1)$ is an odd number and is set to the bar state otherwise. From the operation rule of a prioritized concentrator in Definition 5.3.3, the state of the upper input of the 2×2 switch is $\max[a_0(t), a_1(t)]$ and the state of the lower input of the 2×2 switch is $\min[a_0(t), a_1(t)]$. Thus,

$$b_1(t) = \begin{cases} \max[a_0(t), a_1(t)] & \text{if } q(t-1) \text{ is odd} \\ \min[a_0(t), a_1(t)] & \text{otherwise} \end{cases} . \tag{5.12}$$

Similarly,

$$b_0(t) = \begin{cases} \min[a_0(t), a_1(t)] & \text{if } q(t-1) \text{ is odd} \\ \max[a_0(t), a_1(t)] & \text{otherwise} \end{cases} . \tag{5.13}$$

Now we show that the network element is a 2-to-1 multiplexer with buffer $2B+1$ by verifying the four properties in Definition 5.3.1. Since arriving packets from the two input links are either stored in the network element or transmitted through the two output links, flow conservation of the network element is obviously satisfied. We will verify the other three properties by induction on t with the following additional induction hypothesis.

(P5) If we start from an empty system, i.e., $b_1(0) = q_1(0) = q_2(0) = 0$, then

$$q_2(t) \le b_1(t) + q_1(t) \le q_2(t) + 1, \quad \forall\, t. \tag{5.14}$$

As illustrated in Example 5.3.6, the induction hypothesis in (5.14) says that only a certain set of states are reachable from an empty state.

In view of the induction hypothesis in (5.14), there are four possible cases as described below.

Case 1. $(b_1(t-1), q_1(t-1), q_2(t-1)) = (0,0,0)$:

In this case, we have from (5.7)-(5.10) that

$$d(t) = b_0(t), \tag{5.15}$$

$$\ell(t) = 0, \tag{5.16}$$

$$q_2(t) = 0, \tag{5.17}$$

$$q_1(t) = 0. \tag{5.18}$$

In this case, $q(t-1) = 0$. It then follows from (5.12) and (5.13) that

$$b_0(t) = \max[a_0(t), a_1(t)], \tag{5.19}$$
$$b_1(t) = \min[a_0(t), a_1(t)], \tag{5.20}$$

Thus, we have from (5.17), (5.18) and (5.20) that the next state

$$(b_1(t), q_1(t), q_2(t)) = (\min[a_0(t), a_1(t)], 0, 0).$$

Since $0 \le \min[a_0(t), a_1(t)] \le 1$, the induction hypothesis in (5.14) is satisfied at time t.

Since $q(t-1) = 0$, there should be no packet loss at time t for a 2-to-1 multiplexer with buffer $2B+1$. Equation (5.16) verifies this. Also, there should a packet departure at time t if there is at least one packet arrival at time t. This is shown by (5.15) and (5.19).

As $q(t-1) = 0$, the 2×2 switch is set to the bar state. It is easy to see that the virtual delay for $a_1(t)$ is 0 and the virtual delay for $a_0(t)$ is $a_1(t)$ so that the FIFO order is maintained.

Case 2. $(b_1(t-1), q_1(t-1), q_2(t-1)) = (1, q-1, q)$ or $(b_1(t-1), q_1(t-1), q_2(t-1)) = (0, q, q)$ for some $0 < q \le B$:

In this case, we have from (5.7)-(5.10) that

$$d(t) = 1, \tag{5.21}$$
$$\ell(t) = 0, \tag{5.22}$$
$$q_2(t) = q + b_0(t) - 1, \tag{5.23}$$
$$q_1(t) = q. \tag{5.24}$$

In this case, $q(t-1) = 2q$ and the 2×2 switch is set to the bar state. It then follows from (5.12) and (5.13) that

$$b_0(t) = \max[a_0(t), a_1(t)], \tag{5.25}$$
$$b_1(t) = \min[a_0(t), a_1(t)], \tag{5.26}$$

Thus, we have from (5.23), (5.24), (5.25) and (5.26) that the next state

$$(b_1(t), q_1(t), q_2(t)) = (\min[a_0(t), a_1(t)], q, q + \max[a_0(t), a_1(t)] - 1).$$

Since

$$\max[a_0(t), a_1(t)] - 1 \le \min[a_0(t), a_1(t)] \le \max[a_0(t), a_1(t)],$$

the induction hypothesis in (5.14) is satisfied at time t.

Since $0 < q(t-1) = 2q < 2B+1$, there should a packet departure at time t and there should be no packet loss at time t for a 2-to-1 multiplexer with buffer $2B+1$. Equations (5.21) and (5.22) verify these for this case.

Now we show that the virtual delay for $a_1(t)$ is $q(t-1)$. Since the 2×2 switch is set to the bar state, $a_1(t)$ is routed to the multiplexer with buffer B that is going to be connected at time t. As this multiplexer with buffer B is operated under the FIFO policy, the number of packets that should depart before $a_1(t)$ is $q_1(t-1)+b_1(t-1) = q$. Note that this multiplexer with buffer B is connected to the outputs every two time slots. Thus, the virtual delay of $a_1(t)$ is $2q$, which is exactly $q(t-1)$. On the other hand, if $a_1(t) = 1$, then $a_0(t)$ is routed to the multiplexer that is going to be connected at time $t+1$. As $q_2(t-1) = q$, the virtual delay for $a_0(t) = 2q+1$. Thus, the FIFO order is maintained.

Case 3. $(b_1(t-1), q_1(t-1), q_2(t-1)) = (0, q+1, q)$ or $(b_1(t-1), q_1(t-1), q_2(t-1)) = (1, q, q)$ for some $0 < q < B$:

In this case, we have from (5.7)-(5.10) that

$$d(t) = 1, \tag{5.27}$$

$$\ell(t) = 0, \tag{5.28}$$

$$q_2(t) = q + b_0(t), \tag{5.29}$$

$$q_1(t) = q. \tag{5.30}$$

In this case, $q(t-1) = 2q+1$ and the 2×2 switch is set to the cross state. It then follows from (5.12) and (5.13) that

$$b_0(t) = \min[a_0(t), a_1(t)], \tag{5.31}$$

$$b_1(t) = \max[a_0(t), a_1(t)], \tag{5.32}$$

Thus, we have from (5.29), (5.30), (5.31) and (5.32) that the next state

$$(b_1(t), q_1(t), q_2(t)) = (\max[a_0(t), a_1(t)], q, q + \min[a_0(t), a_1(t)]).$$

Since

$$\min[a_0(t), a_1(t)] \le \max[a_0(t), a_1(t)] \le \min[a_0(t), a_1(t)] + 1,$$

the induction hypothesis in (5.14) is satisfied at time t.

Since $0 < q(t-1) = 2q+1 < 2B+1$, there should a packet departure at time t and there should be no packet loss at time t for a 2-to-1 multiplexer with buffer $2B+1$. Equations (5.27) and (5.28) verify these.

Now we show that the virtual delay for $a_1(t)$ is $q(t-1)$. Since the 2×2 switch is set to the cross state, $a_1(t)$ is routed to the multiplexer with buffer B that is going to be connected at time $t+1$. As the number of packets in this multiplexer is $q_2(t-1) = q$, the virtual delay of $a_1(t)$ is $2q+1$, which is exactly $q(t-1)$. On the other hand, if $a_1(t) = 1$, then $a_0(t)$ is routed to the multiplexer that is going to be connected at time t. As $q_1(t-1) + b_1(t-1) = q+1$, the virtual delay for $a_0(t) = 2q+2$. Thus, the FIFO order is maintained.

Case 4. $(b_1(t-1), q_1(t-1), q_2(t-1)) = (1, B, B)$:

In this case, we have from (5.7)-(5.10) that

$$d(t) = 1, \tag{5.33}$$

$$\ell(t) = b_0(t), \tag{5.34}$$

$$q_2(t) = B, \tag{5.35}$$

$$q_1(t) = B. \tag{5.36}$$

In this case, $q(t-1) = 2B+1$ and the 2×2 switch is set to the cross state. Thus, (5.31) and (5.32) still hold. Thus, we have from (5.35), (5.36), and (5.32) that the next state

$$(b_1(t), q_1(t), q_2(t)) = (\max[a_0(t), a_1(t)], B, B).$$

Since $0 \le \max[a_0(t), a_1(t)] \le 1$, the induction hypothesis in (5.14) is satisfied at time t.

Since $q(t-1) = 2B+1$, there should a packet departure at time t. This is shown in (5.33). On the other hand, since the buffer is full, there should be a packet loss at time t if two packets arrive at time t. Equations (5.34) and (5.31) verify this.

Verification of the virtual delay is the same as that in Case 3.

5.4 N-to-1 buffered multiplexers with switched delay lines

In this section, we extend the results for 2-to-1 multiplexers to N-to-1 multiplexers. In Section 5.3, we construct network elements with 2×2 switches and optical delay lines that emulate exact 2-to-1 multiplexers for both the departure process and the loss process. Exact emulation of N-to-1 multiplexers is much more difficult for $N > 2$. Instead, we only construct network elements with $N \times N$ switches and optical delay lines that generate the same departure processes as those from

N-to-1 multiplexers. Packet losses at our N-to-1 multiplexers might be delayed. Such a construction is called a delayed-loss multiplexer.

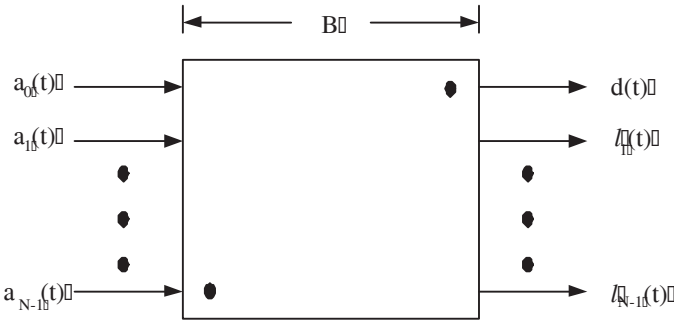


Fig. 5.23. An N-to-1 multiplexer with buffer B

Definition 5.4.1. (Multiplexer) *An N-to-1 multiplexer with buffer B (see Figure 5.23) is a network element with N input links and N output links. We call the first output link of this multiplexer the departure port and the rest of the output links the loss ports. As shown in Figure 5.23, let $a_i(t)$, $i = 0, 1, \ldots, N-1$, be the state of the N input links, $d(t)$ be state of the output link for the departure port, $\ell_i(t)$, $i = 1, 2, \ldots, N-1$, be the state of the i^{th} loss port, and $q(t)$ be the number of packets queued at the multiplexer at time t (at the end of the t^{th} time slot). Then the N-to-1 multiplexer with buffer B satisfies the following four properties:*

(P1) Flow conservation: arriving packets from the N input links are either stored in the buffer or transmitted through the n output links, i.e.,

$$q(t) = q(t-1) + \sum_{i=0}^{N-1} a_i(t) - d(t) - \sum_{i=1}^{N-1} \ell_i(t). \tag{5.37}$$

(P2) Non-idling: there is always a departing packet if there are packets in the buffer or there are arriving packets, i.e.,

$$d(t) = \begin{cases} 0 & \text{if } q(t-1) + \sum_{i=0}^{N-1} a_i(t) = 0 \\ 1 & \text{otherwise} \end{cases}. \tag{5.38}$$

(P3) Maximum buffer usage: arriving packets are lost only when buffer is full, i.e., for $i = 1, \ldots, N-1$,

$$\ell_i(t) = \begin{cases} 1 & \text{if } q(t-1) + \sum_{i=0}^{N-1} a_i(t) \geq B + i + 1 \\ 0 & \text{otherwise} \end{cases} . \quad (5.39)$$

(P4) FIFO with prioritized inputs: packets depart in the first in first out (FIFO) order. The priority of the input links is increasing *in the link number. As such, if there are multiple arriving packets, the packet from the largest input link number is put in the multiplexer first. Specifically, the virtual delay for the input $a_i(t)$ is $q(t-1) + \sum_{j=i+1}^{N-1} a_j(t)$, the sum of the number of packets that is stored in the multiplexer at $t-1$ and the number of higher priority packets that arrives at time t.*

From (P1-3), it is clear that the $q(t)$ process of an N-to-1 multiplexer satisfies the following recursive equation:

$$q(t) = \min[(q(t-1) + a(t) - 1)^+, B], \quad (5.40)$$

where $a(t) = \sum_{i=0}^{N-1} a_i(t)$ is the total number of arrivals at time t, and $x^+ = \max(0, x)$. In view of (5.40), if one does not care about the exact match of the loss processes, one can emulate the departure process of an N-to-1 multiplexer by emulating the $q(t)$ process only. This leads to our definition of delayed-loss multiplexers in Definition 5.4.2.

Definition 5.4.2. (Delayed-loss multiplexer) *An N-to-1 delayed-loss multiplexer with buffer B is a network element with N input links and N output links. As in Definition 5.4.1, the first output link of this multiplexer is the departure port and the rest of the output links are the loss ports (we use the same diagrammatic representation in Figure 5.23). Let $q(t)$ be the number of packets that are queued at the delayed-loss multiplexer at time t (and will be departed from the departure port). Then the N-to-1 delayed-loss multiplexer with buffer B satisfies the recursive equation in (5.40), (P2) and (P4) of Definition 5.4.1.*

As (5.40) is also the governing equation of the N-to-1 multiplexer, (P2) and (P4) imply that the delayed-loss multiplexer and the multiplexer have identical FIFO departure processes (from the departure ports) if both systems are started from empty systems and subject to identical arrival processes.

As in Definition 5.3.2, we define scaled SDL multiplexers in Definition 5.4.3 below. As explained in Section 5.3, scaled SDL multiplexers have the time interleaving property in Proposition 5.1.3 .

Definition 5.4.3. (SDL multiplexer) *A N-to-1 (delayed-loss) multiplexer is called an N-to-1* SDL *(delayed-loss) multiplexer if the multiplexer is built with delay lines (in Definition 5.1.1) and $N \times N$ switches. An N-to-1* SDL *(delayed-loss) multiplexer is with scaling factor k (see Figure 5.24) if the delay in every delay line is k times of that in the original N-to-1* SDL *(delayed-loss) multiplexer.*

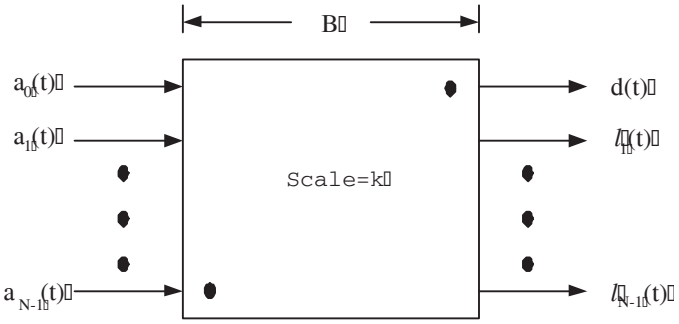


Fig. 5.24. A scaled N-to-1 multiplexer with buffer B and scaling factor k

5.4.1 Prioritized concentrator

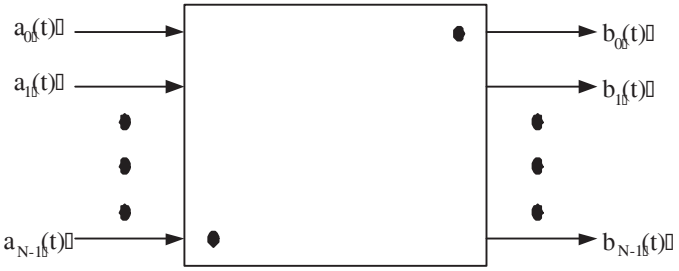


Fig. 5.25. An N-to-N prioritized concentrator

We first generalize the 2×2 prioritized concentrator in Definition 5.3.3.

Definition 5.4.4. (Concentrator) *An $N \times N$ prioritized concentrator (see Figure 5.25) is an $N \times N$ switch with its connection pattern depending on its N inputs. Both the input links and output links are*

numbered from the top to the bottom. The priority of the input links is increasing *in the link number and the priority of the output links is* decreasing *in the link number (the dotted input and the dotted output in the diagrammatic representation have the highest priority). The packets that arrive at high priority input links have priority to be switched to high priority output links. Thus, if there is a packet arrival at input link $N-1$, it is switched to output link 0. If there is no arrival at input link $N-1$ and there is an arrival at input link $N-2$, the arrival at input link $N-2$ is switched to output link 0. Mathematically, the state at output link k is*

$$b_k(t) = \sum_{i=k+1}^{N} a_{N-i}(t) 1_{\left\{\sum_{j=1}^{i-1} a_{N-j}(t) = k\right\}},$$

where $1_{\{A\}}$ is 1 if the event A is true and 0 otherwise.

Note that if there is a packet arriving at the $N-i^{th}$ input at time t (i.e., $a_{N-i}(t) = 1$), then the packet is routed to the k^{th} output if there are exactly k packet arrivals at the inputs $N-1, N-2, \ldots, N-i+1$ at time t (i.e., $\sum_{j=1}^{i-1} a_{N-j}(t) = k$). Thus, $b_k(t) = 1$, $k = 0, 1, \ldots, N-1$, **if and only if there are at least $k+1$ packet arrivals at time t.** The purpose of the $N \times N$ prioritized concentrator is to perform traffic concentration, i.e., to sort inputs $\{a_i(t), i = 0, 1, \ldots, N-1\}$ (according to a pre-assigned priority) so that $b_k(t) = a_i(t)$ for some i and that $b_k(t) \geq b_{k+1}(t)$ for $k = 0, 1, \ldots, N-1$.

We note that an $N \times N$ prioritized concentrator is an N-to-1 multiplexer with buffer 0. It is also an N-to-1 delayed-loss multiplexer with buffer 0.

5.4.2 Recursive construction of N-to-1 multiplexers

In this section, we show how one constructs an N-to-1 delayed-loss multiplexer with a large buffer by time interleaving N N-to-1 delayed-loss multiplexers with small buffers. In Figure 5.26, we consider a network element with N inputs and N outputs. It is a concatenation of a prioritized concentrator, an $N \times N$ switch and a scaled N-to-1 delayed-loss multiplexer with buffer B and scaling factor N. The i^{th} output of the $N \times N$ switch is connected to a delay line with delay i, $i = 0, 1, \ldots, N-1$. We will show in Theorem 5.4.5 such a network element can be operated as an N-to-1 delayed-loss multiplexer with buffer $N(B+1)-1$.

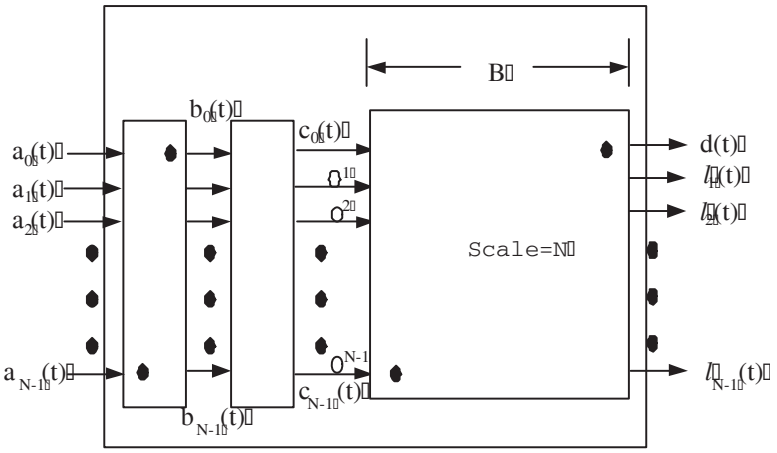


Fig. 5.26. Recursive construction of an N-to-1 delayed-loss multiplexer with buffer $N(B+1)-1$

As described in Proposition 5.1.3, a scaled N-to-1 multiplexer with buffer B and scaling factor N can be operated as time interleaving of N N-to-1 multiplexers with buffer B (see Figure 5.27). The N outputs of these delay lines are connected to the inputs of the N multiplexers in a round robin fashion. The N outputs of the N multiplexers are connected to the N outputs of the network element in the same order. The state of each multiplexer, i.e., the number of packets stored in the multiplexer, remains unchanged when the multiplexer is not connected. Thus, every multiplexer changes its state every N time slots. To ease our presentation, we reorder the N time interleaved multiplexers in every time slot. The i^{th} multiplexer at time t is the multiplexer that is going to be connected at time $t+i$, $i=1,2,\ldots,N-1$. Thus, the first multiplexer at any time is always the multiplexer that is going to be connected in the next time slot.

As shown in Figure 5.27, let $a_i(t)$, $b_i(t)$ and $c_i(t)$, $i=0,1,\ldots,N-1$, be the inputs of the concentrator, the outputs of the concentrator, and the outputs of the $N\times N$ switch. Let $q_i^0(t)$, $i=1,\ldots,N$, **be the number of packets stored at time** t **in the** i^{th} N**-to-1 delayed-loss multiplexer at time** t (i.e., the multiplexer that is going to be connected to the $N\times N$ switch at time $t+i$). Since the N multiplexers are connected in a round robin fashion, the $i+1^{th}$ multiplexer at time $t-1$ becomes the i^{th} multiplexer at time t for $i=1,\ldots,N$. As the

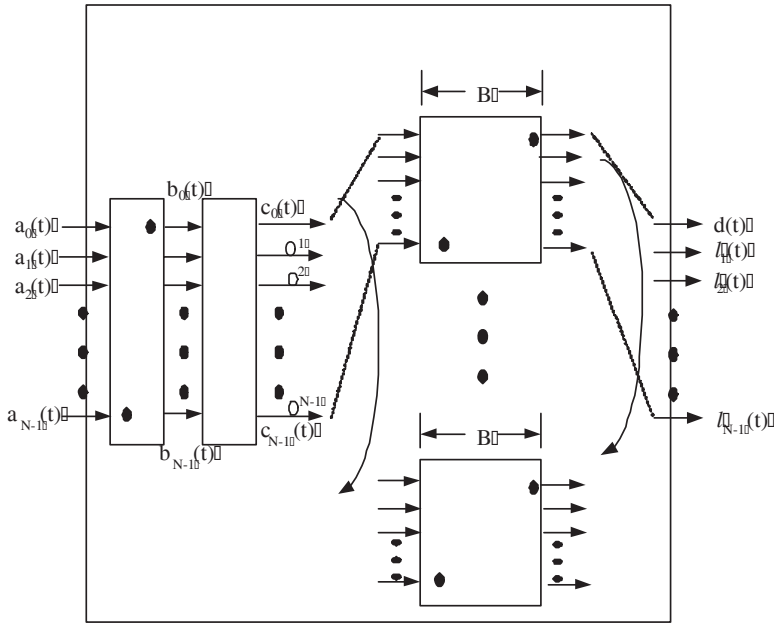


Fig. 5.27. An N-to-1 delayed-loss multiplexer with buffer $N(B+1)-1$

state of a multiplexer remains unchanged when the multiplexer is not connected, we have

$$q_i^0(t) = q_{i+1}^0(t-1), \quad i = 1, \ldots, N-1. \tag{5.41}$$

On the other hand, the first multiplexer at time $t-1$ will be connected at time t and become the N^{th} multiplexer at time t. It then follows from the governing equation for a multiplexer in (5.40) that

$$q_N^0(t) = \min[(q_1^0(t-1) + \sum_{k=0}^{N-2} c_{1+k}(t-1-k) + c_0(t) - 1)^+, B]. \tag{5.42}$$

We define

$$q_i(t-1) = \min[q_i^0(t-1) + \sum_{k=0}^{N-i-1} c_{i+k}(t-1-k), B+1], \quad i = 1, 2 \ldots, N, \tag{5.43}$$

with the convention that the sum equals 0 if the upper index is smaller than the lower index. **The quantity $q_i(t-1)$ represents the number of packets in the system at time $t-1$ that are eligible to leave the system from the departure port at time $t+i-1$ if the**

arrivals were blocked from time t onward. To see this, note that $\sum_{k=0}^{N-i-1} c_{i+k}(t-1-k)$ is the number of packets that are already in the fiber delay lines and will become the inputs to the multiplexer connected at time $t+i-1$ (if there were no further arrivals from time t onward). Recall that $q_i^0(t-1)$ represents the number of packets stored in the multiplexer that is going to be connected at time $t+i-1$. As the connected multiplexer at time $t+i-1$ has buffer B and there is one (possible) departure at time $t+i-1$, the sum of $q_i^0(t-1)$ and $\sum_{k=0}^{N-i-1} c_{i+k}(t-1-k)$ cannot exceed $B+1$.

For this network element, we also define

$$q(t-1) = \sum_{i=1}^{N} q_i(t-1). \tag{5.44}$$

Clearly, $q(t-1)$ **is the total number of packets in the system that will depart from the departure link from time t onward if the arrivals to the system were blocked**. The connection pattern of the $N \times N$ switch in the middle stage of the network element is set according to the value of $q(t-1)$.

As we shall prove later, the network element is a delayed-loss multiplexer with buffer $N(B+1)-1$ (under the operation rule R_N defined below) and $q(t-1)$ is also the number of packets queued in the delayed-loss multiplexer at time $t-1$.

Rule R_N: The connection pattern of the $N \times N$ switch in the middle stage of the network element in Figure 5.27 is set to $P^{q(t-1)}$, where P is the $N \times N$ circular-shift matrix, i.e., for all $i, j = 0, 1, \ldots, N-1$, $P_{i,j} = 1$ when $j = i+1 \bmod N$ and $P_{i,j} = 0$ otherwise.

Note that P^N is the identity matrix. Thus, if $q(t-1) \bmod N = m$, then

$$P^{q(t-1)} = P^m,$$

where the (i,j)-th element of P^m is

$$(P^m)_{ij} = \begin{cases} 1 & \text{if } j = (i+m) \bmod N \\ 0 & \text{otherwise} \end{cases}.$$

Specifically, if output link j of the switch is connected with input link i, then $c_j(t) = b_i(t)$, where $j = (i+m) \bmod N = (i+q(t-1)) \bmod N$.

Intuitively, one may view $q_i(t)$'s as the numbers of customers in the parallel queues in Figure 2.42 and Rule R_N mimics the join-the-shortest-queue policy and the serve-the-longest-queue policy (as the

shifter in the Knockout switch in Section 2.7.2). By so doing, the parallel queues are kept in the most balanced state, i.e., for all t

$$q_1(t) \geq q_2(t) \geq \ldots \geq q_N(t) \geq q_1(t) - 1.$$

As a result, the parallel queues behaves as if it was a single queue with a shared buffer. This leads to the following theorem and its formal proof is shown in Section 5.4.4.

Theorem 5.4.5. *If the network element in Figure 5.26 is operated under Rule R_N and it is started from an empty system, then it is an N-to-1 delayed-loss multiplexer with buffer $N(B+1)-1$.*

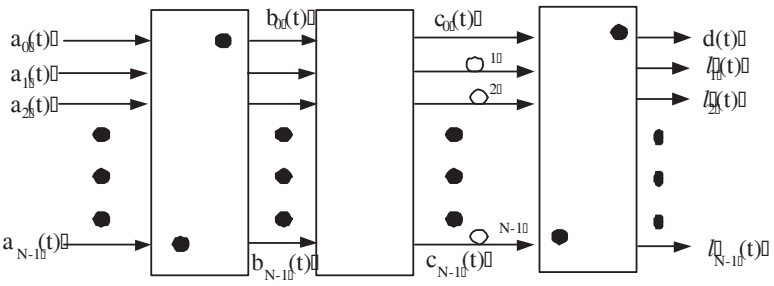


Fig. 5.28. An N-to-1 delayed-loss multiplexer with buffer $N-1$

Example 5.4.6. Since a prioritized concentrator is an N-to-1 multiplexer with buffer 0, it follows from Theorem 5.4.5 that the network element in Figure 5.28 is an N-to-1 delayed-loss multiplexer with buffer $N-1$. For this example, we have from (5.43) that

$$\begin{aligned} q_i(t-1) &= \min[\sum_{k=0}^{N-i-1} c_{i+k}(t-1-k), 1] \\ &= \max_{0 \leq k \leq N-i-1} [c_{i+k}(t-1-k)], \end{aligned} \tag{5.45}$$

for $i = 1, 2, \ldots, N-1$, and $q_N(t-1) = 0$. Thus,

$$q(t-1) = \sum_{i=1}^{N-1} \max_{0 \leq k \leq N-i-1} [c_{i+k}(t-k-1)]. \tag{5.46}$$

The connection pattern of the $N \times N$ switch at time t is then set according to Rule R_N that only depends on $q(t-1)$. We note that the

total number of packets that can be stored in the optical delay lines of the network element in Figure 5.28 is $N(N-1)/2$, which is much larger than the designed buffer size $N-1$. It means that some packets that should have been discarded when they arrive are still stored in the system. In view of (5.45), one can see that at most one packet can be departed from the departure port at time $t+i-1$ and the rest of packets that are stored in the delay lines will be departed from the loss ports. This is how the delayed losses occur!

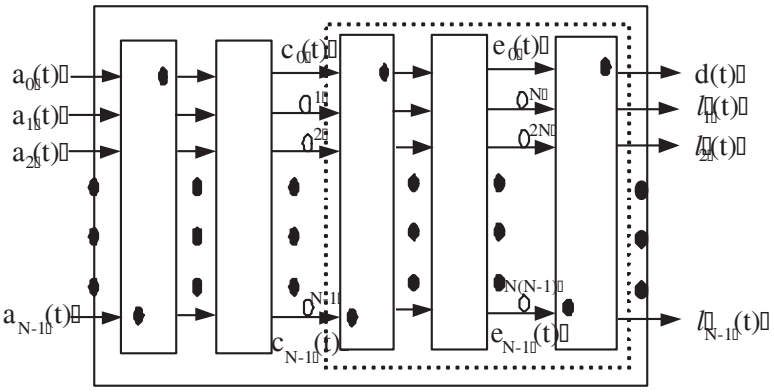


Fig. 5.29. An N-to-1 delayed loss multiplexer with $B = N^2 - 1$.

Example 5.4.7. In Figure 5.29, we illustrate how one applies Theorem 5.4.5 to construct an N-to-1 delayed-loss multiplexer with buffer N^2-1 from the N-to-1 delayed-loss multiplexer with buffer $N-1$ in Figure 5.28. Note that the inner block in Figure 5.29 is exactly the same as the N-to-1 delayed-loss multiplexer with buffer $N-1$ in Figure 5.28 except that the delay in every delay line is scaled N times. As such, it is an N-to-1 delayed-loss SDL multiplexer with buffer $N-1$ and scaling factor N. As a direct consequence of Theorem 5.4.5, the network element in Figure 5.29 is indeed an N-to-1 delayed-loss multiplexer with buffer N^2-1. As shown in (5.46) in Example 5.4.6, $q_j^0(t-1)$, i.e., the number of packets stored in the multiplexer that is going to be connected at time $t+j-1$, has the following form:

$$q_j^0(t-1) = \sum_{i=1}^{N-1} \max_{0 \le k \le N-i-1} [e_{i+k}(t-(k+1)N+j-1)],$$

$$j = 1, 2, \ldots, N. \quad (5.47)$$

Moreover, the second $N \times N$ switch is set according to $q_1^0(t-1)$. To see the operation of the first $N \times N$ switch, we have from (5.43) that

$$q_j(t-1) = \min[q_j^0(t-1) + \sum_{k=0}^{N-j-1} c_{j+k}(t-1-k), B+1],$$
$$j = 1, 2 \ldots, N. \quad (5.48)$$

Thus, the operation of the first $N \times N$ switch is set according to $q(t-1) = \sum_{j=1}^{N} q_j(t-1)$.

By Theorem 5.4.5 and the time interleaving property in Proposition 5.1.3, one can then recursively construct a multi-stage multiplexer with a large buffer. One can further combine the operation of the prioritized concentrator and the $N \times N$ switch at each stage by an $N \times N$ switch. In Figure 5.30, we show the construction of an N-to-1 delayed-loss multiplexer with buffer $N^k - 1$. All its switching patterns are completely determined by the state of the multiplexer.

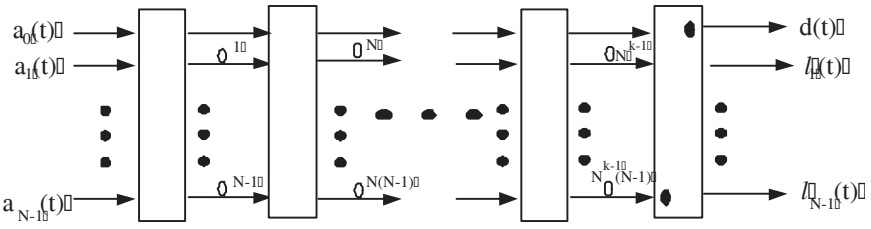


Fig. 5.30. An N-to-1 delayed-loss multiplexer with $B = N^k - 1$.

5.4.3 Self-routing optical multiplexers

Now we have shown from Theorem 5.4.5 a way to control the switching patterns in the N-to-1 delayed-loss multiplexer with buffer $N^k - 1$ in Figure 5.30. One key observation from this is that those packets departing from the departure port have the same delays as the (ideal) N-to-1 multiplexer with buffer $N^k - 1$ since the two departure processes are identical and both of them are FIFO. Moreover, for any packet that departs from the departure port and experiences delay $0 \le d \le N^k - 1$, there is a *unique* path through the network element. The path can be

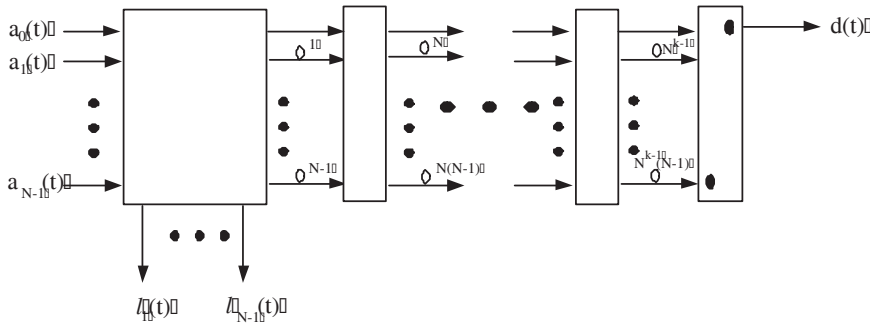


Fig. 5.31. A self-routing N-to-1 multiplexer with $B = N^k - 1$.

determined by the unique decomposition of $d = \sum_{j=1}^{k} r_j N^{j-1}$ and the packet is sent through the network element by taking the $r_j{}^{th}$ output link at the j^{th} $N \times N$ switch. However, the path of a packet that departs from a loss port cannot be determined this way. Thus, if we discard all the packets that depart from the loss ports before entering the network element, then we are left with the packets that depart from the departure port and the paths of those packets can be determined upon their arrivals. As these paths are identical to those from the N-to-1 delayed-loss multiplexer, we conclude that these paths do not conflict with each other, i.e., no more than one packet occupies the same link at any time. This leads to the *self-routing* multiplexer in Figure 5.31.

In Figure 5.31, we replace the first $N \times N$ switch by an $N \times (2N - 1)$ switch in Figure 5.30. The network element in Figure 5.31 keeps track of the number of packets stored in it. If such a number exceeds $N^k - 1$, further arrivals are lost immediately. Specifically, let $q(t)$ be the number of packets stored in the network element. Then $q(t)$ is governed by

$$q(t) = \min[(q(t-1) + \sum_{i=0}^{N-1} a_i(t) - 1)^+, N^k - 1], \tag{5.49}$$

and

$$\ell_i(t) = \begin{cases} 1 & \text{if } q(t-1) + \sum_{i=0}^{N-1} a_i(t) \geq N^k + i \\ 0 & \text{otherwise} \end{cases}, \tag{5.50}$$

for $i = 1, \ldots, N-1$. Let q be the number of packets stored in the network element when a particular packet enters the network element. Then we have $0 \leq q \leq N^k - 1$ and there exists a unique vector $r =$

$(r_1, r_2, \ldots, r_k)$ with $0 \le r_j \le N-1$ for all j such that

$$q = \sum_{j=1}^{k} r_j N^{j-1}.$$

The packet is then routed through the network element by taking the ${r_j}^{th}$ output link at the j^{th} $N \times N$ switch. Note that we now not only match the departure process but also the loss processes. Thus, the network element in Figure 5.31 is an N-to-1 multiplexer with buffer $N^k - 1$.

To see the analogy between our self-routing multiplexer and the classical banyan self-routing network in Section 2.5.2, one may view the virtual delay in our self-routing multiplexer as the "output address" in the banyan self-routing network. By routing packets to different "output addresses," we then resolve conflicts at the multiplexer.

5.4.4 Proof of Theorem 5.4.5

In this section, we prove Theorem 5.4.5. The proof of Theorem 5.4.5 requires the following lemmas. In Lemma 5.4.8, we first derive the governing equations for $q_i(t)$, $i = 1, \ldots, N$. To gain the intuition of our proof, one may view $q_i(t)$'s as the numbers of customers in the parallel queues in Figure 2.42.

Lemma 5.4.8. *The quantities $q_i(t)$, $i = 1, 2, \ldots, N$, satisfy the following recursive equations:*

$$q_i(t) = \min[q_{i+1}(t-1) + c_i(t), B+1], \quad i = 1, 2, \ldots, N-1, \tag{5.51}$$

and

$$q_N(t) = \min[(q_1(t-1) + c_0(t) - 1)^+, B]. \tag{5.52}$$

Proof. We have from (5.41) and (5.43) that for $i = 1, \ldots, N-1$,

$$\begin{aligned} q_i(t) &= \min[q_i^0(t) + \sum_{k=0}^{N-i-1} c_{i+k}(t-k), B+1] \\ &= \min[q_{i+1}^0(t-1) + \sum_{k=0}^{N-i-2} c_{i+1+k}(t-1-k) + c_i(t), B+1] \\ &= \min[q_{i+1}^0(t-1) + \sum_{k=0}^{N-i-2} c_{i+1+k}(t-1-k) + c_i(t), \end{aligned}$$

$$
\begin{aligned}
&\qquad\qquad\qquad\qquad\qquad\qquad B+1, B+1+c_i(t)] \\
&= \min\Big[\min[q^0_{i+1}(t-1) + \sum_{k=0}^{N-i-2} c_{i+1+k}(t-1-k), B+1] \\
&\qquad\qquad\qquad\qquad\qquad\qquad +c_i(t), B+1\Big] \\
&= \min[q_{i+1}(t-1) + c_i(t), B+1].
\end{aligned}
$$

Moreover, it follows from (5.42) and (5.43) that

$$
\begin{aligned}
&q_N(t) = \min[q^0_N(t), B+1] = q^0_N(t) \\
&= \min[(q^0_1(t-1) + \sum_{k=0}^{N-2} c_{1+k}(t-1-k) + c_0(t) - 1)^+, B] \\
&= \min[(q^0_1(t-1) + \sum_{k=0}^{N-2} c_{1+k}(t-1-k) + c_0(t) - 1)^+, \\
&\qquad\qquad\qquad\qquad\qquad\qquad B, B + c_0(t)] \\
&= \min\Big[(\min[q^0_1(t-1) + \sum_{k=0}^{N-2} c_{1+k}(t-1-k), B+1] \\
&\qquad\qquad\qquad\qquad\qquad\qquad +c_0(t) - 1)^+, B\Big] \\
&= \min[(q_1(t-1) + c_0(t) - 1)^+, B]
\end{aligned}
$$

■

Lemma 5.4.9 shows that if the vector $(q_1(t-1), q_2(t-1), \ldots, q_N(t-1))$ is in the most balanced state, i.e., the difference between the largest element and the smallest element is at most 1, then Rule R_N behaves as if it is the join-the-shortest-queue policy and the state is still kept in the most balanced state.

Lemma 5.4.9. *If*

$$q_1(t-1) \geq q_2(t-1) \geq \ldots \geq q_N(t-1) \geq q_1(t-1) - 1, \qquad (5.53)$$

then under Rule R_N

$$
\begin{aligned}
&q_1(t-1) + c_0(t) \geq q_2(t-1) + c_1(t) \geq \ldots \\
&\qquad \geq q_N(t-1) + c_{N-1}(t) \geq q_1(t-1) + c_0(t) - 1. \qquad (5.54)
\end{aligned}
$$

Proof. We define the function $h(k, q(t-1)) = k - q(t-1) \bmod N$. Then under Rule R_N, we have

$$c_i(t) = b_{h(i,q(t-1))}(t), \quad i = 0, 1, \ldots, N-1. \tag{5.55}$$

Note that for any fixed $q(t-1)$, $h(k, q(t-1))$ is increasing with k, except when $k = q(t-1) - 1 \bmod N$ at which $h(k, q(t-1)) = N-1$ and $h(k+1, q(t-1)) = 0$.

We first show that

$$q_{i+1}(t-1)+c_i(t) \geq q_{i+2}(t-1)+c_{i+1}(t), \quad i = 0, 1, \ldots, N-2. \tag{5.56}$$

If $i \neq (q(t-1)-1) \bmod N$, then

$$h(i+1, q(t-1)) \geq h(i, q(t-1)).$$

Since $b_i(t)$ is decreasing in i ($b_i(t)$'s are the outputs from a prioritized concentrator), it then follows that

$$c_i(t) = b_{h(i,q(t-1))}(t) \geq b_{h(i+1,q(t-1))}(t) = c_{i+1}(t). \tag{5.57}$$

Hence, (5.56) follows from (5.53) and (5.57).

On the other hand, if $i = (q(t-1)-1) \bmod N$, then we have from (5.53) that $q_{i+1}(t-1) = q_{i+2}(t-1)+1$ for this case. This implies that

$$\begin{aligned} &q_{i+1}(t-1) + c_i(t) = q_{i+2}(t-1) + 1 + c_i(t) \\ &\geq q_{i+2}(t-1) + c_{i+1}(t) \end{aligned}$$

Now we show that

$$q_N(t-1) + c_{N-1}(t) \geq q_1(t-1) + c_0(t) - 1. \tag{5.58}$$

If $q(t-1) \bmod N \neq 0$, then $h(N-1, q(t-1)) \leq h(0, q(t-1))$. Since $b_i(t)$ is decreasing in i, it then follows that

$$c_{N-1}(t) = b_{h(N-1,q(t-1))}(t) \geq b_{h(0,q(t-1))}(t) = c_0(t).$$

That (5.58) holds then follows from the last inequality in (5.53). On the other hand, if $q(t-1) \bmod N = 0$, then we have from (5.53) that $q_1(t-1) = q_2(t-1) = \ldots = q_N(t-1)$. The inequality in (5.58) holds trivially as $0 \leq c_j(t) \leq 1$ for all j. ■

Lemma 5.4.10 shows that Rule R_N behaves as if it is the serve-the-longest-queue policy and the state $(q_1(t), q_2(t), \ldots, q_N(t))$ is always kept in the most balanced state if the state is started from an empty system.

Lemma 5.4.10. *If the network element in Figure 5.27 is operated under Rule R_N and it is started from an empty system, then for all $t \geq 0$*

$$q_{i+1}(t) \leq q_i(t), \quad i = 1, 2, \ldots, N-1, \tag{5.59}$$

and

$$q_1(t) \leq q_N(t) + 1. \tag{5.60}$$

Proof. Since we assume that the network element starts from an empty system, Eq. (5.59) and (5.60) hold for $t = 0$. Now we assume that they hold for some $t - 1$.

Using the induction hypotheses and Lemma 5.4.9, we have from (5.51) that $q_i(t) \geq q_{i+1}(t)$ for $i = 1, \ldots, N-2$. To see that $q_N(t)+1 \geq q_1(t)$, observe from (5.52) that

$$\begin{aligned} q_N(t) + 1 &= \min[(q_1(t-1) + c_0(t) - 1)^+, B] + 1 \\ &= \min[(q_1(t-1) + c_0(t) - 1)^+ + 1, B + 1] \\ &\geq \min[q_1(t-1) + c_0(t), B + 1]. \end{aligned}$$

Once again, using the induction hypotheses, Lemma 5.4.9 and (5.51), we have that

$$\begin{aligned} q_N(t) + 1 &\geq \min[q_1(t-1) + c_0(t), B + 1] \\ &\geq \min[q_2(t-1) + c_1(t), B + 1] = q_1(t). \end{aligned}$$

It remains to show that $q_{N-1}(t) \geq q_N(t)$. Note from (5.52), the induction hypotheses, the last inequality in (5.54), and (5.51) that

$$\begin{aligned} q_N(t) &= \min[(q_1(t-1) + c_0(t) - 1)^+, B] \\ &\leq \min[(q_N(t-1) + c_{N-1}(t))^+, B] \\ &= \min[q_N(t-1) + c_{N-1}(t), B] \\ &\leq \min[q_N(t-1) + c_{N-1}(t), B + 1] = q_{N-1}(t). \end{aligned}$$

■

(Proof of Theorem 5.4.5) We first show that $q(t)$ satisfies the following recursive equation:

$$q(t) = \min\left[\left(q(t-1) + \sum_{i=0}^{N-1} a_i(t) - 1\right)^+, N(B+1) - 1\right]. \tag{5.61}$$

Since the network element in Figure 5.27 is started from an empty system, we first consider the case that $q(t-1) = 0$. If $q(t-1) = 0$, then $q_i(t-1) = 0$ for all i and $c_i(t) = b_i(t)$, $i = 0, 1, 2 \ldots, N-1$ (according to Rule R_N). Therefore, we have from (5.51) and (5.52) that

$$\begin{aligned}\sum_{i=1}^{N} q_i(t) &= \sum_{i=1}^{N-1} \min[q_{i+1}(t-1) + c_i(t), B+1] \\ &\qquad + \min[(q_1(t-1) + c_0(t) - 1)^+, B] \\ &= \sum_{i=1}^{N-1} b_i(t)\end{aligned}$$

We argue that

$$\sum_{i=1}^{N-1} b_i(t) = (\sum_{i=0}^{N-1} b_i(t) - 1)^+. \tag{5.62}$$

To see this, note that (5.62) holds trivially if $b_0(t) = 1$. On the other hand, if $b_0(t) = 0$, then $b_i(t) = 0$, $i = 1, 2, \ldots, N-1$, as $b_0(t)$ is the largest element in $b_i(t)$. Both sides of (5.62) equal 0. Using (5.62) yields

$$\begin{aligned}q(t) = \sum_{i=1}^{N} q_i(t) &= (\sum_{i=0}^{N-1} b_i(t) - 1)^+ \\ &= (\sum_{i=0}^{N-1} a_i(t) - 1)^+ \\ &= (a(t) - 1)^+ \\ &= \min[(a(t) - 1)^+, N(B+1) - 1].\end{aligned}$$

Thus, (5.61) holds for the case $q(t-1) = 0$.

Now consider the case that $q(t-1) > 0$. As $q_1(t-1)$ is the largest element in $q_i(t-1)$, $i = 1, 2, \ldots, N$ (Lemma 5.4.10), it follows that $q_1(t-1) > 0$. In conjunction with (5.52), we have

$$\begin{aligned}q_N(t) &= \min[q_1(t-1) + c_0(t) - 1, B] \\ &= \min[q_1(t-1) + c_0(t), B+1] - 1.\end{aligned} \tag{5.63}$$

Thus, we have from (5.51) and (5.63) that

$$q(t) = \sum_{i=1}^{N} q_i(t) = \sum_{i=0}^{N-1} \min[q_{i+1}(t-1) + c_i(t), B+1] - 1. \tag{5.64}$$

We argue that

$$\sum_{i=0}^{N-1} \min[q_{i+1}(t-1) + c_i(t), B+1]$$
$$= \min[\sum_{i=0}^{N-1} q_{i+1}(t-1) + c_i(t), N(B+1)]. \quad (5.65)$$

If $q_1(t-1) + c_0(t) \leq B+1$, then it follows from Lemma 5.4.10 and Lemma 5.4.9 that $q_{i+1}(t-1) + c_i(t) \leq B+1$ for all $i = 0, 1, \ldots, N-1$. Thus,

$$\sum_{i=0}^{N-1} \min[q_{i+1}(t-1) + c_i(t), B+1]$$
$$= \sum_{i=0}^{N-1} q_{i+1}(t-1) + c_i(t)$$
$$= \min[\sum_{i=0}^{N-1} q_{i+1}(t-1) + c_i(t), N(B+1)].$$

On the other hand, if $q_1(t-1) + c_0(t) \geq B+2$, then it follows from Lemma 5.4.10 and Lemma 5.4.9 that $q_{i+1}(t-1) + c_i(t) \geq B+1$ for all $i = 0, 1, \ldots, N-1$. Thus,

$$\sum_{i=0}^{N-1} \min[q_{i+1}(t-1) + c_i(t), B+1]$$
$$= \sum_{i=0}^{N-1} B+1 = N(B+1)$$
$$= \min[\sum_{i=0}^{N-1} q_{i+1}(t-1) + c_i(t), N(B+1)].$$

Using (5.65) in (5.64) yields

$$\begin{aligned} q(t) &= \sum_{i=1}^{N} q_i(t) \\ &= \min[\sum_{i=0}^{N-1} q_{i+1}(t-1) + c_i(t), N(B+1)] - 1 \\ &= \min[\sum_{i=1}^{N} q_i(t-1) + \sum_{i=0}^{N-1} a_i(t), N(B+1)] - 1 \\ &= \min[q(t-1) + a(t), N(B+1)] - 1 \\ &= \min[q(t-1) + a(t) - 1, N(B+1) - 1]. \end{aligned}$$

Thus, we have shown that $q(t)$ satisfies (5.61).

Now we show the non-idling property in (P2), i.e.,

$$d(t) = 1_{\{q(t-1)+a(t)>0\}}. \tag{5.66}$$

From the non-idling property for the delayed-loss multiplexer connected to the outputs at time t, it follows that

$$d(t) = 1_{\{q_1(t-1)+c_0(t)>0\}}.$$

Note that $q_1(t-1) > 0$ if and only if $q(t-1) > 0$. Therefore, if $q_1(t-1) > 0$,

$$d(t) = 1 = 1_{\{q(t-1)+a(t)>0\}}.$$

On the other hand, if $q_1(t-1) = q(t-1) = 0$, then $c_0(t) = b_0(t)$ from Rule R_N. As $b_0(t)$ is the largest element in $b_i(t)$, $i = 0, 1, \ldots, N-1$, $c_0(t) > 0$ if and only if $a(t) > 0$. Thus,

$$d(t) = 1_{\{c_0(t)>0\}} = 1_{\{a(t)>0\}}.$$

Finally, we verify the departure order is FIFO in (P4). Without loss of generality, assume that $q(t-1) = mN + k$ for some $m \geq 0$ and $0 \leq k \leq N-1$. From Lemma 5.4.10, it follows that

$$q_i(t-1) = \begin{cases} m+1 & 1 \leq i \leq k \\ m & k+1 \leq i \leq N \end{cases}.$$

Also, we have $b_0(t) = c_k(t)$ from Rule R_N. From (5.51) and (5.52) in Lemma 5.4.8, $c_k(t)$ is added to the end of $q_{k+1}(t-1)$. Thus, the virtual delay for $b_0(t)$ is $Nq_{k+1}(t-1) + k = mN + k = q(t-1)$ as the N multiplexers are served in a round robin fashion. Similarly, if $b_0(t) = 1$, then $b_1(t)$ is added to the end of $q_{k+2 \bmod N}(t-1)$. One can easily verify that the virtual delay for $b_1(t)$ is $q(t-1)+1$. Continuing the same argument shows that the virtual delay for $b_i(t)$ is $q(t-1)+i$ if $b_0(t) = b_1(t) = \ldots = b_{i-1}(t) = 1$. Thus, the FIFO order is maintained.

5.5 FIFO multiplexers with variable length bursts

In the previous section, we have shown how to achieve exact emulation of multiplexers for fixed size packets (or cells). In this section, we address the natural question whether one can use the multiplexers for fixed size packets (or cells) for exact emulation of multiplexers with

variable length bursts. As in electronic buffers, this requires performing burst *segmentation* and *reassembly*. In this section, we assume that burst segmentation is feasible. Each variable length burst can be divided into a contiguous sequence of fixed size *cells*, and each cell can then be transmitted within a time slot in the multiplexer for fixed size cells.

There are two natural places for burst reassembly: (i) after the multiplexer for fixed size cells, and (ii) before the multiplexer for fixed size cells. The former approach is much more difficult to realize by SDL as it has to take the multiplexer into account. Moreover, it incurs additional reassembly delay for each burst. As such, exact emulation for multiplexers with variable length bursts cannot be achieved. Only a time shifted version can be achieved. The latter is the approach we use in this section. Its design objective is to schedule cells in a careful manner so that cells of the same burst depart *contiguously* from the multiplexer for fixed size cells. As such, we add a *cell scheduling block* in front of the multiplexer for fixed size cells. For such an architecture, we show there is an efficient cell scheduling algorithm. Starting from an empty system, we can perform cell scheduling by keeping track of a single state variable, called the *total virtual waiting time*. Moreover, the delay through the cell scheduling block is bounded above by a constant that only depends on the number of inputs and the maximum number of cells in a burst. Such a delay bound provides a limit on the number of fiber delay lines needed in the cell scheduling block.

5.5.1 Cell contiguity problem

Now we would like to extend the multiplexer for fixed size cells to cope with variable length bursts. In this section, we assume that burst segmentation is feasible. Each variable length burst can be divided into a contiguous sequence of fixed size *cells*, and each cell can then be transmitted within a time slot in the multiplexer for fixed size cells. Our objective is to use SDL units to design a multiplexer that achieves exact emulation of a FIFO finite buffer queue with variable length bursts, i.e., the departure process from the multiplexer is the same as that from a FIFO finite buffer queue with variable length bursts. For this, there are two things we need to do. The first is to schedule these bursts under the FIFO policy. The second is to maintain the contiguity of cells in a burst at the output link. The first thing is rather easy to

do. However, maintaining the cell contiguity becomes a problem as shown in the following example in Figure 5.32.

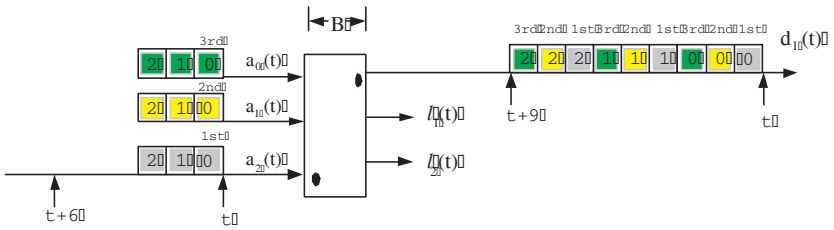


Fig. 5.32. An illustration for the contiguity problem

In Figure 5.32, we show that if we put variable length bursts directly into the multiplexer (for fixed length cells) in Definition 5.4.1, there will be a problem on cell contiguity. In this figure, we consider a multiplexer with 3 inputs and 3 outputs. Assume that the buffer at time t in the multiplexer is empty, i.e., $q(t) = 0$. There are three bursts that arrive at time t. The cells in a burst are indexed by consecutive integers starting from zero. We call the burst arriving to port $a_2(t)$ the first burst, the burst to port $a_1(t)$ the second burst, and the burst to port $a_0(t)$ the third burst. Assume that the buffer B is so large that no cells are lost (a trivial condition is $B \geq 6$ in this case). According to Definition 5.4.1, the order of multiplexing is in the descending order of the input link number for cells that arrive at the same time. Thus, the cell order in the departure port is cell 0 of the three bursts, followed by cell 1 of the three bursts and finally cell 2 of the three bursts, as shown in the figure. Clearly, cells in the same bursts do not depart contiguously. As a result, we cannot naively put variable length bursts directly into multiplexers for fixed size cells. To solve the cell contiguity problem, cells need to be scheduled in a careful manner as illustrated in Figure 5.33.

In Figure 5.33, we add a cell scheduling block in front of the multiplexer for fixed size cells. The function of the cell scheduling block is to route each cell to the *right input* of the multiplexer at the *right time* so that we can receive cells from the same burst contiguously. We illustrate this by considering the same traffic in Figure 5.32. At time t, we can schedule cell 0 of the first burst at input 2 of the multiplexer (for fixed size cells). In order for the cells in the first burst to come out contiguously from the multiplexer, we can not schedule anything

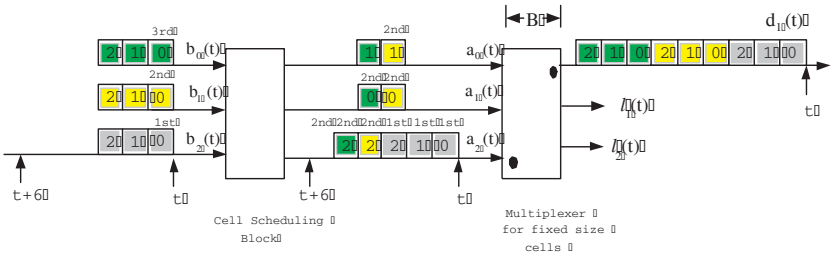


Fig. 5.33. Basic idea for cell scheduling

at input 1 and input 0 of the multiplexer at time t. At time $t+1$, we then schedule cell 1 of the first burst at input 2. Similarly, we do not schedule anything at input 1 and input 0 at time $t+1$. At time $t+2$, we schedule cell 2 of the first burst at input 2. Clearly, the cells in the first burst come out contiguously this way. Now we can schedule cell 0 of the second burst at input 1 and cell 1 of the second burst at input 0 at time $t+2$. Since the multiplexing order is in the descending order of the input link number for cells that arrive at the same time, cell 0 of the second burst will come out from the multiplexer after cell 2 of the first burst. Similarly, cell 1 of the second burst will be out after cell 0 of the second burst. At time $t+3$, we schedule cell 2 of the second burst at input 2, cell 0 of the third burst at input 1, and cell 1 of the third burst at input 0, respectively. At time $t+4$, we then schedule cell 2 of the third burst at input 2. It is easy to verify that the three bursts depart from the multiplexer contiguously, as shown in Figure 5.33. In Section 5.5.2, we will present the design of the scheduling block. In Section 5.5.4, we will further show that the delay through the cell scheduling block is bounded by a constant.

5.5.2 The overall multiplexer architecture

As addressed in the previous section, we need to add a cell scheduling block in order to overcome the cell contiguity problem. In Figure 5.34, we show our architecture for a variable length burst multiplexer with N inputs. It consists of two blocks. The latter is the N-to-1 multiplexer for fixed size cells described in Section 5.4. The former is the cell scheduling block. The function of the cell scheduling is to route each cell to the right input at the right time so that cells of the same burst come out contiguously. To achieve this, the cell scheduling block consists of two stages. As shown in Figure 5.34, there are N $1\times N$ switches

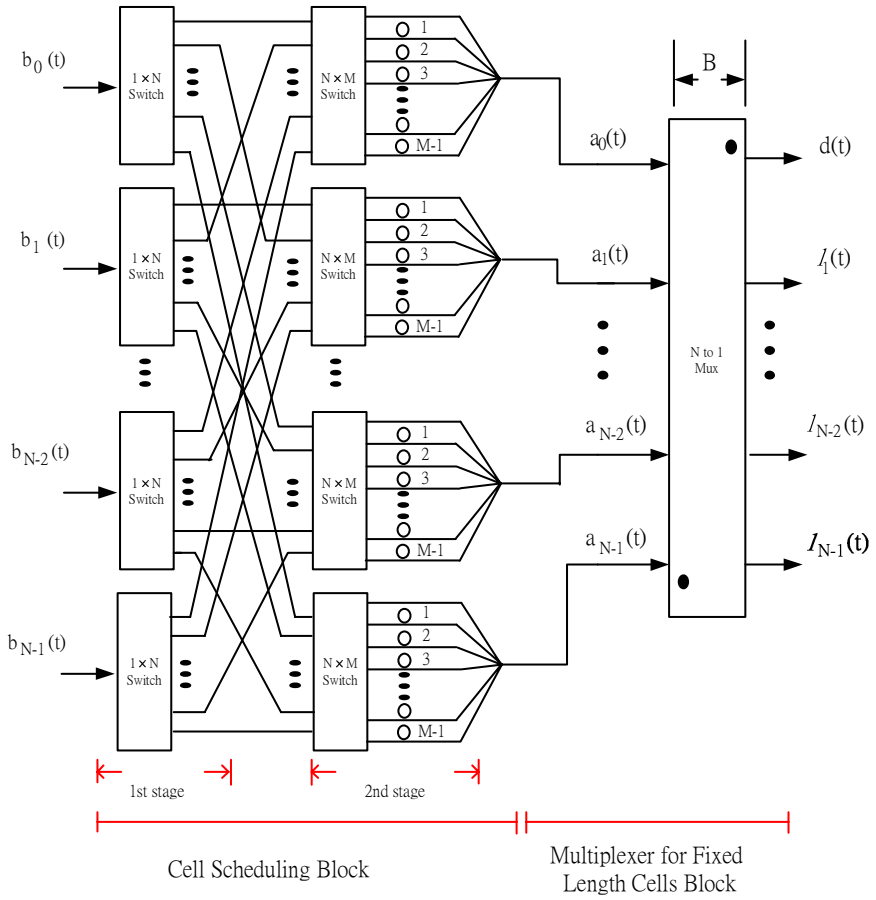


Fig. 5.34. Architecture

at the first stage. The objective of these switches is to route cells to the *right inputs* of the multiplexer for fixed size cells. At the second stage, there are N $N \times M$ switches. The M outputs of each switch are connected to fiber delay lines with delay from 0 to $M-1$. The objective of the second stage is to delay cells so that they arrive at the routed input at the *right time*. The constant $M-1$ is the maximum delay for a cell to go through the cell scheduling block. One key result of this section is that the delay for a cell in the cell scheduling block never exceeds $\left\lfloor \frac{(2N-2)\ell_{\max}-N+1}{N} \right\rfloor$, where $\ell_{\max}$ is the maximum number of cells in a burst. Thus, we can choose M so that $M \geq \left\lfloor \frac{(2N-2)\ell_{\max}-N+1}{N} \right\rfloor + 1$. One easy choice to satisfy this requirement is $M = 2\ell_{\max}$.

5.5.3 The cell scheduling algorithm

In order for the variable length burst multiplexer with N inputs to work properly, we assume that the burst length of a burst is known when the first cell of a burst arrives. This can be done either by adding the burst length information in the header of the first cell or by transmitting such information through a different channel in advance (see e.g., [171, 163]). The arrival time of the first cell in a burst is called the arrival time of that burst. Bursts that arrive at the same time are scheduled in the descending order of their input link numbers (as in the multiplexer for fixed size cells).

Now we describe our cell scheduling algorithm. Note that there are three natural constraints of the cell scheduling algorithm.

(i) Conflict constraint: no more than one cell can be scheduled at the same input (of the multiplexer for fixed size cells) at the same time.

(ii) Causality constraint: no cell can be scheduled before its arrival.

(iii) Contiguity constraint: cells from the same burst should be scheduled so that they leave the multiplexer for fixed length cells block contiguously.

To satisfy the conflict constraint, we have to keep track of the time slots used in every input. For this, we let $V_j(t)$, $j = 0, 1, \cdots, N-1$, be the number of time slots that cannot be scheduled at input j from t onward. In other words, the next available time slot for input j is at time $t + V_j(t)$. Following the queueing context, we call $V_j(t)$ the virtual waiting time of input j (as a cell that is routed to input j at time t

will have to wait for $V_j(t)$ time slots). As we will show later, under our cell scheduling algorithm (described below) the virtual waiting times satisfy the following inequalities for all t:

$$V_{N-1}(t) \geq V_{N-2}(t) \geq \cdots \geq V_1(t) \geq V_0(t) \geq V_{N-1}(t) - 1. \quad (5.67)$$

Initially, we set $V_j(0) = 0$ for all $j = 0, 1, \cdots, N-1$, so that the inequalities in (5.67) are satisfied. Moreover, if there is no burst arrival at time t, then the next available time slot for input j is still at time $t + V_j(t)$. Thus, we have

$$V_j(t+1) = (V_j(t) - 1)^+, \quad j = 0, 1, \ldots, N-1. \quad (5.68)$$

In this case, one can also easily verify that $V_j(t+1)$'s satisfy the inequalities in (5.67).

Let $V(t)$ be the total virtual waiting time at time t, i.e.,

$$V(t) = \sum_{j=0}^{N-1} V_j(t). \quad (5.69)$$

Using the inequalities in (5.67), one can relate the total virtual waiting time to the virtual waiting time of input j as follows:

$$V_j(t) = \begin{cases} \left\lfloor \frac{V(t)}{N} \right\rfloor + 1 & \text{for } j = N-1, N-2, \\ & \quad \cdots, N-k \\ \left\lfloor \frac{V(t)}{N} \right\rfloor & \text{for } j = 0, 1, \cdots, \\ & \quad N-k-1 \end{cases} \quad (5.70)$$

where $k = V(t) \bmod N$. Thus, the total virtual waiting time $V(t)$ is sufficient for the purpose of cell scheduling.

Now suppose that the $m+1^{th}$ burst arrives at time τ_m with length ℓ_m, $m = 0, 1, \ldots$. Let $V(\tau_m^-)$ be the total virtual waiting time immediately before the arrival of the first cell (cell 0) of that burst. As the multiplexing order of the multiplexer (for fixed size cells) is in the descending order of the input link number and in the ascending order of time, cell 0 should be routed to the input with the smallest virtual waiting time and the largest input link number. From (5.70), we know it should be routed to input $N - k_0 - 1$ with $k_0 = V(\tau_m^-) \bmod N$. Moreover, the delay for cell 0 is simply the virtual waiting time of input $N - k_0 - 1$, i.e., $\left\lfloor \frac{V(\tau_m^-)}{N} \right\rfloor$. By so doing, the inequalities in (5.67) are satisfied after cell 0 is scheduled. As the total virtual waiting time is

increased by 1 after cell 0 is scheduled, cell 1 should be scheduled at input $N-k_1-1$ with $k_1=(V(\tau_m^-)+1) \bmod N$. The first available time slot at input $N-k_1-1$ is $\tau_m+\left\lfloor \frac{V(\tau_m^-)+1}{N} \right\rfloor$. As cell 1 arrives at time τ_m+1, the delay for cell 1 would be $\left\lfloor \frac{V(\tau_m^-)+1}{N} \right\rfloor - 1$ if cell 1 is scheduled at input $N-k_1-1$. If $\left\lfloor \frac{V(\tau_m^-)+1}{N} \right\rfloor - 1 \geq 0$, then the causality constraint is satisfied and cell 1 can be scheduled this way. We may continue the process to schedule the other cells in the burst until the causality constraint is violated. In general, cell ℓ should be scheduled at input $N-k_\ell-1$ with $k_\ell=(V(\tau_m^-)+\ell) \bmod N$ (as the total virtual waiting time has been increased by ℓ after scheduling the first ℓ cells). As cell ℓ arrives at time $\tau_m+\ell$, the delay for cell ℓ would be $\left\lfloor \frac{V(\tau_m^-)+\ell}{N} \right\rfloor - \ell$ if cell ℓ is scheduled at input $N-k_\ell-1$. If $\left\lfloor \frac{V(\tau_m^-)+\ell}{N} \right\rfloor - \ell \geq 0$, then the causality constraint is satisfied and cell ℓ can be scheduled this way. If $\left\lfloor \frac{V(\tau_m^-)+\ell}{N} \right\rfloor - \ell < 0$, then the causality constraint is violated. We have to schedule cell ℓ at its arrival time. In order to satisfy the contiguity constraint, cell ℓ is scheduled at input $N-1$, the highest priority input at its arrival time. It is easy to see that if cell ℓ is the first cell such that $\left\lfloor \frac{V(\tau_m^-)+\ell}{N} \right\rfloor - \ell < 0$, then all the subsequent cells in the same burst also satisfy the same inequality. As such, all the subsequent cells have to be scheduled at their arrival times at input $N-1$. To illustrate this, we show in Figure 5.35 how a burst of length 8 is scheduled. Cells 0,1,2,3,4 and 5 can be scheduled without violating the causality constraint. Cells 6 and 7 have to be scheduled at their arrival times.

To summarize, we let

$$I_0(m,\ell) = N-1-((V(\tau_m^-)+\ell) \bmod N). \tag{5.71}$$

Cell ℓ of the $m+1^{th}$ burst is routed to the $I_0(m,\ell)^{th}$ input of the multiplexer for fixed size cells if $\left\lfloor \frac{V(\tau_m^-)+\ell}{N} \right\rfloor \geq \ell$. In this case, its delay is $\left\lfloor \frac{V(\tau_m^-)+\ell}{N} \right\rfloor - \ell$. On the other hand, if $\left\lfloor \frac{V(\tau_m^-)+\ell}{N} \right\rfloor < \ell$, cell ℓ of the $m+1^{th}$ burst is routed to input $N-1$ of the multiplexer for fixed size cells. In this case, its delay is zero. This is formalized in the following algorithm.

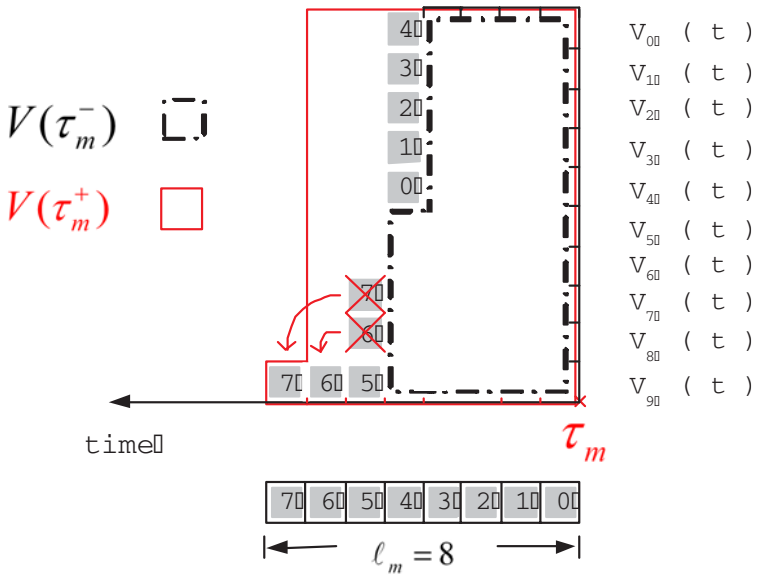


Fig. 5.35. An illustrating example of the cell scheduling algorithm

Algorithm 5.5.1. (Cell scheduling algorithm) Let $I(m,\ell)$ and $D(m,\ell)$ be the routed input (of the multiplexer for fixed size cells) and the delay of cell ℓ of the $m+1^{th}$ burst, $\ell = 0,1,\ldots,\ell_m - 1$, and $m = 0,1,2,\ldots$. Then

$$I(m,\ell) = \begin{cases} I_0(m,\ell) & \text{if } \left\lfloor \frac{V(\tau_m^-)+\ell}{N} \right\rfloor \geq \ell \\ N-1 & \text{otherwise} \end{cases}, \tag{5.72}$$

and

$$D(m,\ell) = \left(\left\lfloor \frac{V(\tau_m^-)+\ell}{N} \right\rfloor - \ell \right)^+. \tag{5.73}$$

Now we discuss how we update the total virtual waiting time. Let $V(\tau_m^+)$ be the total virtual waiting time immediately after the cells in the $m+1^{th}$ burst are scheduled. As described in our cell scheduling algorithm, there are two cases that need to be considered. The first case is that all the cells in that burst are scheduled using the rule specified by $I_0(m,\ell)$. In this case, after the last cell in the burst, i.e., cell $\ell_m - 1$, is scheduled, we have

$$V(\tau_m^+) = V(\tau_m^-) + \ell_m, \tag{5.74}$$

and all the inequalities in (5.67) are still satisfied. The second case is that there exists a cell that does not follow the rule specified by $I_0(m,\ell)$. When this happens, cell $\ell_m - 1$ is scheduled at input $N-1$ at time $\tau_m + \ell_m - 1$. In order to satisfy the contiguity constraint, no cells (from other bursts) can be scheduled before $\tau_m + \ell_m - 1$. Thus, the first available time slot for input j, $j = 0, 1, \ldots, N-2$, is $\tau_m + \ell_m - 1$ and the first available time slot for input $N-1$ is $\tau_m + \ell_m$ (see Figure 5.35 for an illustrating example). Clearly, the inequalities in (5.67) are still satisfied and we have

$$V(\tau_m^+) = (N-1)(\ell_m - 1) + \ell_m = N\ell_m - N + 1. \tag{5.75}$$

These two cases can be combined as follows:

$$V(\tau_m^+) = \max[V(\tau_m^-) + \ell_m, N\ell_m - N + 1]. \tag{5.76}$$

To see this, note that the condition for the first case is equivalent to that cell $\ell_m - 1$ is routed to the $I_0(m, \ell_m - 1)^{th}$ input, i.e.,

$$\left\lfloor \frac{V(\tau_m^-) + \ell_m - 1}{N} \right\rfloor \geq \ell_m - 1. \tag{5.77}$$

As $\ell_m - 1$ is an integer, this is equivalent to

$$\frac{V(\tau_m^-) + \ell_m - 1}{N} \geq \ell_m - 1. \tag{5.78}$$

Thus, in the first case, we have from (5.74) that

$$\begin{aligned} V(\tau_m^+) &= V(\tau_m^-) + \ell_m \\ &= \max[V(\tau_m^-) + \ell_m, N\ell_m - N + 1]. \end{aligned} \tag{5.79}$$

On the other hand, the inequality in (5.78) is reversed in the second case and we have from (5.75) that

$$\begin{aligned} V(\tau_m^+) &= N\ell_m - N + 1 \\ &= \max[V(\tau_m^-) + \ell_m, N\ell_m - N + 1]. \end{aligned} \tag{5.80}$$

Now we describe how we update the total virtual waiting time between two successive bursts. Suppose that the $m + 2^{th}$ burst arrives at τ_{m+1}. There are two cases that need to be considered:
Case 1. $V(\tau_m^+) - N(\tau_{m+1} - \tau_m) > 0$: in this case, we have from (5.70) that $V_{N-1}(\tau_m^+) > \tau_{m+1} - \tau_m$. Note from (5.67) that $V_j(\tau_m^+) \geq V_{N-1}(\tau_m^+) - 1$ for all j. Thus, we have $\tau_m + V_j(\tau_m^+) - \tau_{m+1} \geq 0$ for all $j = 0, 1 \ldots, N-1$. Since the next available time slot of input j is $\tau_m + V_j(\tau_m^+)$, we then have

$$V_j(\tau_{m+1}^-) = \tau_m + V_j(\tau_m^+) - \tau_{m+1}.$$

Summing up over j yields

$$V(\tau_{m+1}^-) = V(\tau_m^+) - N(\tau_{m+1} - \tau_m).$$

Case 2. $V(\tau_m^+) - N(\tau_{m+1} - \tau_m) \leq 0$: in this case, we have from (5.70) that $V_{N-1}(\tau_m^+) \leq \tau_{m+1} - \tau_m$. Note from (5.67) that $V_j(\tau_m^+) \leq V_{N-1}(\tau_m^+)$ for all j. Thus, we have $\tau_m + V_j(\tau_m^+) - \tau_{m+1} \leq 0$ for all $j = 0, 1 \ldots, N-1$. As the next available time slot of input j is $\tau_m + V_j(\tau_m^+) \leq \tau_{m+1}$, we then have $V_j(\tau_{m+1}^-) = 0$ for all $j = 0, 1 \ldots, N-1$. Thus, $V(\tau_{m+1}^-) = 0$.

From these two cases, we then have $V(\tau_{m+1}^-) = (V(\tau_m^+) - N(\tau_{m+1} - \tau_m))^+$. In the following, we summarize the algorithms for updating the total virtual waiting time.

Algorithm 5.5.2. (Algorithms for updating the total virtual waiting time)

(i) The total virtual waiting time after a burst is scheduled is updated as follows:

$$V(\tau_m^+) = \max[V(\tau_m^-) + \ell_m, N\ell_m - N + 1]. \tag{5.81}$$

(ii) The total virtual waiting time between two successive bursts is updated as follows:

$$V(\tau_{m+1}^-) = (V(\tau_m^+) - N(\tau_{m+1} - \tau_m))^+. \tag{5.82}$$

We illustrate how we use the cell scheduling algorithm in the following example.

Example 5.5.3. In Figure 5.36, we consider multiplexing variable length bursts over 3 links, i.e., $N = 3$. As shown in Figure 5.36, there are two bursts coming at time 1 and four bursts coming respectively at time $2, 5, 10, 11$. To break the tie, we choose the burst with length 5 at time 1 to be the first burst. As shown in the figure, now we have $(\tau_0, \ell_0) = (1, 5)$, $(\tau_1, \ell_1) = (1, 3)$, $(\tau_2, \ell_2) = (2, 5)$, $(\tau_3, \ell_3) = (5, 6)$, $(\tau_4, \ell_4) = (10, 3)$, and $(\tau_5, \ell_5) = (11, 3)$. Assume that the buffer in the multiplexer is empty at time 1 and the buffer size B is so large that no cells of the six bursts are lost. Initially, set $V_0(1) = V_1(1) = V_2(1) = 0$. Thus, $V(\tau_0^-) = V(1) = 0$ and the cells in the first burst are all scheduled at their arrival times at input 2. Using the algorithms for updating the total virtual waiting time, we then have $V(\tau_0^+) = 13$ and

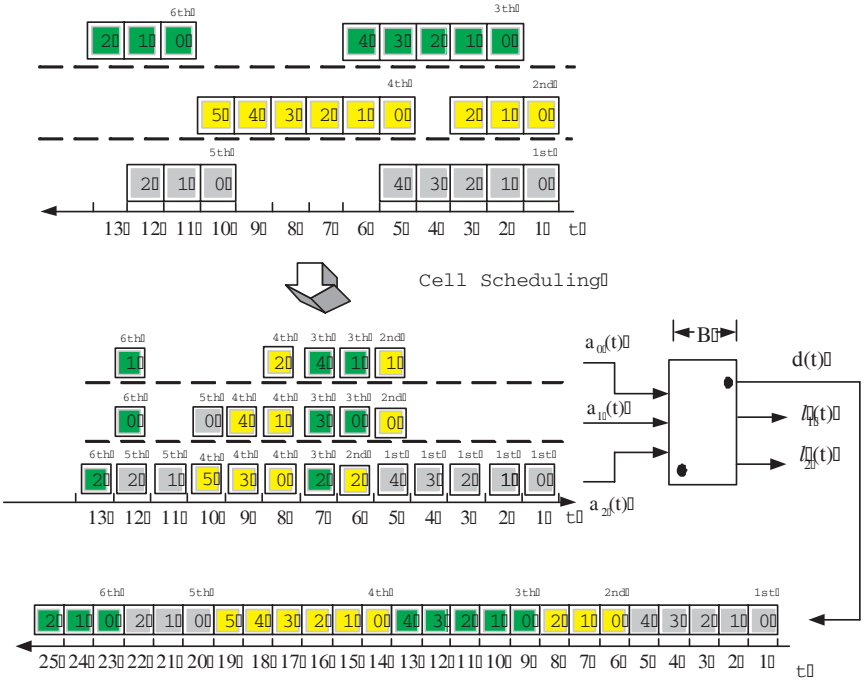


Fig. 5.36. An illustrating example for N=3

$V(\tau_1^-) = V(\tau_0^+) = 13$. According to the cell scheduling algorithm, cell 0 of the second burst is scheduled at input 1 at 5, cell 1 of the second burst is scheduled at input 0 at 5, and cell 2 of the second burst is scheduled at input 2 at 6. Thus, $V(\tau_1^+) = 16$ and $V(\tau_2^-) = 13$. All the cells in the third burst are scheduled using the rule by $I_0(m, \ell)$. As a result, $V(\tau_2^+) = 18$ and $V(\tau_3^-) = 9$. Note that the last cell (cell 5) of the fourth burst does not follow the $I_0(m, \ell)$ rule and it is scheduled at input 2 at its arrival time. Thus, $V(\tau_3^+) = 16$ and $V(\tau_4^-) = 1$. Cell 0 and cell 1 of the fifth burst still follow the $I_0(m, \ell)$ rule. However, cell 2 of the fifth burst does not follow the same rule and it is scheduled at its arrival time at input 2. Thus, $V(\tau_4^+) = 7$ and $V(\tau_5^-) = 4$. The cells in the last burst all follow the $I_0(m, \ell)$ rule.

5.5.4 Delay bound

We have introduced the architecture and the associated cell scheduling algorithm for the variable length burst multiplexer with N inputs. In this section, we will further show that both the total virtual waiting time and the delay through the cell scheduling block are bounded by constants that only depend on the number of inputs and the maximum number of cells in a burst.

Theorem 5.5.4. *Let $\ell_{max} = \sup_{m \geq 0} \ell_m$ be the maximum number of cells in a burst.*

(i) The total virtual waiting time is bounded by $(2N-1)\ell_{\max} - N + 1$ for all t, , i.e.,

$$V(t) \leq (2N-1)\ell_{\max} - N + 1. \tag{5.83}$$

(ii) The delay for every cell through the cell scheduling block is bounded by $\left\lfloor \frac{(2N-2)\ell_{\max} - N + 1}{N} \right\rfloor$, i.e., for all $m = 0, 1, 2, \ldots$, $\ell = 0, 1, \ldots, \ell_m - 1$,

$$D(m, \ell) \leq \left\lfloor \frac{(2N-2)\ell_{\max} - N + 1}{N} \right\rfloor. \tag{5.84}$$

Note that the bounds in Theorem 5.5.4 can be achieved in the worst case. We now construct a worst case to achieve these bounds. To see this, consider the scenario that there are N bursts arriving at time 0 at the N inputs of the variable length burst multiplexer. The burst

lengths of these N bursts are all ℓ_{max}. Thus, we have $\tau_m = 0$, $\ell_m = \ell_{max}$, for $m = 0, 1, \ldots, N-1$. Under our cell scheduling algorithm, we then have

$$V(\tau_0^-) = V(0) = 0,$$
$$V(\tau_m^+) = V(\tau_{m+1}^-) = N(\ell_{\max} - 1) + 1 + m\ell_{\max},$$
$$m = 0, 1, \ldots, N-2,$$

and

$$V(\tau_{N-1}^+) = N(\ell_{\max} - 1) + 1 + (N-1)\ell_{\max}.$$

Thus, $V(\tau_{N-1}^+)$ in this scenario achieves the upper bound in (5.83). Moreover, the delay of cell 0 of the N^{th} burst is $\lfloor \frac{V(\tau_{N-1}^-)}{N} \rfloor$, which is exactly the maximum delay in (5.84).

One important consequence of Theorem 5.5.4 is that the number of the delay lines, M, used in each switch at the second stage of the cell scheduling block is bounded by $\left\lfloor \frac{(2N-2)\ell_{\max} - N + 1}{N} \right\rfloor + 1$. A simple choice is $M = 2\ell_{\max}$. Such a choice is independent of the number of inputs N.

We need the following two lemmas to prove Theorem 5.5.4. In Lemma 5.5.5, we expand the recursive equations in (5.81) and (5.82) to derive closed form expressions of $V(\tau_m^+)$ and $V(\tau_m^-)$. In Lemma 5.5.6, we establish a bound for the multiplexed traffic of N inputs. This bound will be used for the proof of Theorem 5.5.4.

Lemma 5.5.5. *Let $L(n) = \sum_{m=0}^{n-1} \ell_m$ be the total number of cells in the first n bursts. Suppose $V(\tau_0^-) = 0$.*

Then

$$V(\tau_m^+) = \max_{0 \le n \le m} \big(N\ell_n - N + 1 - N(\tau_m - \tau_n) + L(m+1) - L(n+1) \big), \tag{5.85}$$

and

$$V(\tau_m^-) = \max_{0 \le n \le m-1} \big((N-1)\ell_n - N + 1 - N(\tau_m - \tau_n) + L(m) - L(n)\big)^+. \tag{5.86}$$

Proof. We prove (5.85) by induction. Since we assume that $V(\tau_0^-) = 0$, we have from (5.81) that

$$\begin{aligned} V(\tau_0^+) &= \max(\, V(\tau_0^-) + \ell_0, N\ell_0 - N + 1) \\ &= N\ell_0 - N + 1 \end{aligned}$$

and (5.85) is satisfied trivially for $m = 0$.

Now suppose that (5.85) holds for some $m \geq 0$ as the induction hypothesis. It follows from (5.81) that

$$\begin{aligned} & V(\tau_{m+1}^+) \\ & = \max(N\ell_{m+1} - N + 1, (V(\tau_m^+) - N(\tau_{m+1} - \tau_m))^+ + \ell_{m+1}) \\ & = \max(N\ell_{m+1} - N + 1, V(\tau_m^+) - N(\tau_{m+1} - \tau_m) + \ell_{m+1}, \ell_{m+1}) \\ & = \max(N\ell_{m+1} - N + 1, V(\tau_m^+) - N(\tau_{m+1} - \tau_m) + \ell_{m+1}), \end{aligned} \tag{5.87}$$

where we use the fact that $N\ell_{m+1} - N + 1 \geq \ell_{m+1}$. From the induction hypothesis, we then have

$$\begin{aligned} & V(\tau_m^+) - N(\tau_{m+1} - \tau_m) + \ell_{m+1} \\ & = \max_{0 \leq n \leq m} \Big(N\ell_n - N + 1 - N(\tau_m - \tau_n) + \\ & \qquad L(m+1) - L(n+1) \Big) - N(\tau_{m+1} - \tau_m) + \ell_{m+1} \\ & = \max_{0 \leq n \leq m} \Big(N\ell_n - N + 1 - N(\tau_{m+1} - \tau_n) + \\ & \qquad L(m+2) - L(n+1)) \Big) \end{aligned} \tag{5.88}$$

Replacing (5.88) in (5.87) yields

$$\begin{aligned} V(\tau_{m+1}^+) &= \max_{0 \leq n \leq m+1} (N\ell_n - N + 1 - N(\tau_{m+1} - \tau_n) \\ &\qquad + L(m+2) - L(n+1)). \end{aligned}$$

This completes the inductive argument for (5.85).

Since $V(\tau_m^-) = (\, V(\tau_{m-1}^+) - N(\tau_m - \tau_{m-1}) \,)^+$ in (5.82), using (5.85) yields

$$\begin{aligned} & V(\tau_m^-) \\ = \; & (\max_{0 \leq n \leq m-1} (N\ell_n - N + 1 - N(\tau_{m-1} - \tau_n) \\ & + L(m) - L(n+1) \,) - N(\tau_m - \tau_{m-1}) \,)^+ \\ = \; & \max_{0 \leq n \leq m-1} (N\ell_n - N + 1 - N(\tau_m - \tau_n) \end{aligned}$$

$$
\begin{aligned}
& +L(m) - L(n+1))^+ \\
= & \max_{0 \le n \le m-1} ((N-1)\ell_n - N + 1 \\
& -N(\tau_m - \tau_n) + L(m) - (L(n+1) - \ell_n))^+ \\
= & \max_{0 \le n \le m-1} ((N-1)\ell_n - N + 1 \\
& -N(\tau_m - \tau_n) + L(m) - L(n))^+.
\end{aligned}
$$

■

Lemma 5.5.6. *For all $n \le m$,*

$$L(m) - L(n) - N(\tau_m - \tau_n) \le (N-1)\ell_{\max}. \tag{5.89}$$

Proof. Let $\tau_k(n)$ and $\ell_k(n)$, $k = 0, 1, \ldots, N-1$, $n = 0, 1, 2, \ldots$, be the arrival time and the burst length of the $n+1^{th}$ burst at the k^{th} input of the multiplexer with variable length bursts. Also, let $L_k(n) = \sum_{m=0}^{n-1} \ell_k(m)$ be the total number of cells in the first n bursts from the k^{th} input.

Without loss of generality, suppose that in the first n (resp. m) bursts, there are n_k (resp. m_k) bursts from the k^{th} input, $k = 0, 1, \ldots, N-1$. Moreover, suppose that the $m+1^{th}$ burst is from the j^{th} input for some j. Thus, we have

$$L(m) - L(n) = \sum_{k=0}^{N-1} (L_k(m_k) - L_k(n_k)). \tag{5.90}$$

Note that $L_k(m_k) - L_k(n_k) = \sum_{i=n_k}^{m_k-1} \ell_k(i)$ is the total number of cells from the n_k+1^{th} burst to the ${m_k}^{th}$ burst at input k. These cells must arrive during the time interval $[\tau_k(n_k), \tau_k(m_k - 1) + \ell_{\max} - 1]$. Since there is at most one cell arrival within a time slot, we then have

$$L_k(m_k) - L_k(n_k) \le \tau_k(m_k - 1) + \ell_{\max} - \tau_k(n_k). \tag{5.91}$$

As there are exactly n_k bursts from the k^{th} input in the first n bursts, the arrival time of the n_k+1^{th} burst from the k^{th} input cannot be earlier than the arrival time of the $n+1^{th}$ burst, i.e.,

$$\tau_k(n_k) \ge \tau_n. \tag{5.92}$$

Similarly, as there are exactly m_k bursts from the k^{th} input in the first m bursts, the arrival time of the m_k^{th} burst from the k^{th} input cannot be later than the arrival time of the $m+1^{th}$ burst, i.e.,

$$\tau_k(m_k - 1) \leq \tau_m. \tag{5.93}$$

Using (5.92) and (5.93) in (5.91) yields

$$L_k(m_k) - L_k(n_k) \leq \tau_m - \tau_n + \ell_{\max}. \tag{5.94}$$

Now we refine the bound in (5.94) for the j^{th} input. As the $m+1^{th}$ burst is from the j^{th} input, the cells in $L_j(m_j) - L_j(n_j)$ must arrive during the time interval $[\tau_j(n_j), \tau_m - 1]$. Thus,

$$L_j(m_j) - L_j(n_j) \leq \tau_m - \tau_j(n_j) \leq \tau_m - \tau_n. \tag{5.95}$$

It then follows from (5.90), (5.94) and (5.95) that

$$\begin{aligned} L(m) - L(n) &= L_j(m_j) - L_j(n_j) \\ &\quad + \sum_{k \neq j} (L_k(m_k) - L_k(n_k)) \\ &\leq N(\tau_m - \tau_n) + (N-1)\ell_{\max}. \end{aligned}$$

■

Proof of Theorem 5.5.4. (i) Note that $V(t)$ is decreasing between the interarrival time of two successive bursts and that $V(\tau_m^-) \leq V(\tau_m^+)$ for all m. Thus,

$$V(t) \leq \sup_{m \geq 0} V(\tau_m^+).$$

It suffices to show that $V(\tau_m^+) \leq (2N-1)\ell_{\max} - N + 1$ for all $m = 0, 1, 2, \ldots$. Note from (5.85) in Lemma 5.5.5 and (5.89) in Lemma 5.5.6 that

$$\begin{aligned} & V(\tau_m^+) \\ = & \max_{0 \leq n \leq m} (N\ell_n - N + 1 - N(\tau_m - \tau_n) \\ & \quad + L(m+1) - L(n+1)) \\ = & \max_{0 \leq n \leq m} ((N-1)\ell_n - N + 1 - N(\tau_m - \tau_n) \\ & \quad + L(m) + \ell_m - (L(n+1) - \ell_n)) \\ \leq & \max_{0 \leq n \leq m} ((N-1)\ell_n - N + 1 - N(\tau_m - \tau_n) \\ & \quad + L(m) - L(n) + \ell_{\max}) \\ \leq & \max_{0 \leq n \leq m} ((N-1)\ell_n - N + 1 + N\ell_{\max}) \\ \leq & (2N-1)\ell_{\max} - N + 1. \end{aligned}$$

(ii) Note from (5.86) in Lemma 5.5.5 and (5.89) in Lemma 5.5.6 that

$$\begin{aligned} & V(\tau_m^-) \\ = & \max_{0\le n\le m-1}((N-1)\ell_n - N + 1 - N(\tau_m - \tau_n) \\ & +L(m) - L(n))^+ \\ \le & \max_{0\le n\le m-1}((N-1)\ell_n - N + 1 + (N-1)\ell_{\max}) \\ \le & (2N-2)\ell_{\max} - N + 1 \end{aligned}$$

Thus, we have from (5.73) that

$$\begin{aligned} D(m,\ell) &= \left(\left\lfloor \frac{V(\tau_m^-)+\ell}{N}\right\rfloor - \ell\right)^+ \\ &\le \left\lfloor \frac{V(\tau_m^-)}{N}\right\rfloor \\ &= \left\lfloor \frac{(2N-2)\ell_{\max} - N + 1}{N}\right\rfloor. \end{aligned}$$

5.6 FIFO queues

FIFO queues are widely used in every one's daily life. A customer arriving at a FIFO queue joins the tail of the queue. When a customer departs at the head of the queue, every one in a FIFO queue moves up one position. If the buffer of a FIFO queue is finite, then an arriving customer to a full queue is lost. The concept of a discrete-time FIFO queue is formalized in the following definition.

Fig. 5.37. A FIFO queue with buffer B.

Definition 5.6.1. (FIFO queue) *A FIFO queue with buffer B is a network element that has one input link, one control input and two output links (see Figure 5.37). One output link is for departing packets and the other is for lost packets. As shown in Figure 5.37, let $a(t)$ be the state of the input link, $c(t)$ be the state of the control input, $d(t)$ (resp. $\ell(t)$) be state of the output link for departing (resp. lost) packets, and $q(t)$ be the number of packets queued at the FIFO queue at time t (at the end of the t^{th} time slot). Then the FIFO queue with buffer B satisfies the following four properties:*

(P1) Flow conservation: arriving packets from the input link are either stored in the buffer or transmitted through the two output links, i.e.,

$$q(t) = q(t-1) + a(t) - d(t) - \ell(t). \tag{5.96}$$

(P2) Non-idling: if the control input is enabled, i.e., $c(t) = 1$, then there is always a departing packet if there are packets in the buffer or there is an arriving packet, i.e.,

$$d(t) = \begin{cases} 1 & \text{if } c(t) = 1 \text{ and } q(t-1) + a(t) > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{5.97}$$

(P3) Maximum buffer usage: if the control input is not enabled, i.e., $c(t) = 0$, then an arriving packet is lost only when buffer is full, i.e.,

$$\ell(t) = \begin{cases} 1 & \text{if } c(t) = 0,\ q(t-1) = B \text{ and } a(t) = 1 \\ 0 & \text{otherwise} \end{cases}. \tag{5.98}$$

(P4) FIFO: packets depart in the first in first out (FIFO) order.

Analogous to the four properties for a 2-to-1 FIFO multiplexer, one also has from (P1) and (P2) that

$$q(t) = (q(t-1) + a(t) - c(t))^+ - \ell(t).$$

In conjunction with (P3), one further has the following Lindley equation

$$q(t) = \min[(q(t-1) + a(t) - c(t))^+, B]. \tag{5.99}$$

The key difference between the governing equation for a 2-to-1 FIFO multiplexer in (5.4) and that for a FIFO queue in (5.99) is that the delay of a packet in a 2-to-1 FIFO multiplexer can be immediately determined upon its arrival. This is not possible in a FIFO queue as

the delay of an arriving packet depends on the future of the control input $c(t)$. We note that the control input $c(t)$ is also known as the time varying capacity of the discrete-time FIFO queue in the literature (see e.g., [27] and references therein).

5.6.1 A naive construction of FIFO queues with optical memory cells

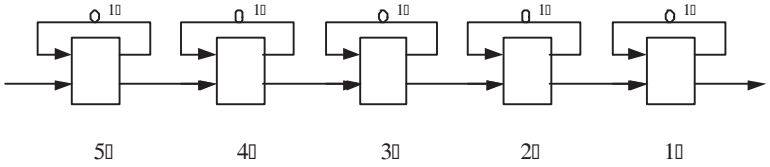


Fig. 5.38. A naive construction of FIFO queues with buffer 5 via optical memory cells

One can easily construct an optical FIFO queue via optical memory cells. As shown in Figure 5.38, it is a concatenation of 5 optical memory cells. To see that such a construction is indeed a FIFO queue with buffer 5, we number the memory cells from 1 to 5 (as shown in Figure 5.38). As each memory cell is capable of storing one packet, these five memory cells corresponding to the 5 positions in the FIFO queue. Initially, the queue is empty. When a packet arrives at an empty queue at time t, the packet leaves the queue during the same time slot if $c(t) = 1$. This requires setting up all the 5 2×2 switches to the "bar" state. On the other hand, if $c(t) = 0$, the arriving packet is stored in the first memory cell. To do so, we set the first 2×2 switch to the "cross" state and the rest of four switches to the "bar" state.

In general, one can construct an optical FIFO queue with buffer B via a concatenation of B optical memory cells. If $c(t) = 0$ and $a(t) = 0$, then all switches are set to the "bar" state so that the state of the queue remains unchanged. If $c(t) = 0$ and $a(t) = 1$, switches 1 to $q(t-1)$ and switches $q(t-1)+2$ to B are set to the "bar" state. Switch $q(t-1)+1$ is set to the "cross" state. The arrival joins the end of the queue. If $c(t) = 1$, switches 1 to $q(t-1)$ are set to the "cross" state and the others are set to the "bar" state. By so doing, every one in the queue moves up one position.

5.6.2 The main idea of the construction

In this section, we show that one might use the time interleaving property to construct a FIFO queue with less complexity than the naive construction in Section 5.6.1. Analogous to the design of the multiplexer in [37], the main idea is based on the following well known queueing result. Consider a system with parallel FIFO queues as shown in Figure 2.42. Suppose that we operate the system as follows: a customer that arrives at the system joins the shortest queue (the queue with the least number of customers), and the server after completing the service of a customer always chooses a customer from the longest queue (the queue with the largest number of customers). By so doing, these parallel FIFO queues are kept in the most balanced state, i.e., at any time the difference between the number of customers in the longest queue and the number of customers in the shortest queue is at most 1. If the service time of all the customers are all identical, then the serving-the-longest-queue policy and the joining-the-shortest-queue policy are simply the **round robin policy**. Moreover, as the system always serves the longest queue, the customer with the longest waiting time is served. Thus, the system with parallel FIFO queues behaves as if it were a single FIFO queue. We note that such an idea was previously used in [170, 93] to construct the multiplexers in the Knockout switches.

In Figure 5.39, we show a direct implementation of a FIFO queue with buffer 20 via four parallel FIFO queues with buffer 5 in Figure 5.38. The first switch is a 1-to-4 demultiplexer (a 1×4 switch) that is used for the joining-the-shortest-queue policy. On the other hand, the last switch is a 4-to-1 multiplexer (a 4×1 switch) that is used for the serving-the-longest-queue policy. Clearly, such a direct implementation of a FIFO queue with buffer BK requires K parallel FIFO queues with buffer B and the total number of memory cells is still $O(BK)$.

As in [37], the trick to reduce the complexity of constructing a FIFO queue is to use the time interleaving property for SDL elements. As shown in Figure 5.40, the delay line in every memory cell is increased by four times. As such, it can be operated as time interleaving of four FIFO queues. Specifically, time slots $1, 5, 9, 13, \ldots$ are for the 1st queue, time slots $2, 6, 10, 14, \ldots$, are for the 2nd queue, time slots $3, 7, 11, 15, \ldots$, are for the 3rd queue, and time slots $4, 8, 12, 16, \ldots$, are for the 4th queue.

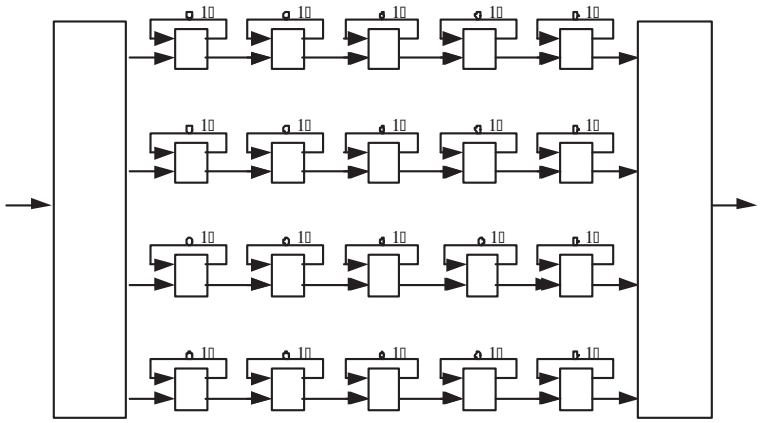


Fig. 5.39. A direct implementation of a FIFO queue with parallel FIFO queues

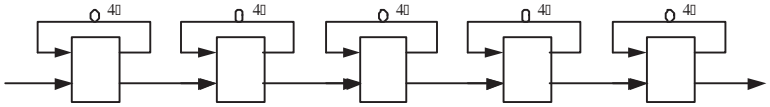


Fig. 5.40. A scaled FIFO queue that can be operated as four parallel FIFO queues

Unfortunately, the construction in Figure 5.40 is not yet a single FIFO queue. This is because there may not be a departure in every time slot in a FIFO queue. Even though the serving-the-longest-queue policy is equivalent to the round-robin policy, it is *not periodic* as the time interleaved queues in Figure 5.40. In order to retrieve the packet in the longest queue, a modification of the first scaled memory cell in Figure 5.40 is needed. In Figure 5.41, the scaled memory cell (with the scaling factor 4) is modified by inserting a 1-to-2 demultiplexer after each one unit of delay. By so doing, one can retrieve any packet from any one of the four parallel queues. Similarly, we also need to modify the last scaled memory cell so that an arriving packet can be placed in the shortest queue. In Figure 5.42, we modify the scaled memory cell by inserting a 2-to-1 multiplexer after each one unit of delay. By so doing, one can place an arriving packet to any one of the four parallel queues.

5.6.3 Three-stage constructions

The modifications in Figure 5.41 and Figure 5.42 can be further simplified. As the serving-the-longest-queue policy is still equivalent to the

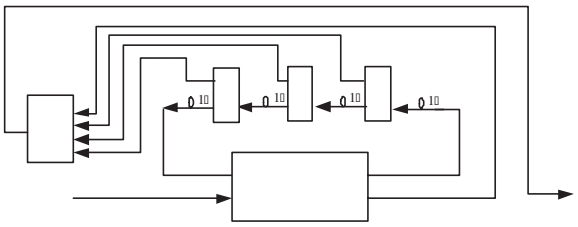


Fig. 5.41. A modification of the right scaled memory cell in Figure 5.40 so that one can retrieve a packet from any one of the four parallel queues.

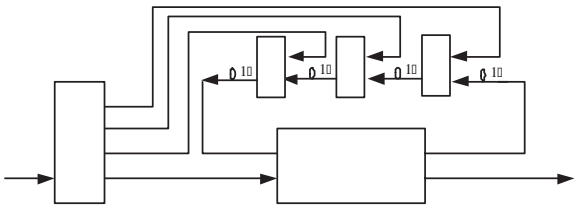


Fig. 5.42. A modification of the left scaled memory cell in Figure 5.40 so that an arriving packet can be placed in any one of the four parallel queues.

round-robin policy, there is no need to design a network element that allows us to retrieve the head-of-line packets *from any of the parallel queues*. Instead, one only needs a network element that allows us to retrieve the head-of-line packets *in the round robin fashion*. For this, we will show that one may simply add a FIFO queue (called the head queue in this section) in front of the parallel queues. Similarly, we will also append a FIFO queue (called the tail queue in the section) after the parallel queues.

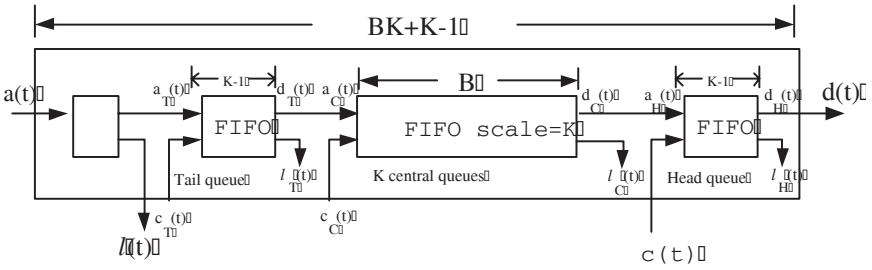


Fig. 5.43. A three-stage construction of a FIFO queue

Specifically, in Figure 5.43, we show a three-stage construction of a FIFO queue with buffer $BK + K - 1$. It is a concatenation of a 1×2

switch, a FIFO queue with buffer $K-1$ (tail queue), a scaled FIFO queue with buffer B and scaling factor K (K parallel central queues), and a FIFO queue with buffer $K-1$ (head queue). The 1×2 switch on the left acts as a 1-to-2 demultiplexer. Its objective is for admission control. An arriving packet is admitted only when the total number of packets inside the network element does not exceed $BK+K-1$ after its admission. Otherwise, an arrival is lost and it is routed to the loss port $\ell(t)$ (as shown in Figure 5.43). By so doing, the maximum number of packets inside the network element is at most $BK+K-1$.

To represent the state of the head queue, we let $a_H(t)$ be the state of its input link, $c_H(t)$ be the state of its control input, $d_H(t)$ be state of its output link for departing packets, $\ell_H(t)$ be state of its output link for lost packets, and $q_H(t)$ be the number of packets queued at the head queue at time t. Similarly, we let $a_T(t)$, $c_T(t)$, $d_T(t)$, $\ell_T(t)$, and $q_T(t)$ denote the corresponding states in the tail queue.

From the time interleaving property for SDL elements, the scaled FIFO queue with buffer B and scaling factor K can be operated as K parallel queues. These K time interleaved parallel queues are connected to the head queue and the tail queue periodically with period K. An illustrating graph is shown in Figure 5.44.

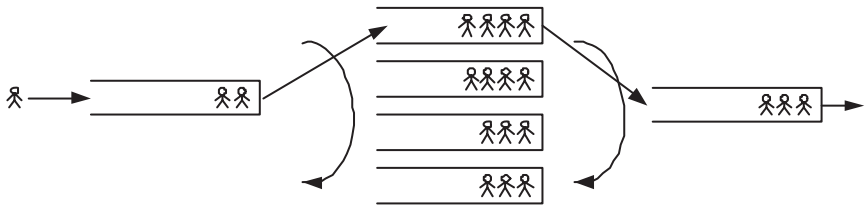


Fig. 5.44. An illustration of periodic connections in the three-stage construction

To simplify our presentation, we let $a_C(t)$ be the state of input link of the central FIFO queue that is connected by the head queue and the tail queue at time t. Also, let $c_C(t)$ be the state of its control input, $d_C(t)$ be state of its output link for departing packets, $\ell_C(t)$ be state of its output link for lost packets, and $q_C(t)$ be the number of packets stored in that central queue. Let $\hat{q}_C(t)$ be the total number of packets stored in the K central queues at time t and

$$q(t) = q_T(t) + \hat{q}_C(t) + q_H(t) \tag{5.100}$$

be the total number of packets stored in the network element. To summarize, a subscript H (resp. T, C) indicates that its is a state variable of the *head* (resp. *tail, connected central*) queue.

From the three-stage construction in Figure 5.43, we have $a_T(t) = a(t)$ (if the arrival is not lost), $d_T(t) = a_C(t)$, $d_C(t) = a_H(t)$, and $d(t) = d_H(t)$.

To operate the three-stage construction in Figure 5.43 as a FIFO queue, we let the control input of the head queue $c_H(t)$ to be the control input of the overall FIFO queue $c(t)$, i.e., $c_H(t) = c(t)$. To complete the operation of the network element, it remains to specify the control input of the tail queue $c_T(t)$ and the control input of the connected central queue $c_C(t)$.

Define a busy period as the period of time that there are packets stored in the central queues, i.e., $\hat{q}_C(t) > 0$. An idle period is the period of time that there are no packets stored in the central queues, i.e., $\hat{q}_C(t) = 0$. Initially, the network element is empty and it is in an idle period.

(R1) (Idle period rule) In an idle period, the tail queue and the central FIFO queues are always enabled as long as the head queue is not full, i.e., $c_T(t) = c_C(t) = 1$ if $\hat{q}_C(t-1) = 0$ and $q_H(t-1) - c(t) < K - 1$.

According to the idle period rule, the tail queue and the scaled FIFO queue are transparent during an idle period and the network element is completely determined by the head queue. Thus, during an idle period, the network element is a FIFO queue with buffer $K - 1$.

(R2) (Initiation of a busy period) When the head queue is full and there is an arriving packet, the packet has to be stored in one of the central queues and this triggers a busy period. Thus, if $\hat{q}_C(t-1) = 0$ and $q_H(t-1) - c(t) = K - 1$, then $c_T(t) = 1$ and $c_C(t) = 0$.

To specify the operation rules in a busy period, we need to keep track of the shortest queue and the longest queue. Let $A(t_1, t_2)$ be the number of arrivals to the central queues between $[t_1, t_2 - 1]$, i.e.,

$$A(t_1, t_2) = \sum_{t=t_1}^{t_2-1} a_C(t). \tag{5.101}$$

Also, let $D(t_1, t_2)$ be the number of departures from the central queues between $[t_1, t_2 - 1]$, i.e.,

$$D(t_1, t_2) = \sum_{t=t_1}^{t_2-1} d_C(t). \tag{5.102}$$

Suppose that a busy period begins at time τ. As the joining-the-shortest-queue policy is simply the round robin assignment of the arriving packets in a busy period, the connected central queue at time t is the shortest queue if and only if

$$(t - \tau - A(\tau, t)) \bmod K = 0.$$

Similarly, the connected central queue at time t is the longest queue if and only if

$$(t - \tau - D(\tau, t)) \bmod K = 0.$$

(R3) (Serving-the-longest-queue rule) In a busy period, there are two conditions that need to be met in order to enable a packet to depart from the connected central queue to the head queue: (i) there is a buffer space in the head queue, and (ii) the central queue being connected is indeed the longest queue. Specifically, suppose that $\hat{q}_C(t-1) > 0$. Then $c_C(t) = 1$ if and only if $q_H(t-1) - c(t) < K-1$ and $(t - \tau - D(\tau, t)) \bmod K = 0$.

(R4) (Joining-the-shortest-queue rule) In a busy period, there are two conditions that need to be met in order to enable a packet to depart from the tail queue to the connected central queue: (i) there is a buffer space in the connected central queue, and (ii) the central queue being connected is indeed the shortest queue. Specifically, suppose that $\hat{q}_C(t-1) > 0$. Then $c_T(t) = 1$ if and only if $q_C(t-1) - c_C(t) < B$ and $(t - \tau - A(\tau, t)) \bmod K = 0$.

In the following theorem, we prove the main result for the three-stage construction of FIFO queues. Its proof is given in Section 5.6.5.

Theorem 5.6.2. *Suppose that the network element in Figure 5.43 is started from an empty system. Under the operation rules specified in (R1)-(R4), it is a FIFO queue with buffer* $BK + K - 1$.

Note that the first 1×2 switch in the three-stage construction in Figure 5.43 is only to make sure that the total number of packets inside the network element does not exceed $BK + K - 1$. In other words, this 1×2 switch can be omitted in the construction as long as the queue never exceeds its buffer size. For this, we call a network element a pre-FIFO queue with buffer B if it behaves exactly the same

as a FIFO queue with buffer B as long as the queue never exceeds its buffer size, i.e., it can realize all the sample paths that do not lead to a buffer overflow. For instance, the naive construction via a concatenation of 5 optical memory cells in Figure 5.38 is indeed a pre-FIFO queue with buffer 5. As there is no internal loss in the three-stage construction in Figure 5.43, the FIFO queues there can be replaced by pre-FIFO queues. As such, one can build a pre-FIFO queue with buffer $BK + K - 1$ by two pre-FIFO queues with buffer $K - 1$ and a scaled pre-FIFO queue with buffer B and scaling factor K. Let $H(K)$ be the number of 2×2 switches needed for constructing a pre-FIFO queue with buffer K. From the three-stage construction, it follows that

$$H(BK + K - 1) = 2H(K - 1) + H(B). \tag{5.103}$$

As an optical memory cell can be used for a pre-FIFO queue with buffer 1, we have $H(1) = 1$. Letting $K = 2$ in (5.103) yields

$$H(2B + 1) = 2 + H(B). \tag{5.104}$$

Solving this yields

$$H(2^n - 1) = 2n - 1. \tag{5.105}$$

In fact, the recursive expansion using $K = 2$ for building a pre-FIFO queue with buffer $2^n - 1$ yields the same implementation for a $2^n \times 2^n$ Benes time slot interchange in Section 5.2.3 (see Figure 5.9 for an implementation). As one can add 1×2 switch in front of a pre-FIFO queue for dropping overflowed packets, a FIFO queue with buffer $2^n - 1$ can be constructed by using $2n$ 2×2 switches with the total fiber length $3 \cdot 2^{n-1} - 2$.

Even though the total number of buffers in the three-stage construction is $BK + 2(K - 1)$, it is not possible to admit more than $BK + K - 1$ packets without violating the properties of a FIFO queue. To see this, suppose that we relax the admission control rule by admitting at most $BK + K$ packets. Consider the following scenario: the head queue and the central queues are all full at time $t - 1$, i.e., $q_H(t - 1) = K - 1$ and $\hat{q}_C(t - 1) = BK$, and the connected central queue at time t is both the shortest queue and the longest queue. Suppose that $c(t) = 0$, $a(t) = 1$ and that $a(s) = c(s) = 1$ for $s = t + 1, \ldots, t + K - 1$. According to the relaxed admission control rule, the packet that arrives at time t is admitted to the tail queue and we have $q_T(t) = 1$, $q_H(t) = K - 1$ and $\hat{q}_C(t) = BK$. As the connected central queue at time t is both the shortest queue and the longest queue, the connected central queue

and the tail queue are not enabled for $s = t+1, \ldots, t+K-1$. Thus, all the subsequent arrivals have to be store at the tail queue. Moreover, the departures from the head queue are not replenished by the packets from the central queues. At time $t+K-2$, we then have $q_H(t+K-2) = 1$, $\hat{q}_C(t+K-2) = BK$ and $q_T(t+K-2) = K-1$. Even though the last packet in the head queue will depart at $t+K-1$, we are not able to admit the arriving packet at $t+K-1$ as the tail queue is full. This violates the maximum buffer usage property for a FIFO queue with buffer $BK+K$.

5.6.4 Extensions of FIFO queues

In this section, we address possible extensions of FIFO queues. First, we show how one constructs a FIFO queue with two inputs from a FIFO queue with a single input. The idea is to add a 2-to-1 FIFO multiplexer (with buffer) in front of a FIFO queue with a single input. As shown in Figure 5.45, we can construct a FIFO queue with two inputs and buffer B via a concatenation of a 2×4 switch, a 2-to-1 multiplexer with buffer B (in Definition 5.3.1) and a FIFO queue with buffer B (in Definition 5.6.1). The objective of the 2×4 switch is for admission control. When the total number of packets inside the network element is B, further arrivals are routed to the loss ports $\ell_0(t)$ and $\ell_1(t)$. By so doing, there are no internal losses inside the network element. The 2-to-1 multiplexer with buffer B then allows packets from the two inputs being multiplexed into the FIFO queue that only has a single input. To see the reason why the 2-to-1 multiplexer is required to have buffer B, consider the worst case sample path that $a_0(t) = a_1(t) = c(t) = 1$ for $t = 1, 2, \ldots, B$. For this particular sample path, the FIFO queue is empty for $t = 1, 2, \ldots, B$ and one half of the arriving packets have to be stored in the 2-to-1 multiplexer.

One can then use a FIFO queue with two inputs to construct a 2-to-1 multiplexer with two classes of priorities (see Figure 5.46). Suppose that there are two classes of packets: high priority packets and low priority packets. High priority packets are not affected by the low priority packets. Low priority packets can be sent out only when there are no high priority packets in the buffered multiplexer. As shown in Figure 5.46, high priority packets that arrive at the two inputs are routed to the upper 2-to-1 multiplexer with buffer B. On the other hand, low priority packets that arrive at the two inputs are routed to the lower FIFO queue with two inputs. The control input of the

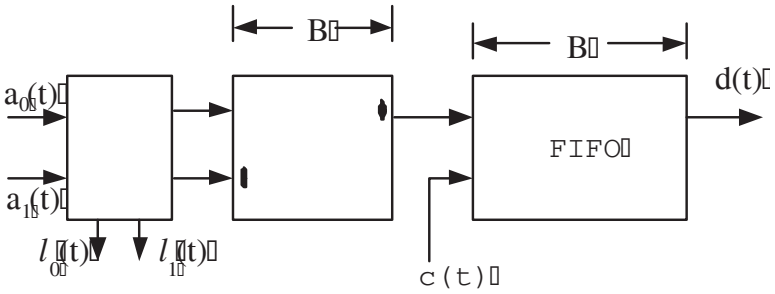


Fig. 5.45. A FIFO queue with two inputs

FIFO queue, i.e., $c(t)$ in Figure 5.46, is enabled only when there are no packets stored in the upper 2-to-1 multiplexer with buffer B.

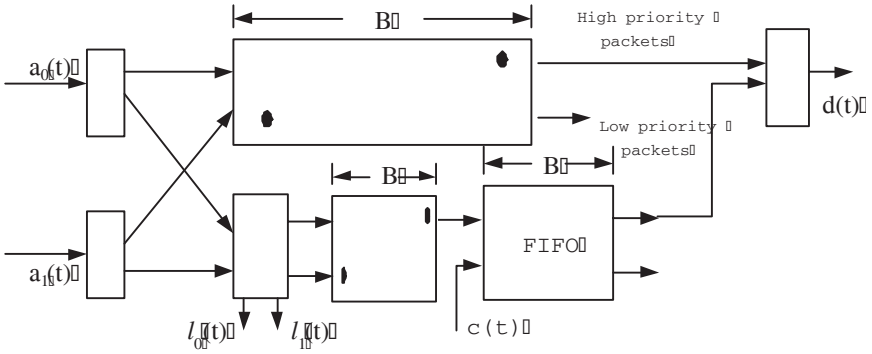


Fig. 5.46. A 2-to-1 multiplexer with two classes of priorities

In view of the extension to 2-to-1 multiplexers with two classes of priorities, one might also extend our results for constructing queueing systems with per flow queueing, such as the Packetized Generalized Processor Sharing (PGPS) in Parekh and Gallager [132] (or the Weighted Fair Queueing (WFQ) in Demers, Keshav, and Shenkar [58]).

5.6.5 Proof of Theorem 5.6.2

In this section, we prove Theorem 5.6.2. To prove the three-stage construction in Figure 5.43 is indeed a FIFO queue with buffer $BK+K-1$, we need to verify the four properties in Definition 5.6.1. From (R1),

the construction is the same as a FIFO queue with buffer $K-1$ in an idle period. It suffices to verify these four properties in a busy period.

(P1) Flow conservation: it is easy to see that under the serving-the-longest-queue rule in (R3), $q_H(t) \leq K-1$ for all t and there is no loss in the head queue, i.e., $\ell_H(t) = 0$. Similarly, under the joining-the-shortest-queue rule in (R4), $q_C(t) \leq B$ for all t and we have $\ell_C(t) = 0$. We will show in (P3) that there is no loss in the tail queue, i.e., $\ell_T(t) = 0$, as long as the total number of packets inside the network element is not greater than $BK + K - 1$. As such, flow conservation can be preserved.

(P2) Non-idling: we will prove this by contradiction. Suppose that a busy period begins at time τ and the non-idling property is violated for the first time at time $t_0 > \tau$. From the non-idling property of the head queue, the head queue must be empty at $t_0 - 1$. Moreover, a packet p in the longest queue (called queue q) cannot be dequeued to the empty head queue as queue q is not connected at t_0. Mathematically, we have (i) the (external) control input is enabled, i.e., $c(t_0) = 1$, (ii) the head queue is empty, i.e., $q_H(t_0 - 1) = 0$, and (iii) the central queues are not empty, i.e., $\hat{q}_C(t_0 - 1) > 0$, and (iv) the connected central queue is not the longest queue, i.e.,

$$(t_0 - \tau - D(\tau, t_0)) \bmod K \neq 0. \tag{5.106}$$

Let t_1 be the time that the last packet is dequeued to the head queue from the central queues. If no packet is dequeued to the head queue, let $t_1 = \tau - 1$. Clearly, $t_1 + 1 \leq t_0$ as it takes at least one time slot to dequeue a packet. Moreover, queue q is the longest queue since $t_1 + 1$. We first claim that the longest queue (queue q) is connected at $t_1 + 1$, i.e.,

$$(t_1 + 1 - \tau - D(\tau, t_1 + 1)) \bmod K = 0. \tag{5.107}$$

As such, $t_1 + 1$ is the *first* time that queue q is connected after the last packet is dequeued to the head queue from the central queues (if there is one). Note that (5.107) holds trivially if no packet is dequeued to the head queue since the beginning of the busy period. On the other hand, according to the serving-the-longest-queue rule in (R3), we have

$$(t_1 - \tau - D(\tau, t_1)) \bmod K = 0. \tag{5.108}$$

As t_1 is the time that the last packet is dequeued to the head queue, we also have $D(\tau, t_1 + 1) = D(\tau, t_1) + 1$. Replacing this in (5.108) yields (5.107).

Secondly, we claim that there exists t_2 with $t_0 - K < t_2 < t_0$ such that queue q is connected at t_2, i.e.,

$$(t_2 - \tau - D(\tau, t_2)) \bmod K = 0. \tag{5.109}$$

As such, t_2 is the *last* time (before t_0) that queue q is connected after the last packet is dequeued to the head queue from the central queues (if there is one). Since there is no packet that is dequeued from the central queues to the head queue from $t_1 + 1$ to t_0, we have

$$D(\tau, t_1 + 1) = D(\tau, t) = D(\tau, t_0) \tag{5.110}$$

for all $t_1 + 1 \le t \le t_0$. Let $m = \lfloor (t_0 - t_1 - 1)/K \rfloor$, where $\lfloor x \rfloor$ is the floor function that returns the largest integer not greater than x. Let

$$t_2 = t_1 + 1 + mK. \tag{5.111}$$

Since $x - 1 < \lfloor x \rfloor \le x$, we have

$$t_0 - K < t_2 \le t_0. \tag{5.112}$$

That (5.109) holds then follows from (5.111), (5.110) and (5.107). Furthermore, as $D(\tau, t_2) = D(\tau, t_0)$, replacing this in (5.109) yields

$$(t_2 - \tau - D(\tau, t_0)) \bmod K = 0.$$

In view of (5.106), we know that $t_2 \neq t_0$. Thus, we have $t_0 - K < t_2 < t_0$.

Now we claim that the head queue must be full at t_2, i.e.,

$$q_H(t_2) = K - 1. \tag{5.113}$$

According to the serving-the-longest-queue rule in (R3), the condition in (5.109) indicates that queue q would be enabled at time t_2 if the other condition $q_H(t_2 - 1) - c(t_2) < K - 1$ were satisfied. As packet p is still in queue q at time t_0, this implies that

$$q_H(t_2 - 1) - c(t_2) = K - 1. \tag{5.114}$$

As $q_H(t) \le K-1$ for all t and $c(t)$ is nonnegative, we have $q_H(t_2-1) = K - 1$ and $c(t_2) = 0$. Thus, the head queue remains unchanged at t_2 and we have $q_H(t_2) = q_H(t_2 - 1) = K - 1$.

Finally we show there is a contradiction to the empty head queue condition $q_H(t_0-1) = 0$. As $t_0-K < t_2 < t_0$, we have $t_0-1-t_2 < K-1$. Since the head queue can be decreased by at most 1 in a time slot, we also have from (5.113) that

$$q_H(t_0 - 1) \ge q_H(t_2) - (t_0 - 1 - t_2) > 0. \tag{5.115}$$

This leads to a contradiction to the empty head queue condition $q_H(t_0 - 1) = 0$ when the non-idling property is violated.

(P3) Maximum buffer usage: once again, we will show by contradiction that there is no loss in the tail queue as long as the total number of packets inside the network element is not greater than $BK + K - 1$, i.e., for all t

$$q_H(t) + \hat{q}_C(t) + q_T(t) \leq BK + K - 1. \tag{5.116}$$

Suppose that the maximum buffer usage property is violated for the first time at time t_0, i.e., a packet arriving at the tail queue is lost at time t_0. When this happens, we have from the maximum buffer usage property of the tail queue that (i) the tail queue must be full, i.e., $q_T(t_0 - 1) = K - 1$ and (ii) the control of the tail queue is not enabled, i.e., $c_T(t_0) = 0$.

Denote by packet p the head-of-line packet of the tail queue at time t_0. Let t_1 be the time that the last packet is dequeued to the central queues from the tail queue. If t_1 is in a busy period, let τ be the beginning of that busy period. Otherwise, let $\tau = t_1$. We first claim that

$$(t_1 + 1 - \tau - A(\tau, t_1 + 1)) \bmod K = 0. \tag{5.117}$$

Note that if $\tau = t_1$, then $A(\tau, t_1+1) = 1$ and (5.117) holds trivially. On the other hand, if τ is the beginning of a busy period, it then follows from the joining-the-shortest-queue rule in (R4) that

$$(t_1 - \tau - A(\tau, t_1)) \bmod K = 0. \tag{5.118}$$

As t_1 is the time that the last packet is dequeued to the central queues from the tail queue, we also have $A(\tau, t_1 + 1) = A(\tau, t_1) + 1$. Replacing this in (5.118) yields (5.117).

Secondly, we claim that there exists t_2 with $t_0 - K < t_2 \leq t_0$ such that

$$(t_2 - \tau - A(\tau, t_2)) \bmod K = 0. \tag{5.119}$$

The argument for (5.119) is similar to that in the proof of the non-idling property. Since there is no packet that is dequeued to the central queues from the tail queue from $t_1 + 1$ to t_0, we have

$$A(\tau, t_1 + 1) = A(\tau, t) = A(\tau, t_0) \tag{5.120}$$

for all $t_1 + 1 \leq t \leq t_0$. Let

$$t_2 = t_1 + 1 + \lfloor (t_0 - t_1 - 1)/K \rfloor K. \tag{5.121}$$

Analogous to the argument for the non-idling property, we have

$$t_0 - K < t_2 \leq t_0. \tag{5.122}$$

That (5.119) holds then follows from (5.121), (5.120) and (5.117).

Now we claim that packet p is in the tail queue at t_2. If packet p has not arrived at the tail queue by t_2, then the tail queue must be empty at t_2, i.e., $q_T(t_2) = 0$, as the last packet departs at $t_1 < t_2$. Since there is at most one packet arrival per time slot, we have from (5.122) that

$$q_T(t_0 - 1) \leq q_T(t_2) + (t_0 - 1 - t_2) < K - 1. \tag{5.123}$$

This leads to a contradiction that $q_T(t_0 - 1) = K - 1$.

Finally we show there is a contradiction to the maximum number of packets inside the network element. In view of (5.119) and the joining-the-shortest-queue rule in (R4), the only reason that packet p did not depart from the tail queue at t_2 is that

$$q_C(t_2 - 1) - c_C(t_2) = B.$$

This implies that the connected central queue is full, i.e., $q_C(t_2-1) = B$ and the control input of the connected central queue is not enabled, i.e., $c_C(t_2) = 0$. As the connected central queue is the shortest queue, all the central queues are full, i.e., $\hat{q}_C(t_2 - 1) = BK$. As τ is the beginning of a busy period, we have

$$A(\tau, t_2) = D(\tau, t_2) + \hat{q}_C(t_2 - 1) = D(\tau, t_2) + BK. \tag{5.124}$$

From (5.119), it then follows that

$$(t_2 - \tau - D(\tau, t_2)) \bmod K = 0. \tag{5.125}$$

Thus, the connected central queue is also the longest queue. According to the serving-the-longest-queue rule in (R3), the condition in (5.125) indicates that the connected central queue would be enabled at time t_2 if the other condition $q_H(t_2 - 1) - c(t_2) < K - 1$ were satisfied. This in turn implies that the head queue is also full, i.e., $q_H(t_2 - 1) = K - 1$ and the head queue is not enabled at t_2, i.e., $c(t_2) = 0$. Since both the head queue and the connected central queue are not enabled at t_2, they remain unchanged at t_2. Thus, we have $q_H(t_2) = K - 1$ and $\hat{q}_C(t_2) = BK$. Adding packet p in the tail queue at t_2, the total number of packets inside the network element at t_2 is at least $BK + K$, which contradicts to (5.116).

(P4) FIFO: since both the tail queue and the head queue are FIFO queues, we only need to consider the order in the central queues. The

FIFO property in the central queues is trivially preserved from the joining-the-shortest-queue rule and the serving-the-longest-queue rule.

5.7 Building optical queues from classical switching theory

We will show how the classical switching theory can be applied to construct optical queues. Intuitively, a sample path of a discrete-time queue with a single input link and a single output link can be viewed as a mapping from the arrivals (at the input link) to the departures (at the output link). To illustrate this, in Figure 5.47 we show a typical sample path of a queue with a single input link and a single output link. The first customer arrives at time $t = 1$ and departs at time $t = 7$, the second customer arrives at time $t = 3$ and departs at time $t = 5$, the third customer arrives at time $t = 5$ and departs at time $t = 9$, and so forth. As shown in Figure 5.47, the queue that realizes this particular sample path can be viewed as a switch that sets up a particular connection pattern between the inputs and the outputs. The difference is that the arrivals at a queue is progressive in time and can be infinite. On the other hand, the inputs of a switch is usually finite and they can be wrapped around (or permuted in any manner).

Despite the difference, switching theory is quite useful in constructing queues considered in this section. There are three types of discrete-time queues in this section: linear compressors, non-overtaking delay lines and flexible delay lines. In the queueing context, a linear compressor is a First In First Out (FIFO) queue with vacations. A non-overtaking delay line is a FIFO queue with known departure times upon arrivals. A flexible delay line is a queue with an infinite number of servers. In the switching context, a linear compressor corresponds to a conditional nonblocking switch that satisfies a certain monotone and consecutive condition. In fact, such a conditional nonblocking switch is also called a linear compressor in switching theory (see e.g., Section 2.5.7 or the book by Li [111]). A non-overtaking delay line also corresponds to a conditional nonblocking switch that satisfies a monotone condition. Such a switch is called a UU (Unimodal-Unimodal) nonblocking switch in [111]. A flexible delay line corresponds to a (strict sense) nonblocking switch.

Our approaches for constructing these queues are inspired by the early works in [118, 141, 90, 140] that map rearrangeable nonblock-

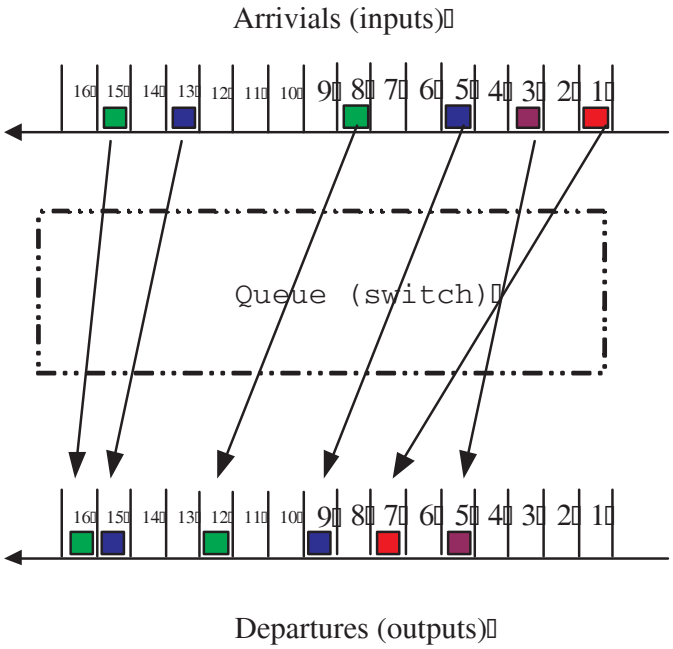


Fig. 5.47. A typical sample path of a discrete-time queue

ing switches to time slot interchanges in Section 5.2. As the two-stage construction for a linear compressor in switching theory, we show that there is also a two-stage construction of a linear compressor in our setting. Moreover, via recursive expansion of the two-stage construction, one can then construct a linear compressor via a banyan type of network. Analogous to the three-stage construction of a UU nonblocking switch, we show that there is a three-stage construction of a non-overtaking delay line. Similarly, there is a three-stage construction of a flexible delay line. Such a result is analogous to the three-stage Clos network for a nonblocking switch. It is known that the Cantor network [21] can be used for constructing a nonblocking switch with less complexity than that by the Clos network. We show that this is also the case by modifying the Cantor network to our setting.

5.7.1 Flexible delay lines, non-overtaking delay lines and Linear compressors

As mentioned earlier, a discrete-time queue with a single input and a single output can also be viewed as a network element that maps

the packets arriving at the input to the output. To be specific, let $\tau^a(n)$ be the arrival time of the n^{th} packet at the input and $\tau^d(n)$ be the departure time of the n^{th} packet at the output. Also, let $\tau^a = \{\tau^a(n), n \geq 1\}$ be the sequence of arrival times and $\tau^d = \{\tau^d(n), n \geq 1\}$ be the sequence of departure times. A discrete-time queue with a single input and a single output then realizes a certain set of mappings (or sample paths in this section) from τ^a to τ^d. For instance, the delay line with delay d in Definition 5.1.1 can be viewed as a queue that realizes the set of mappings with $\tau^d(n) = \tau^a(n) + d$ for all n. In the following, we introduce the concept of flexible delay lines that realize a much larger set of mappings.

Definition 5.7.1. (Flexible delay line) *Suppose that the departure time of a packet is known upon its arrival. A network element with a single input $a(t)$ and a single output $d(t)$ is called a* flexible *delay line with maximum delay d if it realizes the set of mappings (or sample paths) that satisfy*

$$\tau^a(n) \leq \tau^d(n) \leq \tau^a(n) + d, \quad \textit{for all } n, \tag{5.126}$$

and

$$\tau^d(m) \neq \tau^d(n), \quad \textit{for all } m \neq n. \tag{5.127}$$

Furthermore, a flexible delay line is said to be with the range of delay $[d_1, d_2]$ if it satisfies (5.127) and

$$\tau^a(n) + d_1 \leq \tau^d(n) \leq \tau^a(n) + d_2, \quad \textit{for all } n, \tag{5.128}$$

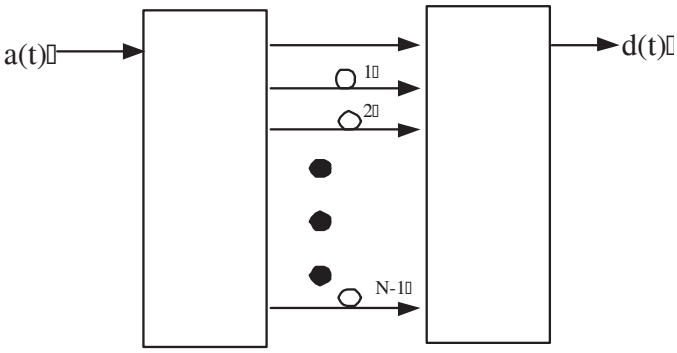


Fig. 5.48. A direct construction of a flexible delay line with maximum delay $N-1$

In Figure 5.48, we show a direct construction of a flexible delay line with maximum delay $N-1$. The first $1 \times N$ switch is a 1-to-N demultiplexer and the second $N \times 1$ switch is an N-to-1 multiplexer. When the n^{th} packet arrives at the input, it will be routed to the fixed delay line with delay $\tau^d(n) - \tau^a(n)$ via the 1-to-N demultiplexer and be multiplexed at the output link via the N-to-1 multiplexer. The cost of such a construction is certainly very high, both in the size of the two switches and the total length of fiber delay lines. Fortunately, for some applications, it may not be necessary to realize such a large set of mappings.

Definition 5.7.2. (Non-overtaking delay line) *Suppose that the departure time of a packet is known upon its arrival. A network element with a single input $a(t)$ and a single output $d(t)$ is called a* non-overtaking *delay line with maximum delay d if it realizes the set of mappings that satisfy (5.126) and*

$$\tau^d(n) < \tau^d(n+1) \tag{5.129}$$

for all n.

The additional constraint in (5.129) indicates that no packet can overtake its previous packet at the output. Clearly, a non-overtaking delay line is a special case of a flexible delay line. As such, it can be built with less complexity. In fact, a non-overtaking delay line with maximum delay d can be realized by a FIFO queue with buffer d. This is done by letting $c(t) = 1$ for every time t that has a packet departure, i.e., $t = \tau_d(n)$ for some n. However, as we mentioned before, the difference between a FIFO queue and a non-overtaking delay line is that a FIFO queue does not need to know the departure time of a packet upon its arrival. With this additional piece of information, we will show that one can construct a non-overtaking delay line that has the self-routing property.

One possible application of non-overtaking delay lines is optical burst switching (see e.g., [160, 171]), in which control signals of packets are sent via a separate electronic network before packets are actually transmitted. By so doing, packet departure times along the route traversing through an optical network can be pre-determined. Another application is for traffic regulation (see e.g., the books [27, 105] and reference therein). Traffic regulation is done by delaying packets in such a way that packets are spaced in a more regular manner. By so doing, congestion can be avoided inside a network.

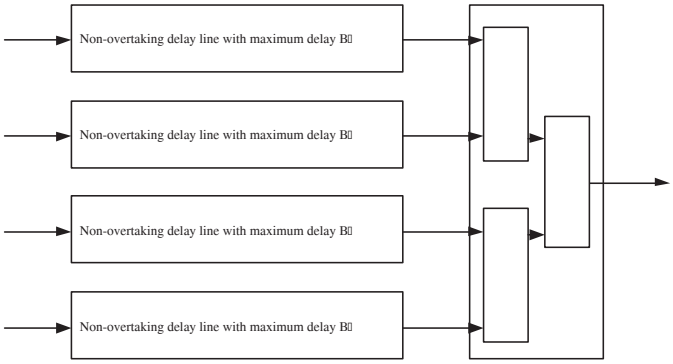


Fig. 5.49. A 4-to-1 multiplexer with buffer B by non-overtaking delay lines

A special case of traffic regulation is an N-to-1 buffered FIFO multiplexer. In Figure 5.49, we show an implementation of a 4-to-1 FIFO multiplexer with buffer B by non-overtaking delay lines. For each input, there is a non-overtaking delay line with maximum delay B before the 4-to-1 un-buffered multiplexer (constructed by three 2×2 switches in Figure 5.49). As the delay of a packet is known upon its arrival to the FIFO multiplexer, we can put packets through the non-overtaking delay lines so that they can be multiplexed by the 4-to-1 un-buffered multiplexer.

More generally, if the control input $c(t)$ of a FIFO queue is deterministic, then such a FIFO queue can be implemented by a non-overtaking delay line. One particular example is the load balanced Birkhoff-von Neumann switch in Chapter 3. In such a switch, the connection patterns of the switch fabrics are periodic and the buffers there are simply FIFO queues with periodic services.

Now we introduce the concept of linear compressors that allow us to construct non-overtaking delay lines.

Definition 5.7.3. (Linear compressor) *Suppose that the departure time of a packet is known upon its arrival. A network element with a single input $a(t)$ and a single output $d(t)$ is called a linear compressor with maximum delay d if it realizes the set of mappings that satisfy (5.126) and the following monotone and consecutive condition: $\tau^d(n) = \tau^d(n-1) + 1$ whenever $\tau^a(n) \leq \tau^d(n-1)$.*

Note that the monotone and consecutive condition and (5.126) imply that $\tau^d(n) > \tau^d(n-1)$ for all n. As such, a linear compressor is

a special case of a non-overtaking delay line. The name, linear compressor, originates from its counterpart for space switches (see e.g., [76, 111]). Historically, it was first shown in [73] that a certain class of banyan networks can realize all permutations that satisfy a monotone and consecutive condition. In that class of banyan networks, an output is *active* if it has a packet to receive. The consecutive condition ensures that there is no gap between two active outputs. The monotone condition indicates that the input addresses of the packets in the consecutive active outputs are increasing. In Chapter 4 of the book by Li [111], it is further shown that this class of banyan networks are equivalent to linear compressors.

The condition $\tau^a(n) \leq \tau^d(n-1)$ means that the n^{th} packet arrives before the $n-1^{th}$ packet departs. If one defines a busy period of a linear compressor as the period of time that there are packets in the linear compressor, then the monotone and consecutive condition implies that the departures in a busy period are monotone and consecutive (see Figure 5.50). Note that the packet that initiates a busy period can have an arbitrary delay (as long as its delay is not greater than the maximum delay). A linear compressor is related to a queue with vacations, i.e., the server of the queue goes on a vacation every time the queue becomes empty. As such, the first customer that initiates a busy period has to wait until the server is back from his/her vacation.

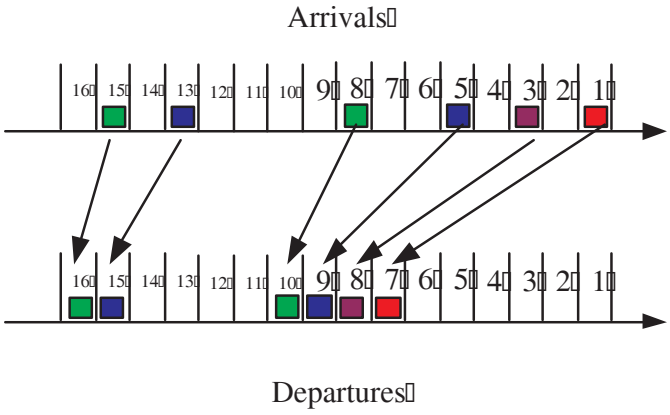


Fig. 5.50. A sample path of a linear compressor

In view of the natural connection between a linear compressor and a queue with vacations, let us consider the 2-to-1 FIFO multiplexer with buffer $2^k - 1$ in Figure 5.22. We claim that it can be used as a linear compressor with maximum delay $2^k - 1$. In this 2-to-1 FIFO multiplexer, we first disable one of the two inputs and the loss output. Define $q(t-1)$ as the virtual delay for a packet that arrives at time t for the linear compressor. We will use $q(t-1)$ to mimic the number of packets stored in the 2-to-1 FIFO multiplexer at $t-1$. Initially the virtual delay is 0, i.e., $q(0) = 0$, and it remains unchanged until the first packet arrives. Consider a packet that initiates a busy period of a linear compressor at time t. This packet will not collide with any other previous packets. As such, the packet can have an arbitrary delay $0 \le d \le 2^k - 1$ and it can be self-routed through the network element via the binary representation of d. Then we update the virtual delay by setting $q(t) = d$. By so doing, we mimic the state that there are d packets stored in the 2-to-1 FIFO multiplexer. In other word, the last packet in the network element will depart at $t + q(t)$. As we have disabled one of the two inputs, the virtual delay $q(t)$ is non-increasing in t until it is decreased to 0. On the other hand, consider a packet that arrives at time t with $q(t-1) > 0$. This packet must arrive before its previous packet departs. As its previous packet will depart at $t - 1 + q(t-1)$, it then follows from the monotone and consecutive condition that the arriving packet will depart at $t + q(t-1)$ and this is exactly the departure time of an arriving packet to the 2-to-1 FIFO multiplexer. Thus, we can use the 2-to-1 FIFO multiplexer as a linear compressor and this is stated in the following corollary.

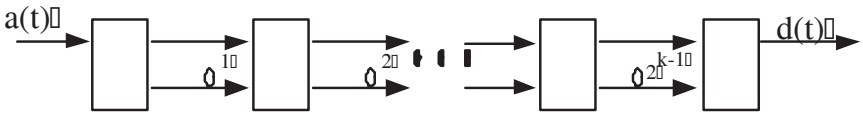


Fig. 5.51. A self-routing linear compressor with maximum delay $2^k - 1$

Corollary 5.7.4. *The construction in Figure 5.51 can be operated as a self-routing linear compressor with maximum delay* $2^k - 1$.

5.7.2 Mirror image and linear decompressor

Definition 5.7.5. (Mirror image) *The mirror image of an SDL element is an SDL element that reverses the direction of every link in the original SDL element. By so doing, the inputs (resp. outputs) of the original SDL element become the outputs (resp. inputs) of its mirror image.*

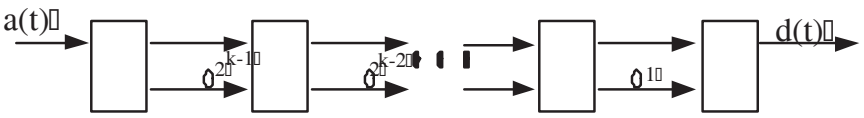


Fig. 5.52. The mirror image of the linear compressor with maximum delay $2^k - 1$ in Figure 5.51

For example, we show in Figure 5.52 the mirror image of the linear compressor with maximum delay $2^k - 1$ in Figure 5.51. For this example, consider a particular sample path $\{a(t), d(t), t = 1, 2, \ldots, \infty\}$. Let $a^r(t) = a(t_0 - t)$ and $d^r(t) = d(t_0 - t)$ be its time reversed sample path from t_0. Intuitively, the time reversed sample path is obtained by playing backward the original sample path from t_0. For instance, in Figure 5.53 we show the time reversed sample path of the sample path in Figure 5.50. As the direction of every link in the mirror image is reversed, playing backward in the original SDL element is equivalent to playing forward in its mirror image. This leads to the following proposition.

Proposition 5.7.6. *If a sample path can be realized by an SDL element, then its time reversed sample path can also be realized by the mirror image of the SDL element.*

As shown in Figure 5.53, the time reversed sample path is obtained by interchanging the arrivals and departures and running the system backward in time. From Proposition 5.7.6, the time reversed sample paths of those realized by a linear compressor can be realized by the mirror image of a linear compressor, called a linear decompressor defined below.

Definition 5.7.7. (Linear decompressor) *Suppose that the departure time of a packet is known upon its arrival. A network element with*

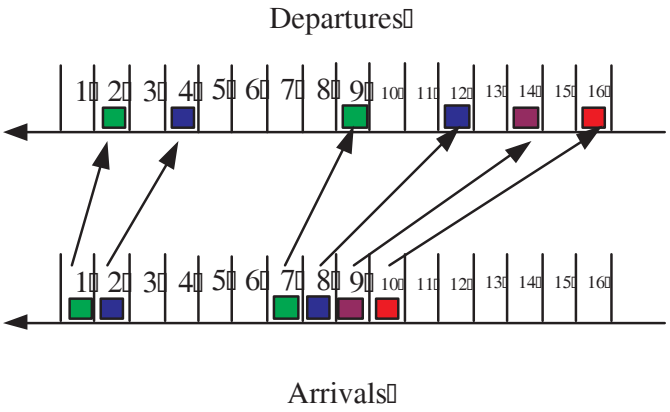


Fig. 5.53. A time reversed sample path of a linear compressor

a single input $a(t)$ and a single output $d(t)$ is called a linear decompressor with maximum delay d if it realizes the set of mappings that satisfy (5.126), (5.129) and the following inverse *monotone and consecutive condition: $\tau^a(n) = \tau^a(n-1) + 1$ whenever $\tau^a(n) \leq \tau^d(n-1)$.*

As the construction in Figure 5.52 is the mirror image of a linear compressor with maximum delay $2^k - 1$, it is a linear decompressor with maximum delay $2^k - 1$. This is stated in the following corollary.

Corollary 5.7.8. *The construction in Figure 5.52 can be operated as a self-routing linear decompressor with maximum delay $2^k - 1$.*

5.7.3 A two-stage construction of a linear compressor

In Figure 5.54, we consider a two-stage construction of a linear compressor with maximum delay $BK - 1$. The first stage is a linear compressor with maximum delay $K-1$. The second stage is a scaled linear compressor with maximum delay $B-1$ and scaling factor K. From the time interleaving property in Proposition 5.1.3, we note that the scaled linear compressor at the second stage can be operated as K time interleaved linear compressors. As shown in Figure 5.55, these K time interleaved linear compressors are connected to the output link of the linear compressor at the first stage and the output link of the network element *periodically* with period K. Moreover, as the delay in each delay line of the scaled linear compressor at the second stage is K times

of that in the original linear compressors, these K time interleaved linear compressors, when connected, only allow delay that is an integer multiple of K, i.e., $d = 0, K, 2K, \ldots, (B-1)K$.

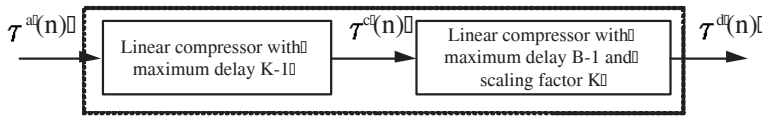


Fig. 5.54. A two-stage construction of a linear compressor

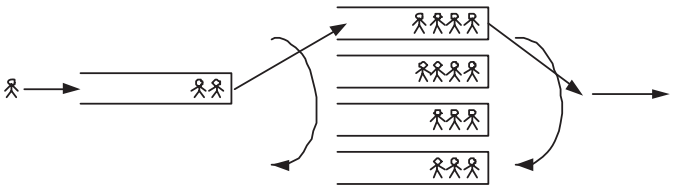


Fig. 5.55. An illustration of periodic connections in the two-stage construction

Now we specify the operation rule for the two-stage construction. As shown in Figure 5.54, let $\tau^a(n)$ be the arrival time of the n^{th} packet, $\tau^c(n)$ be its departure time from the linear compressor at the first stage, and $\tau^d(n)$ be its departure time.

(R1) Initially, we set

$$\tau^c(1) = \tau^a(1) + \Big((\tau^d(1) - \tau^a(1)) \bmod K\Big).$$

If $\tau^c(n-1) < \tau^a(n)$, then we set

$$\tau^c(n) = \tau^a(n) + \Big((\tau^d(n) - \tau^a(n)) \bmod K\Big).$$

Otherwise, we set $\tau^c(n) = \tau^c(n-1) + 1$.

Theorem 5.7.9. *If the network element in Figure 5.54 is started from an empty system, then under (R1) the two-stage construction is a linear compressor with maximum delay* $BK - 1$.

The proof of Theorem 5.7.9 is deferred to the end of this section.

Now we show how one constructs self-routing linear compressors by using optical memory cells. First, we show that an optical memory cell can be used as a linear compressor with maximum delay 1. When there is a packet stored in the optical memory cell, the 2×2 switch is always set to the "cross" state. When a packet arrives at the empty optical memory cell, the 2×2 switch is set to the "bar" state if its delay is 0 and the "cross" state if its delay is 1. As the maximum delay is 1, every packet passes through the fiber delay line with one unit of delay at most once.

With $B = K = 2$, we can apply Theorem 5.7.9 to construct a linear compressor with maximum delay 3 via a concatenation of an optical memory cell (at the first stage) and a scaled optical memory cell with the scaling factor 2 (at the second stage). By recursively expanding the two-stage construction in Theorem 5.7.9 with $K = 2$, one can construct a linear compressor with maximum delay $2^k - 1$ by using a series of k scaled optical memory cells with scaling factors $2^0, 2^1, 2^2, \ldots, 2^{k-1}$ (see Figure 5.56(a)). As a packet passes through every fiber delay line in Figure 5.56(a) at most once, one can simply use the binary representation of packet delay for self-routing. Since a linear decompressor is the mirror image of a linear compressor, one can also construct a self-routing linear decompressor with maximum delay $2^k - 1$ by using a series of k scaled optical memory cells with scaling factors $2^{k-1}, 2^{k-2}, \ldots, 2^1, 2^0$ (see Figure 5.56(b)).

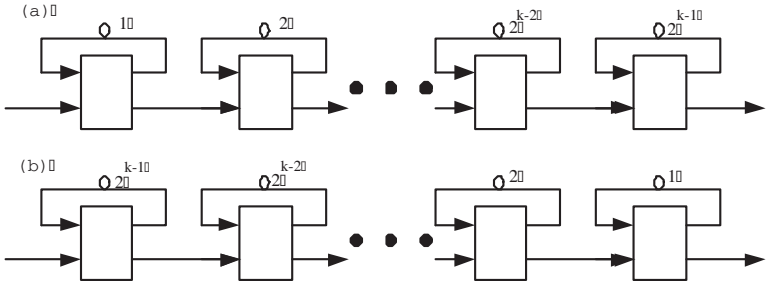


Fig. 5.56. (a) A self-routing linear compressor with maximum delay $2^k - 1$, (b) a self-routing linear decompressor with maximum delay $2^k - 1$.

Proof. In the following, we provide the proof for Theorem 5.7.9. According to Definition 5.7.3 for a linear compressor, we need to show

that the network element in Figure 5.54 can realize all the mappings (or sample paths) that satisfy

$$\tau^a(n) \le \tau^d(n) \le \tau^a(n) + BK - 1, \tag{5.130}$$

$$\tau^d(n) = \tau^d(n-1) + 1, \quad \text{whenever } \tau^a(n) \le \tau^d(n-1). \tag{5.131}$$

In other words, if (5.130) and (5.131) hold for all n, then under the assignment rule in (R1)

$$\tau^a(n) \le \tau^c(n) \le \tau^a(n) + K - 1, \tag{5.132}$$

$$\tau^c(n) = \tau^c(n-1) + 1, \quad \text{whenever } \tau^a(n) \le \tau^c(n-1), \tag{5.133}$$

$$\tau^c(n) \le \tau^d(n) \le \tau^c(n) + (B-1)K, \tag{5.134}$$

$$(\tau^d(n) - \tau^c(n)) \bmod K = 0, \tag{5.135}$$

$$\tau^d(n) = \tau^d(n^*) + K, \quad \text{whenever } \tau^a(n) \le \tau^d(n^*), \tag{5.136}$$

where n^* is the last packet that departs before the n^{th} packet from the same time interleaved linear compressor at the second stage.

We will prove this by induction. For $n = 1$, we have

$$\tau^c(1) = \tau^a(1) + \Big((\tau^d(1) - \tau^a(1)) \bmod K\Big).$$

Thus,

$$\tau^a(1) \le \tau^c(1) \le \tau^a(1) + K - 1$$

and

$$\tau^d(1) - \tau^c(1) = K\lfloor(\tau^d(1) - \tau^a(1))/K\rfloor \le (B-1)K.$$

It is easy to see that (5.132), (5.134) and (5.135) are satisfied. As the network element is started from an empty system, there is no need to check (5.133) and (5.136).

Now suppose that the induction hypotheses in (5.132)-(5.136) hold up to $n-1$. For the n^{th} packet, we need to consider the following cases.
Case 1 $\tau^a(n) > \tau^d(n-1)$:

In this case, the n^{th} packet sees an empty system. The proof is the same as the case for $n = 1$.
Case 2 $\tau^a(n) \le \tau^d(n-1)$ and $\tau^a(n) > \tau^c(n-1)$:

Since $\tau^a(n) > \tau^c(n-1)$, we also have from (R1) that

$$\tau^c(n) = \tau^a(n) + ((\tau^d(n) - \tau^a(n)) \bmod K). \tag{5.137}$$

As argued for the case $n = 1$, it is easy to see that (5.132), (5.134) and (5.135) are satisfied. Since $\tau^a(n) > \tau^c(n-1)$, there is no need to verify (5.133).

To prove (5.136), let $n_0 = \sup\{m < n : \tau^a(m) > \tau^d(m-1)\}$ be the index of the packet that initiates the busy period containing the n^{th} packet. From (5.131), it follows that for all $n_0 < m \leq n$

$$\tau^d(m) = \tau^d(m-1) + 1, \tag{5.138}$$

As illustrated in Figure 5.55, these K time interleaved linear compressors at the second stage are connected to the output link of the linear compressor at the first stage periodically with period K. If $n-K \geq n_0$, then we have from (5.138) that $n^* = n - K$ is the last packet that departs before the n^{th} packet from the same time interleaved linear compressor at the second stage. As such, it follows from (5.138) that

$$\tau^d(n^*) + K = \tau^d(n-K) + K = \tau^d(n). \tag{5.139}$$

On the other hand, if $n - K < n_0$, then the n^{th} packet arrives at an empty linear compressor at the second stage and there is no need to check (5.136).

Case 3 $\tau^a(n) \leq \tau^d(n-1)$ and $\tau^a(n) \leq \tau^c(n-1)$:

Since $\tau^a(n) \leq \tau^c(n-1)$, we also have from (R1) that

$$\tau^c(n) = \tau^c(n-1) + 1. \tag{5.140}$$

Thus, (5.133) is satisfied. Moreover,

$$\tau^a(n) \leq \tau^c(n-1) = \tau^c(n) - 1 \leq \tau^c(n). \tag{5.141}$$

Using the induction hypothesis for $n-1$ in (5.132) and $\tau^a(n-1) < \tau^a(n)$ yields

$$\begin{aligned} \tau^c(n) &= \tau^c(n-1) + 1 \leq \tau^a(n-1) + K - 1 + 1 \\ &\leq \tau^a(n) + K - 1. \end{aligned} \tag{5.142}$$

Thus, (5.132) is satisfied.

Since $\tau^a(n) \leq \tau^d(n-1)$, we have from (5.131) that

$$\tau^d(n) = \tau^d(n-1) + 1, \tag{5.143}$$

From (5.143) and (5.140), it follows that

$$\tau^d(n) - \tau^c(n) = \tau^d(n-1) - \tau^c(n-1).$$

Thus, (5.134) and (5.135) follow from the induction hypotheses for $n-1$.

The argument for (5.136) is the same as that in Case 2. ■

5.7.4 A three-stage construction of a non-overtaking delay line

In Figure 5.57, we consider a three-stage construction of a non-overtaking delay line with maximum delay $BK-1$. The first stage is a linear compressor with maximum delay $K-1$, the second stage is a scaled non-overtaking delay line with maximum delay $B-1$ and scaling factor K, and the third stage is a linear decompressor with maximum delay $K-1$.

As shown in Figure 5.57, let $\tau^a(n)$ be the arrival time of the n^{th} packet, $\tau^c(n)$ be the departure time of the n^{th} packet from the linear compressor, $\tau^b(n)$ be the arrival time of the n^{th} packet at the linear decompressor, and $\tau^d(n)$ be the departure time of the n^{th} packet. In order to show that the three-stage construction in Figure 5.57 can be operated as a non-overtaking delay line, we need to specify $\tau^c(n)$ and $\tau^b(n)$ for every n.

(R2) Initially, we set

$$\tau^c(1) = \tau^a(1) + \Big((\tau^d(1) - \tau^a(1)) \bmod K\Big), \tag{5.144}$$

and

$$\tau^b(1) = \tau^c(1) + \lfloor \frac{\tau^d(1) - \tau^c(1)}{K} \rfloor \times K. \tag{5.145}$$

Note from (5.144) that

$$\tau^d(1) - \tau^c(1) = \lfloor \frac{\tau^d(1) - \tau^a(1)}{K} \rfloor \times K. \tag{5.146}$$

Thus, it follows from (5.145) and (5.144) that

$$\tau^b(1) = \tau^d(1). \tag{5.147}$$

If $\tau^c(n-1) < \tau^a(n)$, then we set

$$\tau^c(n) = \tau^a(n) + \Big((\tau^d(n) - \tau^a(n)) \bmod K\Big), \tag{5.148}$$

and

$$\tau^b(n) = \tau^c(n) + \lfloor \frac{\tau^d(n) - \tau^c(n)}{K} \rfloor \times K. \tag{5.149}$$

By so doing, we also have

$$\tau^b(n) = \tau^d(n). \tag{5.150}$$

Otherwise, we set

$$\tau^c(n) = \tau^c(n-1) + 1, \tag{5.151}$$

and

$$\tau^b(n) = \tau^c(n) + \lfloor \frac{\tau^d(n) - \tau^c(n)}{K} \rfloor \times K. \tag{5.152}$$

Note from (5.145),(5.149) and (5.152) that we always have

$$\tau^b(n) = \tau^c(n) + \lfloor \frac{\tau^d(n) - \tau^c(n)}{K} \rfloor \times K \tag{5.153}$$

for all n. As such, $\tau^b(n) - \tau^c(n)$ is an integer multiple of K and it can be carried by the scaled non-overtaking delay line with scaling factor K in the second stage.

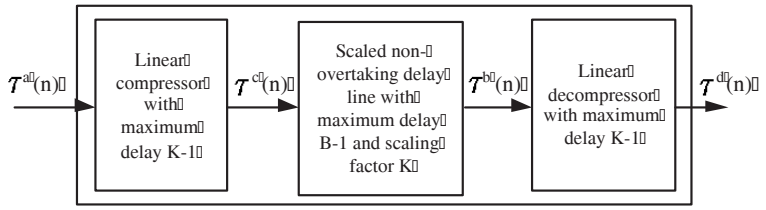


Fig. 5.57. A three-stage construction of a non-overtaking delay line

Theorem 5.7.10. *If the network element in Figure 5.57 is started from an empty system, then under (R2) the three-stage construction is a non-overtaking delay line with maximum delay* $BK - 1$.

The proof of Theorem 5.7.10 is deferred to the end of this section.

For the special case that $B = 1$, the second stage can be omitted. In this case, a non-overtaking delay line with maximum delay $K-1$ can be constructed as a concatenation of a linear compressor with maximum delay $K - 1$ and a linear decompressor with maximum delay $K - 1$. As we have shown how one constructs self-routing linear compressors and decompressors by optical memory cells in Section 5.7.3, it is quite

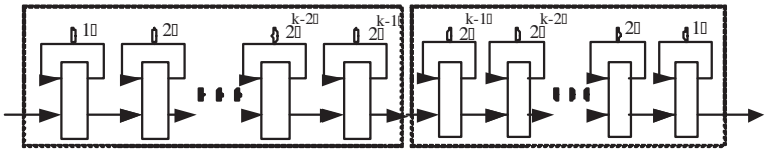


Fig. 5.58. A construction of a self-routing non-overtaking delay line with maximum delay $2^k - 1$ by a concatenation of optical memory cells

straightforward to construct a self-routing non-overtaking delay line by a concatenation of scaled optical memory cells. In Figure 5.58, we show a construction of a self-routing non-overtaking delay line with maximum delay $2^k - 1$ by a concatenation of optical memory cells.

Proof. In the following, we provide the proof for Theorem 5.7.10. We need to show that all the mappings that satisfy (5.126) and (5.129) can be realized by a concatenation of a linear compressor with maximum delay $K - 1$ (at the first stage), a scaled non-overtaking delay line with maximum delay $B-1$ and scaling factor K (at the second stage), and a linear decompressor with maximum delay $K - 1$ (at the third stage). According to Definition 5.7.2 for a non-overtaking delay line, Definition 5.7.3 for a linear compressor and Definition 5.7.7 for a linear decompressor, we need to show that if for all n

$$\tau^a(n) \le \tau^d(n) \le \tau^a(n) + BK - 1, \tag{5.154}$$

$$\tau^d(n) > \tau^d(n-1), \tag{5.155}$$

then under the assignment rule for $\tau^c(n)$ and $\tau^b(n)$ in (R2) that

$$\tau^a(n) \le \tau^c(n) \le \tau^a(n) + K - 1, \tag{5.156}$$

$$\tau^c(n) = \tau^c(n-1) + 1, \quad \text{whenever } \tau^a(n) \le \tau^c(n-1), \tag{5.157}$$

$$\tau^c(n) \le \tau^b(n) \le \tau^c(n) + (B-1)K \tag{5.158}$$

$$(\tau^b(n) - \tau^c(n)) \bmod K = 0, \tag{5.159}$$

$$\tau^b(n) > \tau^b(n-1), \tag{5.160}$$

$$\tau^b(n) \le \tau^d(n) \le \tau^b(n) + K - 1, \tag{5.161}$$

$$\tau^b(n) = \tau^b(n-1) + 1, \quad \text{whenever } \tau^b(n) \le \tau^d(n-1). \tag{5.162}$$

We first show that $\tau^a(n) \leq \tau^c(n)$ for all n. If $n = 1$ or $\tau^c(n-1) < \tau^a(n)$, then we have $\tau^a(n) \leq \tau^c(n)$ from (5.144) and (5.148). On the other hand, if $\tau^c(n-1) \geq \tau^a(n)$, then we have from (5.151) that

$$\tau^a(n) < \tau^c(n-1) + 1 = \tau^c(n).$$

Next, we show by induction that $\tau^c(n) \leq \tau^d(n)$ for all n. Since $\tau^a(1) \leq \tau^d(1)$, this holds trivially for $n = 1$ from (5.144). Now suppose that $\tau^c(n-1) \leq \tau^d(n-1)$. If $\tau^c(n-1) < \tau^a(n)$, then we also have from (5.148) and $\tau^a(n) \leq \tau^d(n)$ that $\tau^c(n) \leq \tau^d(n)$. On the other hand, if $\tau^c(n-1) \geq \tau^a(n)$, then we have from (5.151) that $\tau^c(n) = \tau^c(n-1)+1$. It then follows from the induction hypothesis and $\tau^d(n) > \tau^d(n-1)$ that

$$\tau^c(n) = \tau^c(n-1) + 1 \leq \tau^d(n-1) + 1 \leq \tau^d(n).$$

Now we use $\tau^c(n) \leq \tau^d(n)$ for all n to show that (5.158),(5.159) and (5.161) hold for all n. Note from (5.153) that

$$\tau^b(n) - \tau^c(n) = K\lfloor\frac{\tau^d(n) - \tau^c(n)}{K}\rfloor.$$

Thus, (5.159) is satisfied for all n. Moreover, we have from $\tau^a(n) \leq \tau^c(n)$ for all n and (5.154) that

$$\begin{aligned}
&\tau^b(n) - \tau^c(n) = K\lfloor\frac{\tau^d(n) - \tau^c(n)}{K}\rfloor \\
&\leq K\lfloor\frac{\tau^d(n) - \tau^a(n)}{K}\rfloor \leq K\lfloor\frac{(BK-1)}{K}\rfloor \\
&= (B-1)K. \qquad (5.163)
\end{aligned}$$

Thus, (5.158) is also satisfied for all n. Furthermore,

$$\begin{aligned}
&\tau^d(n) - \tau^b(n) = \tau^d(n) - \tau^c(n) - K\lfloor\frac{\tau^d(n) - \tau^c(n)}{K}\rfloor \\
&= (\tau^d(n) - \tau^c(n)) \bmod K. \qquad (5.164)
\end{aligned}$$

This shows that (5.161) is also satisfied for all n.

We will prove the other four conditions, i.e., (5.156), (5.157), (5.160) and (5.162), by induction on n. For $n = 1$, it follows from (5.144) and (5.154) that (5.156) holds trivially. As this is the first packet, there is no need to verify (5.157), (5.160) and (5.162).

Now suppose that the induction hypotheses in (5.156), (5.157), (5.160) and (5.162) hold up to $n-1$. For the n^{th} packet, we need to consider the following two cases.

Case 1 $\tau^a(n) > \tau^c(n-1)$:

In this case, the assignments for $\tau^c(n)$ and $\tau^b(n)$ are the same as those for $n = 1$. As such, (5.156) holds accordingly. Since $\tau^a(n) > \tau^c(n-1)$, there is no need to verify (5.157). Also, in view of $\tau^b(n) = \tau^d(n)$ in (5.150) and $\tau^d(n) > \tau^d(n-1)$ in (5.155), we have $\tau^b(n) > \tau^d(n-1)$. Thus, there is no need to verify (5.162). In conjunction with $\tau^d(n-1) \geq \tau^b(n-1)$ in (5.161), we derive $\tau^b(n) > \tau^b(n-1)$ for (5.160).

Case 2 $\tau^a(n) \leq \tau^c(n-1)$:

In this case, we have from (5.151) that $\tau^c(n) = \tau^c(n-1)+1$. Thus, (5.157) is satisfied. Moreover, using the induction hypothesis for $n-1$ in (5.156) and the fact that $\tau^a(n) \geq \tau^a(n-1)+1$ yields

$$\begin{aligned}\tau^c(n) &= \tau^c(n-1)+1 \leq \tau^a(n-1)+K-1+1\\ &\leq \tau^a(n)+K-1.\end{aligned}$$

This shows that $\tau^c(n) \leq \tau^a(n)+K-1$ in (5.156).

To see (5.160), note from (5.151) and (5.155) that

$$\tau^d(n)-\tau^d(n-1) \geq 1 = \tau^c(n)-\tau^c(n-1).$$

It then follows from (5.153) and (5.151) that

$$\begin{aligned}&\tau^b(n-1)\\ &= \tau^c(n-1)+\lfloor\frac{\tau^d(n-1)-\tau^c(n-1)}{K}\rfloor\times K\\ &\leq \tau^c(n-1)+\lfloor\frac{\tau^d(n)-\tau^c(n)}{K}\rfloor\times K\\ &< \tau^c(n)+\lfloor\frac{\tau^d(n)-\tau^c(n)}{K}\rfloor\times K\\ &= \tau^b(n).\end{aligned}$$

It remains to verify (5.162). If $\tau^b(n) \leq \tau^d(n-1)$, then it follows from (5.161) (for $n-1$) that

$$\tau^b(n) \leq \tau^d(n-1) \leq \tau^b(n-1)+K-1.$$

Since $(\tau^b(n)-\tau^c(n)) \bmod K = 0$ for all n in (5.159), it then follows from (5.151) that

$$\begin{aligned}&\Big((\tau^b(n)-\tau^b(n-1)) \bmod K\Big)\\ &\quad=\Big((\tau^c(n)-\tau^c(n-1)) \bmod K\Big)=1.\end{aligned}$$

Thus, we have $\tau^b(n) = \tau^b(n-1)+1$. ■

5.7.5 A three-stage construction of a flexible delay line

In Figure 5.59, we show a construction of a flexible delay line with the range of delay $[K-1, BK-1]$. It is a combination of three parallel three-stage constructions. In each three-stage construction, there is a flexible delay line with maximum delay $K-1$ (at the first stage), a scaled flexible delay line with maximum delay $B-1$ and scaling factor K (at the second stage), and a flexible delay line with maximum delay $K-1$ (at the third stage). These three parallel three-stage constructions are joined by a 1×3 switch at the beginning and a 3×1 switch at the end. The 1×3 switch acts as a 1-to-3 demultiplexer so that an arriving packet can choose its path from one of the three three-stage constructions. The departures from these three three-stage constructions are then multiplexed by the 3×1 switch at the end.

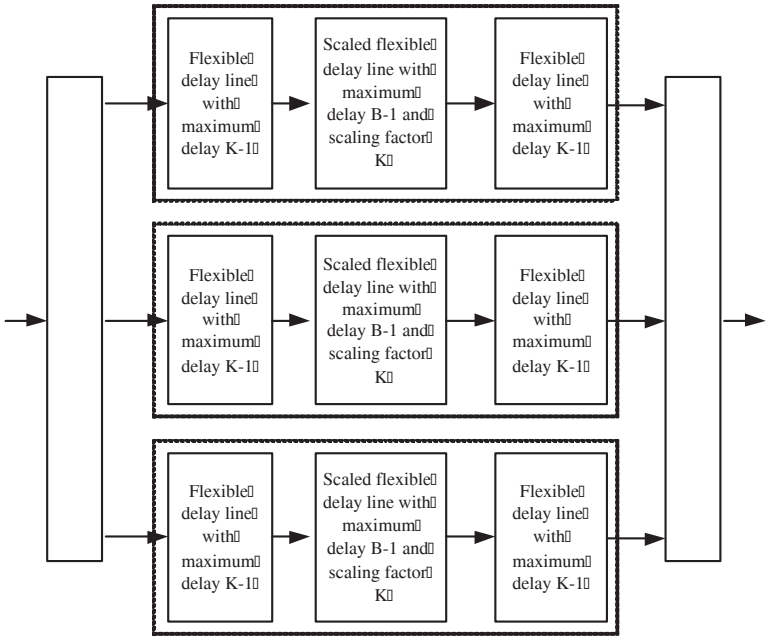


Fig. 5.59. A three-stage construction of a flexible delay line with the range of delay $[K-1, BK-1]$

Theorem 5.7.11. *The three-stage construction in Figure 5.59 is a flexible delay line with the range of delay* $[K-1, BK-1]$.

Proof. The proof for Theorem 5.7.11 is quite similar to that for the classical three-stage nonblocking Clos networks [51]. We will show for each arriving packet, there is a non-conflicting path from the input to the output. Consider the n^{th} packet. Let $\tau^a(n)$ be the arrival time of the n^{th} packet and $\tau^d(n)$ be the departure time of the n^{th} packet. Suppose that the delay of the n^{th} packet, denoted by d, is in the range $[K-1, BK-1]$, i.e., $K-1 \leq d \leq BK-1$. Then we claim that the number of feasible paths for the n^{th} packet to go through one of the three three-stage constructions is K. To see this, note that one can choose $0 \leq d_1 \leq K-1$ to be the delay at the flexible delay line with maximum delay $K-1$ at the first stage. Once d_1 is chosen, there is a unique way to determine the delay at the second stage and the delay at the third stage (as the delay at the second stage must be an integer multiple of K). Specifically, the delay at the third stage is $((d-d_1) \bmod K)$ and the delay at the second stage is $\lfloor (d-d_1)/K \rfloor \times K$.

As there are three parallel constructions, the total number of feasible paths for the n^{th} packet to go through the network element is $3K$. The only places that the n^{th} packet might collide with others are the three output links of the first stage and the three input links of the third stage. As the first stage (in all the three-stage constructions) is a flexible delay line with maximum delay $K-1$, those packets that might collide with the n^{th} packet at the output links of the first stage must arrive during $[\tau^a(n)-(K-1), \tau^a(n)-1]$. These packets will use at most $K-1$ paths among the $3K$ feasible paths for the n^{th} packet. On the other hand, those packets that might collide with the n^{th} packet at the three input links of the third stage must depart during $[\tau^d(n)-(K-1), \tau^d(n)-1]$ and $[\tau^d(n)+1, \tau^d(n)+K-1]$. Similarly, these packets use at most another $2(K-1)$ paths among the $3K$ feasible paths for the n^{th} packet. Thus, there is at least one non-conflicting path for the n^{th} packet. ■

Unlike the classical nonblocking Clos networks, the construction in Figure 5.59 cannot accommodate for the packets with delay smaller than $K-1$. This is because there are not enough feasible paths for those packets with short delay. For instance, for a packet with delay 0, there are only three feasible paths.

To construct a flexible delay line with maximum delay $BK-1$, one can simply add a flexible delay line with maximum delay $K-1$ parallel to the three three-stage constructions (see Figure 5.60). By so doing,

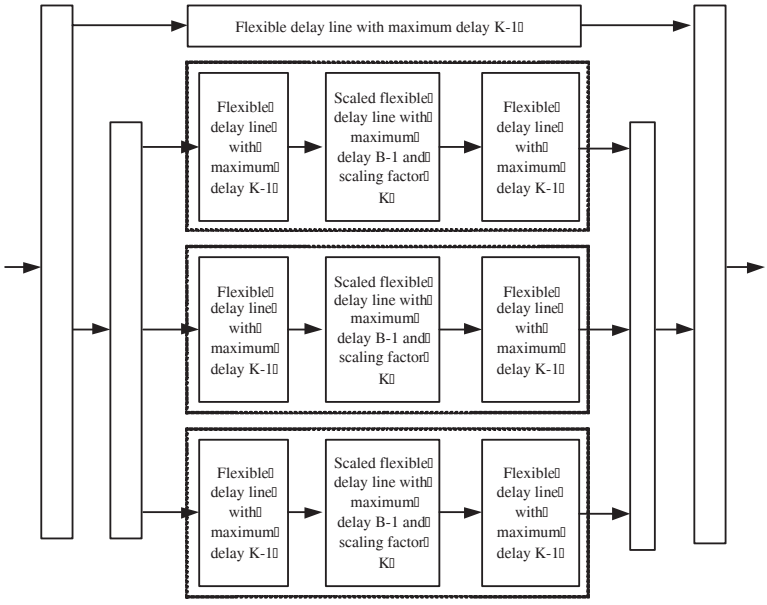


Fig. 5.60. A flexible delay line with maximum delay $BK-1$

those packets with delay smaller than $K-1$ can be routed through the added flexible delay line with maximum delay $K-1$.

We note that one may construct a flexible delay line with maximum delay $K-1$ by a concatenation of $K-1$ optical memory cells (all with one unit of delay). This is because an optical memory cell can be used for storing one packet. When a packet arrives, we may store that packet in an optical memory cell until its departure time. As the maximum delay is $K-1$, there are at most $K-1$ packets that need to be stored at the same time.

To compute the construction complexity, let $H(B)$ be the number of 2×2 switches needed for a flexible delay line with maximum delay B using the construction in Figure 5.60. Note that a 1×3 switch (and a 3×1 switch) can be implemented by two 2×2 switches. Clearly, we have the following recursive equation:

$$H(BK-1) = 7H(K-1) + 3H(B-1) + 6. \tag{5.165}$$

Letting $B = K$ yields

$$H(B^2-1) = 10H(B-1) + 6. \tag{5.166}$$

If we use $B-1$ optical memory cells to construct all the ten (scaled or unscaled) flexible delay lines with maximum delay $B-1$, we can

construct a flexible delay line with maximum delay $B^2 - 1$ by using $10B - 4$ 2×2 switches. By recursive expansion of (5.166), one can construct a flexible delay line with maximum delay B with $O((\log B)^\gamma)$ 2×2 switches, where $\gamma = \log_2 10 \approx 3.321928$.

5.7.6 Constructions of flexible delay lines by Cantor Networks

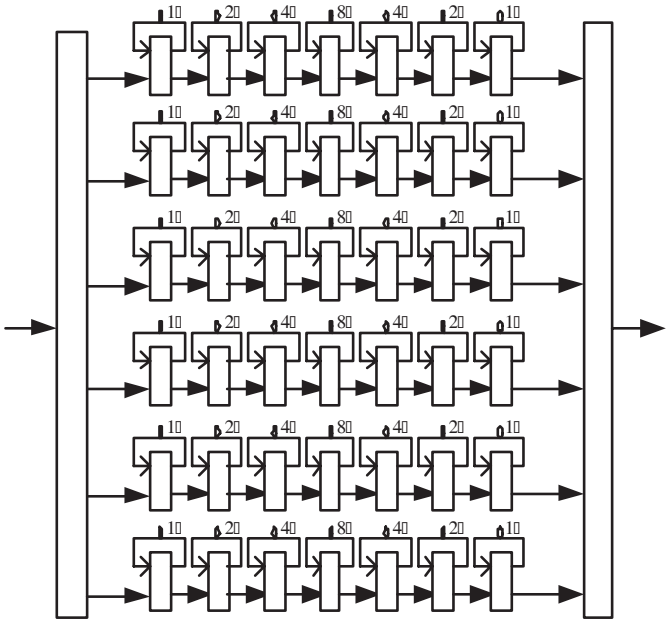


Fig. 5.61. The $(6, 4)$-Cantor network

Instead of using only three parallel three-stage constructions in the previous section, one may consider a combination of multiple multistage constructions as in the Cantor Network [21] for a nonblocking switch. First, we consider a multistage network element constructed by a concatenation of $2k - 1$ scaled optical memory cells. The scaling factor at the j^{th} stage is 2^{j-1} for $j = 1, 2, \ldots, k$ and the scaling factor at the j^{th} stage is 2^{2k-1-j} for $j = k + 1, \ldots, 2k - 1$. Such a network element is called a Benes time slot interchange as it can be used for realizing a $2^k \times 2^k$ Benes time slot interchange in Section 5.2.3. Now we combine m $2^k \times 2^k$ Benes time slot interchanges by adding

a $1 \times m$ switch in the front and an $m \times 1$ switch at the end. Such a network element is called the (m, k)-Cantor network (as it is closely related to the nonblocking Cantor network). In Figure 5.61, we depict the $(6, 4)$-Cantor network.

Theorem 5.7.12. *Suppose that every delay line in the (m, k)-Cantor network can be traversed by a packet at most once. If $m \geq \frac{3}{2}k$, then the (m, k)-Cantor network is a flexible delay line with the range of delay $[2^{k-1} - 1, 2^k - 1]$.*

Proof. The proof for Theorem 5.7.12 is quite similar to the proof for showing that Cantor networks are non-blocking switches (see e.g., [76]). Consider the n^{th} packet. Let $\tau^a(n)$ be its arrival time and $\tau^d(n)$ be its departure time. Suppose that the delay of the n^{th} packet, denoted by d, is in the range $[2^{k-1} - 1, 2^k - 1]$, i.e.,

$$2^{k-1} - 1 \leq d \leq 2^k - 1.$$

First, we claim that the total number of feasible paths for the n^{th} packet is $m2^{k-1}$. To show this, it suffices to argue that the total number of feasible paths through a particular Benes time slot interchange is 2^{k-1}. Let d_1 be the delay to the *input link* of the k^{th} stage of a particular Benes time slot interchange. As each delay line can be traversed at most once, we have $0 \leq d_1 \leq 2^{k-1} - 1$. Once d_1 is chosen, the path to the input link of the k^{th} stage is uniquely determined by the binary representation of d_1. On the other hand, since $2^{k-1} - 1 \leq d \leq 2^k - 1$, we have $0 \leq d - d_1 \leq 2^k - 1$. In view of the constraint that each delay line can be traversed at most once, the path from the input link of the k^{th} stage to the output of the Benes time slot interchange is also uniquely determined by the binary representation of $d - d_1$. Since there are 2^{k-1} choices of d_1, there are 2^{k-1} feasible paths through a Benes time slot interchange.

Let S_1 be the set of *feasible* paths for the n^{th} packet from the input of the Cantor network to the m input links of the optical memory cells at the k^{th} stage. As argued in the previous paragraph, each path in S_1 corresponds to a feasible path from the input to the output. Thus, we have $|S_1| = m2^{k-1}$. Moreover, the delay of a path in S_1 is between 0 and $2^{k-1} - 1$. As such, these paths in S_1 might be in conflict with those packets that arrive during $\tau^a(n) - 1, \tau^a(n) - 2, \ldots, \tau^a(n) - (2^{k-1} - 1)$.

Now we claim that the maximum number of paths in S_1 that are in conflict with the packet arriving at $\tau^a(n) - 1$ is at most 2^{k-2}. First,

note that the packet that arrives at $\tau^a(n)-1$ might be in conflict with the n^{th} packet at various stages (from 2 to k). However, the worst case that results in the maximum number of conflicting paths is to have a conflict at the earliest stage, i.e., the second stage. To achieve this, suppose that the packet that arrives at $\tau^a(n)-1$ is delayed by 1 to the input link of the k^{th} stage in one of the m Benes time slot interchanges. In this case, if the n^{th} packet would still like to use the same Benes time slot interchange, it cannot be delayed by 0 at the first stage. As the number of paths for that Benes time slot interchange with delay 0 at the first stage is 2^{k-2}, the number of conflicting paths in this case is 2^{k-2}.

In general, the maximum number of paths in S_1 that are in conflict with the packet arriving at $\tau^a(n)-s$ for some $2^{j-2} \leq s \leq 2^{j-1}-1$ and $2 \leq j \leq k$, is 2^{k-j}. This is because the earliest conflict can only occur at the input link of the j^{th} stage of a Benes time slot interchange (as the maximum delay for the n^{th} packet to be at the input link of the $(j-1)^{th}$ stage is $2^{j-2}-1$). As such, the maximum number of paths in S_1 that are in conflict with those packets arriving during $[\tau^a(n)-2^{j-2}, \tau^a(n)-(2^{j-1}-1)]$ is $2^{j-2}2^{k-j}$. Hence, the maximum number of paths that are in conflict with those packets arriving during $[\tau^a(n)-1, \tau^a(n)-(2^{k-1}-1)]$ is $(k-1)2^{k-2}$.

Define a *reachable* path as a *feasible* path that is not conflicting with other packets (ahead of the n^{th} packet). Let $\tilde{S}_1$ be the set of *reachable* paths for the n^{th} packet from the input link of the Cantor network to the m input links of the optical memory cells at the k^{th} stage. Clearly, we have

$$|\tilde{S}_1| \geq |S_1| - (k-1)2^{k-2} = m2^{k-1} - (k-1)2^{k-2}. \tag{5.167}$$

Similarly, one can define S_2 (resp. $\tilde{S}_2$) to be the set of *feasible* (resp. *reachable*) paths for the n^{th} packet from the output of the Cantor network to the m output links of the k^{th} stage. As the Cantor network is symmetric (its mirror image is itself), we also know that $|S_2| = m2^{k-1}$. Moreover, these paths in S_2 might be in conflict with those packets that depart during $[\tau^d(n)-1, \tau^d(n)-(2^{k-1}-1)]$ and $[\tau^d(n)+1, \tau^d(n)+(2^{k-1}-1)]$. Analogous to the argument used in the previous paragraph, one can show that the maximum number of paths that are in conflict with those packets departing during $[\tau^d(n)-1, \tau^d(n)-(2^{k-1}-1)]$ is $(k-1)2^{k-2}$. Similarly, the maximum number of paths that are in conflict with those packets departing during $[\tau^d(n)+1, \tau^d(n)+(2^{k-1}-1)]$ is also $(k-1)2^{k-2}$. Thus,

$$|\tilde{S}_2| \geq m2^{k-1} - 2(k-1)2^{k-2}. \tag{5.168}$$

If $m \geq \frac{3}{2}k$, it then follows from (5.167) and (5.168) that

$$|\tilde{S}_1| + |\tilde{S}_2| > m2^{k-1}. \tag{5.169}$$

Since the total number of feasible paths for the n^{th} packet is $m2^{k-1}$, it follows from (5.169) that there must be at least one feasible path that is in the intersection of $\tilde{S}_1$ and $\tilde{S}_2$. Thus, there must be a reachable path from the input of the Cantor network to the output of the Cantor network. ■

As discussed in the previous section, there are not enough feasible paths for packets with short delay, especially in the middle stages of the Cantor network. To construct a flexible delay line with maximum delay $2^k - 1$, one may simply add a flexible delay line with maximum delay $2^{k-1} - 1$ parallel to the $(\lceil \frac{3}{2}k \rceil, k)$-Cantor network as in the previous section. Since the number of 2×2 switches in the $(\lceil \frac{3}{2}k \rceil, k)$-Cantor network is $O(k^2)$, recursively expanding such a construction yields a flexible delay line with maximum delay $2^k - 1$ that needs $O(k^3)$ 2×2 switches.

Here we propose a better alternative to solve the problem for packets with short delay. The idea is to add redundancy to each Benes time slot interchange so that packets with short delay can bypass the middle stages. As illustrated in Figure 5.62, we consider a "layered" (m, k)-Cantor network by inserting a 1×2 switch before the optical memory cell at the j^{th} stage for $j = 1, 2, \ldots, k-1$ and a 2×1 switch after the optical memory cell at the j^{th} stage for $j = k+1, k+2, \ldots, 2k-1$. In addition to these, we add another scaled optical memory cell with scaling factor 2^{j-1} at the j^{th} stage, $j = 1, 2, \ldots, k-1$. For the newly added optical memory cell at the j^{th} stage, its input link is connected to the upper output link of the 1×2 switch before the optical memory cell at the j^{th} stage, and its output link is connected to the upper input link of the 2×1 switch after the optical memory cell at the $2k - j^{th}$ stage. For a packet with delay between $2^{j-1} - 1$ and $2^j - 1$, $j = 1, 2, \ldots, k-1$, it is then routed through the newly added optical memory cell at the j^{th} stage. This is equivalent to embedding an (m, j)-Cantor network, $j = 1, 2, \ldots, k-1$, inside the layered (m, k)-Cantor network. By so doing, a packet with the delay ranging between $2^{j-1} - 1$ and $2^j - 1$ can be routed by the embedded (m, j)-Cantor network. This is stated in the following corollary.

Corollary 5.7.13. *Suppose that every delay line in the layered (m, k)-Cantor network can be traversed by a packet at most once. If $m \geq \frac{3}{2}k$, then the layered (m, k)-Cantor network (see Figure 5.62) is a flexible delay line with maximum delay $2^k - 1$.*

Proof. It suffices to show that a packet with the delay ranging between $2^{j-1}-1$ and 2^j-1 can be routed by the embedded (m, j)-Cantor network. The argument for this is exactly the same as that in the proof of Theorem 5.7.12. ■

Note that the number of 2×2 switches in the layered $(\lceil \frac{3}{2}k \rceil, k)$-Cantor network is still $O(k^2)$. This implies that one can construct a flexible delay line with maximum delay B by using $O((\log B)^2)$ 2×2 switches.

5.8 Priority Queues

Our main objective of this section is the constructions of optical priority queues. Sarwate and Anantharam [146] considered a feedback system in [91] (see Figure 5.63). In such a feedback system, there is an $(M + 1) \times (M + 1)$ crossbar switch and M fiber delay lines with delays d_i, $i = 1, 2, \ldots, M$. If $M = 2k - 1$ for some positive integer k, $d_i = i$ for $i = 1, \ldots, k$, and $d_i = 1$ for $i = k + 1, \ldots, 2k - 1$, then it was shown in [146] that such a system can be used for exact emulation of a priority queue with buffer $\sum_{i=1}^{k} d_i$. In this section, we provide a much simpler and shorter proof than that given in [146]. Our proof not only gives the insights needed to understand why such a construction works, but also leads to a much general result that recovers the choice of the delays of the M fiber delay lines in [146] as a special case.

In the following, we define a (discrete-time) priority queue with buffer B.

Definition 5.8.1. (Priority queue) *A priority queue with buffer B is a network element that has one input link, one control input link, and two output links (see Figure 5.64). One output link is for departing packets and the other is for lost packets. When a packet arrives at the queue, it is associated with a label, called* priority*. We assume that there is a total order for the priorities of all the packets. As shown in Figure 5.64, let $c(t)$ be the state of the control input at time t. When*

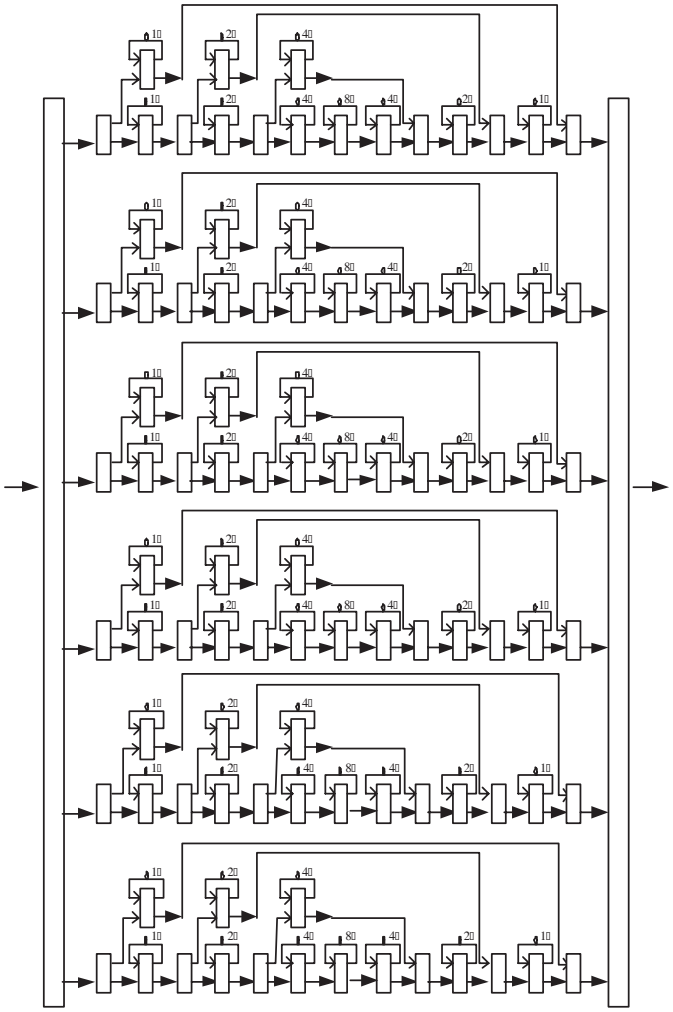


Fig. 5.62. The layered $(6, 4)$-Cantor network

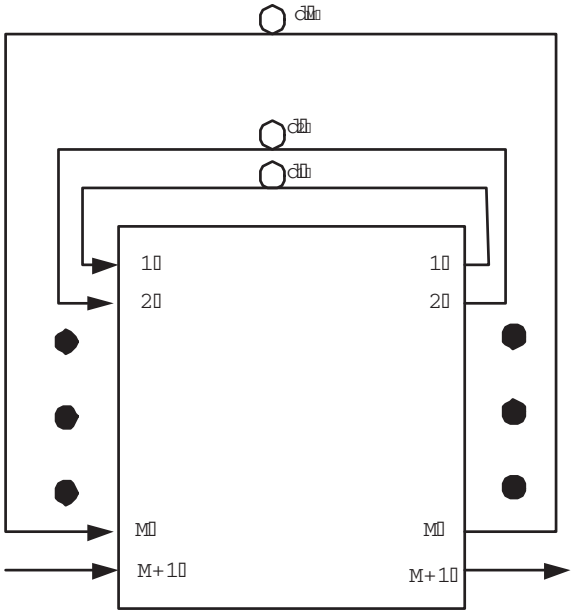


Fig. 5.63. A construction of a priority queue via a single switch and fiber delay lines.

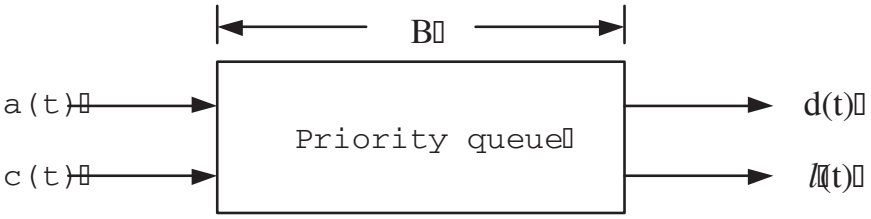


Fig. 5.64. A priority queue with buffer B.

$c(t) = 1$, we say the priority queue is enabled at time t. On the other hand, the priority queue is disabled at time t if $c(t) = 0$. Also, let $a(t)$ be the set of the packet arriving at time t (if any[1]), $d(t)$ be the set of the packet departing at time t (if any), $\ell(t)$ be the set of the lost packet at time t (if any), and $q(t)$ be the set of packets queued at the priority queue at time t (at the end of the t^{th} time slot). Then the priority queue with buffer B satisfies the following five properties:

(P1) Flow conservation: arriving packets from the input link are either stored in the buffer or transmitted through the two output links, i.e.,

$$q(t) = q(t-1) \cup a(t) \backslash (d(t) \cup \ell(t)). \tag{5.170}$$

(P2) Non-idling: if the control input is enabled, i.e., $c(t) = 1$, then there is always a departing packet if there are packets in the buffer or there is an arriving packet, i.e.,

$$|d(t)| = \begin{cases} 1 & \text{if } c(t) = 1 \text{ and } |q(t-1) \cup a(t)| > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{5.171}$$

(P3) Maximum buffer usage: if the control input is not enabled, i.e., $c(t) = 0$, then there is a lost packet only when buffer is full and there is an arriving packet, i.e.,

$$|\ell(t)| = \begin{cases} 1 & \text{if } c(t) = 0 \text{ and } |q(t-1) \cup a(t)| > B \\ 0 & \text{otherwise} \end{cases}. \tag{5.172}$$

(P4) Priority departure: if there is a departing packet at time t, the departing packet is the one with the highest priority among all the packets in $q(t-1) \cup a(t)$.

(P5) Priority loss: if there is a lost packet at time t, the lost packet is the one with the lowest priority among all the packets in $q(t-1) \cup a(t)$.

5.8.1 Complementary Priority Queues

To construct a priority queue, one needs to verify the five properties (P1)–(P5) in Definition 5.8.1. In the following, we introduce a complementary priority queue that reduces these five properties into two simple properties. As such, it is much easier to verify a construction of a complementary priority queue.

[1] This means that $a(t)$ is an empty set if there is no packet arriving at time t, and is a singleton otherwise.

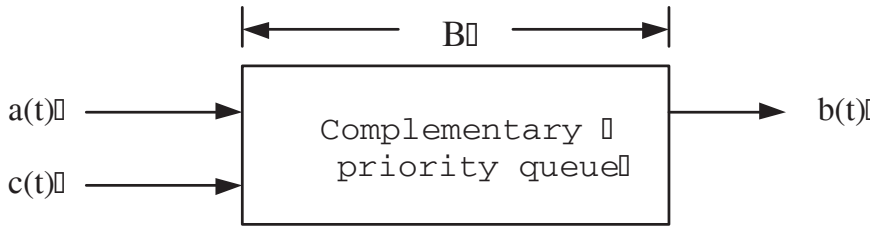


Fig. 5.65. A complementary priority queue.

Definition 5.8.2. (Complementary priority queue) *A complementary priority queue with buffer B is a network element that has one input link, one control input link, and one output link (see Figure 5.65). As in a priority queue, every packet is associated with a label, called* priority*, and there is a total order for the priorities. At time 0, there are B packets stored in the network element. Unlike a priority queue, there is always an arriving packet and a departing packet in every time slot. As shown in Figure 5.65, let $c(t)$ be the state of the control input, $a(t)$ be the set of the packet arriving at time t, $b(t)$ be the set of the packet departing at time t, and $q^c(t)$ be the set of packets queued at the complementary priority queue at time t (at the end of the t^{th} time slot). Then the complementary priority queue with buffer B satisfies the following two properties:*

(C1) Flow conservation: arriving packets from the input link are either stored in the buffer or transmitted through the output link, i.e.,

$$q^c(t) = q^c(t-1) \cup a(t) \backslash b(t). \tag{5.173}$$

(C2) Complementary priority departure: if $c(t) = 1$, then the departing packet is the one with the highest *priority among all the packets in $q^c(t-1) \cup a(t)$. On the other hand, if $c(t) = 0$, then the departing packet is the one with the* lowest *priority among all the packets in $q^c(t-1) \cup a(t)$.*

Clearly, a complementary priority queue and a priority queue are closely related. This is further clarified in Proposition 5.8.3 below.

Proposition 5.8.3. *As shown in Figure 5.66, a priority queue with buffer B can be constructed by a concatenation of a complementary priority queue with buffer B and a 1×2 switch.*

Proof. The key is to view empty time slots as *fictitious* packets that have priorities lower than those of real packets. Moreover, the prior-

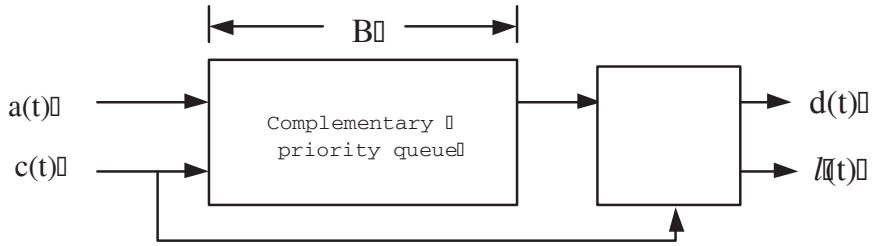


Fig. 5.66. A construction of a priority queue with buffer B via a concatenation of a complementary priority queue with buffer B and a 1×2 switch.

ities among the fictitious packets are decreasing in the order of their arrival times. As such, we have a total order among all the packets, including both the real packets and the fictitious packets. To emulate an empty priority queue at time 0, we can store B fictitious packets in the complementary priority queue.

Now we consider the following two cases.

Case 1. $c(t) = 1$:

In this case, we connect the input of the 1×2 switch to $d(t)$ in Figure 5.66. The complementary priority queue selects the packet with the highest priority among all the packets in $q^c(t-1) \cup a(t)$, where $q^c(t-1)$ is the set of packets stored in the complementary priority queue at the time $t-1$. If there is a real packet in $q^c(t-1) \cup a(t)$, then $d(t)$ contains a real packet as fictitious packets have priorities lower than those of real packets. Moreover, this packet is the one with the highest priority. Thus, (P2) and (P4) in Definition 5.8.1 are satisfied.

Case 2. $c(t) = 0$:

In this case, we connect the input of the 1×2 switch to $\ell(t)$ in Figure 5.66. The complementary priority queue selects the one with the lowest priority among all the packets in $q^c(t-1) \cup a(t)$. If there is a fictitious packet in $q^c(t-1) \cup a(t)$, then $\ell(t)$ contains a fictitious packet (an empty time slot) as fictitious packets have priorities lower than those of real packets. If there is no fictitious packet in $q^c(t-1) \cup a(t)$, then $\ell(t)$ contains a real packet and this real packet is the one with the lowest priority. Thus, (P3) and (P5) in Definition 5.8.1 are also satisfied. ■

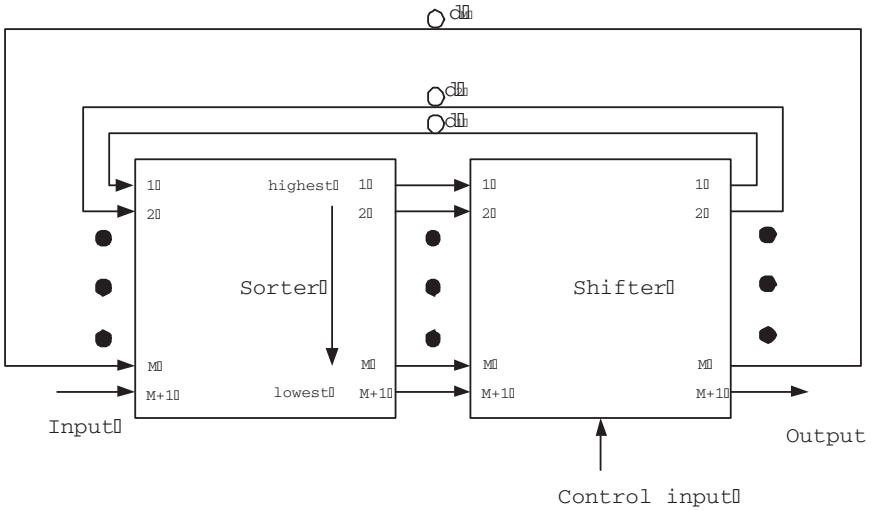


Fig. 5.67. A construction of a complementary priority queue with buffer $\sum_{i=1}^{M} d_i$.

5.8.2 Constructions of Complementary Priority Queues

In Figure 5.67, we show a construction of a complementary priority queue with buffer $\sum_{i=1}^{M} d_i$. In our construction, there are two $(M + 1) \times (M + 1)$ crossbar switches: a sorter (on the left hand side) and a shifter (on the right hand side). The key insight of our construction is based on the following assumption:

(A1) All the packets stored in all the fiber delay lines in Figure 5.67 cannot be either the packet with the highest priority or the packet with the lowest priority until they appear at the inputs of the sorter.

If (A1) holds, then the packets that appear at the inputs of the sorter contain both the packet with the highest priority and the packet with the lowest priority. The sorter then sorts the packets at the $M + 1$ inputs in the order of their priorities. By so doing, the first output from the sorter is the packet with the highest priority and the $(M + 1)^{th}$ output from the sorter is the packet with the lowest priority.

The $(M + 1) \times (M + 1)$ switch on the right hand side is a shifter that only has two connection patterns. When $c(t) = 0$, its connection pattern is realized by the $(M + 1) \times (M + 1)$ identity matrix, i.e., the matrix $I = (I_{ij})$ with $I_{i,j} = 1$ for $i = j$ and $I_{i,j} = 0$ for $i \neq j$. As such, the $(M+1)^{th}$ input of the shifter is connected to the $(M+1)^{th}$ output of

the shifter and the packet with the lowest priority is sent out from the output link. On the other hand, when $c(t) = 1$, its connection pattern is realized by the $(M+1)\times(M+1)$ circular-shift matrix, i.e., the matrix $P = (P_{ij})$ with $P_{i,j} = 1$ for $i = (j \bmod (M+1)) + 1$ and $P_{i,j} = 0$ otherwise. As such, the first input of the shifter is connected to the $(M+1)^{th}$ output of the shifter and the packet with the highest priority is sent out from the output link. Thus, the construction emulates a complementary priority queue if (A1) holds.

The M outputs of the shifter, indexed from $i = 1, \ldots, M$, are connected back to the corresponding M inputs of the sorter via M fiber delay lines with delays d_i, $i = 1, \ldots, M$. For a fiber delay line with delay d, there are d packets stored in that delay line. As such, there are $\sum_{i=1}^{M} d_i$ packets stored in the M fiber delay lines. The question is then how we choose d_i's so that the assumption in (A1) holds. This is answered in Proposition 5.8.4 below.

Proposition 5.8.4. *Suppose that (A1) holds at time 0. If* $0 < d_i \leq \min[i, M+1-i]$ *for all* $i = 1, 2, \ldots, M$*, then the construction in Figure 5.67 is a complementary priority queue with buffer* $\sum_{i=1}^{M} d_i$*.*

Proof. It suffices to argue by induction that (A1) holds for all time. Suppose that (A1) holds up to time $t-1$. As such, we have exactly emulated a complementary priority queue with buffer $\sum_{i=1}^{M} d_i$ up to time t. For both cases $c(t) = 0$ and $c(t) = 1$, we also know that the priorities of the packets at the M outputs of the shifter, indexed from $1, 2, \ldots, M$, are decreasing. Let us consider the packet at the i^{th} output of the shifter. Call this packet the tagged packet. For the tagged packet, there are $i-1$ packets that have priority higher than its priority and there are $M-i$ packets that have priority lower than its priority. As the construction departs a packet in a time slot, the tagged packet cannot be the packet with the highest priority or the packet with the lowest priority for the next $\min[i-1, M-i]$ time slots. As the delay of the i^{th} delay line d_i is less than or equal to $\min[i, M+1-i]$, it follows that the tagged packet cannot be either the packet with the highest priority or the packet with the lowest priority until it appears at the input of the sorter. Thus, the assumption (A1) holds at time t. ■

We first note that the purpose of having two $(M+1) \times (M+1)$ crossbar switches in our construction is for the ease of the presentation and the proof. In practice, one can combine these two switches into

one to reduce the hardware cost. Also, note that for $M = 2k - 1$, the maximum buffer size that can be achieved by Proposition 5.8.4 is to set $d_i = i$ for $i = 1, 2, \ldots, k$, and $d_i = 2k - i$ for $i = k+1, k+2, \ldots, 2k-1$. For this, we have buffer size $\sum_{i=1}^{2k-1} d_i = k^2$. And for $M = 2k$, the maximum buffer size that can be achieved by Proposition 5.8.4 is to set $d_i = i$ for $i = 1, 2, \ldots, k$, and $d_i = 2k+1-i$ for $i = k+1, k+2, \ldots, 2k$. For this, we have buffer size $\sum_{i=1}^{2k} d_i = k^2 + k$. One less efficient way, as originally proposed in [146], is to choose $M = 2k - 1$ and set $d_i = i$ for $i = 1, 2, \ldots, k$, and $d_i = 1$ for $i = k + 1, k + 2, \ldots, 2k - 1$, which gives buffer size $\sum_{i=1}^{k} d_i = (k^2 + k)/2$. As shown in Proposition 5.8.3, a complementary priority queue (in conjunction with a 1×2 switch) can be used for emulating a priority queue. Proposition 5.8.4 recovers the result in [146] as a special case. Finally, we note that if one would like to drop the arriving packet when the buffer is full, one can simply add a 1×2 switch at the input as discussed for FIFO queues in Section 5.6.

5.9 Notes

The development of optical packet switching via switched delay lines (SDL) starts from early 90's. It was first demonstrated by M. J. Karol [91], that SDL elements can be used as a buffer for a shared-memory optical packet switch. The buffer in [91] is built by SDL elements with feedbacks (like the optical memory cell in Section 5.1). However, no proofs were given for exact emulation of a shared-memory switch. The use of SDL elements for time slot interchanges in Section 5.2 seems to be introduced by M. J. Marcus [118] (as pointed out by N. Pippenger [140]). See also [141, 90] for subsequent developments. Pippenger called such a network a juggling network as packets are just like balls being thrown by a sequence of jugglers. A huge project (see [23, 24]), called CORD (contention resolution by delay lines), was started by I. Chlamtac et al in Boston University. Once again, no formal proofs for exact emulation of an output-buffered switch (or multiplexer) were given in [23, 24].

It seems that J. T. Tsai and R. L. Cruz [158, 55] are the first to construct an exact 2-to-1 First In First Out (FIFO) multiplexer with SDL elements. The multiplexer in [158, 55], named COD (Cascaded Optical Delay-Lines), only requires local information for the control of the connection patterns of 2×2 switches. However, the number of

2×2 switches in such an architecture is proportional to the buffer size. A more efficient design, called Logarithm Delay-Line Switch, is proposed by D. K. Hunter, M. C. Chia and I. Andonovic in [83]. The 2-to-1 FIFO multiplexer in [83] turns out to be the recursively expanded version of the 2-to-1 FIFO multiplexer presented in Section 5.3. As addressed in Section 5.3, the number of 2×2 switches needed for such an architecture is only $O(\log B)$, where B is the buffer size. In [82], SLOB (Switch with Large Optical Buffers) is proposed for the extension of optical buffered switches with N input/output ports ($N \geq 2$). Such an architecture also uses output buffer emulation and relies on a special hardware, called a primitive switching element (PSE). Each PSE itself is an $N \times 2N$ output-buffered switch with buffer size $N - 1$. Unlike the Logarithm Delay-Line Switch, the routing path of a packet in SLOB cannot be uniquely determined upon its arrival. This makes control of the PSEs much more difficult. In fact, a small control message must be transmitted electronically for each packet, representing the remaining delay over the remaining PSE's. Finally, we note that a "packing" and "scheduling" optical switch that uses the framed Birkhoff-von Neumann decomposition was introduced by E. A. Varvarigos [162].

Recursive constructions of buffered multiplexers in Section 5.3 and Section 5.4 were proved by C.-S. Chang, D.-S. Lee and C.-K. Tu in [37]. Our development of FIFO multiplexers with variable length bursts in Section 5.5 follows that in [38]. The construction of optical FIFO queues in Section 5.6 was originally proposed by C.-S. Chang, Y.-T. Chen, and D.-S. Lee in [32]. The connection of the classical switching theory and the construction of optical queues in Section 5.7 was established in the paper by C.-S. Chang, Y.-T. Chen, J. Cheng, and D.-S. Lee [31]. The construction of optical priority queues in Section 5.8 was first proposed by Sarwate and Anantharam [146]. The simple proof for such a construction in Section 5.8 was taken from the paper by H.-C. Chiu, C.-S. Chang, J. Cheng, and D.-S. Lee [46]. For additional references of optical packet switches, we refer to the review papers [81, 80, 169].

We note that optical packet switches are not as popular as optical crossconnects. Optical crossconnects are based on the technology of wavelength division multiplexing (WDM). In fact, up to the time of writing the book, most existing optical networks are optical crossconnects. There are several reasons for this: (i) it is not clear that

people need the granularity of packet switching at the level of optical networks, (ii) WDM (and Densed WDM) is a more economical technology than optical packet switching with SDL, (iii) it is technologically difficult to build optical buffers (even with the SDL elements), and (iv) developing a theory for optical switches with SDL elements is in general very challenging as it is switching (and scheduling) both in time and space. However, recent advances in optical technologies have demonstrated compact optical buffers using slow light in semiconductor nanostructures (see e.g., [42]). In the future, the construction of a scaled optical memory cell may not be as bulky as one might expect.

Problems

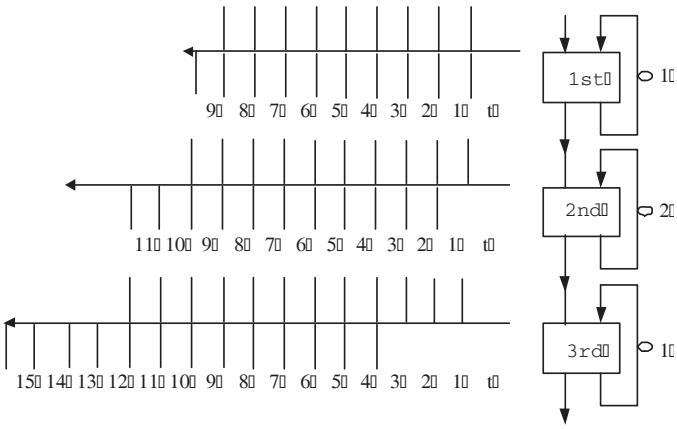


Fig. 5.68. A 4×4 Benes time slot interchange

1. In Figure 5.68, we show a 4×4 Benes time slot interchange. Consider the following permutation

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}.$$

   Find all the connection patterns in the 2×2 switches that realize π in Figure 5.68.
2. As in Example 5.2.2, find a feasible set of connection patterns in the 8×8 Benes time slot interchange for the following permutation matrix:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

3. Instead of designing sorters in space as addressed in Chapter 2, one can also design sorters in time. It is shown in Section 5.2.3 that a memory cell can be used for a 2×2 time slot interchange. Assume that a memory cell is also able to read and compare the "output addresses" of two incoming packets to the 2×2 switch in the memory cell. Show that one can use a concatenation of $\log_2 N$ (scaled) memory cells to design an $N \times N$ bitonic sorter with SDL elements.
4. Continue from the previous problem. Show that one can use a concatenation of $\frac{1}{2}\log_2 N(\log_2 N + 1)$ (scaled) memory cells to design an $N \times N$ Batcher sorting network with SDL elements.
5. Continue from the previous problem. Show that one can use a concatenation of $\frac{1}{2}\log_2 N(\log_2 N + 3)$ (scaled) memory cells to design an $N \times N$ Batcher-banyan network with SDL elements.
6. Use the optical Benes time slot interchange in Section 5.2.3 to design the Cantor network in Problem 20 of Chapter 2 with SDL elements.

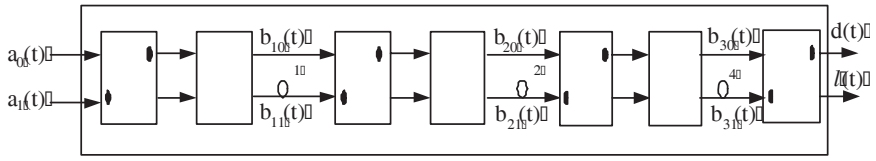


Fig. 5.69. A 2-to-1 multiplexer with $B = 7$.

7. In Figure 5.69, we show the construction for a 2-to-1 SDL multiplexer with buffer 7. Determine the connection patterns of the three switches at time t by the state variables $b_{11}(t-1), b_{21}(t-1), b_{21}(t-2), b_{31}(t-1), b_{31}(t-2), b_{31}(t-3)$ and $b_{31}(t-4)$.
8. Continue from the previous problem. Note that the state of the network element at time $t-1$ is $(b_{11}(t-1), q_1(t-1), q_2(t-1))$.

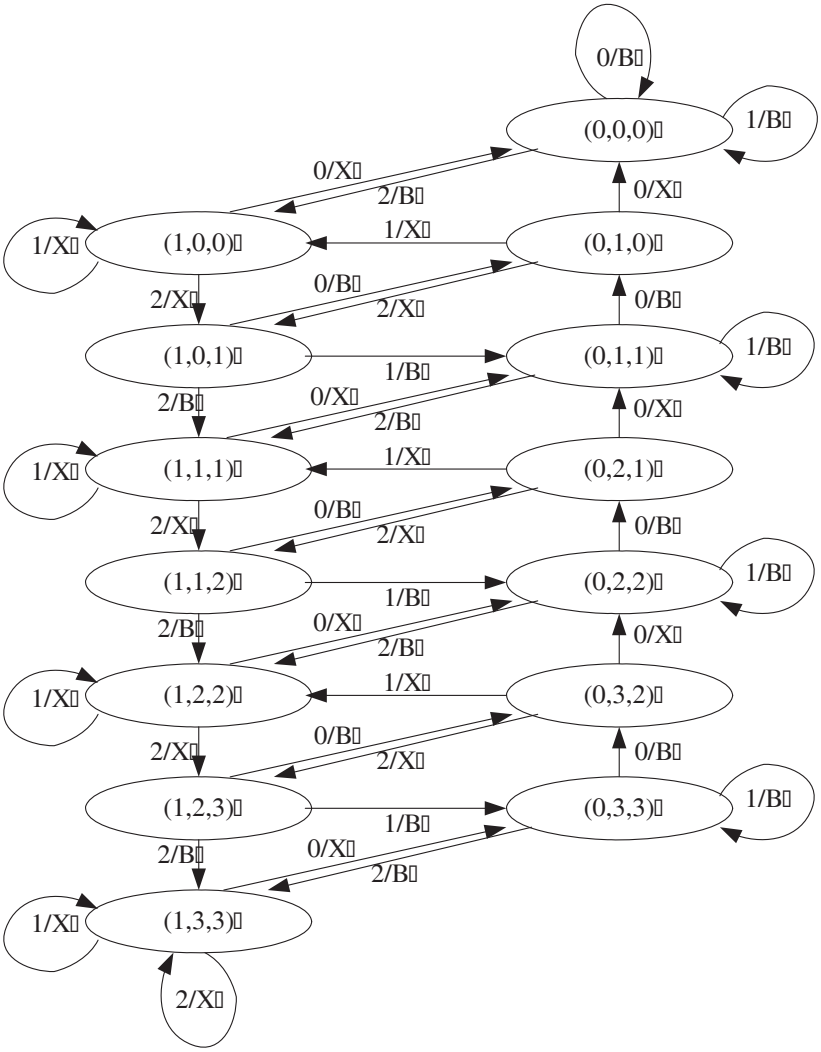


Fig. 5.70. The state transition diagram for $B = 7$.

where

$$q_1(t-1) = b_{21}(t-2) + b_{31}(t-4) + b_{31}(t-2)$$

and

$$q_2(t-1) = b_{21}(t-1) + b_{31}(t-3) + b_{31}(t-1).$$

Verify that the 2-to-1 multiplexer with buffer 7 has the state transition diagram in Figure 5.70.

9. Continue from the previous problem. Suppose that $a_0(t) = a_1(t) = 1$ for $t = 1, 2, \ldots, 7, 8$ and $a_0(t) = a_1(t) = 0$ from $t = 9$ onward. Trace the routes (with respect to time) for every packet that arrives at the buffered multiplexer.
10. Verify (5.47) in Example 5.4.7.

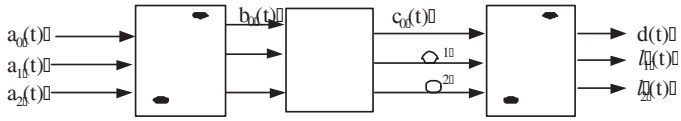


Fig. 5.71. A 3-to-1 delayed-loss multiplexer with buffer 2

11. In Figure 5.71, we show a 3-to-1 delayed-loss multiplexer with buffer 2. Suppose that the arrivals are described in Table 5.1.

time t	1	2	3	4	5
$a_0(t)$	1	1	1	0	0
$a_1(t)$	1	1	1	0	0
$a_2(t)$	0	1	1	0	0

Table 5.1. Arrivals to the 3-to-1 delayed-loss multiplexer with buffer 2

a) Find the connection patterns for the 3×3 switch (in the middle of Figure 5.71) for $t = 1, 2, 3$.
b) Find $d(t), \ell_1(t)$ and $\ell_2(t)$ for $t = 1, 2, 3, 4, 5$.

12. In Figure 5.72, we consider multiplexing variable length bursts over 3 links, i.e., $N = 3$. As shown in Figure 5.72, there are two bursts coming at time 1 and one burst coming at time 2. To break the tie, we choose the burst from the link $b_1(t)$ to be the first burst. As shown in this figure, now we have $(\tau_0, \ell_0) = (1, 3)$, $(\tau_1, \ell_1) = (1, 3)$ and $(\tau_2, \ell_2) = (2, 3)$. Assume that the buffer in the multiplexer is

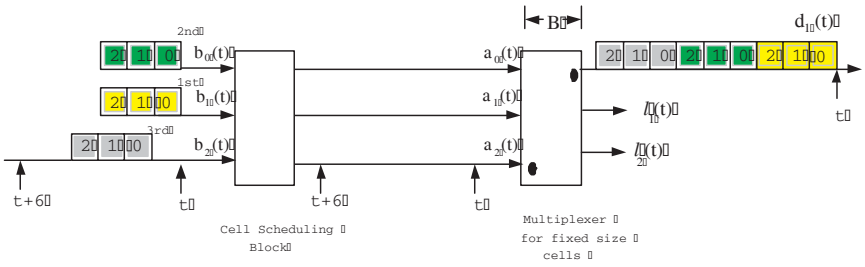


Fig. 5.72. Cell scheduling for variable length bursts

empty at time 1 and the buffer B is so large that no cells of these three bursts are lost. Initially, set $V_0(1) = V_1(1) = V_2(1) = 0$. Thus, $V(\tau_0^-) = V(1) = 0$.

a) Find $V(\tau_0^+)$.
b) Find $V(\tau_1^+)$.
c) Find $V(\tau_2^-)$.

13. Design an $N \times N$ output buffered switch using N-to-1 buffered SDL multiplexers in Section 5.4. Compute the complexity of your design in terms of the number of 2×2 switches. (Hint: need N N-to-1 buffered multiplexers and N $1 \times N$ switches.)
14. Design an $N \times N$ output buffered switch using non-overtaking delay lines. Compute the complexity of your design in terms of the number of 2×2 switches. (Hint: use the crosspoint buffer architecture in Section 2.6.1.)
15. Design an $N \times N$ output buffered switch using flexible delay lines. Compute the complexity of your design in terms of the number of 2×2 switches. (Hint: use the parallel buffer architecture in Section 2.6.2).
16. Design an optical Knockout switch (see Section 2.7.3) using SDL elements. (Hint: the fast Knockout concentrator/sorter in Figure 2.45 or the Knockout concentrator/sorter in Figure 2.51 can be built with SDL elements)
17. Consider an $N \times N$ load balanced Birkhoff-von Neumann switch with one-stage buffering in Section 3.1. For each central buffer, there are N Virtual Output Queues (VOQs) with each VOQ for each output. In Figure 5.73, there is an SDL element that is a concatenation of $1 \times N$ switch and a scaled N-to-1 (delayed-loss) multiplexer with buffer B and scaling factor N. The i^{th} output of the $1 \times N$ switch is connected to a delay line with delay i,

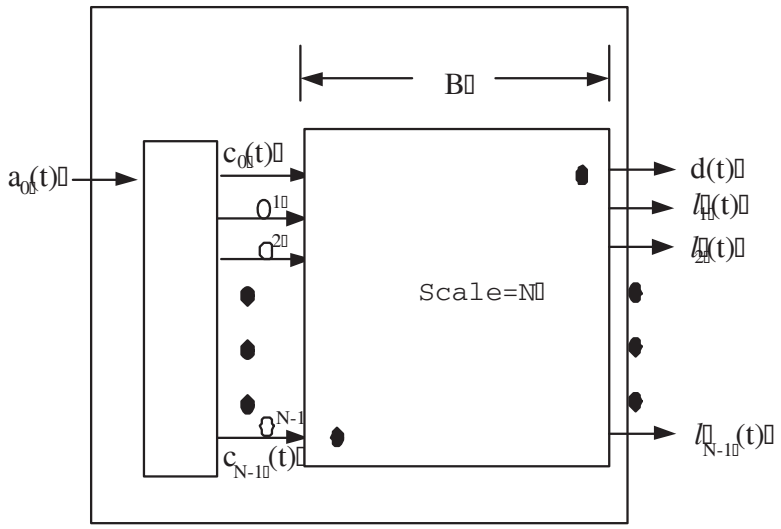


Fig. 5.73. A SDL construction of N VOQs in a central buffer of the load balanced Birkhoff von Neumann switch with one stage buffering

$i = 0, 1, 2 \ldots, N-1$. Find the operation rule for the $1 \times N$ switch so that the SDL element in Figure 5.73 can be operated as the N VOQs in a central buffer of an $N \times N$ load balanced Birkhoff-von Neumann switch with one-stage buffering. (Hint: the connection patterns of the two switch fabrics in the load balanced Birkhoff-von Neumann switch are periodic with the period N. If a packet is destined for a particular output port, it should be routed to the appropriate fiber delay line.)

18. Continue from the previous problem. Use the VOQs in the previous problem to design an $N \times N$ load balanced Birkhoff-von Neumann switch with one-stage buffering via SDL elements. Compare this with your design for the corresponding output-buffered switch in Problem 13. Without the uniform cost assumption for memory, is the load balanced Birkhoff-von Neumann switch still a good switch architecture?
19. The 4×4 time slot interchange in Figure 5.68 can be operated as a pre-FIFO queue with buffer 3. In Table 5.2, we consider the input pattern $a(t)$ and the control input pattern $c(t)$ for the pre-FIFO queue with buffer 3 for $t = 1, 2, \ldots, 8$. Find the connection patterns for the three 2×2 switches in Figure 5.68 for $t = 1, 2, \ldots, 8$.

time t	1	2	3	4	5	6	7	8
$a(t)$	1	1	1	0	1	0	0	0
$c(t)$	0	0	0	1	0	1	1	1
1st stage (Tail)								
2nd stage (Central)								
3rd stage (Head)								

Table 5.2. Arrivals for the FIFO queue with buffer 3

20. In Figure 5.58, we have shown a non-overtaking delay line with maximum delay $2^k - 1$. Note that there are two scaled optical memory cells with scaling factor 2^{k-1}. However, as the maximum delay is $2^k - 1$, it is impossible for a packet to route through the delay lines of these two scaled optical memory cells. Thus, we only need one of them!!! This implies that the 4×4 time slot interchange in Figure 5.68 can be operated as a non-overtaking delay line with maximum delay 3. In Table 5.3 (see also Table 5.4 for another representation), we consider the arrival times and departure times of four packets for the non-overtaking delay line with maximum delay 3.
    a) Let $\tau^c(n)$ be the departure time of the n^{th} packet from the linear compressor with maximum delay 3. Find $\tau^c(n)$ for $n = 1, 2, 3$ and 4.
    b) Find the connection patters for the three 2×2 switches for $t = 1, 2, \ldots, 8$.

Packets	1	2	3	4
$\tau^a(n)$	1	3	4	6
$\tau^d(n)$	3	4	6	8
$\tau^c(n)$				

Table 5.3. Arrivals and departures for the non-overtaking delay line with maximum delay 3

21. In Figure 5.74, we show a self-routing linear compressor with maximum delay 7. In Table 5.5 (see also Table 5.6 for another representation), we consider the arrival times and departure times of three packets for the linear compressor with maximum delay 7. Find the connection patters for the four 2×2 switches in Figure 5.74 for $t = 1, 2, \ldots, 7$.

time t	1	2	3	4	5	6	7	8
$a(t)$	1	0	1	1	0	1	0	0
$d(t)$	0	0	1	1	0	1	0	1
1st stage								
2nd stage								
3rd stage								

Table 5.4. Arrivals and departures for the non-overtaking delay lines with maximum delay 3

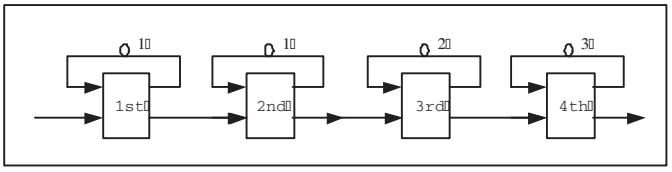


Fig. 5.74. A construction of a self-routing linear compressor with maximum delay 7 by a concatenation of optical memory cells

Packets	1	2	3
$\tau^a(n)$	1	4	6
$\tau^d(n)$	5	6	7

Table 5.5. Arrivals and departures for the linear compressor with maximum delay 7

time t	1	2	3	4	5	6	7
$a(t)$	1	0	0	1	0	1	0
$d(t)$	0	0	0	0	1	1	1
1st stage							
2nd stage							
3rd stage							
4rd stage							

Table 5.6. Arrivals and departures for the linear compressor with maximum delay 7

22. Show that a concatenation of memory cells (see Figure 5.38) can be used as an optical random access memory as in Figure 5.1.
23. Consider an N-to-1 priority queue with buffer B. For such a queue, there are N input links, one control input $c(t)$, and $N+1$ output links. The $N+1$ output links consist of one output link for departing packets and N output links for lost packets. The definition of an N-to-1 priority queue is basically the same as that in Definition 5.8.1. An N-to-1 priority queue also satisfies the flow conservation property in (P1), the non-idling property in (P2) and the priority departure property in (P4). The key difference is that there might be multiple packet losses in a time slot as there are multiple packet arrivals in a time slot. For this, one needs to modify the maximum buffer usage property in (P3) and the priority loss property in (P5). Let $c(t)$ be the state of the control input, $q(t)$ be the set of packets stored in the buffer at time t and $a(t)$ be the set of packets arriving at time t. If $|q(t-1) \cup a(t)| - c(t) > B$, then there are $|q(t-1) \cup a(t)| - c(t) - B$ lost packets at time t. When there are ℓ lost packets in a time slot (for some $\ell \geq 1$), these ℓ lost packets are selected from the ℓ lowest priority packets among the packets in $q(t-1) \cup a(t)$. Show that the construction in Figure 5.75 can be operated as an N-to-1 priority queue (Hint: all the packets stored in all the fiber delay lines in Figure 5.75 cannot be either the packet with the highest priority or one of the N lowest priority packets until they appear at the inputs of the sorter. See [46] for a detailed proof).
24. ($\mathcal{C}$-transform [47]) Consider an M-vector $\mathcal{D}_M = (d_1, d_2, \ldots, d_M)$ with $d_i \in \mathbf{N},\ i = 1, 2, \ldots, M$. Define a mapping $\mathcal{C} : \mathbf{N} \cup \{0\} \mapsto \{0,1\}^M$ as follows:

$$\mathcal{C}(x) = \Big(I_1(x), I_2(x), \ldots, I_M(x)\Big), \tag{5.174}$$

where

$$I_M(x) = \begin{cases} 1, & \text{if } x \geq d_M, \\ 0, & \text{otherwise,} \end{cases} \tag{5.175}$$

and for $i = M-1, M-2, \ldots, 1$, $I_i(x)$ is given recursively by

$$I_i(x) = \begin{cases} 1, & \text{if } x - \sum_{k=i+1}^{M} I_k(x) \cdot d_k \geq d_i, \\ 0, & \text{otherwise.} \end{cases} \tag{5.176}$$

Then $\mathcal{C}(x)$ is called the $\mathcal{C}$-transform of x with respect to $\mathcal{D}_M$. For the 5-vector

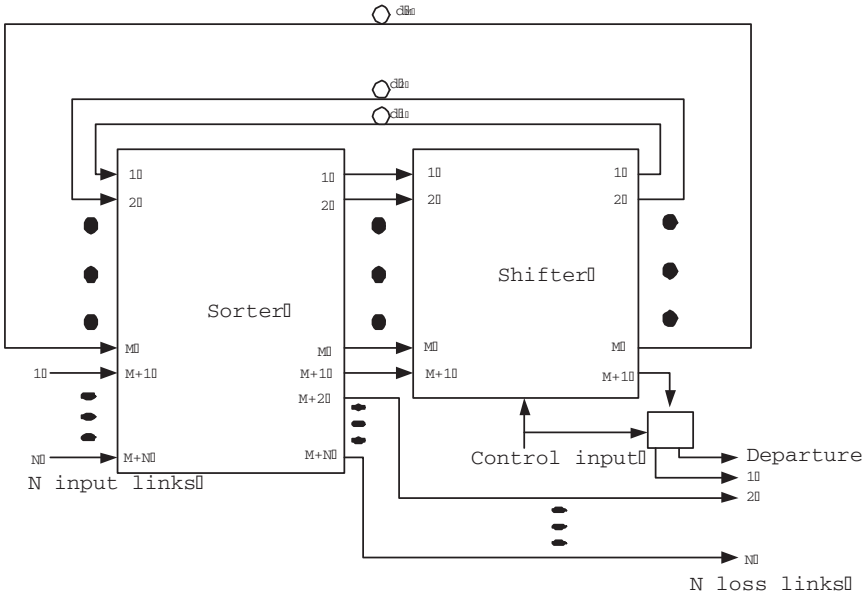


Fig. 5.75. A construction of an N-to-1 priority queue with buffer $\sum_{i=1}^{M} d_i$.

$$\mathcal{D}_5 = (1, 2, 3, 6, 10),$$

find the $\mathcal{C}$-transform of x for $0 \le x \le 22$.

25. ($\mathcal{C}$-transform as a projection) Continue from the previous problem. The inverse mapping $\mathcal{C}^{-1} : \{0,1\}^M \mapsto \mathbf{N} \cup \{0\}$ is defined as follows:

$$\mathcal{C}^{-1}\Big(\mathcal{C}(x)\Big) = \sum_{k=1}^{M} I_k(x) \cdot d_k. \tag{5.177}$$

The mapping $\mathcal{C}^{-1}\Big(\mathcal{C}(x)\Big)$ is called the inverse $\mathcal{C}$-transform. Suppose that $x, y \in \mathbf{N} \cup \{0\}$ and $1 \le i \le M$. Show the following properties for the $\mathcal{C}$-transform.

(i) $0 \le \sum_{k=i}^{M} I_k(x) \cdot d_k \le \mathcal{C}^{-1}(\mathcal{C}(x)) \le x$.

(ii) (Uniqueness) $\sum_{k=i}^{M} I_k(x) \cdot d_k = \sum_{k=i}^{M} I_k(y) \cdot d_k$ if and only if $I_k(x) = I_k(y),\ k = i, i+1, \ldots, M$.

(iii) (Monotonicity) If $0 \le x \le y$, then $\sum_{k=i}^{M} I_k(x) \cdot d_k \le \sum_{k=i}^{M} I_k(y) \cdot d_k$.

(iv) If $x = \sum_{k=i}^{M} I_k(y) \cdot d_k$ for some y, then $x = \sum_{k=i}^{M} I_k(x) \cdot d_k$.

26. (Complete decomposition) Continue from the previous problem. Assume that $d_1 = 1$, and $1 \le d_{i+1} \le \sum_{k=1}^{i} d_k + 1$, $i = 1, 2, \ldots, M-1$. Show that

$$x = \mathcal{C}^{-1}\Big(\mathcal{C}(x)\Big) = \sum_{k=1}^{M} I_k(x) \cdot d_k, \ \ for\ all\ 0 \le x \le \sum_{k=1}^{M} d_k.$$

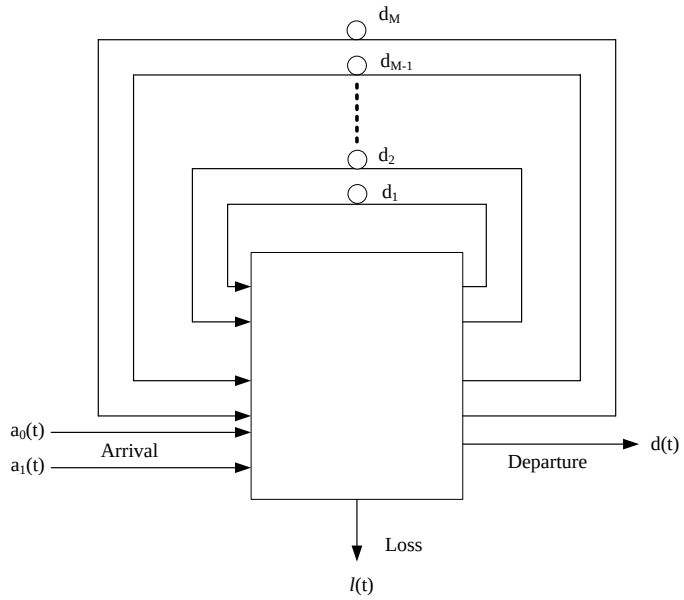


Fig. 5.76. A 2-to-1 multiplexer with buffer $\sum_{i=1}^{M} d_i$.

27. (2-to-1 multiplexer [47]) In Figure 5.76, we show a construction of a 2-to-1 multiplexer by a single switch with feedback. The feedback system consists of an $(M+2)\times(M+2)$ optical crossbar switch and M fiber delay lines with delays $d_1, d_2, \ldots, d_M$. The $(M+2)\times(M+2)$ optical crossbar switch is capable of realizing all the $(M+2)!$ permutations between its inputs and outputs. Among the $M+2$ outputs of the crossbar switch, two of them are used as the output port and the loss port of the 2-to-1 multiplexer. The remaining M outputs are connected to the M fiber delay lines with delays $d_1, d_2, \ldots, d_M$, respectively. Similarly, among the $M+2$ inputs of the crossbar switch, two of them are used as the two inputs of the 2-to-1 multiplexer. The remaining M inputs are connected to the M fiber delay lines that are fed back from the outputs. As the delay of a packet is known upon its arrival, packet delay can be used for routing. Suppose that the delay of a packet that arrives at time t is x. One first finds out a specific decomposition of x (using

the $\mathcal{C}$-transform of x in Problem 24) such that

$$x = d_{i_1} + d_{i_2} + \cdots + d_{i_k},$$

with $1 \le i_1 < i_2 < \cdots < i_k \le M$. Then the packet arriving at time t is routed to the delay line with delay d_{i_1} at time t, to the delay line with delay d_{i_2} at time $t + d_{i_1}, \ldots$, and to the delay line with delay d_{i_k} at time $t + \sum_{\ell=1}^{k-1} d_{i_\ell}$. Assume that $d_1 = 1$, and $d_i \le d_{i+1} \le 2d_i,\ i = 1, 2, \ldots, M-1$. Show that there is no collision at any fiber delay line at any time and the construction indeed achieves the exact emulation of a 2-to-1 FIFO multiplexer with buffer $\sum_{i=1}^{M} d_i$.

References

1. D. P. Agrawal, "Graph theoretical analysis and design of multistage interconnection networks," *IEEE Transactions on Computers*, Vol. 32, pp. 637-648, 1983.
2. R. Agrawal, R. L. Cruz, C. Okino, and R. Rajan, "Performance bounds for flow control protocols," April, 1998. Available from http://www.ece.wisc.edu/~agrawal.
3. H. Ahmadi and W. E. Denzel, "A survey of modern high-performance switching techniques," *IEEE Journal of Selected Areas in Communications*, Vol 7. pp. 1091-1103, 1989.
4. M. Ajmone Marsan, A. Bianco, P. Giaccone, E. Leonardi and F. Neri, "On the throughput of input-queued cell-based switches with multicast traffic," *Proceedings of IEEE INFOCOM*, 2001, pp. 1664-1672.
5. M. Ajtai, J. Komlos, and E. Szemeredi, "An $O(n \log n)$ sorting network," *Proceedings of the* 15^{th} *ACM Symposium on Theory of Computing*, pp. 1-9, 1983.
6. 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.
7. M. Andrews and M. Vojnovic, "Scheduling reserved traffic in input-queued switches: new delay bounds via probabilistic techniques," *Proceedings of IEEE INFOCOM*, 2003.
8. M. Andrews and L. Zhang, "Achieving stability in networks of input-queued switches," *Proceedings of IEEE INFOCOM*, 2001.
9. M. Avriel. *Nonlinear Programming: Analysis and Methods*, Prentice-Hall, 1976.
10. D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell and J. McManus, "Requirements for Traffic Engineering Over MPLS," Informational, RFC 2702, September, 1999.
11. F. Baccelli and P. Bremaud. *Elements of Queueing Theory*. New York: Springer-Verlag, 1994.
12. L. A. Bassalygo and M. S. Pinsker, "Complexity of optimal non-blocking switching networks without rearrangement," *Problems of Information Translations*, New York, Plenum, Vol. 9, pp. 64-66, 1974.
13. K. E. Batcher, "Sorting networks and their applications," *Proc. 1968 Spring Joint Comput. Conf.*, Vol. 32, pp. 307-314, 1968.
14. V. E. Benes. *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York: Academic Press, 1965.
15. C. Berge, *The Theory of Graphs and Its Applications* (translated by A. Doig). New York: Wiely, 1962.
16. J. C. Bermond, J. M. Fourneau, and A. Jean-Marie, "Equivalence of multistage interconnection networks," *Information Processing Letters*, Vol. 26, pp. 45-50, 1987.

17. G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Univ. Nac. Tucumán Rev. Ser. A*, Vol. 5, pp. 147-151, 1946.
18. R. Boorstyn, A. Burchard, J. Liebeherr, and C. Oottamakorn, "Effective Envelopes: Statistical Bounds on Multiplexed Traffic in Packet Networks," *Proceedings of IEEE INFOCOM*, 2000, Vol. 3, pp. 1223-1232, Tel Aviv, Israel.
19. J. Bucklew. *Large Deviation Techniques in Decision, Simulation and Estimation.* New York, NY: J. Wiley & Sons, Inc., 1990.
20. F. Callegati, "Approximate modeling of optical buffers for variable length packets," *Photonic Network Communications*, Vol. 3, pp. 383-390, 2001.
21. D. G. Cantor, "On nonblocking switching networks," *Networks*, Vol. 1, pp. 367-377, 1971.
22. I. Chlamtac and A. Fumagalli, "QUADRO-star: High performane optical WDM star networks," *Proceedings of IEEE GLOBAECOM'91*, Phoenix, AZ, Dec. 1991.
23. I. Chlamtac, A. Fumagalli, L. G. Kazovsky, P. Melman, W. H. Nelson, P. Poggiolini, M. Cerisola, A. N. M. M. Choudhury, T. K. Fong, R. T. Hofmeister, C. L. Lu, A. Mekkittikul, D. J. M. Sabido IX, C. J. Suh and E. W. M. Wong, "Cord: contention resolution by delay lines," *IEEE Journal on Selected Areas in Communications*, Vol. 14, pp. 1014-1029, 1996.
24. I. Chlamtac and A. Fumagalli, and C.-J. Suh, "Multibuffer delay line architectures for efficient contention resolution in optical switching nodes," *IEEE Transactions on Communications*, Vol. 48, pp. 2089-2098, 2000.
25. C.-S. Chang, "Stability, queue length and delay of deterministic and stochastic queueing networks," *IEEE Transactions on Automatic Control*, Vol.39, pp. 913-931, 1994.
26. C.-S. Chang, "On deterministic traffic regulation and service guarantees: a systematic approach by filtering," *IEEE Transactions on Information Theory*, Vol. 44, pp. 1097-1110, 1998.
27. C.-S. Chang. *Performance Guarantees in Communication Networks.* Springer-verlag: London, 2000.
28. C.-S. Chang, W.-J. Chen and H.-Y. Huang, "On service guarantees for input buffered crossbar switches: a capacity decomposition approach by Birkhoff and von Neumann," *Proceedings of IEEE IWQoS*, 1999, pp. 79-86, London, U.K.
29. C.-S. Chang, W.-J. Chen and H.-Y. Huang, "Birkhoff-von Neumann input buffered crossbar switches," *Proceedings of IEEE INFOCOM*, 2000, pp. 1614-1623, Tel Aviv, Israel.
30. C.-S. Chang, W.-J. Chen and H.-Y. Huang, "Birkhoff-von Neumann input-buffered crossbar switches for guaranteed-rate services," *IEEE Transactions on Communications*, vol. 49, issue 7, pp. 1145-1147, July 2001.
31. C.-S. Chang, Y.-T. Chen, J. Cheng, and D.-S. Lee, "Multistage constructions of linear compressors, non-overtaking delay lines, and flexible delay lines," *Proceedings of IEEE INFOCOM*, 2006.
32. C.-S. Chang, Y.-T. Chen, and D.-S. Lee, "Constructions of optical FIFO queues," joint special issue of *IEEE Transactions on Information Theory and IEEE/ACM Transactions on Networking*, Vol. 52, No. 6, p. 2838-2843, June 2006.
33. C.-S. Chang and D.-S. Lee, "Quasi-circuit switching and quasi-circuit switches," *unpublished manuscript.*
34. 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.

35. C.-S. Chang, D.-S. Lee and C.-M. Lien, "Load Balanced Birkhoff-von Neumann Switches, Part II: Multi-stage Buffering," *Computer Communications*, Vol. 25, pp. 623-634, 2002.
36. C.-S. Chang, D.-S. Lee, and Y.-J. Shih, "mailbox switch: a scalable two-stage switch architecture for conflict resolution of ordered packets," *Proceedings of IEEE INFOCOM*, 2004.
37. C.-S. Chang, D.-S. Lee and C.-K. Tu, "Recursive construction of FIFO optical multiplexers with switched delay lines," *IEEE Transactions on Information Theory*, Dec. 2004.
38. C.-S. Chang, D.-S. Lee and C.-K. Tu, "Using switched delay lines for exact emulation of FIFO multiplexers with variable length bursts," *IEEE Journal on Selected Areas in Communications*, Vol. 24, No. 4, p. 108-117, April 2006. conference version presented in *Proceedings of IEEE INFOCOM*, 2003.
39. C.-S. Chang, D.-S. Lee, and C.-L. Yu, "Generalization of the Pollaczek-Khinchin formula for throughput analysis of input-buffered switches," *Proceedings of IEEE INFOCOM*, 2005.
40. C.-S. Chang, D.-S. Lee, and C.-Y. Yue, "Providing guaranteed rate services in the load balanced Birkhoff-von Neumann switches," *IEEE Transactions on Networking*, Vol. 14, No. 3, p. 644-656, June 2006. conference version presented in *Proceedings of IEEE INFOCOM*, 2003.
41. C.-S. Chang and H.-J. Wang, "Large deviations for large capacity loss networks with fixed routing and polyhedral admission sets," *Discrete Event Dynamic Systems*, Vol. 7, pp. 391-418, 1997.
42. C. J. Chang-Hasnain, P.-C. Ku, J. Kim, and S.-L. Chuang, "Variable optical buffer using slow light in semiconductor nanostructures," *Proceedings of IEEE*, Vol. 91, No. 11, pp. 1884-1897, 2003.
43. H. J. Chao, C. H. Lam and E. Oki. *Broadband Packet Switching Technologies: A Practical Guide to ATM Switches and IP Routers*. John Wiley & Sons, Inc., 2001.
44. H. J. Chao, J. Song, N. S. Artan, G. Hu, and S. Jiang, "Byte-focal: a practical load-balanced switch," *preprint*.
45. A. Charny, P. Krishna, N. Patel and R. Simcoe, "Algorithms for providing bandwidth and delay guarantees in input-buffered crossbars with speedup," *IEEE IWQoS'98*, pp. 235-244, Napa, California, 1998.
46. H.-C. Chiu, C.-S. Chang, J. Cheng, and D.-S. Lee, "A simple proof for the constructions of optical priority queues," *QUEUEING SYSTEMS: Theory and Applications*, Vol. 56, p. 73-77, June 2007.
47. C.-C. Chou, C.-S. Chang, D.-S. Lee, and J. Cheng, "A necessary and sufficient condition for the construction of 2-to-1 optical FIFO multiplexers by a single crossbar switch and fiber delay lines," *IEEE Transactions on Information Theory*, Vol. 52, No. 10, pp. 4519–4531, October 2006.
48. G. L. Choudhury and W. Whitt, "Long-tail buffer-content distributions in broadband networks," *Performance Evaluation*, Vol. 30, pp. 177-190, 1997.
49. Y. S. Chow and H. Teicher, *Probability Theory: Independence, Interchangeability, Martingales*. New York: Springer-Verlag, 1988.
50. S.-T. Chuang, A. Goel, N. McKeown and B. Prabhakar, "Matching output queueing with a combined input output queued switch," *Proceedings of IEEE INFOCOM*, 1999, pp. 1169-1178, New York.
51. C. Clos, "A study of nonblocking switching networks," *BSTJ*, Vol. 32, pp. 406-424, 1953.
52. T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley & Sons, 1991.

53. R. L. Cruz, "A calculus for network delay, Part I: Network elements in isolation," *IEEE Tran. Inform. Theory*, Vol. 37, pp. 114-131, 1991.
54. R. L. Cruz, "A calculus for network delay, Part II: Network analysis," *IEEE Transactions on Information Theory*, Vol. 37, pp. 132-141, 1991.
55. R. L. Cruz and J. T. Tsai, "COD: alternative architectures for high speed packet switching," *IEEE/ACM Transactions on Networking*, Vol. 4, pp. 11-20, February 1996.
56. J. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," *Proceedings of IEEE INFOCOM*, 2000, pp. 556-564, Tel Aviv, Isreal.
57. A. Dembo and O. Zeitouni. *Large Deviations Techniques and Applications*. Boston: Jones and Barlett Publishers, 1992.
58. A. Demers, S. Keshav, and S. Shenkar, "Analysis and simulation of a fair queueing algorithm," *Proceedings of SIGCOMM*, 1989, pp. 1-12, Austin, TX.
59. A. M. Duguid. *Structural properties of switching networks*. Progr. Rep. BTL-7, Brown University, 1959.
60. L. Dulmage and I. Halperin, "On a theorem of Frobenius-König and J. von Neumann's game of hide and seek, *Trans. Roy. Soc. Canada III(3)*, Vol. 49, pp. 23-29, 1955.
61. R. S. Ellis. *Entropy, Large Deviations, and Statistical Mechanics*. New York: Spinger-Verlag, 1985.
62. A.I. Elwalid and D. Mitra, "Effective bandwidth of general Markovian traffic sources and admission control of high speed networks," *IEEE/ACM Transactions on Networking*, Vol. 1, pp. 329-343, 1993.
63. A. Elwalid, D. Mitra and R. Wentworth, "A new approach for allocating buffers and bandwidth to heterogeneous, regulated traffic in an ATM node," *IEEE Journal on Selected Areas in Communications*, Vol. 13, pp. 1115-1127, 1995.
64. O. Gabber and Z. Galil, "Explicit construction of linear size superconcentrators," *J. Comput. Syst. Sci.,* Vol. 22, pp. 407-420, 1981.
65. J. Gärtner, "On large deviations from invariant measure," *Theory Probab. Appl.*, Vol. 22, pp. 24-39, 1977.
66. P. Gazdzicki, I. Lambadaris, and R. R. Mazumdar, "Blocking probabilities for large multirate Erlang loss systems," *Adv. Appl. Probab.*, Vol. 25, pp. 997–1009, 1993.
67. S. J. Golestani, "Congestion-free communication in high speed packet networks," *IEEE Transactions on Communications*, Vol. 39, pp. 1802-1812, 1991.
68. S. J. Golestani, "Duration-limited statistical multiplexing of delay sensitive traffic in packet networks," *Proceedings of IEEE INFOCOM*, 1991, pp. 323-332.
69. S. J. Golestani, "Network delay analysis of a class of fair queueing algorithms," *IEEE Journal on Selected Areas of Communications*, Vol. 13, pp. 1057-1076, 1995.
70. P. Gupta, S. Lin and N. McKeown, "Routing lookups in hardware at memory access speeds," *Proceedings of IEEE INFOCOM*, 1998.
71. P. Hall, "Distinct representatives of subsets," *J. London Math. Soc.*, Vol. 10, pp. 26-30, 1932.
72. R. Händel, M. N. Huber, S. Schröder. *ATM Networks: Concepts, Protocols, Applications*. Addison-Wesley Publishers Ltd., 1994.
73. A. Huang and S. Knauer, "Starlite: a wideband digital switch," *Proc. Globecom*, 1984.
74. N.-F. Huang and S. M. Zhao, "A fast IP routing lookup scheme for gugabit switch routers," *IEEE Journal on Selected Areas in Communications*, pp. 1093-1104, 1999.

75. J. Y. Hui, "Resource allocation for broadband networks," *IEEE Select. Areas Commun.*, Vol. 6, pp. 1598-1608, 1988.
76. J. Y. Hui. *Switching and Traffic Theory for Integrated Broadband Networks*, Boston: Kluwer Academic Publishers, 1990.
77. J. Y. Hui and E. Arthurs, "A broadband packet switch for integrated transport," *IEEE Journal on Selected Areas of Communications*, Vol. 5, 1987.
78. J. Y. Hui and T. Renner, "Queueing strategies for multicast packet switching," *IEEE GLOBECOM'90*, Vol. 3, pp. 1431-1437, 1990.
79. A. Hung, G. Kesidis and N. McKeown, "ATM input-buffered switches with guaranteed-rate property," *Proc. IEEE ISCC'98*, Athens, pp. 331-335, 1998.
80. D. K. Hunter and I. Andonovic, "Approaches to optical Internet packet switching," *IEEE Communication Magazine*, Vol. 38, pp. 116-122, 2000.
81. D. K. Hunter, M. C. Chia and I. Andonovic, "Buffering in optical packet switches," *IEEE Journal of Lightwave Technology*, Vol. 16, pp. 2081-2094, 1998.
82. D. K. Hunter, W. D. Cornwell, T. H. Gilfedder, A. Franzen and I. Andonovic, "SLOB: a switch with large optical buffers for packet switching," *IEEE Journal of Lightwave Technology*, Vol. 16, pp. 1725-1736, 1998.
83. D. K. Hunter, D. Cotter, R. B. Ahmad, D. Cornwell, T. H. Gilfedder, P. J. Legg and I. Andonovic, "2×2 buffered switch fabrics for traffic routing, merging and shaping in photonic cell networks," *IEEE Journal of Lightwave Technology*, Vol. 15, pp. 86-101, 1997.
84. F. K. Hwang. *The Mathematical Theory of Nonblocking Switching Networks*, Singapore: World Scientific Publishing Co., 1998
85. T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, Vol. 27, pp. 1449-1455, 1979.
86. S. Iyer, A. Awadallah and N. McKeown, "Analysis of a packet switch with memories running at slower than line speed," *Proceedings of IEEE INFOCOM*, 2000.
87. S. Iyer and N. McKeown, "Making parallel packet switch practical," *Proceedings of IEEE INFOCOM*, 2001, Anchorage, Alaska, U.S.A.
88. J.-J. Jaramillo, F. Milan, and R. Srikant, "Padded frames: a novel algorithm for stable scheduling in load-balanced switches," *Technical Report*, University of Illinois at Urbana-Champaign, 2005.
89. P. R. Jelenković and A. A. Lazar, "Subexponential asymptotics of a Markov-modulated random walk with queueing applications," *J. Appl. Prob.*, June, 1998.
indexLee, K. Y.
90. H. F. Jordan, D. Lee, K. Y. Lee, S. V. Ramanan, "Serial array time slot interchangers and optical implementations," *IEEE Transactions on Computers*, ,Vol. 43 , No. 11 , pp. 1309-1318, 1994.
91. M. Karol, "Shared-memory optical packet (ATM) switch," SPIE Vol. 2024 Multigigabit Fiber Communications Systems, pp. 212-222, 1993.
92. M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input Versus Output Queueing on a Space-Division Packet Switch," *IEEE Transactions on Communications*, Vol. COM35, NO.12, Dec. 1987.
93. M. J. Karol and M. G. Hluchyj, "The Knockout Packet Switch: Principles and Performance," *Proceedings of the 12th Conf. on Local Computer Nets*, Minneapolis, MN, Oct. 1987: 16-22.
94. G. Kesidis, J. Walrand and C.-S. Chang, "Effective bandwidths for multiclass Markov fluids and other ATM sources," *IEEE/ACM Transactions on Networking*, Vol. 1, pp. 424-428, 1993.
95. I. Keslassy, "The load-balanced router," *PhD Thesis. Stanford University*, 2004.

96. I. Keslassy, C.-S. Chang, N. McKeown, D.-S. Lee, "Optimal load balancing," *Proceedings of IEEE INFOCOM*, 2005.
97. I. Keslassy, S.-T. Chuang and N. McKeown, "A load-balanced switch with an arbitrary number of linecards," *Proceedings of IEEE INFOCOM*, 2004.
98. I. Keslassy and N. McKeown, "Maintaining packet order in two-stage switches," *Proceedings of IEEE INFOCOM*, 2002.
99. I. Keslassy, S.-T. Chung, K. Yu, D. Miller, M. Horowitz, O. Slogaard, N. McKeown, "Scaling Internet routers using optics," *Proceedings of ACM SIGCOMM*, 2003, Karlsruhe, Germany.
100. I. Keslassy, M. Kodialam, T. V. Lakshman, D. Stiliadis, "On guaranteed smooth scheduling for input-queued switches," *Proceedings of IEEE INFOCOM*, 2003.
101. C. Kolias and L. Kleinrock, "Throughput analysis for multiple input-queueing in ATM switches," in *Broadband Communications*, L. Mason and A. Casaca, Eds, Chapman & Hall, London, U.K., pp. 382-393, 1996.
102. P. Krishna, N. S. Patel, A. Charny and R. Simcoe, "On the speedup required for work-conserving crossbar switches," *IEEE IWQoS'98*, pp. 225-234, Napa, California, 1998.
103. J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison Wesley Longman, Inc., 2001.
104. J. Y. Le Boudec, "Application of network calculus to guarantee service networks," *IEEE Transactions on Information Theory*, Vol. 44, pp. 1087-1096, 1998.
105. J. Y. Le Boudec and P. Thiran. *Network Calculus*. Springer Verlag, Lecture Notes in Computer Science, LNCS 2050.
106. H. Y. Lee, F. K. Hwang and J. Capinelli, "A new decomposition algorithm for rearrangeable Clos interconnection networks," *IEEE Transactions on Communications*, Vol. 44, pp. 1572-1578, 1997.
107. T. T. Lee, "Non-blocking copy networks for multicast packet switching," *IEEE Journal of Selected Areas on Communications*, Vol. 6, pp. 1455-1467, 1988.
108. T. T. Lee and C. H. Lam, "Path switching-a quasi-static routing scheme for large scale ATM packet switches," *IEEE Journal on Selected Areas of Communications*, Vol. 15, pp. 914-924, 1997.
109. W. E. Leland, M. S. Taqqu, W. Willinger and D.V. Wilson, "On the self-similar Nature of Ethernet Traffic," *IEEE/ACM Transactions on Networking*, Vol. 2, pp. 1-15, 1994.
110. S. Li and N. Ansari, "Input-queued switching with QoS guarantees," *Proceedings of IEEE INFOCOM*, 1999, pp. 1152-1159, New York, 1999.
111. S.-Y. R. Li. *Algebraic Switching Theory and Broadband Applications*. Academic Press, 2001.
112. S.-Y. R. Li and C.-M. Lau, "Concentrators in ATM switching," *Comp. Sys. Sci. Eng.*, Vol. 6, pp. 335-342, 1996.
113. Y. Li, S. Panwar and H. J. Chao, "On the performance of a dual round-robin switch," *Proceedings of IEEE INFOCOM*, 2001, pp. 1688-1697.
114. D. V. Lindley, "The theory of queues with a single server," *Proc. Camb. Phil. Soc.*, Vol. 48, pp. 277-289, 1952.
115. J. D. C. Little, "A proof for the queueing formula $L = \lambda W$," *Operations Research*, Vol. 16, pp. 651-665, 1961.
116. R. M. Loynes, "The stability of a queue with non-independent inter-arrival and service times," *Proc. Camb. Phil. Soc.*, Vol. 58, pp. 497-520, 1962.
117. J. van Lunteren and T. Engbersen, "Fast and scalable packet classification," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 560-571, May 2003.

118. M. J. Marcus, "Designs for time slot interchangers," *Proceedings of the National Electronics Conf.*, Vol. 26, pp. 812-817, 1970.
119. G. A. Marguis, "Explicit constructions of concentrations," *Problems Peredachi Informatsii*, Vol. 9, No. 4, 1973. (In English: Problems of Information Translations, New York, Plenum, 1975).
120. A. W. Marshall and I. Olkin, *Inequalities: Theory of Majorization and Its Applications*. New York: Academic Press, 1979.
121. N. McKeown, "Scheduling algorithms for input-queued cell switches," *PhD Thesis. University of California at Berkeley*, 1995.
122. N. McKeown, "iSLIP: A Scheduling Algorithm for Input-Queued Switches," *IEEE Transactions on Networking*, Vol 7, No.2, April 1999.
123. N. McKeown, V. Anantharam and J. Walrand, "Achieving 100% throughput in an input-queued switch," *Proceedings of IEEE INFOCOM*, 1996, pp. 296-302, 1996.
124. A. Mekkittikul and N. McKeown, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," *Proceedings of IEEE INFOCOM*, 1998.
125. L. Mirsky, *Transversal Theory*. New York: Academic Press, 1971.
126. D. Mitra and R. A. Cieslak, "Randomized parallel communications on an extension of the omega network," *Journal of the Association for Computing Machinery*, Vol. 34, No. 4, pp. 802-824, 1987.
127. M. G. Nadkarni. *Basic Ergoidc Theory*. Berlin: Birkhaüser, 1998.
128. R. Nelson, "Stochastic catastrophe theory in computer performance modeling," *Journal of the Association for Computing Machinery*, Vol. 34, pp. 661-685, 1987.
129. R. Nelson, *Probability, Stochastic Processes, and Queueing Theory: the Mathematics of Computer Performance Modeling*. Springer-Verlag: New York, 1995.
130. A. G. Pakes, "On the tail of waiting-time distribution," *J. Appl. Prob.*, Vol. 12, pp. 555-564, 1975.
131. C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithm and Complexity*. New Jersey: Prentice-Hall, 1982.
132. A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated service networks: the single-node case," *IEEE/ACM Transactions on Networking*, Vol. 1, pp. 344-357, 1993.
133. A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated service networks: the multiple node case," *IEEE/ACM Transactions on Networking*, vol. 2, pp. 137-150, 1994.
134. A. Pattavina. *Switching Theory: Architecture and Performance in Broadband ATM Networks*. New York: John Wiley & Sons Ltd., 1998.
135. M. C. Paull, "Reswitching of connection networks," *Bell Syst. Tech. J.*, Vol. 41, pp. 833-855, 1962.
136. T.B. Pei and C. Zukowski, "VLSI implementation of routing tables: tries and CAMs," *Proceedings of the IEEE INFOCOM*, 1991.
137. K. Petersen. *Ergodic Theory*. Cambridge: University Press, 1983.
138. L. L. Peterson and B. Davie. *Computer Networks: A Systems Approach*. San Francisco: Morgan Kaufmann Publishers, 2000.
139. M. S. Pinsker, "On the complexity of a concentrator," *Proceedings of the Seventh International Teletraffic Congress*, Stockholm, 1973.
140. N. Pippenger, "Juggling networks," *Canada-France Conference on Parallel and Distributed Computing*, pp. 1-12, 1994.
141. S. V. Ramanan, H. F. Jordan, J. R. A. Sauer, "A new time domain, multistage permutation algorithm," *IEEE Transactions on Information Theory*, Vol. 36 , No. 1 , pp. 171-173, 1990.

142. M. R. N. Ribeiro and M. J. O'Mahony "Traffic management in photonic packet switching nodes by priority assignment and selective discarding," *Computer Communications*, Vol 24, pp. 1689-1701, 2001.
143. R. T. Rockafeller. *Convex Analysis*. Princeton: Princeton University Press, 1970.
144. K. W. Ross. *Multiservice Loss Models for Broadband Telecommunication Networks*. Springer-Verlag: New York, 1995.
145. S. M. Ross, *Stochastic Processes*. New York: J. Wiley & Sons, 1983.
146. A. D. Sarwate and V. Anantharam, "Exact emulation of a priority queue with a switch and delay lines," to appear in *Queueing Systems Theory and Applications*, 2005.
147. M. Schwartz. *Broadband Integrated Networks*. Prentice Hall, 1996.
148. C. E. Shannon, "A mathematical theory of communication," *Bell System Tech. J.*, Vol. 27, (pt. I) pp. 379-423, (pt. 2) pp. 623-656, 1948.
149. S. Shenker, C. Patridge and R. Guerin, "Specification of guaranteed quality of service," (IETF RFC 2212) ftp://ds.internic.net/rfc/rfc2212.txt, 1997.
150. A. Shwartz and A. Weiss. *Large Deviations for Performance Analysis: Queues, Communication and Computing.* 1994.
151. D. Slepian, "Two theorems on a particular crossbar switching network," unpublished BTL memo.
152. D. Stiliadis and A. Varma, "Providing bandwidth guarantees in an input-buffered crossbar switch," *Proceedings of IEEE INFOCOM*, 1995, pp. 960-968.
153. I. Stoica and H. Zhang, "Exact emulation of an output queueing switch by a combined input output queueing switch," *Proceedings of IEEE IWQoS*, 1998, pp. 218-224, Napa, California.
154. D. Stoyan, *Comparison Methods for Queues and Other Stochastic Models*, Berlin: J. Wiley & Sons, 1983.
155. 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.
156. L. Tancevski, S. Yegnanarayanan, G. Castanon, et al. "Optical routing of asynchronous, variable length packets" *Journal on Selected Areas in Communications*, Vol. 18, pp. 2084-2093, 2000.
157. G. Thomas, "Bifurcated queueing for throughput enhancement in input-queued switches," *IEEE Commun. Lett.*, Vol. 1, pp. 56-57, 1997.
158. J. T. Tsai, "COD: architectures for high speed time-based multiplexers and buffered packet switches," Ph.D. Dissertation, University of California, San Diego, 1995.
159. C.-Y. Tu, C.-S. Chang, D.-S. Lee, and C.-T. Chiu, "Design a simple and high performance switch using the two-stage architecture," *Proceedings of Globecom*, 2005, St. Louis, Missouri.
160. J. S. Turner, "Terabit burst switching," *Journal of High Speed Networks*, 1999.
161. L. G. Valiant, "A scheme for fast parallel communication," *SIAM J. Comput.*, Vol. 11, No. 2, pp. 350-361, 1982.
162. E. A. Varvarigos, "The 'Packing' and 'Scheduling' switch architectures for almost-all optical lossless networks," *IEEE Journal of Lightwave Technologies*, vol. 16 (no. 10), pp. 1757-67, Oct. 1998.
163. E. A. Varvarigos and V. Sharma, "An efficient reservation connection control protocol for gigabit networks, " *Computer Networks and ISDN Systems* ,vol. 30, (no. 12), 13 July 1998, pp. 1135-1156.
164. J. von Neumann, "A certain zero-sum two-person game equivalent to the optimal assignment problem, " *Contributions to the Theory of Games,* Vol. 2, pp. 5-12, Princeton University Press, Princeton, New Jersey, 1953.

165. S. X. Wai and V.P. Kumar, "On the multiple shared memory module approach to ATM switching," *Proceedings of IEEE INFOCOM*, 1992, pp. 116-123, Florence, Italy.
166. W. Whitt, "Tail probability with statistical multiplexing and effective bandwidths in multi-class queues," *Telecommunication Systems*, Vol. 2, pp. 71-107, 1993.
167. C.-L. Wu and T.-Y. Feng, "On a class of multistage interconnection networks," *IEEE Transactions on Computers*, Vol. 29, pp. 694-702, 1980.
168. C.-L. Wu and T.-Y. Feng, "The university of the shuffle-exchange networks," *IEEE Transactions on Computers*, Vol. 30, pp. 324-332, 1981.
169. S. Yao, B. Mukherjee, and S. Dixit, "Advances in photonic packet switching: An overview," *IEEE Communication Magazine*, Vol. 38, pp. 84-94, 2000.
170. Y. S. Yeh, M. G. Hluchyj, and A. S. Acampora, "The Knockout switch: a simple, modular architecture for high-performance packet switching," *IEEE Journal of Selected Areas in Communications*, Vol. SAC-5, pp. 1274-1283, 1987.
171. M. Yoo, C. Qiao and S. Dixit, "QoS performance of optical burst switching in IP-over-ATM networks," *IEEE Journal on Selected Areas in Communications*, Vol. 18, pp. 2062-2071, 2000.
172. K. Y. Yun, K. W. James, R. H. Fairlie-Cuninghame, S. Chakraborty, and R. L. Cruz, "A self-timed real-time sorting network," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 8, pp. 356-363, 2000.

Index