

A Permutation Engine Switching Node

S. P. Monacos

Center for Space Microelectronics Technology
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91109

and

A, A. Sawchuk

Signal and Image Processing Institute
Department of Electrical Engineering
University of Southern California
Los Angeles, CA. 90089-2564

Submitted to Dr. Allan Gottlieb

PE Switching Node

Steve P. Monacos
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, California 91109
MS 300-329
(818)354-9161

Abstract

We describe an asynchronous, strictly non-blocking crossbar node topology and distributed routing algorithm that is particularly suited to optoelectronic implementation. The node has: i) asynchronous, strictly non-blocking performance; ii) constant delay routing of any input to any output; iii) a simple distributed routing control mechanism based strictly on the desired destination (i.e. any permutation of destination addresses in time and input position will route to the desired output port); iv) wrap-around queuing capability to handle blocking when two or more packets want to route to the same output port; and v) an easily scalable architecture. The crossbar node operates in a bit-synchronous, packet-asynchronous mode (incoming packets need only be reclocked for bit alignment but not packet alignment). The distributed control and switching hardware for this crossbar may be built using current electronic and/or optical/electrooptic technology. We call the fundamental component of this system a permutation engine (PE) due to its ability to sort properly any permutation of (destination addresses at the input ports before reaching the output ports,

List of Symbols

k - kay

??2 - *cm*

?1 - *cn*

N - *en*

0-- zero

1 - one

K - kay

7' - tee

1-- ic

O -oh

c -- Sec

r -- are

d - dcc

i -- ic

3.. jay

th - eth

α -- alpha

l - ic

l -- ell

s - C_{ss}

14 - you

v -- vcc

Optical interconnection networks offer the potential for bandwidth in the tens to hundreds of gigabits/sec over a given data path. The primary limitation of optical systems is that complex logic and data buffering is difficult to perform in the optical domain. Our goal is to maintain the data in all-optical form as much as possible with a minimum of electronic/optical conversions.

In this paper we describe a switching *node* architecture that is suitable for optical implementation because routing decisions are simple, local operations, and buffering is done by fiber loops and fiber inter-node connections. The *permutation engine* (PE) described in section 3 is the fundamental building block for the switching node described in section 4, and consists of several stages of bypass/exchange switches and associated internal connections. The switching node in section 4 is composed of a cascade of PE modules. The PE modules can route packet asynchronous traffic from input ports to output ports without internal contention, and have constant latency (delay) regardless of the path taken from input/output. We begin by defining some relevant terms used in this work,

1.1 Terminology

We define a data packet as an indivisible time slot of information consisting of an N -bit binary packet header; the payload or data for this header; and a trailer marking the end of the packet. The header contains the address of the desired destination port represented as a binary number with the most significant bit (MSB) first. A typical packet format is shown in Fig. 1.

A *packet asynchronous* system does not require that packet headers (and the packet themselves) be synchronized in time. While the architectures described here are packet asynchronous, they do require *bit synchronous* data, i.e., *bits flowing* through the network are clocked synchronously through the nodes and PE modules.

Output contention occurs when packets within a PE or node simultaneously require the same output port. A PE or node is called *rearrangably non-blocking* if any permutation of the destination headers of n simultaneous packets at the input ports are properly routed to the desired output ports in the absence of contention. An n input network with one input/output port connection in use is defined as *asynchronous, strictly non-blocking* if any of the remaining $n-1$ packet headers are properly

routed to the comet output ports in the absence of output contention. In this case, packet asynchronous inputs will always route to the proper output ports (as specified by their headers), regardless of the input port used and the time at which the packet enters the network. The fundamental switching element of the PE is a *bypass/exchange* switch, having two inputs and two outputs. The two states of the bypass/exchange switch are shown in Fig. 2.

The remainder of this paper is organized as follows. Section 2 reviews various network architectures and their basic properties. Section 3 discusses the basic permutation engine (PE) module topology and its fundamental routing procedure, called routing algorithm A. Section 4 discusses the use of cascaded PE modules to make a switching node for a network and an augmented routing procedure to handle output port contention, called routing algorithm B. Conclusions are given in section 5.

2 Network Architectures

There are several elements which are needed to realize a useful all-optical data path switching node: 1) single packet routing from any input to any output in a non-blocking fashion; 2) constant routing latency regardless of input/output path; 3) integration of a simple, distributed, high-speed routing mechanism within the switching matrix; 4) the ability to route asynchronous packet traffic; 5) an embedded contention resolution scheme within the topology and routing controller to avoid intervention by a slow external controller; and 6) implementation with current or near-future technology.

2.1 Crossbar

The unidirectional crossbar network shown in Fig. 3 is a strictly non-blocking network with n input ports (at the left) and n output ports (on the bottom). Also shown is an external electronic controller which routes packets through the network and resolves output contention problems [15]. Such a scheme has the drawback of requiring a complicated electronic control mechanism [4,12]. This configuration not only constricts switching speed [4] but also the scalability of the crossbar due to the complexity of the controller [12]. This topology also requires the use of electronic buffers to handle output port contention and is not well suited to routing optical packets,

2.2 Multistage Interconnection Networks

Figure 4 shows a unidirectional shuffle/exchange or omega network [5]. This network is an example of a *multistage interconnection network* (MIN) composed of cascaded stages of bypass/exchange switches (circles) with fixed shuffle permutations between them². This network is called a *full-access* network, because a path always exists from any of 2^k input ports to any of 2^k output ports by some setting of the internal bypass/exchange switches. Each stage of a k -stage shuffle exchange network has 2^{k-1} 2×2 switches for a total of $k2^{k-1}$ switches with 2^k input lines incident at the first stage. After k switch stages, 2^k packets emerge from the output lines. Unfortunately, many simultaneous connection states of inputs to outputs (input/output *permutations*) are blocking states due to internal contention at the bypass/exchange switches [5]. The shuffle/exchange omega network can be rearrangably non-blocking with $3k-1$ switch stages [16]. The presence of packet asynchronous traffic further complicates the problem.

2.3 Self-Routing Networks

The omega (or cascaded shuffle/exchange) network shown in Fig. 4 can be used to make a *self-routing* network [8]. Packets enter the network from the left on various lines numbered 0 through $n-1$. There are $k = \log_2(n)$ stages of the network and they are numbered from the left as $1, \dots, m, \dots, k$. At the bypass/exchange switches located at stage m of the network, the correct routing of the packet to its destination port is done by examining the m th most significant bit of the destination header shown in Fig 1 [2,8]. If this bit is 0, the packet takes the upper output port at the 2×2 bypass/exchange switch⁸. If the bit is 1, the packet takes the lower output port [8]. Using this procedure, any single packet entering the network at any input port will be routed correctly after k stages of bypass/exchange switches to the correct output port.

2.4 Sorting Networks

Multistage networks related to the shuffle/exchange omega networks can be used for sorting of numerical data and self-routing of packets to output ports [1,8]. Previous work on multistage sorting networks has discussed global control algorithms [9, 16]. A self-routing Benes network was developed by Nassimi and Sahn [10] but requires packet synchronous inputs and can not process all possi-

ble input permutations in a non-blocking fashion. The self-routing crossbar by Cloonan and Lentine allows for all possible permutations but also requires synchronous packets for proper operation [3]. Moreover, this crossbar has variable routing latency which is not well suited to an all-optical data path.

The use of bitonic sorting networks for packet synchronous interconnections was previously proposed by Batcher [1]. A bitonic sequence is defined as the juxtaposition of an ascending sequence with a descending sequence [1]. By performing a merging operation on the two sequences, a single ascending or descending sequence can be obtained [1]. This merging operation can be performed by a bitonic sorter to rearrange a bitonic sequence into monotonic order [1].

These ideas are illustrated by the 8-input Batcher bitonic sorting network shown with a randomly selected set of inputs as an example in Fig. 5 [17]. The circles in this figure represent 2-input/2-output compare and exchange modules. These devices are similar to the 2x2 bypass/exchange switches shown in Fig. 2 along with logic to compare two numbers and route them to an output port [1,17]. The arrows in Fig. 5 indicate whether the larger number is routed up or down [17]. The first three stages of switches perform the sort-by-merging operation resulting in the bitonic sequence 0,2,3,7,6,5,4,1. The last three switch stages are an 8-number bitonic sorter. The end result is that the Batcher bitonic sorter rearranges packets in monotonic order of the packet headers [11]. Any arbitrary sequence can be sorted into a monotonic sequence by using bitonic sorters to merge numbers two at a time, then four at a time and so on until the entire sequence of numbers is merged [1].

A Banyan network can be placed after a Batcher bitonic sorter to take a monotonically ordered set of inputs and route them to their respective output ports in the absence of output contention [8,17,11]. If a Batcher sorting network is cascaded with a Banyan network with a perfect shuffle input stage (shown in Fig. 6), the resulting network is a rearrangeably non-blocking network for simultaneous packets [8, 11]. In effect, the Batcher network performs a reordering of inputs while the Banyan network selects a fixed output port for a given packet address.

3 Permutation Engine (PE) Modules

Here we discuss the permutation engine as a fundamental component of the switching node architecture presented later in section 4. The PE has constant delay in routing any input to any output, and suitable cascades of PE modules form an asynchronous, strictly non-blocking node with packet asynchronous traffic. The PE is a self-routing network analogous to the shuffle/exchange omega network described in section 2.3 and has the topology and routing algorithm described in sections 3.1 and 3.2 respectively. A single permutation engine can be rearrangably non-blocking. A cascade of $n/2$ PE modules for n even or $(n+1)/2$ PE modules for n odd forms an asynchronous, strictly non-blocking node

3.1 Permutation Engine Topology

At the present time, the easiest technique for implementing optical buffering is to delay the data through a length of fiber. To realize a variable delay requires the use of optoelectronic conversions in the data path, which limits the network data rate. Thus, we do not perform dynamic retiming of data entering the network and require packet asynchronous operation with fixed routing latency regardless of the input/output connection. Thus, the control mechanism for the network must route traffic in an asynchronous, strictly non-blocking fashion regardless of the packet timing at its inputs. For routing without output contention, this condition implies that any permutation of input/output connections must be achievable and that any unused input/output connections must not be blocked by existing packet traffic. We address both of these requirements with the PE.

3.1.1 Graph Representation

For an $n \times n$ self routing network, given a set of pairings between all input ports and all output ports, there is a unique path between each input/output port pair and a total of n such paths. Here a path is defined as the set of links and switches which connect the input port to the desired output port. We use *graph theory* to establish some details of the permutation engine structure. A *graph* generally consists of nodes (called *vertices*) interconnected by links (called *edges*). We associate a path *intersection graph* with a given set of connections between input/output ports of a module containing

switches. The path intersection graph defines the intersection of each input/output path in the module with respect to the other $n-1$ such paths,

The path intersection graph is constructed by: 1) associating a *vertex* with each path to an output port in the network; and 2) associating an *edge* linking these vertices if their corresponding paths for a given connection pattern intersect. Equivalently, we say that the two paths intersect if they share a bypass/exchange switch, regardless of the setting of the switch. If there is more than one intersection between two input/output paths, additional edges linking the corresponding vertices are added. The degree of a vertex (which is the number of edges connected to it), represents the number of other input/output paths which intersect the path corresponding to this vertex.

As an example of this concept, the graph for the 3×3 crossbar in Fig. 3 is shown in Fig. 7 with an arbitrarily chosen connection pattern. The resulting path intersection graph for this topology and connection pattern is shown in Fig. 8a. The packet incident on input port 0 is destined for output port 1. The internal links in Fig. 7 with label 1 show how this packet routes from input port 0 to output port 1 and define the vertex with label 1 in the path intersection graph of Fig. 8a. The routing paths associated with output ports 0 and 2 are defined likewise.

An $n \times n$ self routing module is defined as *fully interconnected* if any y input/output path intersects the remaining $n-1$ input/output paths. This condition translates into requiring that all vertices of the path intersection graph have degree at least $n-1$ and that those $n-1$ edges connect to the remaining $n-1$ vertices. From this definition, we see that the path intersection graph in Fig. 8a is not fully connected. For a self routing 4×4 network, the minimum required path intersection graph is called a K_4 graph, which is shown in Fig. 8b. This is a fully connected graph having four vertices. Since each vertex corresponds to one input/output path, and each edge corresponds to one path intersection, these graphs correspond to a network topology with a connection pattern such that each path intersects every other path once and only once. Hence any pair of inputs can exchange outputs without affecting the remaining traffic in the network. This construction realizes a fully interconnected network with the minimum amount of interconnections as there is only one edge connecting a pair of vertices. Our goal is to define a switching module with a self routing algorithm to realize a K_n path intersection graph for n inputs and n outputs.

3.1.2 Topology and Link Labelling Scheme

The permutation engine is based on a mesh topology described by Shamir and Caulfield [13] which achieves n -input to n -output rearrangably non-blocking permutations. We will refer to this type of network as a SC network. This arrangement uses $n(n-1)/2$ compare and exchange modules arranged as n stages [13,14,6]. For application in a packet network, these compare and exchange modules are the same as those in the Batcher sorting network of Fig. 5, with additional logic to compare two headers and set the switch state so that the data and trailer in Fig. 3 follow the header to the correct output port.

The permutation engine uses a modified version of the SC topology along with a distributed routing algorithm. Figure 9 shows an example of the basic n -input, n -output PE for $n=4$. In order to achieve constant routing latency, we augment the SC topology with elements (shown as square boxes in Fig. 9) whose delay is equal to the switching time of a 2×2 bypass/exchange switch. For any possible input/output path, the mesh switching latency is just nT , where n is the number of I/O ports and T is the switching time of one 2×2 bypass/exchange switch. Notice that the input ports are numbered from top to bottom in increasing order, while the output (destination) ports are numbered in reverse order. The labels along the top of the mesh denote columns of links interconnecting bypass/exchange switches. For any n , the number of 2×2 bypass/exchange switches is $n(n-1)/2$ and the number of unit delay elements is n .

In Fig. 9, we define the m _{th} column of links as the set of links connected to the inputs of the m _{th} stage of 2×2 switches. The i _{th} row of links is the set of $n+1$ links defined by selecting the i _{th} link from the top in each column of links. We define the row and column pointers used to index a link in the mesh topology of Fig. 9 by r and c respectively. The left hand column of links (with label 0 in Fig. 9) corresponds to the $c=0$ column, and the right hand column of links (with label 4) to the $c=4$ column. The top and bottom links in each column have row pointers $r=0$ and $r=n-1$ respectively, where n is the number of input ports.

Each internal link in Fig. 9 carries a single label $d_{r,c}$ defined by a set of equations as follows. When the labels (r,c) are (even,odd) numbers or (odd,even) numbers respectively, we obtain

$$d_{r,c} = r + c \quad \text{for } r + c \leq n - 1 \quad (1)$$

$$d_{r,c} = 2n - r - c - 1 \quad \text{for } r + c > n - 1 \quad (2)$$

where n is the number of input ports. When the labels (r,c) are (even ,even) or (odd,odd) numbers respectively, we obtain

$$d_{r,c} = r - c \quad \text{for } r \geq c \quad (3)$$

$$d_{r,c} = c - r - 1 \quad \text{for } r < c. \quad (4)$$

These $d_{r,c}$ labels correspond to *optimal paths* to a given destination port and are used in the packet routing algorithm.

Before presenting a routing algorithm, it is useful to examine a particular special case represented in Fig. 9. Here an input packet at port i is destined for output port j , where $i=j$ and both i and j take on values from 0 to $n-1$. This is the identity permutation of inputs to outputs. In this particular case, the 2×2 switches are all set to the exchange state, the PE has a fully connected K_n path intersection graph, and the $d_{r,c}$ labels defined in Eqs. (1)–(4) correspond to the paths followed by input packets to a destination d . The *optimal paths* are the set of paths and labels $d_{r,c}$ for this particular case.

A routing path through switches in the exchange state is preferred over a path through switches in the bypass state. With all switches in the exchange state as in Fig. 9, if input 1 goes to output 1, input 0 can route to outputs 0, 2 or 3. Switch bypass states can result in the omission of one or more edges of the K_4 path intersection graph of the identity permutation shown in Fig. 9. This result is most easily seen by setting all the 2×2 switches in Fig. 9 to the bypass state. For this case, the maximum number of edges connected to a vertex in the path intersection graph is two (as compared to three edges for the identity permutation shown in Fig. 9). Consequently, the resulting path intersection graph is not K_n and the interconnection pattern is not fully connected.

The basic feature of the permutation engine is this link labeling scheme that is used in the routing procedure. This is confirmed by examining the path intersection graph for this topology with the given link labelling scheme defining the connection pattern. For the 4×4 permutation engine in Fig.

9, with the switches set for the identity permutation the resulting path intersection graph is the fully connected K_4 , shown in Fig. 8b.

3.2 Routing Procedure

The next sections provide the basic 2x2 switch control method used in the routing procedure. Our goal in routing is a simple, distributed control mechanism using only local traffic destination information and avoiding header modification. This approach allows for local, high speed arbitration at the 2x2 dynamic switching elements arranged in a multistage switching matrix. The routing latency penalty for using many switch stages is alleviated by the speed achieved from the simple 2x2 switch control philosophy.

3.2.1 Basic Routing Rule

It is useful to compare the PE routing procedure with that of the Shamir and Caulfield (SC) network [7]. One stage of the SC network sorts packets on n input ports by comparing output port addresses of a pair of adjacent packets, with $n/2$ or $n/2-1$ such comparisons per stage, and routing the packets with the larger destination address up [13]. Thus n packets sent through an n stage SC network will result in ordering the largest destination output at the top of the network and the smallest destination output at the bottom. Here the larger output port address is always routed up at a 2x2 switch, but this network requires packets at all the input ports simultaneously to properly route packets to an output destination.

To address this shortcoming, the internal link labels of the PE in Fig. 9 are used to define the up/down decision at each 2x2 switch stage. At any given 2x2 switch in the PE, a packet has one or more available paths to an output port d . A packet is said to route in a non-blocking fashion if it avoids blocking another input/output connection. This mode of operation requires using 2x2 switch exchange states to route packets for the reasons discussed in section 3.1.2. The routing principles discussed next are graphically summarized by the optimal paths and the constraints on switch bypass states shown in the boxed regions of Fig. 9.

Because packets generally do not enter the PE at optimal path inputs, we define the *basic routing rule* as follows. A packet entering at input port i routes along the set of links with label i (using switch

exchange states) as shown by the identity permutation in Fig. 9. A 2x2 switch bypass state is used only when the optimal path for destination d is reached or when the boxed rules in Fig. 9 allow a bypass state. Once a packet reaches its optimal path, it routes along this path (using switch exchange states) to the designated output port d . This routing procedure minimizes the number of 2x2 switch bypass states to route a packet from any input port i to the desired output port d to minimize the blocking of *subsequent* packets. Additionally, for simultaneous packets, a switch bypass state is used for intermediate link contention resolution - i.e. when two packets require the same output of a 2x2 switch at the same time.

3.2.2 Routing Algorithm A

Routing algorithm A uses the optimal paths defined in section 3.1.2 to make routing decisions. The algorithm also defines a set of diagonal planes called *bounding optimal paths* labeled by arrows in Fig. 9. Each diagonal plane is labeled with a rule which constrains the routing of a packet onto a given plane using a switch bypass state. This rule corresponds to an upper/lower bound on the destination port number d for packets routing to the plane using a switch bypass state. These rules are defined by the following set of equations. For diagonal planes which ascend from the lower left to the upper right of the PE topology in Fig. 9, the upper bound on the destination tag of a packet, d , for packets routing to one of these planes using a switch bypass state is given by

$$d < r + c + 1 \quad \text{for } r + c < n - 1 \quad (5)$$

$$d < 2n - r - c - 3 \quad \text{for } r + c \geq n - 1 \quad (6)$$

where n is the number of input ports and d is the destination label of a packet. The row and column pointers, (r,c) denote the current location of the packet header within the PE topology and correspond to the link incident on the top input of a 2x2 switch whose top output lies on a diagonal plane defined by Eqs. (5) and (6). For diagonal planes which descend from the upper left to the lower right of the PE topology, the lower bound on d for packets routing to one of these planes using a switch bypass state is given by

$$d > r - c - 1 \quad \text{for } r > c \quad (7)$$

$$d > c - r + 1 \quad \text{for } r \leq c \quad (8)$$

where (r,c) denote the current location of the packet header within the PE topology and correspond to the link incident on the bottom input of a 2x2 switch whose bottom output lies on a diagonal plane defined by Eqs. (7) and (8),

in applying these rules, a bypass switch state may be needed. Packets with destination address d greater than the rule for a bounding optimal path (defined by Eqs. (7) and (8)) can always route to their desired output from the region above that bounding optimal path in the absence of contention. We call this region the *favorable routing region*. This is similarly true for packets with address d less than the rule for bounding optimal path (in Eqs. (5) and (6)) in the region below this path. If a packet is forced outside of its favorable routing region, it may not be able to reach the desired output port within a single PE. Thus the routing algorithm allows a switch bypass state if this is the only way to keep a packet in its favorable routing region. It is because of this restriction on switch bypass states that intermediate link contention results in a 2x2 switch bypass state,

Our fundamental routing procedure is applied at every bypass/exchange switch and is called algorithm A. It is given below and shown as a flow chart in Fig.10. We divide the procedure into two parts. Part 1 applies to packets entering the top input of any bypass/exchange switch in the network. Part 2 applies to packets entering the bottom input of any bypass/exchange switch. We label the current link occupied by a packet with its (row, column) position indices (r,c) . The index i points to the i th link from the top in the $(c+1)$ th column of links, with $d_{c+1,i}$ being the value of the link label as shown in Fig. 9.

Each part of the fundamental routing procedure is further broken down into two basic rules. For part 1, these rules are described as follows. In rule A 1, we first check to see if either 2x2 switch output is currently in use. If yes, then route to the unused output. If not, then increment r and c by one, and check to see if any of the links in column $c+1$ and row i , for $i=r+1$ to $n-1$ match the header d of the packet. If they do, we use the exchange state. If rule A 1 fails to request a 2x2 switch exchange state, we apply rule A2 to determine if a bypass state is not allowed. For part 1, rule A2 is embodied in

Eqs. (5) and (6). For packets in the upper left half of the network, we check if Eq. (5) is satisfied, For packets in the lower right half of the network, we check if Eq. (6) is satisfied, If the appropriate equation is satisfied, we use the exchange state. If not, we use the bypass state, These rules are also applicable to part 2 of the procedure as shown diagrammatically in Fig. 10,

We present several examples to illustrate the basic routing rule, As a first example, assume that a packet destined for output port 2 is incident on the top input of the switch labelled "a" in Fig. 9, Thus, the packet header is located at the link with row/column pointers (1,1) respectively. Since the packet is above its optimal path, part 1, rule A1 in Fig. 10 requires a switch exchange state be used to route the packet to the bottom output of switch a in the absence of contention. When the packet intersects the optimal path at the next 2x2 switch, it uses a switch bypass state to follow this optimal path to output port 2.

As a second example, a packet destined for output port 2 is now incident on the bottom input of switch a in Fig. 9. Here the packet header is located at the link with row/column pointers (2,1) respectively. The optimal path for output port 2 is below this packet and part 2, rule A1 in Fig. 10 does not require a switch exchange state. Part 2, rule A2, however, uses the bounding optimal path rule defined by Eq. (8), which specifies that if $d > 0$, then a switch exchange state must be used, Because the bounding optimal path defines the output 0 boundary, a switch exchange state routes the packet into its favorable routing region because output port 2 is above output port 0. Using a switch bypass state would block a subsequent packet trying to route from input ports 0 or 1 to output port 0. Hence, any packet destined for output ports above port 0 are not allowed to move onto this optimal path using a switch bypass state.

Part 1, rule A2 is applied as shown in of Fig. 10 for switch bypass states from the top input of a switch. If a given packet wants a switch bypass state but is destined for a port below the port defined by the bounding optimal path, then that packet must route to the bottom switch output. By substituting a switch exchange state for a bypass state at these boundaries, a packet moves into its favorable routing region and avoids potentially blocking a later packet whose optimal path is outside of this region.

A third example illustrates contention resolution at a 2x2 switch. Such a situation could occur, for example, if the top input of switch a , in Fig. 9, is destined for output port 2 and the bottom input to the switch is destined for output port 0. Both packets want to use the bottom output port of switch a and the bottom input packet is allowed to use a switch bypass state since it is destined for output port 0. In this case, an exchange state would prevent the packet destined for output port 0 from reaching this output port, but a bypass state does not prevent the top packet from reaching port 2. The packet for port 2 would be in a similar position at the next switch, but now it requires a switch bypass state, which has priority. It is this combination of the PE topology, the link labeling scheme and the fundamental routing algorithm A which defines of the permutation engine,

3.23 Space-Time Progression Diagrams

in order to investigate the routing of packets as they proceed through the network, we define a routing table called an $n \times n$ space-time progression diagram (STPD), where n is the number of I/O ports. Figures 11 and 12 show examples of 4x4 and 5x5 STPDs for sequential packet routing. The left-hand column of numbers in the STPD are the input port numbers of the permutation engine and the right-hand column of numbers correspond to the output port numbers (listed in reverse order to match the topology of the network). The numbers at the top of the STPDs in Fig. 11 designate the link column labels for the four input topology in Fig. 9. An STPD shows the time evolution of packet traffic, where the numbers shown within the STPD correspond to the destination address, d , for each packet.

in Figs. 11a and 12a, a single packet is allowed to route completely (from left to right) before a second packet is injected into the permutation engine. As each packet is routed, the switches affected by its routing path are held locked for the duration. This situation is typical of long packets arriving at staggered times and demonstrates the permutation engine's non-blocking potential for individual packet routing.

Closer examination of Figs. 11 and 12, however, illustrates what happens when packets do not enter at the optimal inputs for the desired outputs. in Fig. 12c, the packets destined for outputs 1 and 3 do not pass through a common 2x2 switch. Hence, if after routing all the packets in Fig. 12c, packets

1 and 3 were removed from the STPD, a new packet beginning at input port 0 could not reach output port 1 due to the current switch settings.

3.2.4 Cascading Additional PEs for Asynchronous, Strictly Non-blocking Routing

Fig. 13 presents an example which shows that rearrangably non-blocking routing (in the absence of output port contention) with one PE is not always possible with routing algorithm A in section 3.2.2. Rearrangably non-blocking routing with one PE is always possible if routing algorithm A is modified to perform a search for two optimal paths. However, our main goal is to provide asynchronous, strictly non-blocking routing, and we achieve this by cascading additional PE modules to provide additional stages of switches and possible routing paths. Routing algorithm A in section 3.2.2 is useful in determining the routing in these cascaded networks.

To achieve asynchronous, strictly non-blocking operation, we require additional PEs in cascade. A total of $n/2$ cascaded PE modules are needed for n even or $(n+1)/2$ cascaded PE modules are needed for n -odd to achieve asynchronous, strictly non-blocking operation as we now demonstrate. In forming the cascade, we use *reflected* PEs alternately with ordinary PEs. A reflected PE is defined by the top-to-bottom reflection of the original PE such that the input ports are now numbered in descending order from top to bottom and the outputs numbered in ascending order. An example of a 5-input permutation engine, cascaded with its reflected version is shown in Fig. 14. In Fig. 14, we define the first permutation engine as the sorting PE and the second as the interconnection PE. The sorting PE tries to order simultaneous packets in descending order of their headers at its output ports while the interconnection PE provides the interconnection between the paths of these packets. The routing algorithm for the reflected PE is similar to the original except that the larger destination addresses route down instead of up.

The problem with the routing example in Fig. 12c is that the path intersection graph in this particular case is not a fully connected K_5 graph. The cascade of PEs with reflected PEs as shown in Fig. 14 tries to achieve a K_5 path intersection graph for all possible connection patterns. The most difficult routing problem for a single PE occurs when a packet destined for the i_h output port enters at the $(n+1-i)$, input port - i.e. it begins and ends the route at the same row position. This input/output

condition imposes the tightest constraint on the number of other input/output paths a particular route will traverse.

In the first (sorting) PE in Fig. 14, a packet that enters at port 4 destined for output port O is shown by the dashed arrows. The routing path shown in the sorting PE is the one selected by routing algorithm A and illustrates the maximum number of other input/output paths intersected by this route in the first PE. As shown in Fig. 12c, paths 1 and 3 do not cross a common 2x2 switch in a single PE. The paths for these packets do cross a common switch in the Fig. 14 configuration due to the ordering of packets by the first PE and subsequent reverse ordering of packets by the second PE. In essence, the first PE performs sorting while the second PE performs interconnections.

In Fig. 15, we further illustrate the requirements for cascade PEs by examining a 6-input permutation engine used to route six simultaneous input packets to output ports with identical addresses as shown in Fig. 15a. Here each path intersects all other paths. We label this case as connection pattern I.

Connection pattern II represents a difficult routing situation in that incoming packets are destined for output ports in the same row. To transform I into II in Fig. 15, we reorder the packets at the inputs to the PE using a *packet swap* operation illustrated by the following example. Beginning with connection pattern I, we remove from the PE two packets destined for output ports O and 5. These packets are again input into the PE but with packet O incident on input 5 and packet 5 incident on input O. These two packets route through the PE and result in a switch bypass state where these two paths intersect. By performing a second packet swap between packets 1 and 4, followed by a third swap with packets 2 and 3, we obtain connection pattern II shown in Fig. 15b. We call this STPD the *worst case connection pattern* because it minimizes the number of other input/output paths crossed by any given packet for the PE routing procedure in section 3.2.

The four parameters to consider for asynchronous, strictly non-blocking routing are the network topology, the routing algorithm, packet timing and packet position at the inputs to the network. Routing algorithm A, described in section 3.2.2, will always produce connection pattern II regardless of the relative time at which packets enter the PE. The importance of this observation is that muting

within the PE does not depend on relative packet timing to realize this connection pattern, By focusing on connection pattern II as a starting point, we remove any consideration of packet timing for this particular ordering of the packet headers at the inputs to the PE.

The swapping of packets is the most difficult routing case to handle because the network is fully utilized except for the paths to be swapped, For an n -input network, we do not affect the remaining $n-2$ packet paths when two packets are swapped. By starting with connection pattern II and generating a new interconnection pattern (using packet swaps), we also remove the specifics of the routing algorithm from the analysis. Hence, this methodology shifts the analysis of asynchronous, strictly non-blocking operation to the network topology and packet position at the inputs to the network,

As can be seen in connection pattern II Fig.15b, each packet will cross only three of the remaining five packet routes. To continue with this illustration, we select the three pairs of packets to be swapped based on the criteria that the packet pairs do not cross through a common 2×2 switch in Fig. 15b. One of two possible pairings is packets 0 and 3, 1 and 5, 2 and 4. Swapping each pair (two at a time) at the inputs of Fig. 15b, we realize the STPD shown in Fig. 16a, and label this STPD as connection pattern II 1. To obtain proper routing for this connection pattern, we must cascade a reflected permutation engine. The connection pattern for this additional PE is shown in Fig. 16b and labelled connection pattern IV.

Since there are two input/output paths not crossed by a given packet in connection pattern II, we can swap three different pairs of packets, where again each pair does not share a common 2×2 switch in connection pattern III, The remaining packet pairings are 0 and 5, 1 and 4, and 2 and 3. Asynchronous, strictly non-blocking operation requires three PE modules, and the resulting STPDs for this case are shown in Fig. 17. As can be seen in Fig. 17, all input/output paths cross all other input/output paths. These pairings result in the greatest number of packet swaps which do not share a common 2×2 switch in the first PE module and pose the greatest possibility of blocking.

For the worst case connection pattern with n even, using $n/2$ packet swaps as discussed above results in misrouting all packets at the outputs of the first PE, as seen in connection pattern III of Fig. 16 for $n=6$. The second PE provides the crossover function for each pair of packets, with subsequent

PEs routing packets along their optimal paths. If we now select the next set of packet swaps (which again do not share a common 2x2 switch in the first PE), the third PE will guarantee the crossover of paths since this PE previously routed packets along their optimal paths. By continuing this procedure, the number of additional PEs required for asynchronous, strictly non-blocking operation in general is just the number of input/output paths not crossed by a packet in the first PE with connection pattern II. For n even the number of additional PE modules is $n/2-1$ for a total of $n/2$ modules. For n odd the additional number of PE modules is $(n-1)/2$ for a total of $(n+1)/2$ modules. Thus for the case of $n=6$ with connection pattern II, a total of three PEs are needed for asynchronous, strictly non-blocking operation.

4 Output Port Contention

Here we address the routing capability with partial simultaneous packets and output port contention. As noted earlier, it is difficult to implement delay, storage, buffering and retiming of data directly in the optical domain to resolve these problems. We now describe Routing Algorithm B used with cascaded PE modules and optical fiber delay loops to address these issues,

4.1 Routing Algorithm B

At the present time, fiber delay lines are a convenient form of optical storage. By adding redundant I/O ports to a cascade of permutation engines, we buffer misrouted packets by using recirculating fiber delay links. An example of this idea is shown in Fig. 18 for a 3-input, 3-output switching node using a cascade of four 7-input PEs and four recirculating links. This device accepts asynchronous optical packets from three sources and routes these packets to three destinations using the permutation engine topology and an augmented version of the routing algorithm A called routing algorithm B.

Four PE stages are needed for asynchronous, strictly non-blocking operation with a 7-input network. The use of four redundant I/O ports between the host ports shown in Fig. 18, allows for routing of two simultaneous packets routing to host ports 1 and 3 and three packets routing to host port 2. One possible application of this switching node is shown in Fig. 18. In this figure, a host is any device that can send packets into the switching node and receive packets coming from the node.

For the 7-input PE configuration in Fig. 18, packets from external devices enter the node at PE input ports 0,3 or 6, and route through the cascaded PEs to the PE output ports 0, 3 or 6, Hence a valid header for incoming packets in the Fig. 18 scheme contains a 0,3 or 6 on] y. The remaining PE output ports 1,2,4 and 5 receive packets only because ports 0, 3 or 6 are in USC.

To accommodate the redundant output ports, we modify routing algorithm A in section 3.2.2 to select a 2x2 switch exchange state based on the *range* of outputs defined by the desired output and the redundant outputs, We define d_l and d_s define the largest and smallest valued redundant output ports respectively for the desired output port d . We also define u as

$$u = |2x2 \text{ switch top output link label} - d| \quad (9)$$

and v as

$$v = |2x2 \text{ switch bottom output 1 ink label} - d|. \quad (10)$$

They denote the absolute value of the difference between the desired output port d and the upper/lower 2x2 switch output link labels denoted by $d_{r,c}$.

Routing Algorithm B, shown in Fig. 19, is again composed of two parts, where Part 1 applies to packets entering the top input of any bypass/exchange switch and Part 2 to packets entering the bottom input of any bypass/exchange switch. Each part is also composed of two basic routing rules called B1 and B2, which are similar in function to rules A1 and A2 in section 3.2.2. In mlc B], we first check to see if either 2x2 switch output is currently in USC. If yes, then route to the unused output, If not, then increment r and c by one, and check to see if any of the link labels in column $c+1$ and row i , for $i=r+1$ to $n-1$ meet the condition $d_s \leq d_{i,c+1} < d_l$. If they do and $d_{r,c+1} > d_l$ or $d_{r,c+1} < d_s$, then use the exchange state, In this case, one of the link labels (starting with the label $d_{r+1,c+1}$) is in the desired range of output ports while the link label at the top switch output, $d_{r,c+1}$, is not,

If the link label at the top switch output is in the desired range of output ports, then we also check if $u > v$ or $u = v$ and $n -] - d > r$ to use a switch exchange state. For these two cases, a packet can use either switch output to reach an output port within the. desired range of output ports. in making the final decision, we compare u and v to determine if the. bottom switch output (as defined by its

associated link label) is closer to the desired output port d than the top switch output. If so ($u > v$), we use an exchange state. If $u=v$, the two switch outputs (as defined by their associated link labels) are equidistant from the desired output port, To resolve this case, we check to see if the row index of the desired output port, $n-1-d$, is greater than the row index of the packet, r . If this is the case, use an exchange state.

If an exchange state is not desired, we apply rule B2 to determine if a switch bypass state is allowed, For Part 1, rule 132 is embodied in the equations,

$$d_l < r + c + 1 \quad \text{for } r + c < n - 1 \quad (11)$$

$$d_l < 2n - r - c - 3 \quad \text{for } r + c \geq n - 1 . \quad (12)$$

For Part 2 rule B2 is given by,

$$d_s > r - c - 1 \quad \text{for } r > c \quad (13)$$

$$d_s > c - r + 1 \quad \text{for } r \leq c . \quad (14)$$

By comparing Eqs. (5) and (6) to Eqs. (11) and (12), we see that the largest valued redundant output port, d_l , is used to check bypass states from the top input of a switch to the top output. Similarly, comparing Eqs. (7) and (8) to Eqs. (13) and (14), the smallest valued redundant output port, d_s , is used to check bypass states from the bottom input of a switch to the bottom output, in essence, we have relaxed the constraints on switch bypass states due to the freedom in selecting from a range of output ports.

The redundant output ports for the switching node in Fig. 18 are 1, 2, 4 and 5. The range of output ports for each host output are given by: host 1 out, $d_l=1, d_s=0$; host 2 out, $d_l=4, d_s=2$; and host 3 out, $d_l=6, d_s=5$. We can select different values for d_l and d_s for each host output depending on the amount of traffic destined for each host. Additionally, the number of I/O ports per PE module can be increased to provide more redundant I/O ports. The number of redundant ports and the size of the fiber delays used to recirculate the data are parameters best defined by traffic statistics.

We present a simple routing example in Fig. 20 to illustrate switching node operation with output port contention. in this example, two coincident packets are destined for PE output port 0. The num-

bers within the boxes indicate the *priority* of the packets in reaching PE output port 0. The priority expresses the order in which the packets should be routed to port 0. In Fig. 20, the packet with boxed label one has the highest priority. Due to the priority of the 2x2 switch bypass state over the exchange state in combination with the routing rules used, the vertical ordering of packets destined for output port 0 is maintained. Since output port 0 is the top most output of the last PE (Fig. 20d), the top most packet (denoted by the boxed one) will have priority over the other packet.

By maintaining the vertical ordering of packets destined for output port 0, a first-in-first-out (FIFO) operation of the recirculate ports is achieved. This mode of operation is also applicable to packets destined for output port 6. Vertical reordering of multiple packets destined for intermediate output ports (such as port 3 in Fig. 20), however, can occur due to the constraints on switch bypass states. More importantly, when multiple packets desire the same output port, one of these packets will acquire that port and the remaining packets will migrate to the redundant outputs directly adjacent to the desired output port, where they are recirculated.

in the routing example of Fig. 20, by looping the recirculate port at PE output 1 in a row parallel fashion to PE input 1 (as shown in Fig. 18), a queue (organized as a FIFO delay line) is realized allowing for two simultaneous packets to route to the host 1 output. After the first pass through the network, the packet with the boxed label two will recirculate to PE input port 1.

The configuration in Fig. 18 acts like a 3x3 crossbar with a one stage queue for output ports 1 and 3, and a two stage queue for output port 2, where a stage of the queue is one recirculating link. By increasing/decreasing the amount of loopback (fiber) delay in the recirculate ports, the overall size of each queue stage can be adjusted to suit system parameters. Making the loops progressively longer for recirculate ports farther from a host output port results in additional queue length (and delay) proportional to the traffic congestion for that output port.

It is also possible to provide a global control mechanism by monitoring traffic in the recirculating links. We can achieve flow control for the network by not sending packets into the network (destined for a given host output) when the recirculating link(s) for this host contain packets. This monitoring

function is simple to implement and provides a means for achieving a request/acknowledge handshake well suited to an all-optical data path scheme.

5 Conclusions

It is desirable to keep information passed between communication switching nodes in optical form as much as possible. Ideally, all information passed between switching nodes (both control headers and data) is composed of optical packets without any electronic connections for control of the switching nodes. The requirement within the node is to keep the packet (header and data) in optical form. This requirement allows for very high link bandwidths and a variety of data formats without modifications to the hardware in the data path but imposes the constraint that a buffer is typically a simple fiber loop used to delay the packet. It is desirable to avoid the use of variable optical delay lines due to the complexity and expense of these devices.

Given the above requirements, the switching nodes must route the optical packets without any timing constraints on when the packets enter the node. To meet this need, we have described a topology and routing algorithm which allow for an asynchronous, strictly non-blocking crossbar node with asynchronous packet traffic. The fundamental component of this node is a PE module, which consists of several stages of bypass/exchange switches and associated internal connections arranged in a mesh topology. A node is composed of a cascade of PE modules to route packets from input ports to output ports with constant routing delay regardless of the input/output path taken. In the PE methodology, there are no complicated processing elements, no flow control, no request/acknowledge mechanism, and no header modification. Optical packets are routed on the fly to avoid packet synchronization issues and electronic buffering in the data path. In future work, we will address the optical/optoelectronic implementation of a PE module and details of control and switching hardware.

6 Acknowledgements

The research described in this paper was performed by the Center for Space Microelectronics Technology, Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the Ballistic Missile Defense Organization, Innovative Science and Technology Office., through an agreement with the National Aeronautics and Space Administration.

7 Technical References

1. Batcher, K. E. Sorting Networks and their applications. In Bhargava, A. (Ed.). *Integrated Broad-band Networks*. Artech House Inc., Norwood, MA, 1991.
2. Bhu yan, L. N. interconnection Networks for Parallel and Distributed Processing. *IEEE Computer*, 20, 6 (June 1987), 9–12,
3. Cloonan, 'T'. J., and Lentine, A. L., Self-routing crossbar packet switch employing free-space optics for chip-to-chip interconnections. *Applied Optics*. 30,26 (September 1991), 3721-3733,
4. Hinton, H. S. Architectural Considerations for Photonic Switching Networks. *IEEE J. on Selected Areas in Communications*. 6,7 (August 1988), 1209-1226.
5. Jenkins, B. K., and Sawchuk, A. A. Dynamic Optical Interconnections for Parallel Processors. *Proc. Optical Computing Symposium, Society of Photo- Optical Instrumentation Engineers '86 Optoelectronics and Laser Applications in Science and Engineering*, SPIE, Los Angeles, CA, 1986, pp. 143-153.
6. Johnson, K. M., Surette, M. R., and Shamir, J. Optical interconnection network using polarization-based ferroelectric liquid crystal gates. *Applied Optics*. 27,9 (May 1988), 1727–1733.
7. Ica, C. T. Multi- $\log_2 N$ Self-Routing Networks anti Their Applications in High Speed Electronic and Photonic Switching Systems. *Proc. IEEE INFOCOM*. 1989, pp. 877–886.
8. Lee, 'T'. T. Nonblocking Copy Networks for Multicast Packet Switching. *IEEE Journal on Selected Areas in Communications*. 6,9 (December 1988), 1455-1467.
9. Lev, G. F., Pippenger, N., and Valiant, L. G. A Fast Parallel Algorithm for Routing in Permutation Networks. *IEEE Transactions on Computers*. C-30 (February 1981), 93-100.
10. Nassimi, D., and Sahni, S. A Self-Routing Benes Network and Parallel Permutation Algorithms. *IEEE Transactions on Computers*, C-30 (May 1981) 332–340.
- 11 Newman, P. ATM Technology for Corporate Networks. *IEEE Communications*. (April 1992), 90-101.

12. Sawchuk, A. A., Jenkins, B. K., Raghavendra, C. S., and Varma, A. Optical Crossbar Networks, *IEEE Computer*, 20,6 (June 1987), 50-60.
13. Shamir, J, and Caulfield, H. J. High-efficiency rapidly programmable optical interconnections. *Applied Optics*. 26,6 (March 1987), 1032-1037.
14. Shamir, J. Three-dimensional optical interconnection gate array. *Applied Optics*. 26,16 (August 1 987), 3455-3457,
15. Varma, A. Optical Switched-Star Networks for Computer-System Interconnection. in Tada, K., and Hinton, H. S. (Eds.). *Photonic Switching II*. Springer-Verlag, Berlin, Heidelberg, 1990.
16. Wu, C. L., and Feng, T. Y. The Universality of the Shuffle-Exchange Network, *IEEE Transactions on Computers*. C-30 (May 1981), 324-332,
17. Zcgura, E. W. Architectures for ATM Switching Systems, *IEEE Communications*. (February 1993), 28-37,

Steve P. Monacos received the A.A. degree from Pasadena City College in 1979, the B.S. and M.S. degrees from UCLA in Electrical Engineering in 1981 and 1984, and the Engineers degree in Electrical Engineering from USC in 1990. He is currently working towards a Ph.D. in Electrical Engineering at USC. He joined the Jet Propulsion Laboratory in 1982 where he has worked on a variety of chip and board level electronics for tracking system applications, multi-media hardware and communication networks. He is currently engaged in research on routing methodologies and network architectures for application to high-speed optical communication networks in a wide area network environment.

Alexander A. Sawchuk received the S. II. degree in Electrical Engineering from the Massachusetts Institute of Technology, Cambridge, MA, in 1966, and the M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA, in 1968 and 1972 respectively. He is currently Professor in the Department of Electrical Engineering and Associate Director of the Signal and Image Processing Institute at the University of Southern California. He was Director of the Signal and Image Processing Institute from 1978 to 1988 and 1990 to 1991. His research interests include optical computing, optical interconnections and networks, and digital image processing. He is the author of more than 120 technical publications in these fields.

8 List of Figures

Figure 1. Packet format.

Figure 2. 2x2 switch state definitions. The arrows in the circles indicate the flow of data for each switch state.

Figure 3. 3x3 unidirectional crossbar. The left-hand column of numbers corresponds to the input port numbers and the row of numbers at the bottom are the output port numbers.

Figure 4. 3 stage onlca/shuffle exchange network. The circles are bypass/exchange switches. Data flows from input ports at the left to output ports at the right,

Figure 5. 8-input Batcher sorting network¹⁵.

Figure 6. 8-input Banyan network with perfect shuffle input¹⁵. This network is isomorphic to the shuffle exchange network.

Figure 7. 3x3 crossbar graph and connection pattern. The numbers outside the dashed box are the input/output port numbers. The numbers inside the dashed box indicate the routing from the i_{th} input to the j_{th} output.

Figure 8. 3x3 crossbar path intersection graph and fully interconnected path intersection graph for a 4x4 network. The vertices (shown as black dots) represent the three output ports. The two edges indicate that the packet destined for output 2 crosses the paths of the packets destined for output ports 0 and 1. The packets routing to outputs 0 and 1 do not pass through a common 2x2 switch.

Figure 9. 4-input permutation engine. The circles denote 2x2 switches and the rectangles are unit delay elements. The packet header specifies the desired output port and is denoted by "d". Each number at the top of the mesh labels one column of links connecting two stages of 2x2 switches,

Figure 10. Routing Algorithm A flow chart.

Figure 11. 4x4 STPDs for sequential packets, STPD (a) shows the routing path for a packet incident on input port 1 and destined for output port 0 with no other packets in the network. STPDs (b) through (d) show the time evolution of subsequent packets entering the network and destined for unused output ports.

Figure 12. 5x5 STPDs for sequential packets. These diagrams are similar to those in Fig, 11 but for a 5-input network. The lines in STPD (c) highlight the intersection (or lack thereof) between the various input/output paths.

Figure 13. 5x5 STPD with rearrangably non-blocking routing.

Figure 14. Asynchronous, strictly non-blocking configuration.

Figure 15. 6x6 STPDs for best/worst case routing.

Figure 16. 6x6 STPDS for PE pair.

Figure 17. 6x6 STPDS for three PE modules.

Figure 18. 3x3 switching node connected to three host devices. This network has three I/O connections to external devices. The reference to a 7-input PE reflects the size of the PE topology. The four additional connections provide recirculating storage suited to optical data,

Figure 19. Routing Algorithm B flow chart.

Figure 20. Routing with partial simultaneous packets and output port contention.



Figure 1

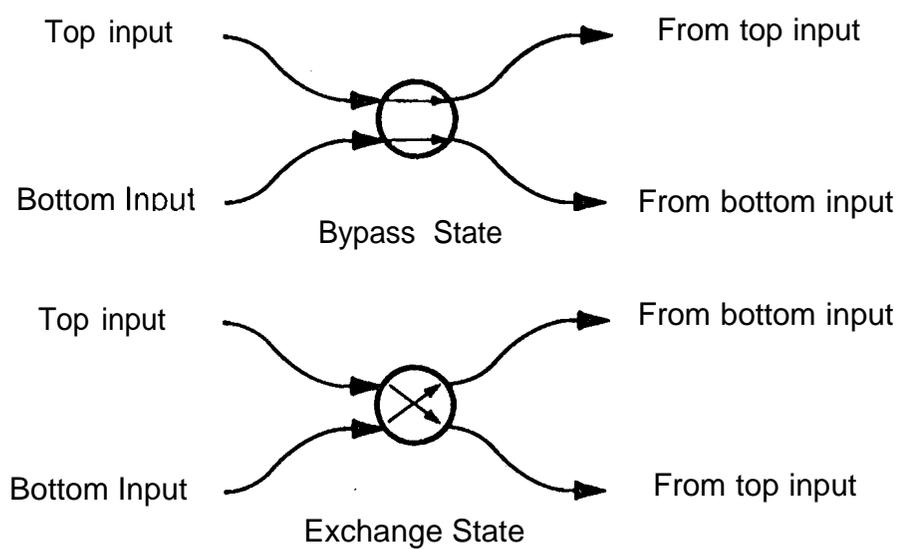


Figure 2

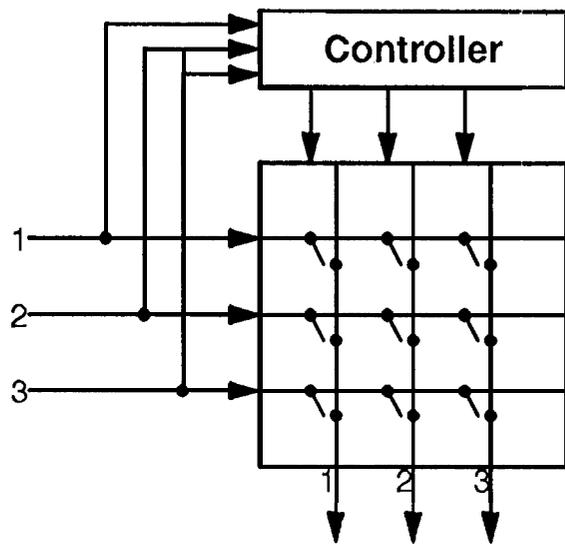


Figure 3

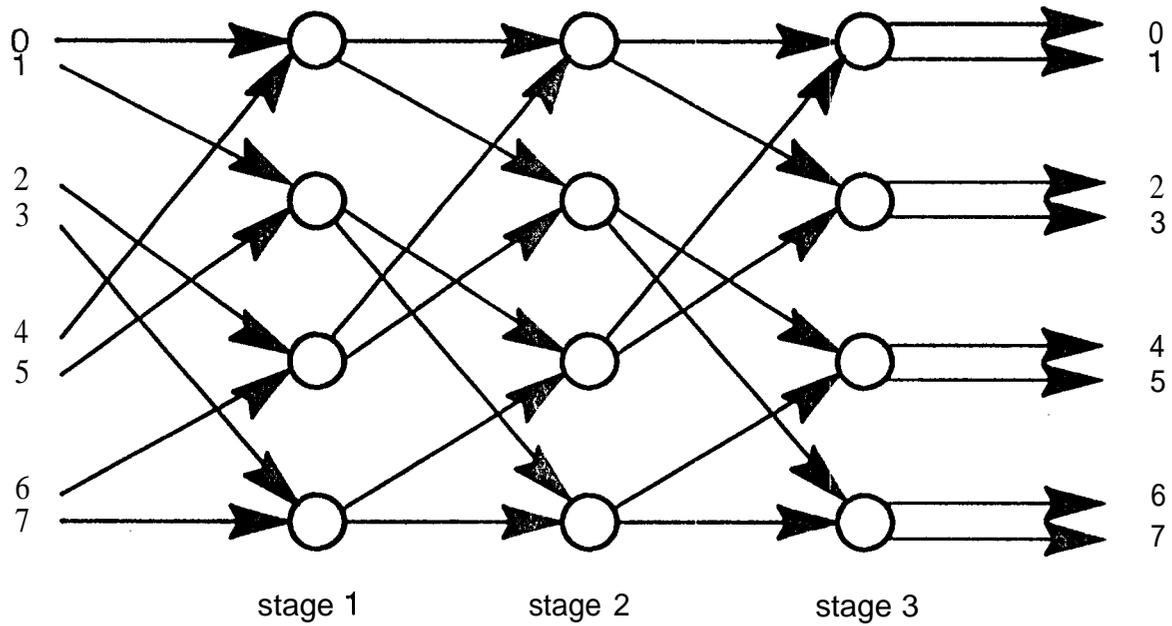


Figure 4

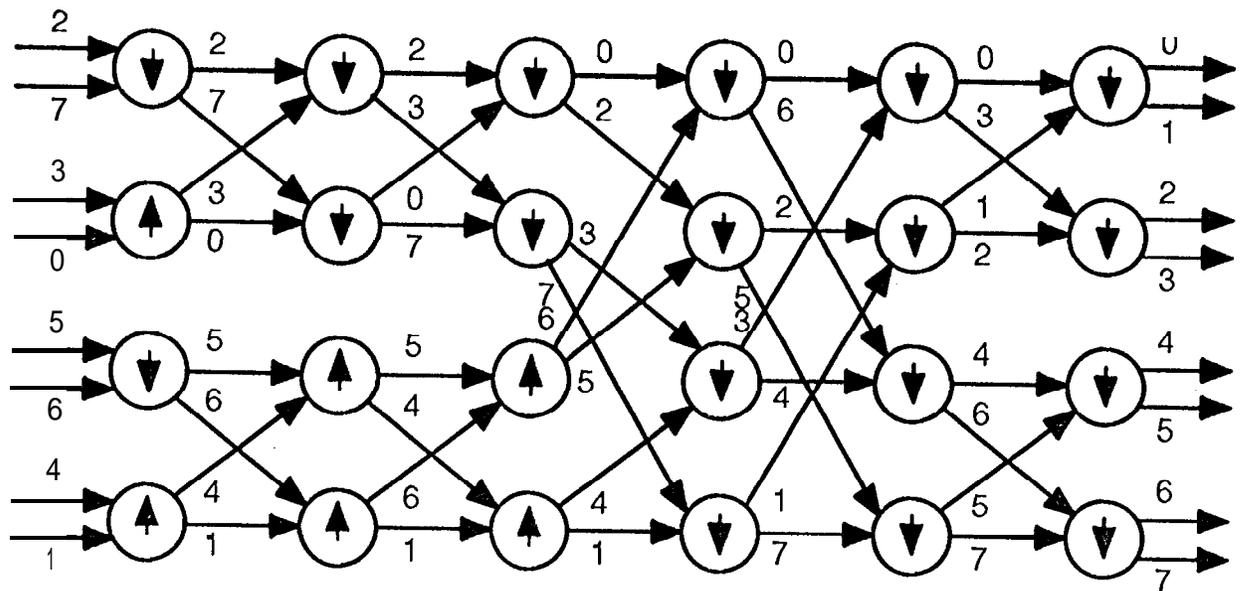


Figure 5

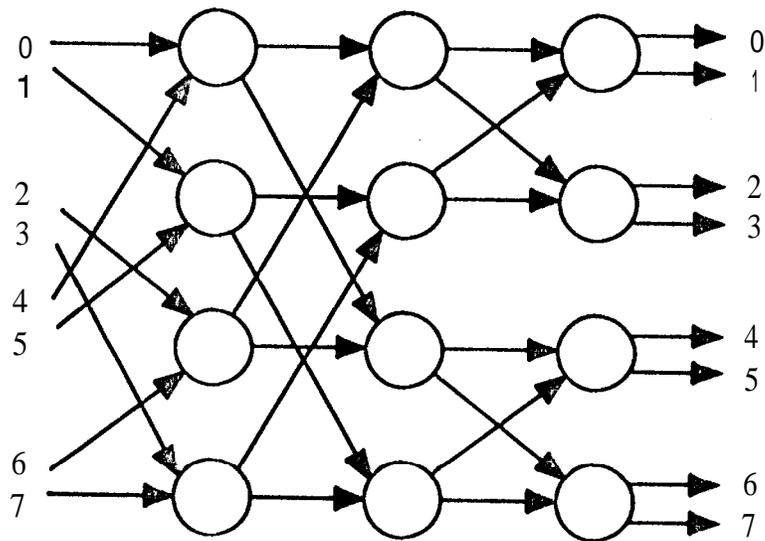


Figure 6

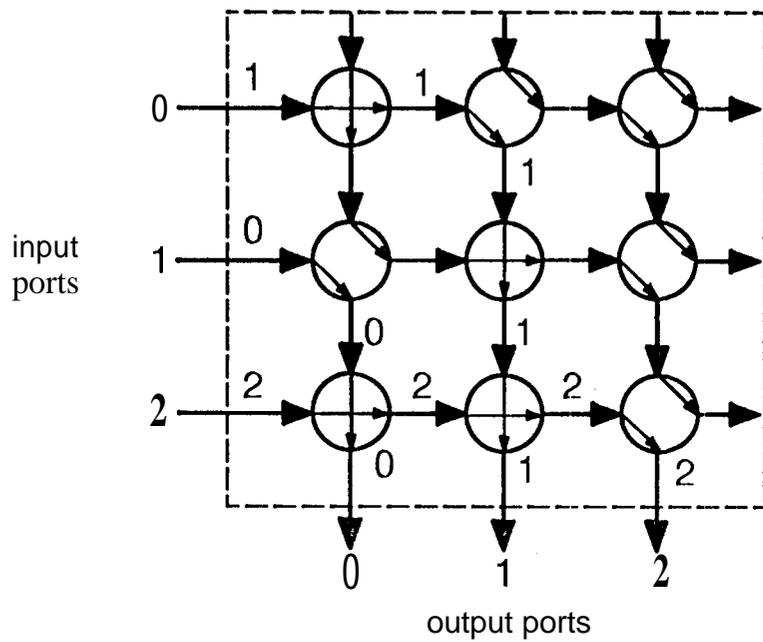


Figure 7

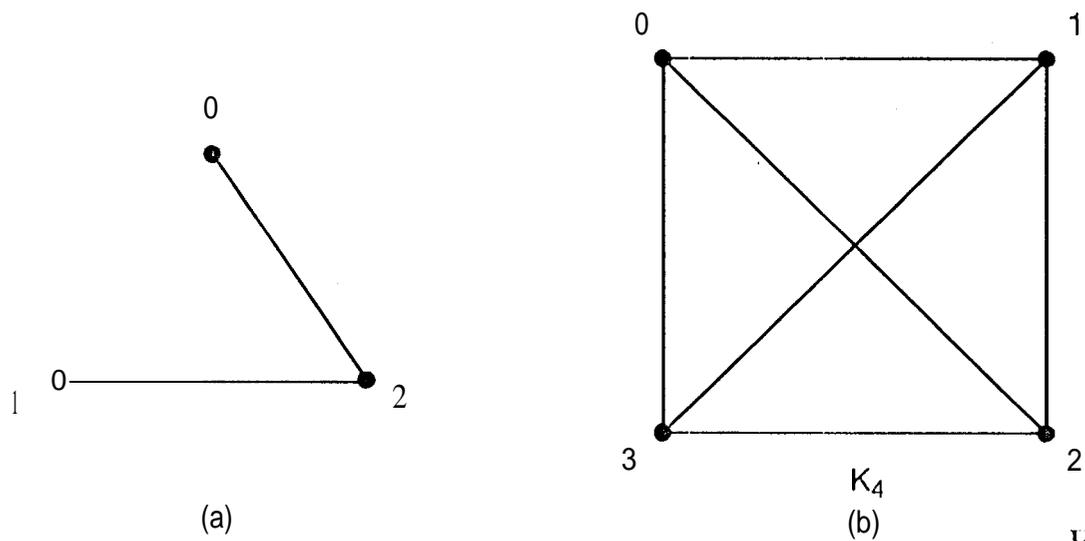


Figure 8

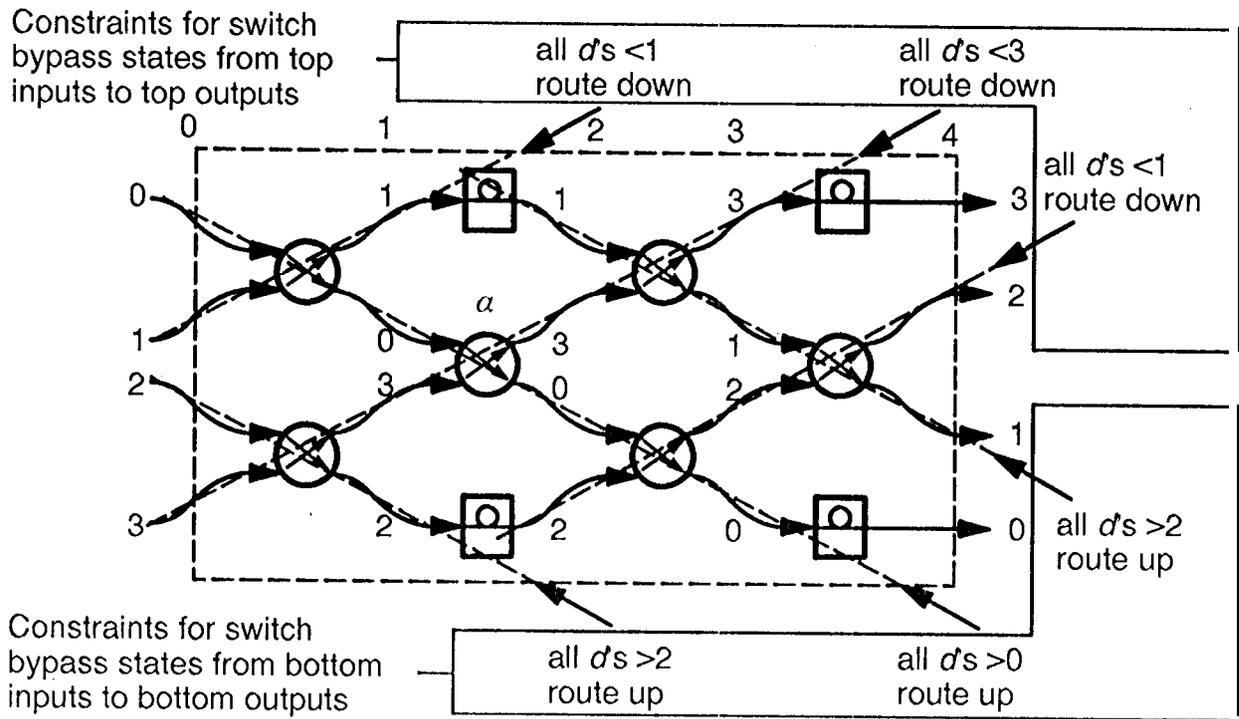


Figure 9

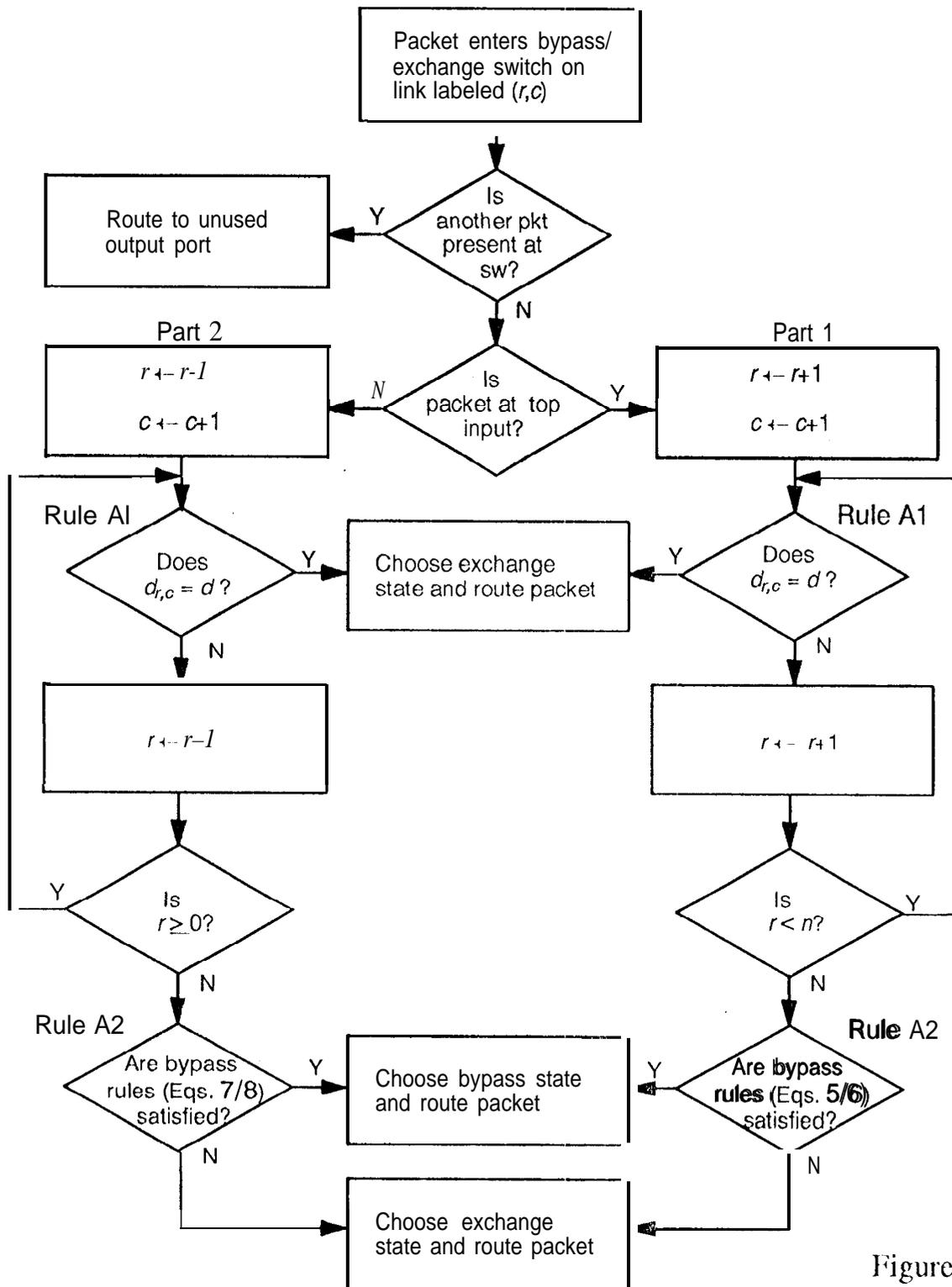


Figure 10

	0	1	2	3	4	
0	-	-	-	-	-	3
1	0	0	-	-	-	2
2	-	-	0	-	-	1
3	-	-	-	0	0	0

(a)

	0	1	2	3	4	
6	-	-	-	-	-	3
1	00	00	-	-	-	2
2	-	-	00	22	-	1
3	22	22	20	00	00	0

(b)

	0	1	2	3	4	
0	3	3	3	3	3	3
1	0	0	-	-	2	2
2	-	-	0	2	-	1
3	2	2	2	0	0	0

(c)

	0	1	2	3	4	
0	3	3	3	3	3	3
1	00	00	11	11	22	22
2	11	11	00	22	11	11
3	22	22	20	00	00	00

(d)

Figure 11

	0	1	2	3	4	5	
0	-	-	-	-	-	-	4
1	0	0	-	-	-	-	3
2	-	-	0	-	-	-	2
3	-	-	-	0	-	-	1
4	-	-	-	-	0	0	0

(a)

	0	1	2	3	4	5	
0	-	-	-	-	-	-	4
1	0	0	-	-	4	-	3
2	-	-	0	4	-	-	2
3	4	4	4	0	-	-	1
4	-	-	-	-	()	()	0

(b)

	0	1	2	3	4	5	
0	3	3	3	3	3	4	4
1	0	0	-	-	4	3	3
2	-	-	0	4	-	-	2
3	4	4	4	0	-	-	1
4	-	-	-	-	0	0	0

(c)

	0	1	2	3	4	5	
0	3	3	3	3	3	4	4
1	0	0	2	2	4	3	3
2	2	2	0	4	2	2	2
3	4	4	4	0	-	-	1
4	-	-	-	-	0	0	0

(d)

	0	1	2	3	4	5	1	
0	3	3	3	3	3	4	4	4
1	0	0	2	2	4	3	3	3
2	2	2	0	4	2	2	2	2
3	4	4	4	0	1	1	1	1
4	1	1	1	1	0	0	0	0

(e)

Figure 12

n	1	7	3	4	5	
0	2	1	1	2	2	4
1	1	2	2	1	4	2
2	4	0	0	4	1	3
3	0	4	4	0	3	1
4	3	3	3	3	0	0

Figure 13

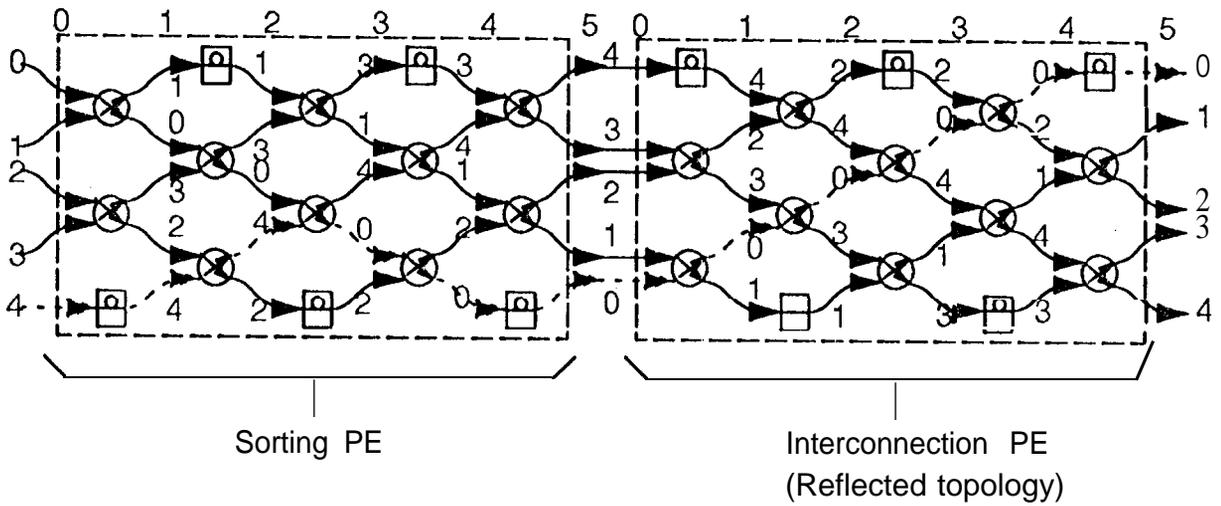
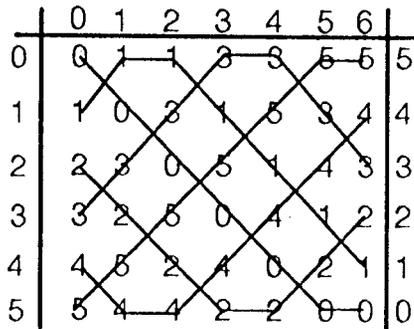
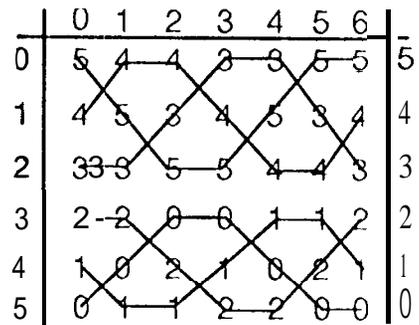


Figure 14



Connection pattern I

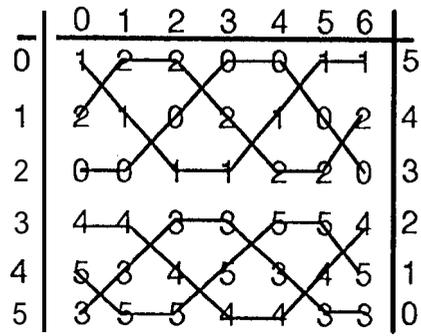
(a)



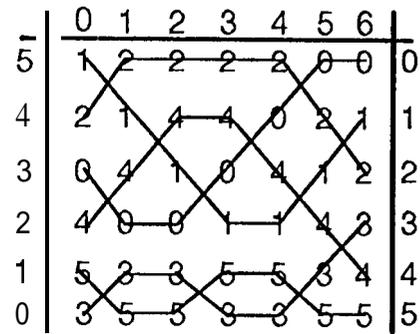
Connection pattern II

(b)

Figure 15



Connection pattern III



Connection pattern IV

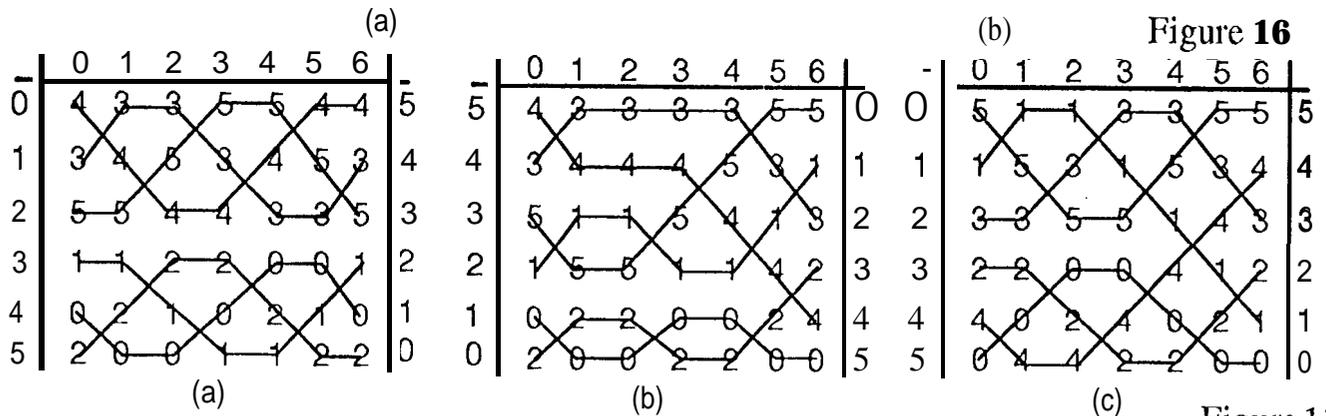


Figure 16

Figure 17

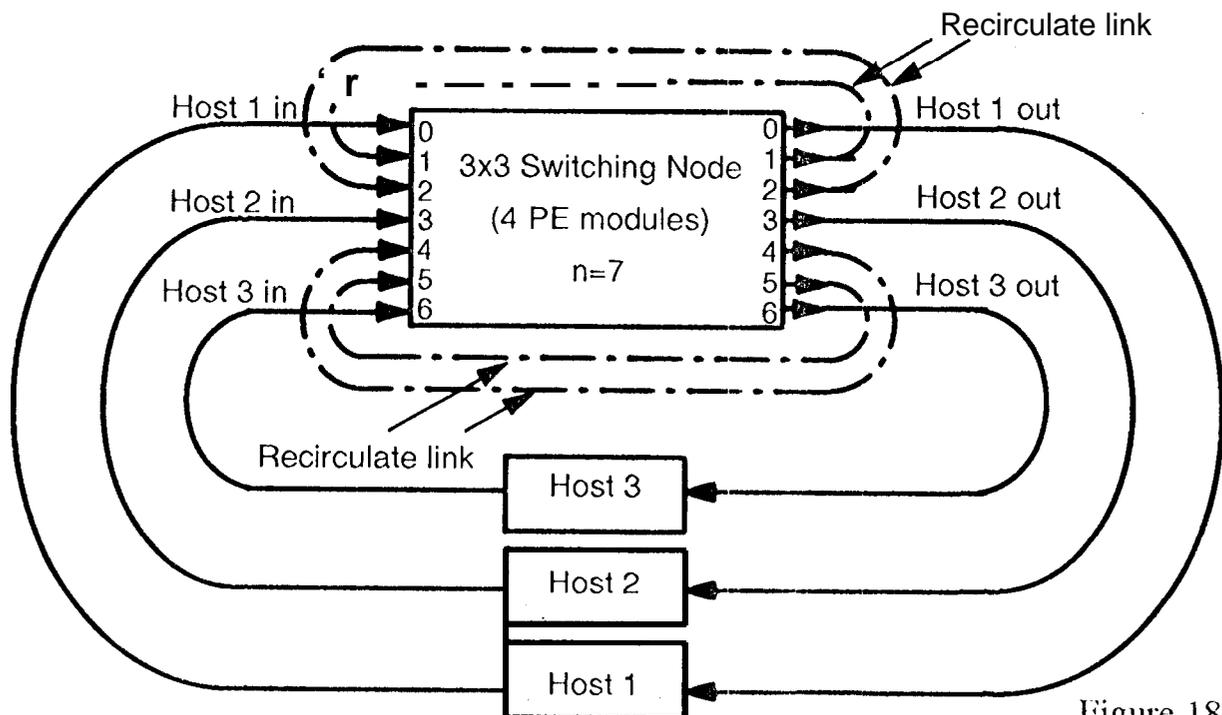


Figure 18

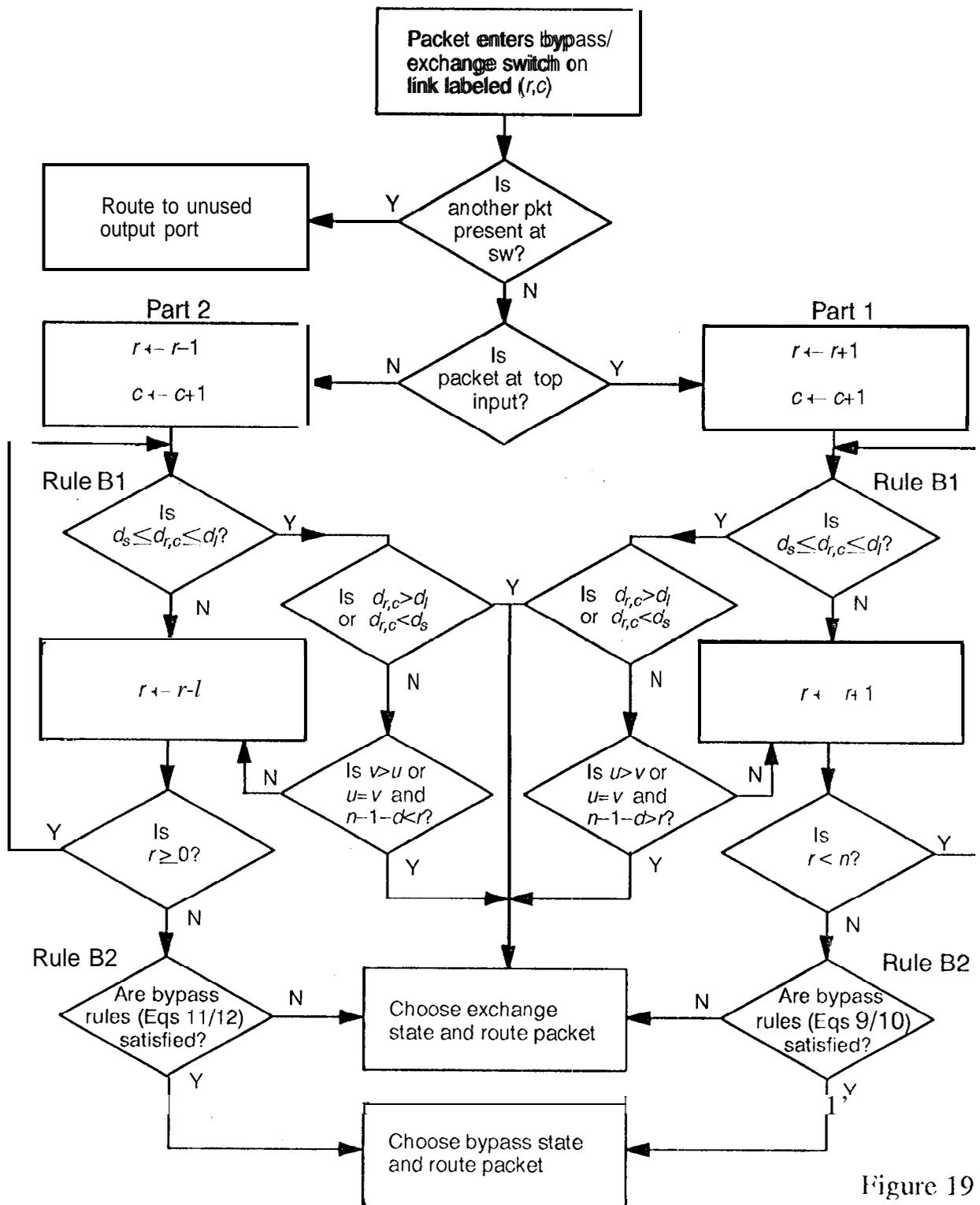
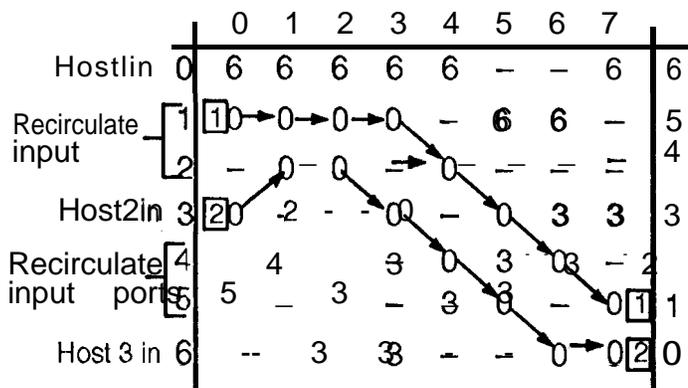
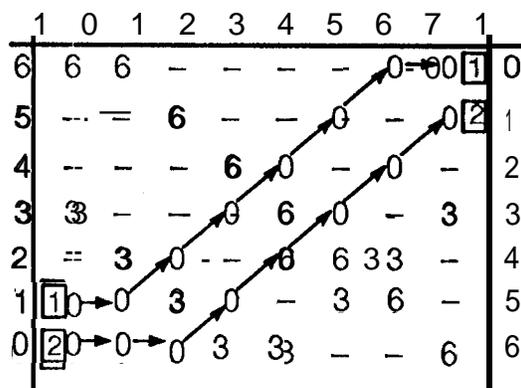


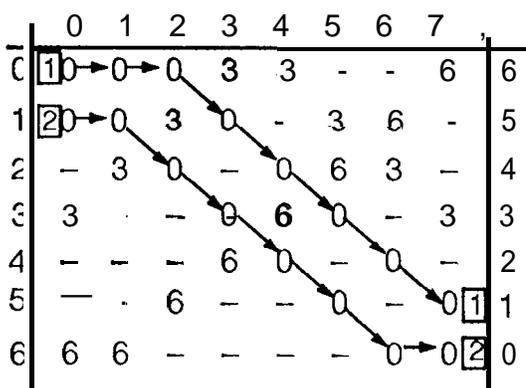
Figure 19



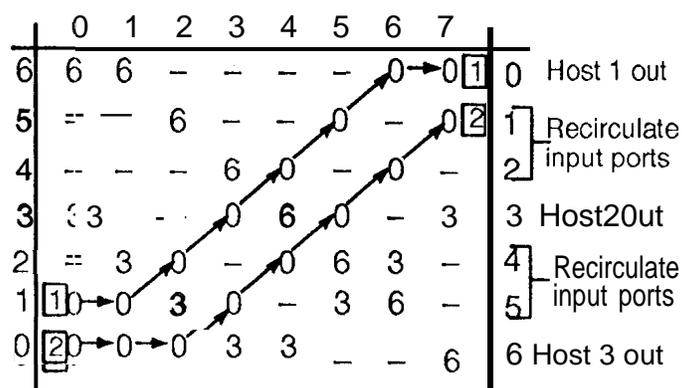
(a)



(b)



(c)



(d)

Figure 20