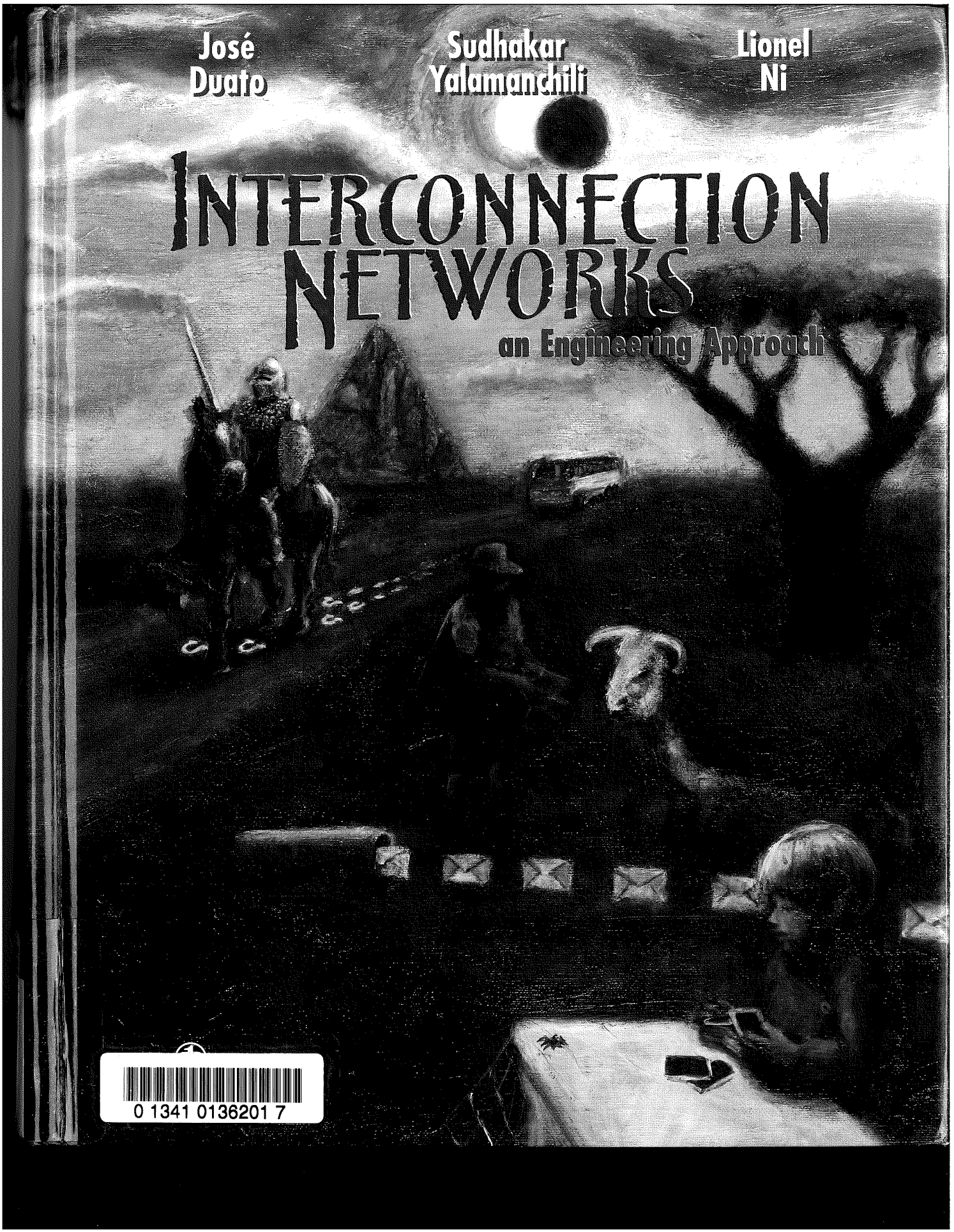

José Duato
Sudhakar Yalamanchili
Lionel Ni
INTERCONNECTION NETWORKS
an Engineering Approach
0 1341 0136201 7

Interconnection Networks

An Engineering Approach

José Duato
Sudhakar Yalamanchili
Lionel Ni

Los Alamitos, California

Washington • Brussels • Tokyo

Library of Congress Cataloging-in-Publication Data

Duato, José
Interconnection networks: an engineering approach / José Duato, Sudhakar Yalamanchili, Lionel Ni.
p. cm.
Includes bibliographical references.
ISBN 0-8186-7800-3
1. Computer networks. 2. Multiprocessors. I. Yalamanchili.
II. Ni, Lionel M. III. Title.
TK5105.5.D88 1997
004.6—dc21
97-20502
CIP

IEEE Computer Society Press Order Number BP07800
Library of Congress Number 97-20502
ISBN 0-8186-7800-3

Additional copies may be ordered from:

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: +1-714-821-8380
Fax: +1-714-821-4641
Email: cs.books@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: +1-908-981-1393
Fax: +1-908-981-9667
mis.custserv@computer.org

IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
BELGIUM
Tel: +32-2-770-2198
Fax: +32-2-770-8505
euro.ofc@computer.org

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama
Minato-ku, Tokyo 107
JAPAN
Tel: +81-3-3408-3118
Fax: +81-3-3408-3553
tokyo.ofc@computer.org

Editor-in-Chief: Mohamed Fayad
Publisher: Matt Loeb
Acquisitions Editor: Bill Sanders
Developmental Editor: Cheryl Baltes
Advertising/Promotions: Tom Fink
Production Editor: Lisa O'Conner
Cover Artist: John Marshall

About the cover: *Interconnection Networks* by John Marshall, oil on canvass, 1997, is based on a concept by José Duato. The painting includes symbols representing important elements in the field. To find out more about the symbols in the painting, visit the book's website at http://computer.org/books.

Printed in the United States of America

IEEE COMPUTER SOCIETY
IEEE

Chapter 1

Introduction

Interconnection networks are currently being used for many different applications, ranging from internal buses in very large-scale integration (VLSI) circuits to wide area computer networks. Among others, these applications include backplane buses and system area networks, telephone switches, internal networks for asynchronous transfer mode (ATM) switches, processor/memory interconnects for vector supercomputers, interconnection networks for multicomputers and distributed shared-memory multiprocessors, clusters of workstations, local area networks, metropolitan area networks, wide area computer networks, and networks for industrial applications. Additionally, the number of applications requiring interconnection networks is continuously growing. For example, an integral control system for a car requires a network connecting several microprocessors and devices.

The characteristics and cost of these networks considerably depend on the application. There are no general solutions. For some applications, interconnection networks have been studied in depth for decades. This is the case for telephone networks, computer networks, and backplane buses. These networks are covered in many books. However, there are some other applications that have not been fully covered in the existing literature. This is the case for the interconnection networks used in multicomputers and distributed shared-memory multiprocessors.

The lack of standards and the need for very high performance and reliability pushed the development of interconnection networks for multicomputers. This technology was transferred to distributed shared-memory multiprocessors, improving the scalability of those machines. However, distributed shared-memory multiprocessors require an even higher network performance than multicomputers, pushing the development of interconnection networks even more. More recently, this network technology began to be transferred to local area networks (LANs). Also, it has been proposed as a replacement for backplane buses, creating the concept of system area network. Hence, the advances in interconnection networks for multicomputers are the basis for the development of interconnection networks for other architectures and environments. Therefore, there is a need for structuring the concepts and solutions for this kind of interconnection networks. Obviously, when this technology is transferred to another environment, new issues arise that have to be addressed.

Moreover, several of these networks are evolving very quickly, and the solutions proposed for different kinds of networks are overlapping. Thus, there is a need for formally stating the basic concepts, the alternative design choices, and the design trade-offs for most of those networks. In this book, we take that challenge and present in a structured way the basic underlying concepts of most interconnection networks, as well as the most interesting solutions currently implemented or proposed in the literature. As indicated above, the network technology developed for multicomputers has been transferred to other environments. Therefore, in this book we will mainly describe techniques

developed for multicomputer networks. Most of these techniques can also be applied to distributed shared-memory multiprocessors, and to local and system area networks. However, we will also describe techniques specifically developed for these environments.

1.1 Parallel Computing and Networks

The demand for even more computing power has never stopped. Although the performance of processors has doubled in approximately every three-year span from 1980 to 1996, the complexity of the software as well as the scale and solution quality of applications have continuously driven the development of even faster processors. A number of important problems have been identified in the areas of defense, aerospace, automotive applications, and science, whose solution requires tremendous amount of computational power. In order to solve these grand challenge problems, the goal has been to obtain computer systems capable of computing at the teraflops (10^{12} floating-point operations per second) level. Even the smallest of these problems requires gigaflops (10^9 floating-point operations per second) of performance for hours at a time. The largest problems require teraflops performance for more than a thousand hours at a time.

Parallel computers with multiple processors are opening the door to teraflops computing performance to meet the increasing demand of computational power. The demand includes more computing power, higher network and input/output (I/O) bandwidths, and more memory and storage capacity. Even for applications requiring a lower computing power, parallel computers can be a cost-effective solution. Processors are becoming very complex. As a consequence, processor design cost is growing so fast that only a few companies all over the world can afford to design a new processor. Moreover, design cost should be amortized by selling a very high number of units. Currently, personal computers and workstations dominate the computing market. Therefore, designing custom processors that boost the performance one order of magnitude is not cost-effective. Similarly, designing and manufacturing high-speed memories and disks is not cost-effective. The alternative choice consists of designing parallel computers from commodity components (processors, memories, disks, interconnects, etc.). In these parallel computers, several processors cooperate to solve a large problem. Memory bandwidth can be scaled with processor computing power by physically distributing memory components among processors. Also, redundant arrays of inexpensive disks (RAID) allow the implementation of high-capacity reliable parallel file systems meeting the performance requirements of parallel computers.

However, a parallel computer requires some kind of communication subsystems to interconnect processors, memories, disks and other peripherals. The specific requirements of these communication subsystems depend on the architecture of the parallel computer. The simplest solution consists of connecting processors to memories and disks as if there were a single processor, using system buses and I/O buses. Then, processors can be interconnected using the interfaces to local area networks. Unfortunately, commodity communication subsystems have been designed to meet a different set of requirements, i.e., those arising in computer networks. Although networks of workstations have been proposed as an inexpensive approach to build parallel computers, the communication subsystem becomes the bottleneck in most applications.

Therefore, designing high-performance interconnection networks becomes a critical issue to exploit the performance of parallel computers. Moreover, as the interconnection network is the only subsystem that cannot be efficiently implemented by using commodity components, its design becomes very critical. This issue motivated the writing of this book. Up to now, most manufacturers designed custom interconnection networks (nCUBE-2, nCUBE-3, Intel Paragon, Cray T3D, Cray T3E, Thinking Machines Corp. CM-5, NEC Cenju-3, IBM SP2). More recently, several high-

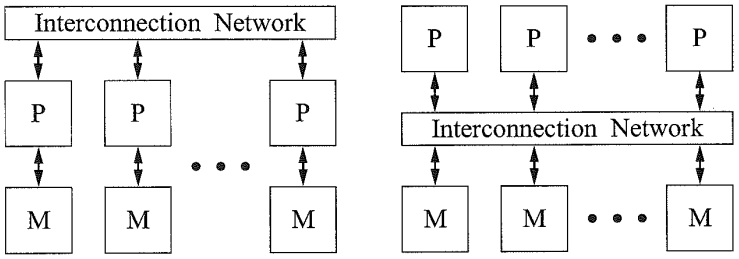


Figure 1.1. Schematic representation of parallel computers: (a) A multicomputer. (b) A UMA shared-memory multiprocessor. (M = memory; P = processor.)

performance switches have been developed (Autonet, Myrinet, ServerNet) and are being marketed. These switches are targeted to workstations and personal computers, offering the customer the possibility of building an inexpensive parallel computer by connecting cost-effective computers through high-performance switches. The main issues arising in the design of networks for both approaches are covered in this book.

1.2 Parallel Computer Architectures

In this section, we briefly introduce the most popular parallel computer architectures. This description will focus on the role of the interconnection network. A more detailed description is beyond the scope of this book.

The idea of using commodity components for the design of parallel computers led to the development of *distributed-memory multiprocessors*, or *multicomputers* in early 1980s. These parallel computers consist of a set of processors, each one connected to its own local memory. Processors communicate between them by passing messages through an interconnection network. Figure 1.1a shows a simple scheme for this architecture. The first commercial multicomputers utilized commodity components, including Ethernet controllers to implement communication between processors. Unfortunately, commodity communication subsystems were too slow, and the interconnection network became the bottleneck of those parallel computers. Several research efforts led to the development of interconnection networks that are several orders of magnitude faster than Ethernet networks. Most of the performance gain is due to architectural rather than technological improvements.

Programming multicomputers is not an easy task. The programmer has to take care of distributing code and data among the processors in an efficient way, invoking message-passing calls whenever some data are needed by other processors. On the other hand, *shared-memory multiprocessors* provide a single memory space to all the processors, simplifying the task of exchanging data among processors. Access to shared memory has been traditionally implemented by using an interconnection network between processors and memory (Figure 1.1b). This architecture is referred to as *uniform memory access* (UMA) architecture. It is not scalable because memory access time includes the latency of the interconnection network, and this latency increases with system size.

More recently, shared-memory multiprocessors followed some trends previously established for multicomputers. In particular, memory has been physically distributed among processors, therefore reducing the memory access time for local accesses and increasing scalability. These parallel computers are referred to as *distributed shared-memory multiprocessors* (DSM). Accesses to remote memory

are performed through an interconnection network, very much like in multicomputers. The main difference between DSMs and multicomputers is that messages are initiated by memory accesses rather than by calling a system function. In order to reduce memory latency, each processor has several levels of cache memory, thus matching the speed of processors and memories. This architecture provides *nonuniform memory access* (NUMA) time. Indeed, most of the nonuniformity is due to the different access time between caches and main memories, rather than the different access time between local and remote memories. The main problem arising in DSMs is cache coherence. Several hardware and software cache coherence protocols have been proposed. These protocols produce additional traffic through the interconnection network.

The use of custom interconnects makes multicomputers and DSMs quite expensive. So, *networks of workstations* (NOW) have been proposed as an inexpensive approach to build parallel computers. NOWs take advantage of recent developments in LANs. In particular, the use of ATM switches has been proposed to implement NOWs. However, ATM switches are still expensive, which has motivated the development of high-performance switches, specifically designed to provide a cost-effective interconnect for workstations and personal computers.

Although there are many similarities between interconnection networks for multicomputers and DSMs, it is important to keep in mind that performance requirements may be very different. Messages are usually very short when DSMs are used. Additionally, network latency is important because memory access time depends on that latency. However, messages are typically longer and less frequent when using multicomputers. Usually the programmer is able to adjust the granularity of message communication in a multicomputer. On the other hand, interconnection networks for multicomputers and NOWs are mainly used for message passing. However, the geographical distribution of workstations usually imposes constraints on the way processors are connected. Also, individual processors may be connected to or disconnected from the network at any time, thus imposing additional design constraints.

1.3 Network Design Considerations

Interconnection networks play a major role in the performance of modern parallel computers. There are many factors that may affect the choice of an appropriate interconnection network for the underlying parallel computing platform. These factors include:

1. *Performance requirements.* Processes executing in different processors synchronize and communicate through the interconnection network. These operations are usually performed by explicit message passing or by accessing shared variables. Message *latency* is the time elapsed between the time a message is generated at its source node and the time the message is delivered at its destination node. Message latency directly affects processor idle time and memory access time to remote memory locations. Also, the network may *saturate* — it may be unable to deliver the flow of messages injected by the nodes, limiting the effective computing power of a parallel computer. The maximum amount of information delivered by the network per time unit defines the *throughput* of that network.

2. *Scalability.* A scalable architecture implies that as more processors are added, their memory bandwidth, I/O bandwidth, and network bandwidth should increase proportionally. Otherwise the components whose bandwidth does not scale may become a bottleneck for the rest of the system, decreasing the overall efficiency accordingly.

3. *Incremental expandability.* Customers are unlikely to purchase a parallel computer with a full set of processors and memories. As the budget permits, more processors and memories may be

added until a system's maximum configuration is reached. In some interconnection networks, the number of processors must be a power of 2, which makes them difficult to expand. In other cases, expandability is provided at the cost of wasting resources. For example, a network designed for a maximum size of 1,024 nodes may contain many unused communication links when the network is implemented with a smaller size. Interconnection networks should provide incremental expandability, allowing the addition of a small number of nodes while minimizing resource wasting.

4. *Partitionability.* Parallel computers are usually shared by several users at a time. In this case, it is desirable that the network traffic produced by each user does not affect the performance of other applications. This can be ensured if the network can be partitioned into smaller functional subsystems. Partitionability may also be required for security reasons.

5. *Simplicity.* Simple designs often lead to higher clock frequencies and may achieve higher performance. Additionally, customers appreciate networks that are easy to understand because it is easier to exploit their performance.

6. *Distance span.* This factor may lead to very different implementations. In multicomputers and DSMs, the network is assembled inside a few cabinets. The maximum distance between nodes is small. As a consequence, signals are usually transmitted using copper wires. These wires can be arranged regularly, reducing the computer size and wire length. In NOWs, links have very different lengths and some links may be very long, producing problems such as coupling, electromagnetic noise, and heavy link cables. The use of optical links solves these problems, equalizing the bandwidth of short and long links up to a much greater distance than when copper wire is used. Also, geographical constraints may impose the use of irregular connection patterns between nodes, making distributed control more difficult to implement.

7. *Physical constraints.* An interconnection network connects processors, memories, and/or I/O devices. It is desirable for a network to accommodate a large number of components while maintaining a low communication latency. As the number of components increases, the number of wires needed to interconnect them also increases. Packaging these components together usually requires meeting certain physical constraints, such as operating temperature control, wiring length limitation, and space limitation. Two major implementation problems in large networks are the arrangement of wires in a limited area, and the number of pins per chip (or board) dedicated to communication channels. In other words, the complexity of the connection is limited by the maximum wire density possible, and by the maximum pin count. The speed at which a machine can run is limited by the wire lengths, and the majority of the power consumed by the system is used to drive the wires. This is an important and challenging issue to be considered. Different engineering technologies for packaging, wiring, and maintenance should be considered.

8. *Reliability and repairability.* An interconnection network should be able to deliver information reliably. Interconnection networks can be designed for continuous operation in the presence of a limited number of faults. These networks are able to send messages through alternative paths when some faults are detected. In addition to reliability, interconnection networks should have a modular design, allowing hot upgrades and repairs. Nodes can also fail or be removed from the network. In particular, a node can be powered off in a network of workstations. Thus, NOWs usually require some reconfiguration algorithm for the automatic reconfiguration of the network when a node is powered on or off.

9. *Expected workloads.* Users of a general-purpose machine may have very different requirements. If the kind of applications that will be executed in the parallel computer are known in advance, it may be possible to extract some information on usual communication patterns, message sizes, network load, etc. That information can be used for the optimization of some design parameters. When it is not possible to get information on expected workloads, network design should be robust, i.e., design parameters should be selected in such a way that performance is good over a wide range of traffic conditions.

10. *Cost constraints.* Finally, it is obvious that the "best" network may be too expensive. Design decisions very often are trade-offs between cost and other design factors. Fortunately, cost is not always directly proportional to performance. Using commodity components whenever possible may considerably reduce the overall cost.

1.4 Classification of Interconnection Networks

Among other criteria, interconnection networks have been traditionally classified according to the operating mode (synchronous or asynchronous), and network control (centralized, decentralized, or distributed). Nowadays, multicomputers, multiprocessors, and NOWs dominate the parallel computing market. All of these architectures implement asynchronous networks with distributed control. Therefore, we will focus on other criteria that are currently more significant.

A classification scheme is shown in Figure 1.2 which categorizes the known interconnection networks into four major classes based primarily on network topology: shared-medium networks, direct networks, indirect networks, and hybrid networks. For each class, the figure shows a hierarchy of subclasses, also indicating some real implementations for most of them. This classification scheme is based on the classification proposed in [252], and it mainly focuses on networks that have been implemented. It is by no means complete as other new and innovative interconnection networks may emerge as technology further advances, such as mobile communication and optical interconnections.

In *shared-medium networks*, the transmission medium is shared by all communicating devices. An alternative to this approach consists of having point-to-point links directly connecting each communicating device to a (usually small) subset of other communicating devices in the network. In this case, any communication between nonneighboring devices requires transmitting the information through several intermediate devices. These networks are known as *direct networks.* Instead of directly connecting the communicating devices between them, *indirect networks* connect those devices by means of one or more switches. If several switches exist, they are connected between them using point-to-point links. In this case, any communication between communicating devices requires transmitting the information through one or more switches. Finally, *hybrid* approaches are possible. These network classes and the corresponding subclasses will be described in the following sections.

1.5 Shared-Medium Networks

The least complex interconnect structure is one in which the transmission medium is shared by all communicating devices. In such *shared-medium networks*, only one device is allowed to use the network at a time. Every device attached to the network has requester, driver, and receiver circuits to handle the passing of address and data. The network itself is usually passive, since the network itself does not generate messages.

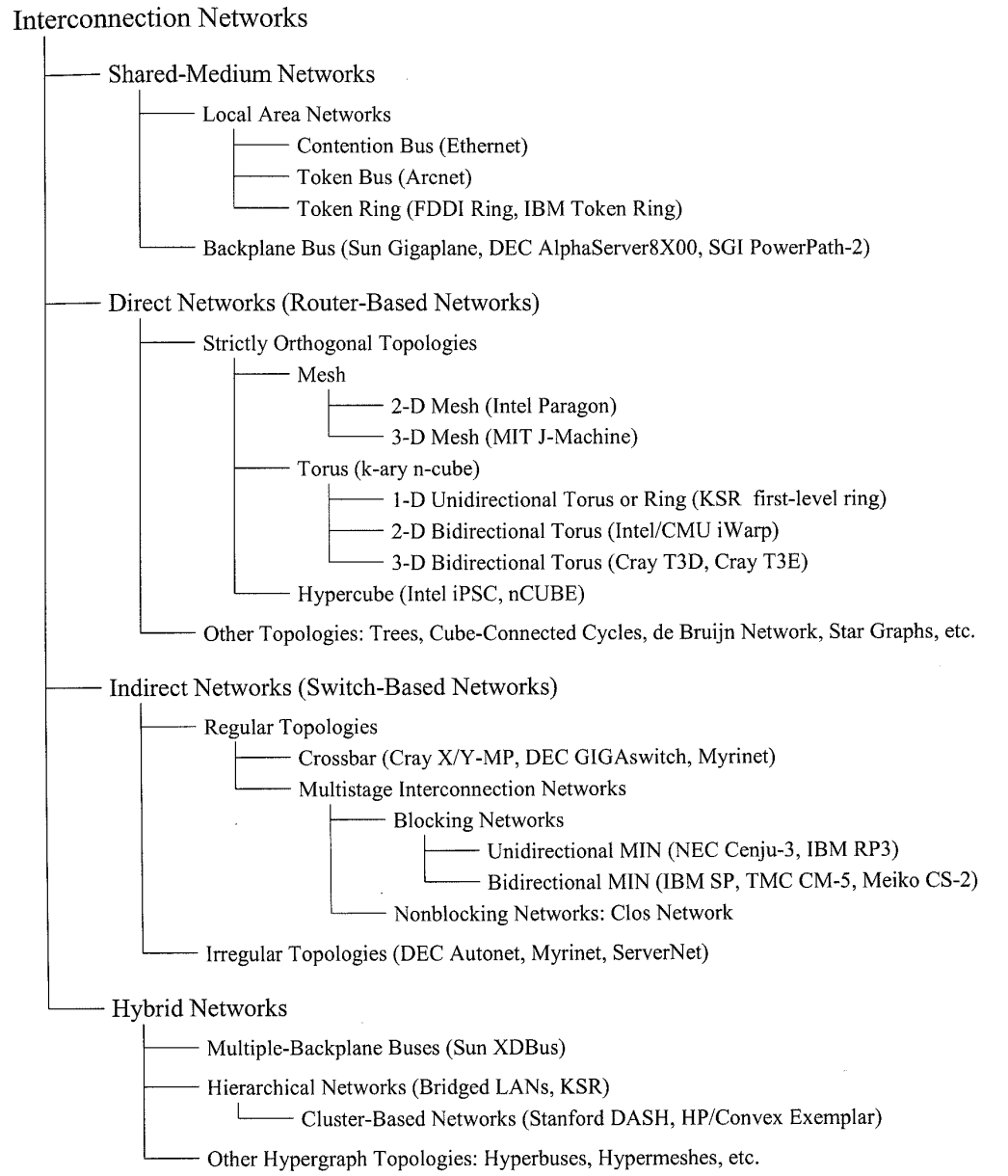


Figure 1.2. Classification of interconnection networks. (1-D = one-dimensional; 2-D = two-dimensional; 3-D = three-dimensional; CMU = Carnegie Mellon University; DASH = Directory Architecture for Shared-Memory; DEC = Digital Equipment Corp.; FDDI = Fiber Distributed Data Interface; HP = Hewlett-Packard; KSR = Kendall Square Research; MIN = Multistage Interconnection Network; MIT = Massachusetts Institute of Technology; SGI = Silicon Graphics Inc.; TMC = Thinking Machines Corp.)

An important issue here is the *arbitration strategy* that determines the mastership of the shared-medium network to resolve network access conflicts. A unique characteristic of a shared medium is its ability to support atomic *broadcast* in which all devices on the medium can monitor network activities and receive the information transmitted on the shared medium. This property is important to efficiently support many applications requiring one-to-all or one-to-many communication services, such as barrier synchronization and snoopy cache coherence protocols. Due to limited network bandwidth, a single shared medium can only support limited number of devices before the medium becomes a bottleneck.

Shared-medium networks constitute a well established technology. Additionally, their limited bandwidth restricts their use in multiprocessors. So, these networks will not be covered in this book, but we will present a short introduction in the following sections. There are two major classes of shared-medium networks: local area networks, mainly used to construct computer networks that span physical distances no longer than a few kilometers, and backplane buses, mainly used for internal communication in uniprocessors and multiprocessors.

1.5.1 Shared-Medium Local Area Networks

High-speed LANs can be used as a networking backbone to interconnect computers to provide an integrated parallel and distributed computing environment. Physically, a shared-medium LAN uses copper wires or fiber optics in a bit-serial fashion as the transmission medium. The network topology is either a bus or a ring. Depending on the arbitration mechanism used, different LANs have been commercially available. For performance and implementation reasons, it is impractical to have a centralized control or to have some fixed access assignment to determine the bus master who can access the bus. Three major classes of LANs based on distributed control are described below.

Contention Bus

The most popular bus arbitration mechanism is to have all devices to compete for the exclusive access right of the bus. Due to the broadcast nature of the bus, all devices can monitor the state of the bus, such as idle, busy, and collision. Here the term "collision" means that two or more devices are using the bus at the same time and their data collided. When the collision is detected, the competing devices will quit transmission and try later. The most well-known contention-based LAN is Ethernet which adopts carrier-sense multiple access with collision detection (CSMA/CD) protocol. The bandwidth of Ethernet is 10 Mbps and the distance span is 250 meters (coaxial cable). As processors are getting faster, the number of devices that can be connected to Ethernet is limited to avoid the network bottleneck. In order to break the 10 Mbps bandwidth barrier, Fast Ethernet can provide 100 Mbps bandwidth.

Token Bus

One drawback of the contention bus is its nondeterministic nature as there is no guarantee of how much waiting time is required to gain the bus access right. Thus, the contention bus is not suitable to support real-time applications. To remove the nondeterministic behavior, an alternate approach involves passing a token among the network devices. The owner of the token has the right to access the bus. Upon completion of the transmission, the token is passed to the next device based on some scheduling discipline. By restricting the maximum token holding time, the upper bound that a device has to wait for the token can be guaranteed. Arcnet supports token bus with a bandwidth of 2.5 Mbps.

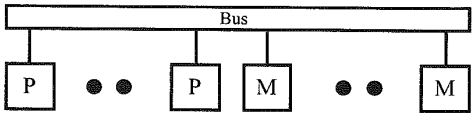


Figure 1.3. A single-bus network. (M = memory; P = processor.)

Token Ring

The idea of token ring is a natural extension of token bus as the passing of the token forms a ring structure. IBM token ring supports bandwidths of both 4 and 16 Mbps based on coaxial cable. Fiber Distributed Data Interface (FDDI) provides a bandwidth of 100 Mbps using fiber optics.

1.5.2 Shared-Medium Backplane Bus

A *backplane bus* is the simplest interconnection structure for bus-based parallel computers. It is commonly used to interconnect processor(s) and memory modules to provide UMA architecture. Figure 1.3 shows a single-bus network. A typical backplane bus usually has 50 – 300 wires and is physically realized by printed lines on a circuit board or by discrete (backplane) wiring. Additional costs are incurred by interface electronics, such as line drivers, receivers, and connectors.

There are three kinds of information in the backplane bus: data, address, and control signals. Control signals include bus request signal and request grant signal, among many others. In addition of the width of data lines, the maximum bus bandwidth that can be provided is dependent on the technology. The number of processors that can be put on a bus depends on many factors, such as processor speed, bus bandwidth, cache architecture, and program behavior.

Methods of Information Transfer

Both data and address information must be carried in the bus. In order to increase the bus bandwidth and provide a large address space, both data width and address bits have to be increased. Such an increase implies another increase in the bus complexity and cost. Some designs try to share address and data lines. For *multiplexed transfer*, address and data are sent alternatively. Hence, they can share the same physical lines and require less power and fewer chips. For *nonmultiplexed transfer*, address and data lines are separated. Thus, data transfer can be done faster.

In *synchronous bus* design, all devices are synchronized with a common clock. It requires less complicated logic and has been used in most existing buses. However, a synchronous bus is not easily upgradable. New faster processors are difficult to fit into a slow bus.

In *asynchronous buses*, all devices connected to the bus may have different speeds and their own clocks. They use a handshaking protocol to synchronize with each other. This provides independence for different technologies and allows slower and faster devices with different clock rates to operate together. This also implies buffering is needed, since slower devices cannot handle messages as fast as faster devices.

Bus Arbitration

In a single-bus network, several processors may attempt to use the bus simultaneously. To deal with this, a policy must be implemented that allocates the bus to the processors making such requests. For performance reasons, bus allocation must be carried out by hardware arbiters. Thus, in order to perform a memory access request, the processor has to exclusively own the bus and become the *bus*

master. To become the bus master, each processor implements a *bus requester*, which is a collection of logic to request control of the data transfer bus. On gaining control, the requester notifies the requesting master.

Two alternative strategies are used to release the bus:

- Release-when-done: release the bus when data transfer is done
- Release-on-request: hold the bus until another processor requests it

Several different *bus arbitration algorithms* have been proposed, which can be classified into *centralized* or *distributed*. A centralized method has a central *bus arbiter*. When a processor wants to become the bus master, it sends out a bus request to the bus arbiter which then sends out a request grant signal to the requesting processor. A bus arbiter can be an encoder-decoder pair in hardware design. In distributed method, such as daisy chain method, there is no central bus arbiter. The bus request signals form a daisy chain. The mastership is released to the next device when data transfer is done.

Split Transaction Protocol

Most bus transactions involve request and response. This is the case for memory read operations. After a request is issued, it is desirable to have a fast response. If a fast response time is expected, the bus mastership is not released after sending the request, and data can be received soon. However, due to memory latency, the bus bandwidth is wasted while waiting for a response. In order to minimize the waste of bus bandwidth, the *split transaction protocol* has been used in many bus networks.

In this protocol, the bus mastership is released immediately after the request, and the memory has to gain mastership before it can send the data. Split transaction protocol has a better bus utilization but its control unit is much more complicated. *Buffering* is needed in order to save messages before the device can gain the bus mastership.

To support shared-variable communication, some atomic read/modify/write operations to memories are needed. With the split transaction protocol, the atomic read/modify/write can no longer be indivisible. One approach to solve this problem is to disallow bus release for those atomic operations.

Bus Examples

Several examples of buses and the main characteristics are listed below.

- Gigaplane used in Sun Ultra Enterprise X000 Server (ca. 1996): 2.6 Gbyte/s peak, 256 bits data, 42 bits address, split-transaction protocol, 83.8 MHz clock.
- DEC AlphaServer8X00, i.e., 8200 and 8400 (ca. 1995): 2.1 Gbyte/s, 256 bits data, 40 bits address, split-transaction protocol, 100 MHz clock (1 foot length).
- SGI PowerPath-2 (ca. 1993): 1.2 Gbyte/s, 256 bits data, 40 bits address, 6 bits control, split-transaction protocol, 47.5 MHz clock (1 foot length).
- HP9000 Multiprocessor Processor Memory Bus (ca. 1993): one Gbyte/s, 128 bits data, 64 bits address, 13 inches, pipelined-bus, 60 MHz clock.

1.6 Direct Networks

Scalability is an important issue in designing multiprocessor systems. Bus-based systems are not scalable as the bus becomes the bottleneck when more processors are added. The *direct network* or *point-to-point network* is a popular network architecture that scales well to a large number of processors. A direct network consists of a set of *nodes*, each one being directly connected to a (usually small) subset of other nodes in the network. Figures 1.5 through 1.7 show several direct networks. The corresponding interconnection patterns between nodes will be studied below. Each node is a programmable computer with its own processor, local memory, and other supporting devices. These nodes may have different functional capabilities. For example, the set of nodes may contain vector processors, graphics processors, and I/O processors. Figure 1.4 shows the architecture of a generic node. A common component of these nodes is a *router*, which handles message communication among nodes. For this reason, direct networks are also known as router-based networks. Each router has direct connections to the router of its neighbors. Usually, two neighboring nodes are connected by a pair of unidirectional channels in opposite directions. A bidirectional channel may also be used to connect two neighboring nodes. Although the function of a router can be performed by the local processor, dedicated routers have been used in high-performance multicomputers, allowing overlapped computation and communication within each node. As the number of nodes in the system increases, the total communication bandwidth, memory bandwidth, and processing capability of the system also increase. Thus, direct networks have been a popular interconnection architecture for constructing large-scale parallel computers.

Each router supports some number of input and output channels. *Internal* channels or *ports* connect the local processor/memory to the router. Although it is common to provide only one pair of internal channels, some systems use more internal channels in order to avoid a communication bottleneck between the local processor/memory and the router [39]. *External* channels are used for communication between routers. By connecting input channels of one node to the output channels of other nodes, the direct network is defined. Unless otherwise specified, the term "channel" will refer to an external channel. Two directly connected nodes are called *neighboring* or *adjacent* nodes. Usually, each node has a fixed number of input and output channels, and every input channel is paired with a corresponding output channel. Through the connections among these channels, there are many ways to interconnect these nodes. Obviously, every node in the network should be able to reach every other node.

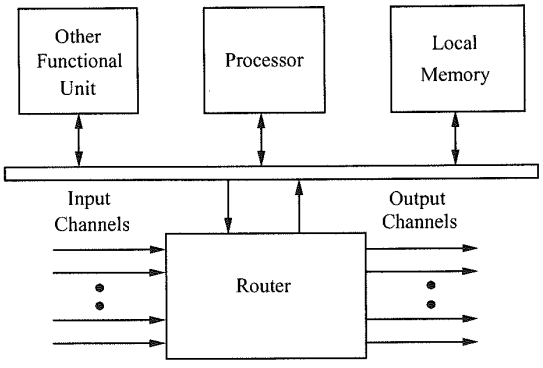


Figure 1.4. A generic node architecture.

1.6.1 Characterization of Direct Networks

Direct networks have been traditionally modeled by a graph $G(N, C)$, where the vertices of the graph N represent the set of processing nodes, and the edges of the graph C represent the set of communication channels. This is a very simple model that does not consider implementation issues. However, it allows the study of many interesting network properties. Depending on the properties under study, a bidirectional channel may be modeled either as an edge or as two arcs in opposite directions (two unidirectional channels). The latter is the case for deadlock avoidance in Chapter 3. Let us assume that a bidirectional channel is modeled as an edge. Some basic network properties can be defined from the graph representation:

- *Node degree:* Number of channels connecting that node to its neighbors.
- *Diameter:* The maximum distance between two nodes in the network.
- *Regularity:* A network is *regular* when all the nodes have the same degree.
- *Symmetry:* A network is *symmetric* when it looks alike from every node.

A direct network is mainly characterized by three factors: topology, routing, and switching. The *topology* defines how the nodes are interconnected by channels, and is usually modeled by a graph as indicated above. For direct networks, the ideal topology would connect every node to every other node. No message would even have to pass through an intermediate node before reaching its destination. This fully connected topology requires a router with N links (including the internal one) at each node for a network with N nodes. Therefore, the cost is prohibitive for networks of moderate to large size. Additionally, the number of physical connections of a node is limited by hardware constraints such as the number of available pins and the available wiring area. These engineering and scaling difficulties preclude the use of such fully connected networks even for small network sizes. As a consequence, many topologies have been proposed, trying to balance performance and some cost parameters. In these topologies, messages may have to traverse some intermediate nodes before reaching the destination node.

From the programmer's perspective, the unit of information exchange is the *message*. The size of messages may vary depending on the application. For efficient and fair use of network resources, a message is often divided into packets prior to transmission. A *packet* is the smallest unit of communication that contains the destination address and sequencing information, which are carried in the packet *header*. For topologies in which packets may have to traverse some intermediate nodes, the *routing* algorithm determines the path selected by a packet to reach its destination. At each intermediate node, the routing algorithm indicates the next channel to be used. That channel may be selected among a set of possible choices. If all the candidate channels are busy, the packet is blocked and cannot advance. Obviously, efficient routing is critical to the performance of interconnection networks.

When a message or packet header reaches an intermediate node, a *switching* mechanism determines how and when the router switch is set, i.e., the input channel is connected to the output channel selected by the routing algorithm. In other words, the switching mechanism determines how network resources are allocated for message transmission. For example, in circuit switching, all the channels required by a message are reserved before starting message transmission. In packet switching, however, a packet is transmitted through a channel as soon as that channel is reserved but the next channel is not reserved (assuming that it is available) until the packet releases the channel it is currently using. Obviously, some buffer space is required to store the packet until the next channel is reserved. That buffer should be allocated before starting packet transmission. So,

buffer allocation is closely related to the switching mechanism. Flow control is also closely related to the switching and buffer allocation mechanisms. The *flow control* mechanism establishes a dialog between sender and receiver nodes, allowing and stopping the advance of information. If a packet is blocked, it requires some buffer space to be stored. When there is no more available buffer space, the flow control mechanism stops information transmission. When the packet advances and buffer space is available, transmission is started again. If there is no flow control and no more buffer space is available, the packet may be dropped, or derouted through another channel.

The above factors affect the network performance. They are not independent of each other but are closely related. For example, if a switching mechanism reserves resources in an aggressive way (as soon as a packet header is received), packet latency can be reduced. However, each packet may be holding several channels at the same time. So, such a switching mechanism may cause severe network congestion and, consequently, make the design of efficient routing and flow control policies difficult. The network topology also affects performance, as well as how the network traffic can be distributed over available channels. In most cases, the choice of a suitable network topology is restricted by wiring and packaging constraints.

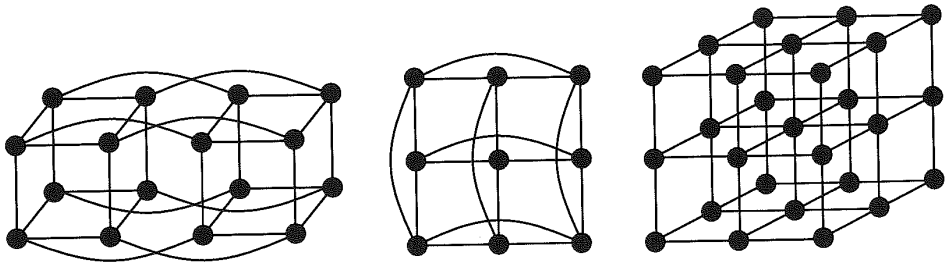
1.6.2 Popular Network Topologies

Many network topologies have been proposed in terms of their graph theoretic properties. However, very few of them have ever been implemented. Most of the implemented networks have an orthogonal topology. A network topology is *orthogonal* if and only if nodes can be arranged in an orthogonal n-dimensional space, and every link can be arranged in such a way that it produces a displacement in a single dimension. Orthogonal topologies can be further classified as strictly orthogonal and weakly orthogonal. In a *strictly orthogonal* topology, every node has at least one link crossing each dimension. In a *weakly orthogonal* topology, some nodes may not have any link in some dimensions. Hence, it is not possible to cross every dimension from every node. Crossing a given dimension from a given node may require moving in another dimension first.

Strictly Orthogonal Topologies

The most interesting property of strictly orthogonal topologies is that routing is very simple. Thus, the routing algorithm can be efficiently implemented in hardware. Effectively, in a strictly orthogonal topology nodes can be numbered by using their coordinates in the n-dimensional space. As each link traverses a single dimension and every node has at least one link crossing each dimension, the distance between two nodes can be computed as the sum of dimension offsets. Also, the displacement along a given link only modifies the offset in the corresponding dimension. Taking into account that it is possible to cross any dimension from any node in the network, routing can be easily implemented by selecting a link that decrements the absolute value of the offset in some dimension. The set of dimension offsets can be stored in the packet header, and updated (by adding or subtracting one unit) every time the packet is successfully routed at some intermediate node. If the topology is not strictly orthogonal, however, routing may become much more complex.

The most popular direct networks are the *n-dimensional mesh*, the *k-ary n-cube* or *torus*, and the *hypercube*. All of them are strictly orthogonal. Formally, an n-dimensional mesh has $k_0 \times k_1 \times \cdots \times k_{n-2} \times k_{n-1}$ nodes, k_i nodes along each dimension i, where $k_i \geq 2$ and $0 \leq i \leq n-1$. Each node X is identified by n coordinates, $(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$, where $0 \leq x_i \leq k_i - 1$ for $0 \leq i \leq n-1$. Two nodes X and Y are neighbors if and only if $y_i = x_i$ for all i, $0 \leq i \leq n-1$, except one, j, where $y_j = x_j \pm 1$. Thus, nodes have from n to $2n$ neighbors, depending on their location in the mesh. Therefore, this topology is not regular.



(a) 2–ary 4–cube (hypercube) (b) 3–ary 2–cube (c) 3–ary 3–D mesh

Figure 1.5. Strictly orthogonal direct network topologies.

In a bidirectional k-ary n-cube [71], all nodes have the same number of neighbors. The definition of a k-ary n-cube differs from that of an n-dimensional mesh in that all of the k_i are equal to k and two nodes X and Y are neighbors if and only if $y_i = x_i$ for all i, $0 \leq i \leq n-1$, except one, j, where $y_j = (x_j \pm 1) \bmod k$. The change to modular arithmetic in the definition adds wraparound channels to the k-ary n-cube, giving it regularity and symmetry. Every node has n neighbors if $k = 2$ and $2n$ neighbors if $k > 2$. When $n = 1$, the k-ary n-cube collapses to a bidirectional *ring* with k nodes.

Another topology with regularity and symmetry is the hypercube, which is a special case of both n-dimensional meshes and k-ary n-cubes. A hypercube is an n-dimensional mesh in which $k_i = 2$ for $0 \leq i \leq n-1$, or a 2-ary n-cube, also referred to as a binary n-cube.

Figure 1.5a depicts a binary 4-cube or 16-node hypercube. Figure 1.5b illustrates a 3-ary 2-cube or two-dimensional (2-D) torus. Figure 1.5c shows a 3-ary three-dimensional (3-D) mesh, resulting by removing the wraparound channels from a 3-ary 3-cube.

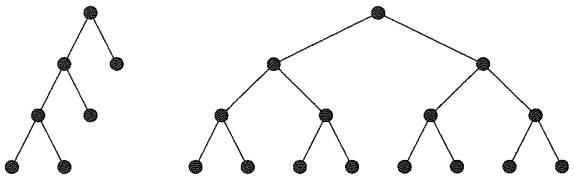
Two conflicting requirements of a direct network are that it must accommodate a large number of nodes while maintaining a low network latency. This issue will be addressed in Chapter 7.

Other Direct Network Topologies

In addition to the topologies defined above, many other topologies have been proposed in the literature. Most of them were proposed with the goal of minimizing the network diameter for a given number of nodes and node degree. As will be seen in Chapter 2, for pipelined switching techniques network latency is almost insensitive to network diameter, especially when messages are long. So it is unlikely that those topologies are implemented. In the following paragraphs, we present an informal description of some relevant direct network topologies.

A popular topology is the *tree*. This topology has a *root* node connected to a certain number of descendant nodes. Each of these nodes is in turn connected to a disjoint set (possibly empty) of descendants. A node with no descendants is a *leaf* node. A characteristic property of trees is that every node but the root has a single parent node. Therefore, trees contain no cycles. A tree in which every node but the leaves has a fixed number k of descendants is a k-ary tree. When the distance from every leaf node to the root is the same, i.e., all the branches of the tree have the same length, the tree is *balanced*. Figures 1.6a and 1.6b show an unbalanced and a balanced binary tree, respectively.

The most important drawback of trees as general-purpose interconnection networks is that the root node and the nodes close to it become a bottleneck. Additionally, there are no alternative paths between any pair of nodes. The bottleneck can be removed by allocating a higher channel



(a) Binary Tree (b) Balanced Binary Tree

Figure 1.6. Some tree topologies.

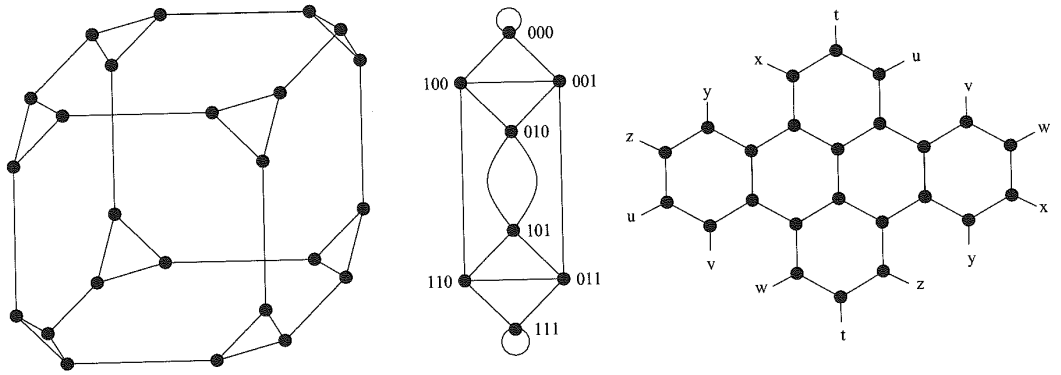
bandwidth to channels located close to the root node. The shorter the distance to the root node, the higher the channel bandwidth. However, using channels with different bandwidth is not practical, specially when message transmission is pipelined. A practical way to implement trees with higher channel bandwidth in the vicinity of the root node (fat trees) will be described in Section 1.7.5.

One of the most interesting properties of trees is that, for any connected graph, it is possible to define a tree that spans the complete graph. As a consequence, for any connected network, it is possible to build an acyclic network connecting all the nodes by removing some links. This property can be used to define a routing algorithm for any irregular topology. However, that routing algorithm may be inefficient due to the concentration of traffic across the root node. A possible way to circumvent that limitation will be presented in Section 4.9.

Some topologies have been proposed with the purpose of reducing node degree while keeping the diameter small. Most of these topologies can be viewed as a hierarchy of topologies. This is the case for the cube-connected cycles [283]. This topology can be considered as an n-dimensional hypercube of virtual nodes, where each virtual node is a ring with n nodes, for a total of $n2^n$ nodes. Each node in the ring is connected to a single dimension of the hypercube. Therefore, node degree is fixed and equal to three: two links connecting to neighbors in the ring, and one link connecting to a node in another ring through one of the dimensions of the hypercube. However, the diameter is of the same order of magnitude as that of a hypercube of similar size. Figure 1.7a shows a 24-node cube-connected cycles network. It is worth to note that cube-connected cycles are weakly orthogonal because the ring is a one-dimensional network, and displacement inside the ring does not change the position in the other dimensions. Similarly, a displacement along a hypercube dimension does not affect the position in the ring. However, it is not possible to cross every dimension from each node.

Many topologies have been proposed with the purpose of minimizing the network diameter for a given number of nodes and node degree. Two well-known topologies proposed with this purpose are the de Bruijn network and the star graphs. In the *de Bruijn* network [300] there are d^n nodes, and each node is represented by a set of n digits in base d. A node $(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$, where $0 \leq x_i \leq d-1$ for $0 \leq i \leq n-1$ is connected to nodes $(x_{n-2}, \ldots, x_1, x_0, p)$ and $(p, x_{n-1}, x_{n-2}, \ldots, x_1)$, for all p such that $0 \leq p \leq d-1$. In other words, two nodes are connected if the representation of one node is a right or left shift of the representation of the other. Figure 1.7b shows an eight-node de Bruijn network. When networks are very large, this network topology achieves a very low diameter for a given number of nodes and node degree. However, routing is complex. Additionally, the average distance between nodes is high, close to the network diameter. Finally, some nodes have links connecting to themselves. All of these issues make the practical application of these networks very difficult.

A *star graph* [6] can be informally described as follows. The vertices of the graph are labeled by permutations of n different symbols, usually denoted as 1 to n. A permutation is connected to every other permutation that can be obtained from it by interchanging the first symbol with any



(a) Cube-Connected Cycles (b) De Bruijn Network (c) Star Graph

Figure 1.7. Other direct network topologies.

of the other symbols. A star graph has $n!$ nodes and node degree is equal to $n-1$. Figure 1.7c shows a star graph obtained by permutations of four symbols. Although a star graph has a lower diameter than a hypercube of similar size, routing is more complex. Star graphs were proposed as a particular case of Cayley graphs [6]. Other topologies like hypercubes and cube-connected cycles are also particular cases of Cayley graphs.

1.6.3 Examples

Several examples of parallel computers with direct networks and the main characteristics are listed below.

- Cray T3E: Bidirectional 3-D torus, 14-bit data in each direction, 375 MHz link speed, 600 Mbytes/s per link.
- Cray T3D: Bidirectional 3-D torus, up to 1,024 nodes ($8 \times 16 \times 8$), 24-bit links (16-bit data, 8-bit control), 150 MHz, 300 Mbytes/s per link.
- Intel Cavallino: Bidirectional 3-D topology, 16-bit-wide channels operating at 200 MHz, 400 Mbytes/s in each direction.
- SGI SPIDER: Router with 20-bit bidirectional channels operating on both edges, 200 MHz clock, aggregate raw data rate of 1 Gbyte/s. Support for regular and irregular topologies.
- MIT M-Machine: 3-D mesh, 800 Mbytes/s for each network channel.
- MIT Reliable Router: 2-D mesh, 23-bit links (16-bit data), 200 MHz, 400 Mbytes/s per link per direction, bidirectional signaling, reliable transmission.
- Chaos Router: 2-D torus topology, bidirectional 8-bit links, 180 MHz, 360 Mbytes/s in each direction.
- Intel iPSC-2 Hypercube: Binary hypercube topology, bit-serial channels at 2.8 Mbytes/s.

1.7 Indirect Networks

Indirect or *switch-based networks* are another major class of interconnection networks. Instead of providing a direct connection among some nodes, the communication between any two nodes has to be carried through some *switches*. Each node has a network adapter that connects to a network switch. Each switch can have a set of *ports*. Each port consists of one input and one output link. A (possibly empty) set of ports in each switch are either connected to processors or left open, whereas the remaining ports are connected to ports of other switches to provide connectivity between the processors. The interconnection of those switches defines various network topologies.

Switch-based networks considerably evolved over time. A wide range of topologies have been proposed, ranging from regular topologies used in array processors and shared-memory UMA multiprocessors to the irregular topologies currently used in NOWs. Both network classes will be covered in this book. Regular topologies have regular connection patterns between switches while irregular topologies do not follow any predefined pattern. Figures 1.19 and 1.21 show several switch-based networks with regular topology. The corresponding connection patterns will be studied below. Figure 1.8 shows a typical switch-based network with irregular topology. Both network classes can be further classified according to the number of switches a message has to traverse before reaching its destination. Although this classification is not important in the case of irregular topologies, it makes a big difference in the case of regular networks because some specific properties can be derived for each network class.

1.7.1 Characterization of Indirect Networks

Indirect networks can also be modeled by a graph $G(N, C)$, where N is the set of switches, and C is the set of unidirectional or bidirectional links between the switches. For the analysis of most properties, it is not necessary to explicitly include processing nodes in the graph. Although a similar model can be used for direct and indirect networks, a few differences exist between them. Each switch in an indirect network may be connected to zero, one, or more processors. Obviously, only the switches connected to some processor can be the source or the destination of a message. Additionally, transmitting a message from a node to another node requires crossing the link between

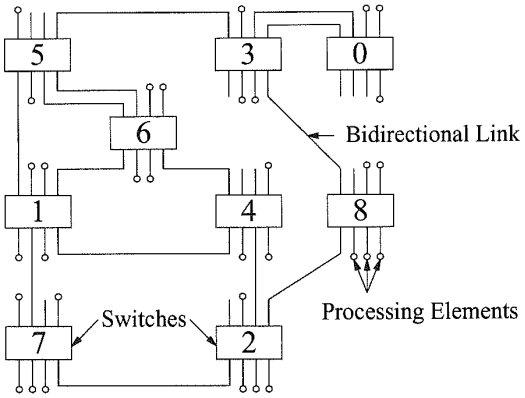


Figure 1.8. A switch-based network with irregular topology.

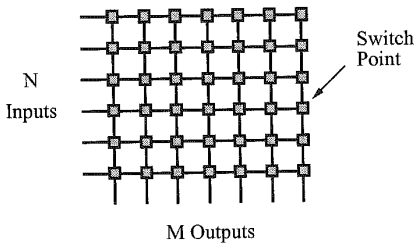


Figure 1.9. An $N \times M$ crossbar.

the source node and the switch connected to it, and the link between the last switch in the path and the destination node. Therefore, the distance between two nodes is the distance between the switches directly connected to those nodes plus two units. Similarly, the diameter is the maximum distance between two switches connected to some node plus two units. It may be argued that it is not necessary to add two units because direct networks also have internal links between routers and processing nodes. However, those links are external in the case of indirect networks. This gives a consistent view of the diameter as the maximum number of external links between two processing nodes. In particular, the distance between two nodes connected through a single switch is two instead of zero.

Similar to direct networks, an indirect network is mainly characterized by three factors: topology, routing, and switching. The topology defines how the switches are interconnected by channels, and can be modeled by a graph as indicated above. For indirect networks with N nodes, the ideal topology would connect those nodes through a single $N \times N$ switch. Such a switch is known as a *crossbar*. Although using a single $N \times N$ crossbar is much cheaper than using a fully connected direct network topology (requiring N routers, each one having an internal $N \times N$ crossbar), the cost is still prohibitive for large networks. Similar to direct networks, the number of physical connections of a switch is limited by hardware constraints such as the number of available pins and the available wiring area. These engineering and scaling difficulties preclude the use of crossbar networks for large network sizes. As a consequence, many alternative topologies have been proposed. In these topologies, messages may have to traverse several switches before reaching the destination node. In regular networks, these switches are usually identical and have been traditionally organized as a set of *stages*. Each stage (but the input/output stages) is only connected to the previous and next stages using regular connection patterns. Input/output stages are connected to the nodes as well as to another stage in the network. These networks are referred to as *multistage* networks, and have different properties depending on the number of stages, and how those stages are arranged.

The remaining issues discussed in Section 1.6.1 (routing, switching, flow control, buffer allocation, and their impact on performance) are also applicable to indirect networks.

1.7.2 Crossbar Networks

Crossbar networks allow any processor in the system to connect to any other processor or memory unit so that many processors can communicate simultaneously without contention. A new connection can be established at any time as long as the requested input and output ports are free. Crossbar networks are used in the design of high-performance small-scale multiprocessors, in the design of routers for direct networks, and as basic components in the design of large-scale indirect networks. A crossbar can be defined as a switching network with N inputs and M outputs, which allows up to

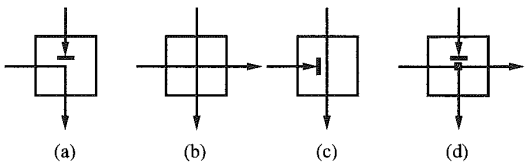


Figure 1.10. States of a switch point in a crossbar network.

$\min\{N, M\}$ one-to-one interconnections without contention. Figure 1.9 shows an $N \times M$ crossbar network. Usually, $M = N$ except for crossbars connecting processors and memory modules.

The cost of such a network is $O(NM)$, which is prohibitively high with large N and M. Crossbar networks have been traditionally used in small-scale shared-memory multiprocessors, where all processors are allowed to access memories simultaneously as long as each processor reads from, or writes to, a different memory. When two or more processors contend for the same memory module, arbitration lets one processor proceed while the others wait. The arbiter in a crossbar is distributed among all the switch points connected to the same output. However, the arbitration scheme can be less complex than the one for a bus because conflicts in crossbar are the exception rather than the rule, and therefore easier to resolve.

For a crossbar network with distributed control, each switch point may have four states as shown in Figure 1.10. In Figure 1.10a, the input from the row containing the switch point has been granted access to the corresponding output while inputs from upper rows requesting the same output are blocked. In Figure 1.10b, an input from an upper row has been granted access to the output. The input from the row containing the switch point does not request that output, and can be propagated to other switches. In Figure 1.10c, an input from an upper row has also been granted access to the output. However, the input from the row containing the switch point also requests that output and is blocked. The configuration in Figure 1.10d is only required if the crossbar has to support multicasting (one-to-many communication).

The advent of VLSI permitted the integration of hardware for thousands of switches into a single chip. However, the number of pins on a VLSI chip cannot exceed a few hundreds, which restricts the size of the largest crossbar that can be integrated into a single VLSI chip. Large crossbars can be realized by partitioning them into smaller crossbars, each one implemented using a single chip. Thus, a full crossbar of size $N \times N$ can be implemented with $(N/n)(N/n)$ $n \times n$ crossbars.

1.7.3 Multistage Interconnection Networks

Multistage interconnection networks (MINs) connect input devices to output devices through a number of switch stages, where each switch is a crossbar network. The number of stages and the connection patterns between stages determine the routing capability of the networks.

MINs were initially proposed for telephone networks and later for array processors. In these cases, a central controller establishes the path from input to output. In cases where the number of inputs equals the number of outputs, each input synchronously transmits a message to one output, and each output receives a message from exactly one input. Such unicast communication patterns can be represented as a permutation of the input addresses. For this application, MINs have been popular as alignment networks for storing and accessing arrays in parallel from memory banks. Array storage is typically skewed to permit conflict-free access, and the network is used to unscramble the arrays during access. These networks can also be configured with the number of inputs greater than the number of outputs (concentrators) and vice versa (expanders). On the other hand, in asynchronous

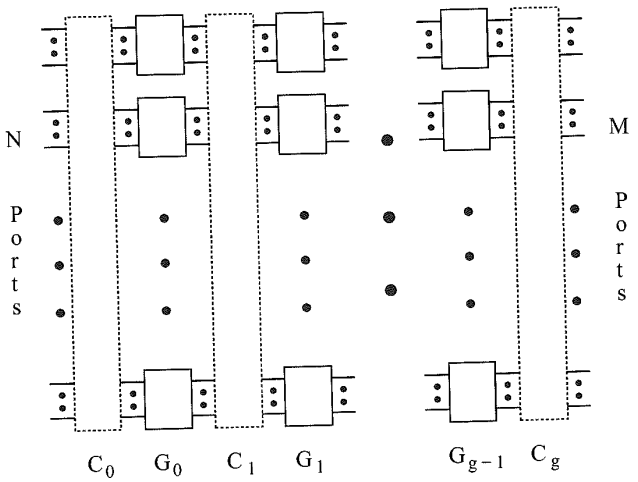


Figure 1.11. A generalized MIN with N inputs, M outputs, and g stages.

multiprocessors, centralized control and permutation routing are infeasible. In this case, a routing algorithm is required to establish the path across the stages of a MIN.

Depending on the interconnection scheme employed between two adjacent stages and the number of stages, various MINs have been proposed. MINs are good for constructing parallel computers with hundreds of processors and have been used in some commercial machines.

1.7.4 A Generalized MIN Model

There are many ways to interconnect adjacent stages. Figure 1.11 shows a generalized multistage interconnection network with N inputs and M outputs. It has g *stages*, G_0 to G_{g-1}. As shown in Figure 1.12, each stage, say G_i, has w_i switches of size $a_{i,j} \times b_{i,j}$, where $1 \leq j \leq w_i$. Thus, stage G_i has p_i inputs and q_i outputs, where

$$p_i = \sum_{j=1}^{w_i} a_{i,j} \quad \text{and} \quad q_i = \sum_{j=1}^{w_i} b_{i,j}$$

The connection between two adjacent stages, G_{i-1} and G_i, denoted C_i, defines the *connection pattern* for $p_i = q_{i-1}$ links, where $p_0 = N$ and $q_{g-1} = M$. A MIN thus can be represented as

$$C_0(N)G_0(w_0)C_1(p_1)G_1(w_1)\ldots G_{g-1}(w_{g-1})C_g(M)$$

A connection pattern $C_i(p_i)$ defines how those p_i links should be connected between the $q_{i-1} = p_i$ outputs from stage G_{i-1} and the p_i inputs to stage G_i. Different connection patterns give different characteristics and topological properties of MINs. The links are labeled from 0 to $p_i - 1$ at C_i.

From a practical point of view, it is interesting that all the switches are identical, thus amortizing the design cost. Banyan networks are a class of MINs with the property that there is a unique path between any pair of source and destination [133]. An N-node ($N = k^n$) Delta network is a subclass of banyan networks, which is constructed from identical $k \times k$ switches in n stages, where each stage contains $\frac{N}{k}$ switches. Many of the known MINs, such as Omega, flip, cube, butterfly, and baseline,

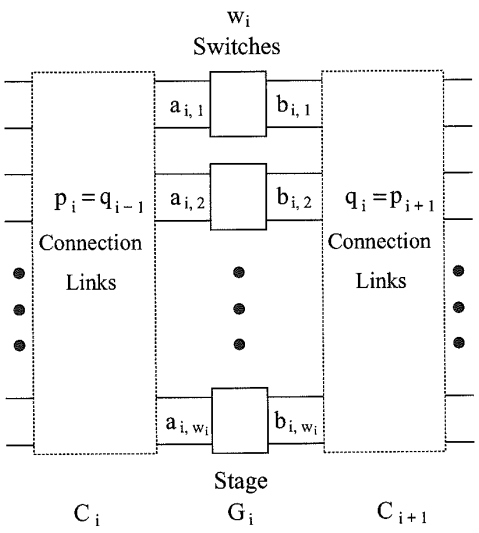


Figure 1.12. A closer view of stage G_i.

belong to the class of Delta networks [272] and have been shown to be topologically and functionally equivalent [351]. A good survey of those MINs can be found in [317]. Some of those MINs are defined below.

When switches have the same number of input and output ports, MINs also have the same number of input and output ports. Since there is a one-to-one correspondence between inputs and outputs, these connections are also called *permutations*. Five basic permutations are defined below. Although these permutations were originally defined for networks with 2×2 switches, for most definitions we assume that the network is built by using $k \times k$ switches, and that there are $N = k^n$ inputs and outputs, where n is an integer. However, some of these permutations are only defined for the case where N is a power of 2. With $N = k^n$ ports, let $X = x_{n-1}x_{n-2}\ldots x_0$ be an arbitrary port number, $0 \leq X \leq N-1$, where $0 \leq x_i \leq k-1$, $0 \leq i \leq n-1$.

Perfect Shuffle Connection

The *perfect k-shuffle* connection σ^k is defined by

$$\sigma^k(X) = (kX + \left\lfloor \frac{kX}{N} \right\rfloor) \bmod N$$

A more cogent way to describe the perfect k-shuffle connection σ^k is

$$\sigma^k(x_{n-1}x_{n-2}\ldots x_1x_0) = x_{n-2}\ldots x_1x_0x_{n-1}$$

The perfect k-shuffle connection performs a cyclic shifting of the digits in X to the left for one position. For $k = 2$, this action corresponds to perfectly shuffling a deck of N cards, as demonstrated in Figure 1.13a for the case of $N = 8$. The perfect shuffle cuts the deck into two halves from the center and intermixes them evenly. The *inverse perfect shuffle* does the opposite as defined below:

$$\sigma^{k^{-1}}(x_{n-1}x_{n-2}\ldots x_1x_0) = x_0x_{n-1}\ldots x_2x_1$$

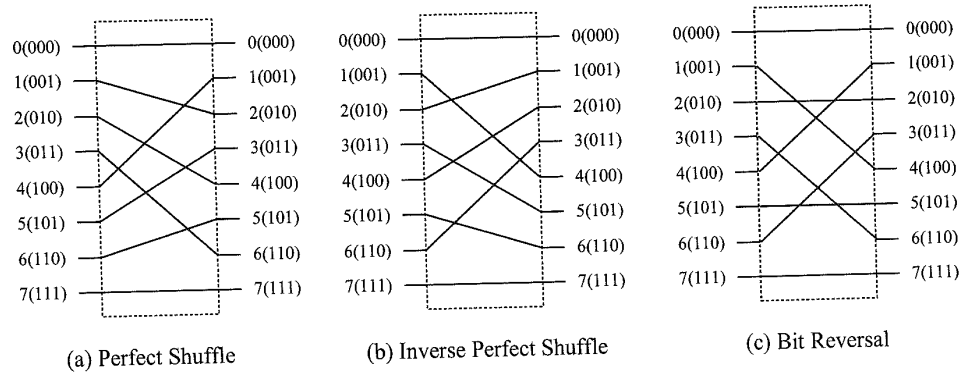


Figure 1.13. The perfect shuffle, inverse perfect shuffle, and bit reversal connections for $N = 8$.

Digit Reversal Connection

The *digit reversal* permutation ρ^k is defined by

$$\rho^k(x_{n-1}x_{n-2}\dots x_1x_0) = x_0x_1\dots x_{n-2}x_{n-1}$$

This permutation is usually referred to as *bit reversal*, clearly indicating that it was proposed for $k = 2$. However, its definition is also valid for $k > 2$. Figure 1.13c demonstrates a bit reversal connection for the case of $k = 2$ and $N = 8$.

Butterfly Connection

The ith k-ary *butterfly* permutation β_i^k, for $0 \le i \le n-1$, is defined by

$$\beta_i^k(x_{n-1}\dots x_{i+1}x_ix_{i-1}\dots x_1x_0) = x_{n-1}\dots x_{i+1}x_0x_{i-1}\dots x_1x_i$$

The ith butterfly connection interchanges the zeroth and ith digits of the index. Figure 1.14 shows the butterfly connection for $k = 2$, and $i = 0$, 1, and 2 with $N = 8$. Note that β_0^k defines a straight one-to-one connection and is also called *identity connection*, I.

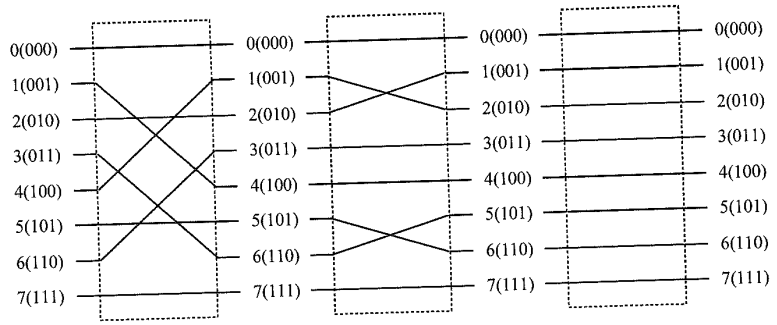
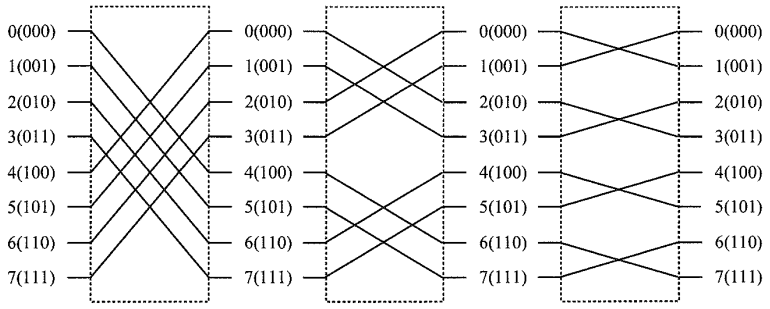


Figure 1.14. The butterfly connection for $N = 8$.



(a) Second Cube (b) First Cube (c) Zeroth Cube

Figure 1.15. The cube connection for $N = 8$.

Cube Connection

The ith *cube* connection E_i, for $0 \leq i \leq n-1$, is defined only for $k = 2$ by

$$E_i(x_{n-1} \dots x_{i+1} x_i x_{i-1} \dots x_0) = x_{n-1} \dots x_{i+1} \overline{x}_i x_{i-1} \dots x_0$$

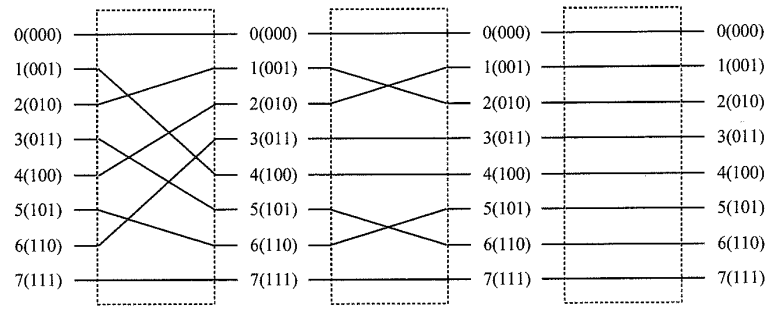
The ith cube connection complements the ith bit of the index. Figure 1.15 shows the cube connection for $i = 0$, 1, and 2 with $N = 8$. E_0 is also called the *exchange* connection.

Baseline Connection

The ith k-ary *baseline* permutation δ_i^k, for $0 \leq i \leq n-1$, is defined by

$$\delta_i^k(x_{n-1} \dots x_{i+1} x_i x_{i-1} \dots x_1 x_0) = x_{n-1} \dots x_{i+1} x_0 x_i x_{i-1} \dots x_1$$

The ith baseline connection performs a cyclic shifting of the $i+1$ least significant digits in the index to the right for one position. Figure 1.16 shows the baseline connection for $k = 2$, and $i = 0$, 1, and 2 with $N = 8$. Note that δ_0^k also defines the identity connection I.



(a) Second Baseline (b) First Baseline (c) Zeroth Baseline

Figure 1.16. The baseline connection for $N = 8$.

1.7.5 Classification of Multistage Interconnection Networks

Depending on the availability of paths to establish new connections, MINs have been traditionally divided into three classes:

1. *Blocking.* A connection between a free input/output pair is not always possible because of conflicts with the existing connections. Typically, there is a unique path between every input/output pair, thus minimizing the number of switches and stages. However, it is also possible to provide multiple paths to reduce conflicts and increase fault tolerance. These blocking networks are also known as *multipath.*

2. *Nonblocking.* Any input port can be connected to any free output port without affecting the existing connections. Nonblocking networks have the same functionality as a crossbar. They require multiple paths between every input and output, which in turn leads to extra stages.

3. *Rearrangeable.* Any input port can be connected to any free output port. However, the existing connections may require rearrangement of paths. These networks also require multiple paths between every input and output but the number of paths and the cost is smaller than in the case of nonblocking networks.

Nonblocking networks are expensive. Although they are cheaper than a crossbar of the same size, their cost is prohibitive for large sizes. The best known example of nonblocking multistage network is the Clos network, initially proposed for telephone networks. Rearrangeable networks require less stages or simpler switches than a nonblocking network. The best known example of rearrangeable network is the Beneš network. Figure 1.17 shows an 8×8 Beneš network. For 2^n inputs, this network requires $2n-1$ stages, and provides 2^{n-1} alternative paths. Rearrangeable networks require a central controller to rearrange connections, and were proposed for array processors. However, connections cannot be easily rearranged on multiprocessors because processors access the network asynchronously. So, rearrangeable networks behave like blocking networks when accesses are asynchronous. Thus, this class has not been included in Figure 1.2. We will mainly focus on blocking networks.

Depending on the kind of channels and switches, MINs can be split into two classes [253]:

1. *Unidirectional MINs.* Channels and switches are unidirectional.

2. *Bidirectional MINs.* Channels and switches are bidirectional. This implies that information can be transmitted simultaneously in opposite directions between neighboring switches.

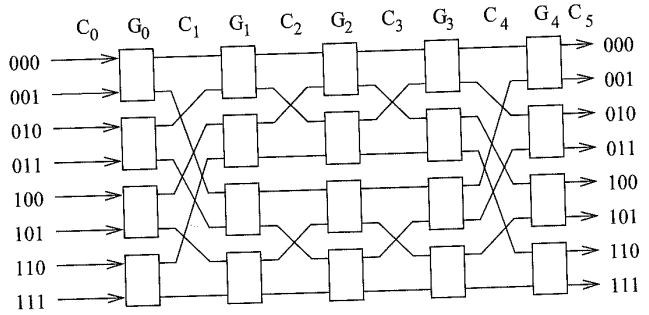
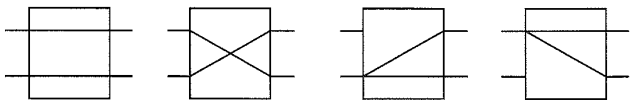


Figure 1.17. An 8×8 Beneš network.



(a) Straight (b) Exchange (c) Lower Broadcast (d) Upper Broadcast

Figure 1.18. Four possible states of a 2×2 switch.

Additionally, each channel may be either multiplexed or replaced by two or more channels. In the latter case, the network is referred to as *dilated MIN*. Obviously, the number of ports of each switch must increase accordingly.

Unidirectional Multistage Interconnection Networks

The basic building blocks of unidirectional MINs are unidirectional switches. An $a \times b$ switch is a crossbar network with a inputs and b outputs. If each input port is allowed to connect to exactly one output port, at most $\min\{a, b\}$ connections can be supported simultaneously. If each input port is allowed to connect to many output ports, a more complicated design is needed to support the so-called *one-to-many* or *multicast* communication. In the *broadcast* mode or *one-to-all* communication, each input port is allowed to connect to all output ports. Figure 1.18 shows four possible states of a 2×2 switch. The last two states are used to support one-to-many and one-to-all communications.

In MINs with $N = M$, it is common to use switches with the same number of input and output ports, i.e., $a = b$. If $N > M$, switches with $a > b$ will be used. Such switches are also called *concentration switches*. In the case of $N < M$, *distribution switches* with $a < b$ will be used.

It can be shown that with N input and output ports, a unidirectional MIN with $k \times k$ switches requires at least $\lceil \log_k N \rceil$ stages to allow a connection path between any input port and any output port. By having additional stages, more connection paths may be used to deliver a message between an input port and an output port at the expense of extra hardware cost. Every path through the MIN crosses all the stages. Therefore, all the paths have the same length.

Four topologically equivalent unidirectional MINs are considered below. These MINs are a class of Delta networks.

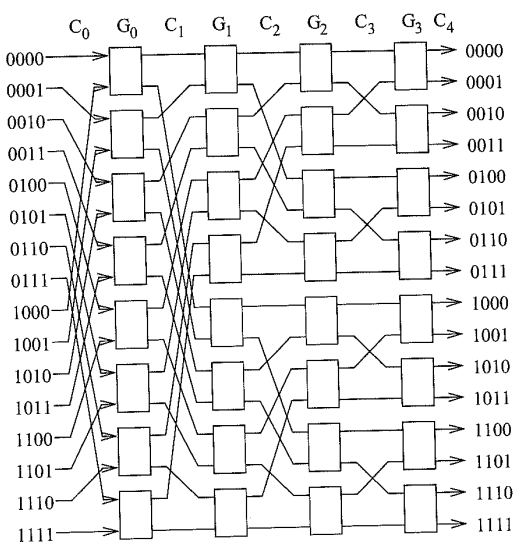
Baseline MINs. In a baseline MIN, connection pattern C_i is described by the $(n-i)$th baseline permutation δ^k_{n-i} for $1 \leq i \leq n$. Connection pattern C_0 is selected to be σ^k.

Butterfly MINs. In a butterfly MIN, connection pattern C_i is described by the ith butterfly permutation β^k_i for $0 \leq i \leq n-1$. Connection pattern C_n is selected to be β^k_0.

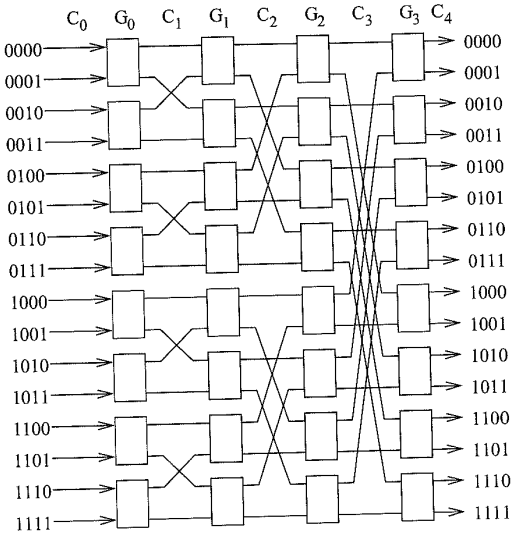
Cube MINs. In a cube MIN (or multistage cube network [317]), connection pattern C_i is described by the $(n-i)$th butterfly permutation β^k_{n-i} for $1 \leq i \leq n$. Connection pattern C_0 is selected to be σ^k.

Omega network. In an Omega network, connection pattern C_i is described by the perfect k-shuffle permutation σ^k for $0 \leq i \leq n-1$. Connection pattern C_n is selected to be β^k_0. Thus, all the connection patterns but the last one are identical. The last connection pattern produces no permutation.

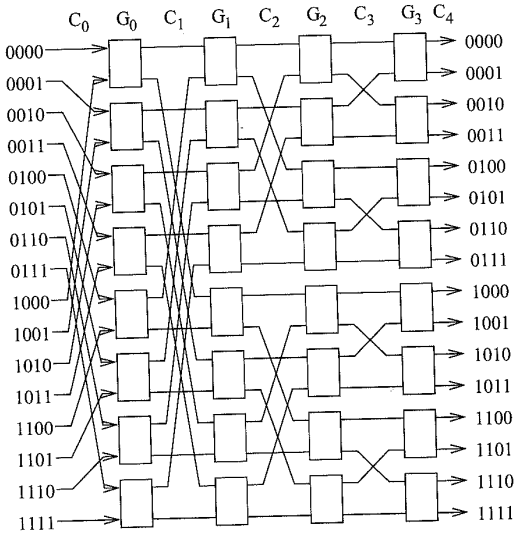
The topological equivalence of these MINs can be viewed as follows: Consider that each input link to the first stage is numbered using a string of n digits $s_{n-1}s_{n-2}\ldots s_1 s_0$, where $0 \leq s_i \leq k-1$, for $0 \leq i \leq n-1$. The least significant digit s_0 gives the address of the input port at the corresponding switch and the address of the switch is given by $s_{n-1}s_{n-2}\ldots s_1$. At each stage, a given switch is able to connect any input port with any output port. This can be viewed as changing the value of



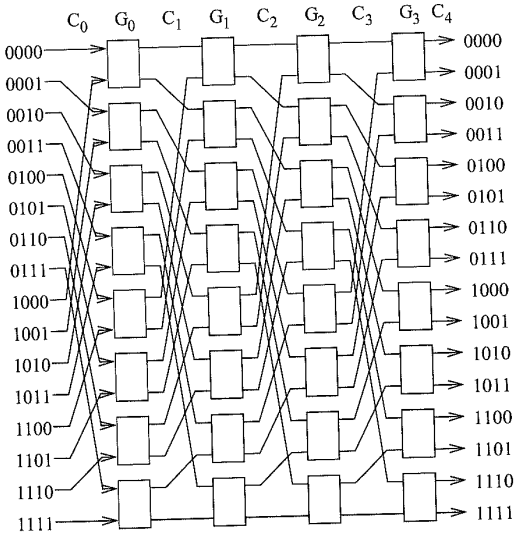
(a) Multistage Baseline Network



(b) Multistage Butterfly Network



(c) Multistage Cube Network



(d) Omega Network

Figure 1.19. Four 16×16 unidirectional multistage interconnection networks.

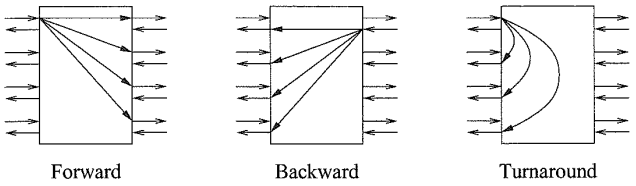


Figure 1.20. Connections in a bidirectional switch.

the least significant digit of the address. In order to be able to connect any input to any output of the network, it should be possible to change the value of all the digits. As each switch is only able to change the value of the least significant digit of the address, connection patterns between stages are defined in such a way that the position of digits is permuted, and after n stages all the digits have occupied the least significant position. Therefore, the above defined MINs differ in the order in which address digits occupy the least significant position.

Figure 1.19 shows the topology of four 16×16 unidirectional multistage interconnection networks: (a) baseline network, (b) butterfly network, (c) cube network, and (d) omega network.

Bidirectional Multistage Interconnection Networks

Figure 1.20 illustrates a bidirectional switch in which each port is associated with a pair of unidirectional channels in opposite directions. This implies that information can be transmitted simultaneously in opposite directions between neighboring switches. For ease of explanation, it is assumed that processor nodes are on the left-hand side of the network, as shown in Figure 1.21. A bidirectional switch supports three types of connections: *forward*, *backward*, and *turnaround* (see Figure 1.20). As turnaround connections between ports at the same side of a switch are possible, paths have different lengths. An eight-node butterfly bidirectional MIN (BMIN) is illustrated in Figure 1.21.

Paths are established in BMINs by crossing stages in forward direction, then establishing a turnaround connection, and finally crossing stages in backward direction. This is usually referred to as *turnaround routing*. Figure 1.22 shows two alternative paths from node S to node D in an eight-node butterfly BMIN. When crossing stages in forward direction, several paths are possible. Each switch can select any of its output ports. However, once the turnaround connection is crossed,

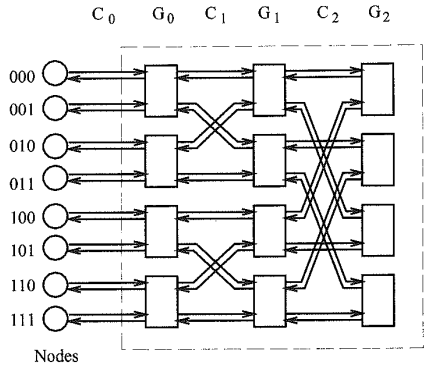


Figure 1.21. An eight-node butterfly bidirectional MIN.

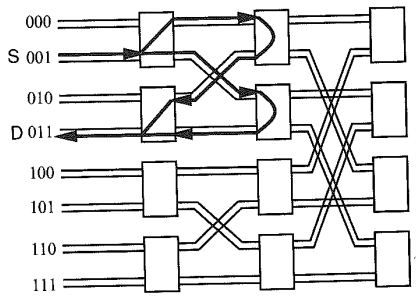


Figure 1.22. Alternative paths in an eight-node butterfly bidirectional MIN.

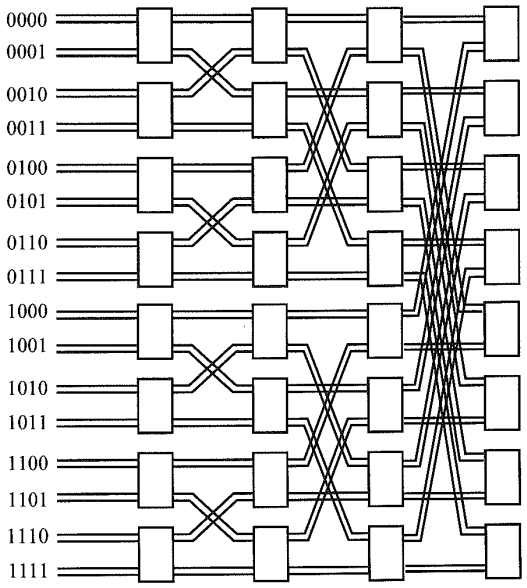
a single path is available up to the destination node. In the worst case, establishing a path in an n-stage BMIN requires crossing $2n - 1$ stages. This behavior closely resembles that of the Beneš network. Indeed, the baseline BMIN can be considered as a folded Beneš network.

As shown in Figure 1.23, a butterfly BMIN with turnaround routing can be viewed as a *fat tree* [201]. In a fat tree, processors are located at leaves, and internal vertices are switches. Transmission bandwidth between switches is increased by adding more links in parallel as switches become closer to the root switch. When a message is routed from one processor to another, it is sent up (in forward direction) the tree to the least common ancestor of the two processors, and then sent down (in backward direction) to the destination. Such a tree routing well explains the turnaround routing mentioned above.

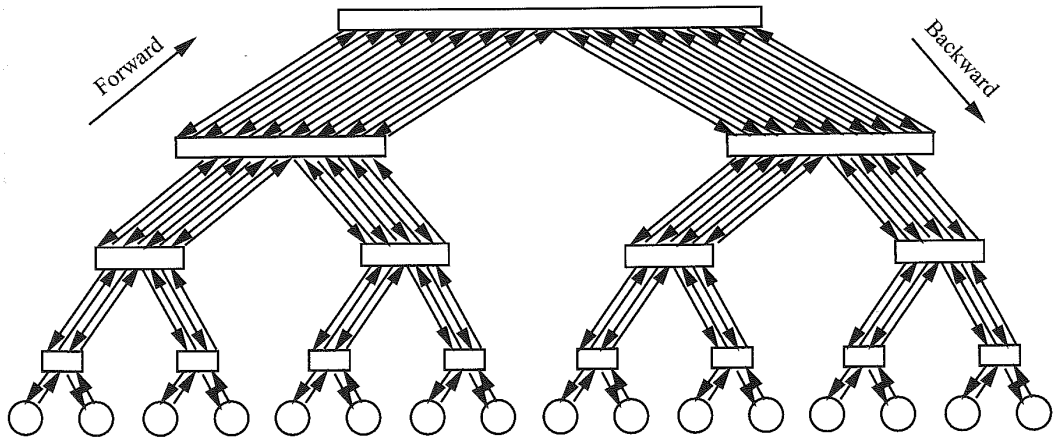
1.7.6 Examples

Several examples of parallel computers with indirect networks and commercial switches to build indirect networks are listed below.

- Myricom Myrinet: Supports regular and irregular topologies, 8×8 crossbar switch, 9-bit channels, full-duplex, 640 Mbits/s per link.
- Thinking Machines CM-5: Fat tree topology, 4-bit bidirectional channels at 40 MHz, aggregate bandwidth in each direction of 20 Mbytes/s.
- Inmos C104: Supports regular and irregular topologies, 32×32 crossbar switch, serial links, 100 Mbits/s per link.
- IBM SP2: Crossbar switches supporting Omega network topologies with bidirectional, 16-bit channels at 150MHz, 300 Mbytes/s in each direction.
- SGI SPIDER: Router with 20-bit bidirectional channels operating on both edges, 200 MHz clock, aggregate raw data rate of 1 Gbyte/s. Offers support for configurations as nonblocking multistage network topologies as well as irregular topologies.
- Tandem ServerNet: Irregular topologies, 8-bit bidirectional channels at 50 MHz, 50 Mbytes/s per link.



(a) A 16–Node Butterfly BMIN Built with 2 x 2 Switches



(b) A 16–Node Fat Tree

Figure 1.23. Fat tree and butterfly BMIN.

1.8 Hybrid Networks

In this section, we briefly describe some network topologies that do not fit into the classes described above. In general, hybrid networks combine mechanisms from shared-medium networks and direct or indirect networks. Therefore, they increase bandwidth with respect to shared-medium networks, and reduce the distance between nodes with respect to direct and indirect networks. There exist well-established applications of hybrid networks. This is the case for bridged LANs. However, for systems requiring very high performance, direct and indirect networks achieve better scalability than hybrid networks because point-to-point links are simpler and faster than shared-medium buses. Most high-performance parallel computers use direct or indirect networks. Recently hybrid networks have been gaining acceptance again. The use of optical technology enables the implementation of high-performance buses. Currently, some prototype machines are being implemented based on electrical as well as optical interconnects.

Many hybrid networks have been proposed for different purposes. The classification proposed for these networks is mainly dictated by the application fields. In general, hybrid networks can be modeled by a hypergraph [23], where the vertices of the hypergraph represent the set of processing nodes, and the edges represent the set of communication channels and/or buses. Note that an edge in a hypergraph can interconnect an arbitrary number of nodes. When an edge connects exactly two nodes then it represents a point-to-point channel. Otherwise it represents a bus. In some network designs, each bus has a single driving node. No other device is allowed to drive that bus. In this case, there is no need for arbitration. However, it is still possible to have several receivers at a given time, thus retaining the broadcast capability of buses. Obviously, every node in the network must drive at least one bus, therefore requiring a number of buses not lower than the number of nodes. In this case, the network topology can be modeled by a directed hypergraph.

1.8.1 Multiple Backplane Buses

Due to limited shared-medium network bandwidth, a shared-medium network can only support a small number of devices, has limited distance, and is not scalable to support a large system. Some approaches have been used or studied to remove such bottlenecks. One approach to increase network bandwidth is to have multiple buses as shown in Figure 1.24. However, concern about wiring and interface costs is a major reason why multiple buses have seen little use so far in multiprocessor design [246]. Due to the limitations of electrical packaging technology, it is unlikely to have a multiple-bus network with more than four buses. However, many more buses are likely feasible with other packaging technologies such as wavelength division multiplexing on fiber optics [279].

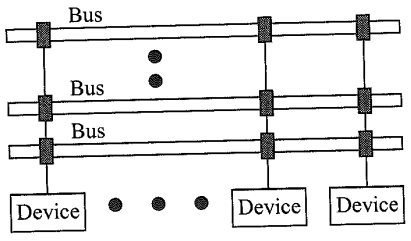


Figure 1.24. A multiple-bus network.

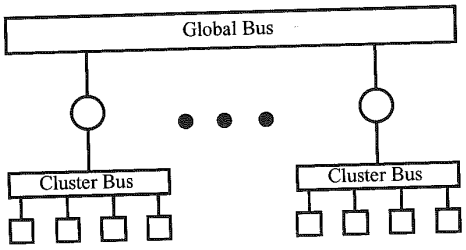


Figure 1.25. Two-level hierarchical buses.

1.8.2 Hierarchical Networks

Another approach to increase the network bandwidth is to have a hierarchical structure as shown in Figure 1.25. Different buses are interconnected by routers or bridges to transfer information from one side of the network to the other side of the network. These routers or bridges can filter the network traffic by examining the destination address of each passing message. The hierarchical network can expand the network area and handle more devices, but it is no longer a simple shared-medium network. This approach is used in bridged LANs. Usually, a higher bandwidth is available at the global bus. Otherwise, it may become a bottleneck. This can be achieved by using a faster technology. This is the case for the backbone LAN in some bridged LANs. Hierarchical networks have also been proposed as the interconnection scheme for shared-memory multiprocessors. Again, the global bus may become a bottleneck. For example, the Encore Gigamax addressed this problem by having an optical fiber bus to increase the bandwidth of the global bus.

1.8.3 Cluster-Based Networks

Cluster-based networks also have a hierarchical structure. Indeed, they can be considered as a subclass of hierarchical networks. Cluster-based networks combine the advantages of two or more kinds of networks at different levels in the hierarchy. For example, it is possible to combine the advantages of buses and point-to-point links by using buses at the lower level in the hierarchy to form clusters, and a direct network topology connecting clusters at the higher level. This is the case for the Stanford Directory Architecture for Shared-Memory (DASH) [203]. Figure 1.26 shows the basic architecture of this parallel computer. At the lower level, each cluster consists of four processors connected by a bus. At the higher level, a 2-D mesh connects the clusters. The broadcast capability of the bus is used at the cluster level to implement a snoopy protocol for cache coherence. The direct network at the higher level overcomes the bandwidth constraints of the bus, considerably increasing the scalability of the machine.

Other combinations are possible. Instead of combining buses and direct networks, the HP/Convex

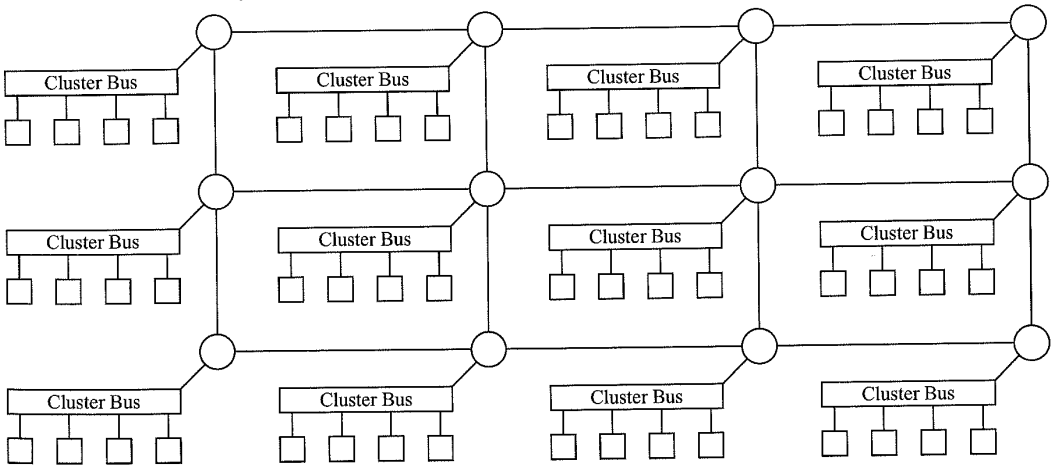


Figure 1.26. Cluster-based 2-D mesh.

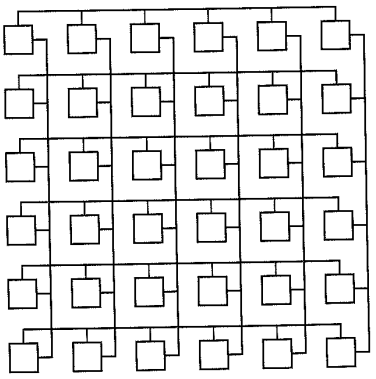


Figure 1.27. A two-dimensional hypermesh.

Exemplar multiprocessor combines indirect and direct networks. This multiprocessor has 5×5 nonblocking crossbars at the lower level in the hierarchy, connecting four functional blocks and one I/O interface to form clusters or *hypernodes*. Each functional block consists of two processors, two memory banks and interfaces. These hypernodes are connected by a second level *coherent toroidal interconnect* made out of multiple rings using Scalable Coherent Interface (SCI). Each ring connects one functional block from all the hypernodes. At the lower level of the hierarchy, the crossbars allow all the processors within a hypernode to access the interleaved memory modules in that hypernode. At the higher level, the rings implement a cache coherence protocol.

1.8.4 Other Hypergraph Topologies

Many other hybrid topologies have been proposed [25, 336, 348]. Among them, a particularly interesting class is the *hypermesh* [335]. A hypermesh is a regular topology consisting of a set of nodes arranged into several dimensions. Instead of having direct connections to the neighbors in each dimension, each node is connected to all the nodes in each dimension through a bus. There are several ways to implement a hypermesh. The most straightforward way consists of connecting all the nodes in each dimension through a shared bus. Figure 1.27 shows a 2-D hypermesh. In this network, multiple buses are arranged in two dimensions. Each node is connected to one bus in each dimension. This topology was proposed by Wittie [348], and was referred to as *spanning-bus hypercube*. This topology has a very low diameter, and average distance between nodes scales very well with network size. However, the overall network bandwidth does not scale well. Additionally, the frequent changes of bus mastership incur significant overheads.

An alternative implementation that removes the above mentioned constraints consists of replacing the single shared bus connecting the nodes along a given dimension by a set of as many buses as nodes in that dimension. This is the approach proposed in the Distributed Crossbar Switch Hypermesh (DCSH) [222, 223]. Figure 1.28 shows one dimension of the network. Each bus is driven by a single node. Therefore, there are no changes in mastership. Also, bandwidth scales with the number of nodes. Two major concerns, however, are the high number of buses required in the system and the very high number of input ports required at each node. Although the authors propose an electrical implementation, this topology is also suitable for optical interconnects.

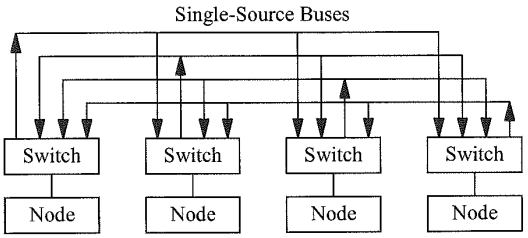


Figure 1.28. A one-dimensional Distributed Crossbar Switch Hypermesh.

1.9 A Unified View of Direct and Indirect Networks

Up to this point, most researchers and manufacturers considered direct and indirect networks as two completely different approaches for the interconnection of a set of nodes. However, the last few years have seen developments in router architecture, algorithms, and switch design evolve to a point that is blurring the distinction between these two classes of networks. For performance driven environments, as networks converge on the use of pipelined message transmission and source based routing algorithms, the major differences between the switch architectures for direct and indirect networks is becoming more a question of topology. Direct networks have typically used routers with routing algorithms implemented in the network. Switch-based designs have had routing performed at the source. Modern routers are being designed to support simple, fast, reprogrammable routing decisions within the network with substantial control of routing at the message source. This expands the base of applications for the (sizeable) investment in router/switch designs. This flexibility is also evidenced by the fact that the traditional high-performance multiprocessor router/switch designs are moving up to implementations involving workstation and personal computer (PC) clusters.

In particular, during the last few years several manufacturers have introduced switches to interconnect a set of processors. These switch-based networks allow the user to define the topology. Let us consider the particular case in which each switch has a single processor directly attached to it, and the remaining ports are used to connect with other switches. This network does not differ substantially from a direct network because each switch can be considered as a router associated with the corresponding node. Indeed, the functionality of current routers and switches is practically identical. Moreover, some manufacturers proposed the use of the same switch to implement either direct or indirect networks. This is the case of the Inmos C104 switch [228] and the SGI SPIDER [118]. For interconnection networks using the former switch, Inmos proposed the use of bidirectional multistage interconnection networks as well as direct topologies like meshes. In the latter case, each switch is connected to a single node through a set of independent ports. Also, two adjacent switches may use several ports in parallel to communicate between them. The SGI SPIDER router implementation was designed to support configurations as nonblocking multistage networks, as irregular topologies, and as routers for conventional multiprocessor topologies. Such flexibility is typically achieved using topology-independent routing algorithms and flexible switching techniques.

Therefore, we can view networks using point-to-point links as a set of interconnected switches, each one being connected to zero, one, or more nodes. Direct networks correspond to the case where every switch is connected to a single node. Crossbar networks correspond to the case where there is

a single switch connected to all the nodes. Multistage interconnection networks correspond to the case where switches are arranged into several stages and the switches in intermediate stages are not connected to any processor. However, other choices are possible under this unified view of direct and indirect interconnection networks. Effectively, nothing prevents the designer from interconnecting switches using a typical direct network topology, and connecting several processors to each switch. This is the case for the Cray T3D, which implements a 3-D torus topology, connecting two processors to each router.

The unified view allows the development of generic routing algorithms. In these algorithms, the destination address specifies the destination switch as well as the port number in that switch that is connected to the destination processor. Note that routing algorithms for direct networks are a particular case of these generic algorithms. In this case, there is a single processor connected to each switch. So, the output port in the last switch is not required, assuming that processors are connected to the same port in every switch. This unified view also allows the application of many results developed for direct networks to indirect networks, and vice versa. For example, the theory of deadlock avoidance presented in Chapter 3 was initially developed for direct networks but can also be applied to indirect networks with minor changes, as will be shown in that chapter. Obviously, not all the results can be directly applied to both network classes. Additionally, applying some results to other network classes may require a considerable effort. In the remaining chapters of the book, most techniques will be described for the class of networks for which they were initially proposed. However, it is important to keep in mind that most of those techniques can also be applied to other network topologies by taking into account the unified view proposed in this section.

Chapter 2

Message Switching Layer

Interprocessor communication can be viewed as a hierarchy of services starting from the physical layer that synchronizes the transfer of bit streams to higher-level protocols layers that perform functions such as packetization, data encryption, data compression, etc. Such a layering of communication services is common in the local and wide area network communities. While there currently may not be a consensus on a standard set of layers for multiprocessor systems, we find it useful to distinguish between three layers in the operation of the interconnection network: the *routing layer*, the *switching layer*, and the *physical layer*. The physical layer refers to link-level protocols for transferring messages and otherwise managing the physical channels between adjacent routers. The switching layer utilizes these physical layer protocols to implement mechanisms for forwarding messages through the network. Finally, the routing layer makes routing decisions to determine candidate output channels at intermediate router nodes and thereby establish the path through the network. The design of routing protocols and their properties, e.g., deadlock and livelock freedom, are largely determined by the services provided by the switching layer.

This chapter focuses on the techniques that are implemented within the network routers to realize the switching layer. These techniques differ in several respects. The *switching techniques* determine when and how internal switches are set to connect router inputs to outputs and the time at which message components may be transferred along these paths. These techniques are coupled with *flow control* mechanisms for the synchronized transfer of units of information between routers and through routers in forwarding messages through the network. Flow control is tightly coupled with *buffer management* algorithms that determine how message buffers are requested and released, and as a result determine how messages are handled when blocked in the network. Implementations of the switching layer differ in decisions made in each of these areas, and in their relative timing, i.e., when one operation can be initiated relative to the occurrence of the other. The specific choices interact with the architecture of the routers and traffic patterns imposed by parallel programs in determining the latency and throughput characteristics of the interconnection network.

As we might expect, the switching techniques employed in multiprocessor networks initially followed those techniques employed in local and wide area communication networks, e.g., circuit switching and packet switching. However, as the application of multiprocessor systems spread into increasingly compute-intensive domains, the traditional layered communication designs borrowed from LANs became a limiting performance bottleneck. New switching techniques and implementations evolved that were better suited to the low latency demands of parallel programs. This chapter reviews these switching techniques and their accompanying flow control and buffer management algorithms.

2.1 Network and Router Model

In comparing and contrasting alternative implementations of the switching layer we are interested in evaluating their impact on the router implementations. The implementations in turn determine the cycle time of router operation and therefore the resulting message latency and network bandwidth. The architecture of a generic router is shown in Figure 2.1 and is comprised of the following major components.

- *Buffers.* These are first-in first-out (FIFO) buffers for storing messages in transit. In the above model, a buffer is associated with each input physical channel and each output physical channel. In alternative designs, buffers may be associated only with inputs (input buffering) or outputs (output buffering). The buffer size is an integral number of flow control units.

- *Switch.* This component is responsible for connecting router input buffers to router output buffers. High-speed routers will utilize crossbar networks with full connectivity, while lower-speed implementations may utilize networks that do not provide full connectivity between input buffers and output buffers.

- *Routing and arbitration unit.* This component implements the routing algorithms, selects the output link for an incoming message, and accordingly sets the switch. If multiple messages simultaneously request the same output link this component must provide for arbitration between them. If the requested link is busy, the incoming message remains in the input buffer. It will be routed again after the link is freed and if it successfully arbitrates for the link.

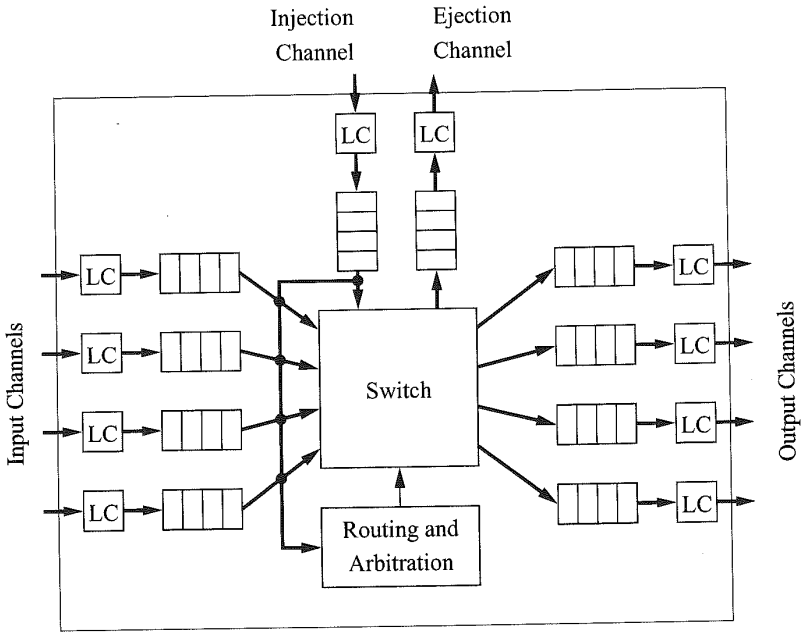


Figure 2.1. Generic router model. (LC = Link controller.)

- *Link controllers (LC).* The flow of messages across the physical channel between adjacent routers is implemented by the link controller. The link controllers on either side of a channel coordinate to transfer units of flow control.
- *Processor interface.* This component simply implements a physical channel interface to the processor rather than to an adjacent router. It consists of one or more injection channels from the processor and one or more ejection channels to the processor. Ejection channels are also referred to as delivery channels or consumption channels.

From the point of view of router performance we are interested in two parameters [57]. When a message first arrives at a router, it must be examined to determine the output channel over which the message is to be forwarded. This is referred to as the *routing delay*, and typically includes the time to set the switch. Once a path has been established through a router by the switch, we are interested in the rate at which messages can be forwarded through the switch. This rate is determined by the propagation delay through the switch (intrarouter delay), and the signaling rate for synchronizing the transfer of data between the input and output buffers. This delay has been characterized to as the the *internal flow control* latency [57]. Similarly, the delay across the physical links (interrouter delay) is referred to as the *external flow control* latency. The routing delay and flow control delays collectively determine achievable message latency through the switch, and along with contention by messages for links, determines the network throughput.

The following section addresses some basic concepts in the implementation of the switching layer, assuming the generic router model shown in Figure 2.1. The remainder of the chapter focuses on alternative implementations of the switching layer.

2.2 Basic Concepts

Switching layers can be distinguished by the implementation and relative timing of flow control operations and switching techniques. In addition, these operations may be overlapped with the time to make routing decisions.

Flow control is a synchronization protocol for transmitting and receiving a unit of information. The *unit of flow control* refers to that portion of the message whose transfer must be synchronized. This unit is defined as the smallest unit of information whose transfer is requested by the sender and acknowledged by the receiver. The request/acknowledgment signaling is used to ensure successful transfer and the availability of buffer space at the receiver. Note that there is no restriction on when requests or acknowledgments are actually sent or received. Implementation efficiency governs the actual exchange of these control signals, e.g., the use of block acknowledgments. For example, it is easy to think of messages in terms of fixed-length packets. A packet is forwarded across a physical channel or from the input buffers of a router to the output buffers. Note that these transfers are atomic in the sense that sufficient buffering must be provided so that a packet is either transferred in its entirety, or transmission is delayed until sufficient buffer space becomes available. In this example, the *flow* of information is managed and *controlled* at the level of an entire packet.

Flow control occurs at two levels. In the preceding example, *message flow control* occurs at the level of a packet. However, the transfer of a packet across a physical channel between two routers make take several steps or cycles, e.g., the transfer of a 128-byte packet across a 16-bit data channel. The resulting multicycle transfers use *physical channel flow control* to forward a message flow control unit across the physical link connecting routers.

Switching techniques differ in the relationship between the sizes of the physical and message flow control units. In general, each message may be partitioned into fixed-length *packets*. Packets in

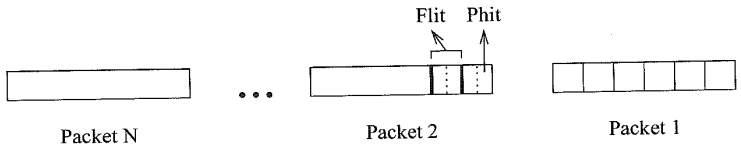


Figure 2.2. Alternative flow control units in a message.

turn may be broken into message flow control units or *flits* [78]. Due to channel width constraints, multiple physical channel cycles may be used to transfer a single flit. A *phit* is the unit of information that can be transferred across a physical channel in a single step or cycle. Flits represent logical units of information as opposed to phits which correspond to physical quantities, i.e., the number of bits that can be transferred in parallel in a single cycle. An example of a message comprised of N packets, 6 flits/packet and 2 phits/flit is shown in Figure 2.2.

The relationships between the sizes of phits, flits, and packets differs across machines. Many machines have the phit size equivalent to the flit size. In the IBM SP2 switch [327], a flit is 1 byte and is equivalent to a phit. Alternatively, the Cray T3D [311] utilizes flit-level message flow control where each flit is comprised of eight 16-bit phits. The specific choices reflect trade-offs in performance, reliability, and implementation complexity.

Example 2.1

There are many candidate synchronization protocols for coordinating phit transfers across a channel, and Figure 2.3 illustrates an example of a simple four-phase asynchronous handshaking protocol. Only one direction of transfer is shown. Router $R1$ asserts the RQ signal when information is to be transferred. Router $R2$ responds by reading the data and asserting the ACK signal. This leads to deasserting RQ by $R1$ which in turn causes $R2$ to deassert ACK. This represents one cycle of operation wherein 1 phit is transferred across the channel.

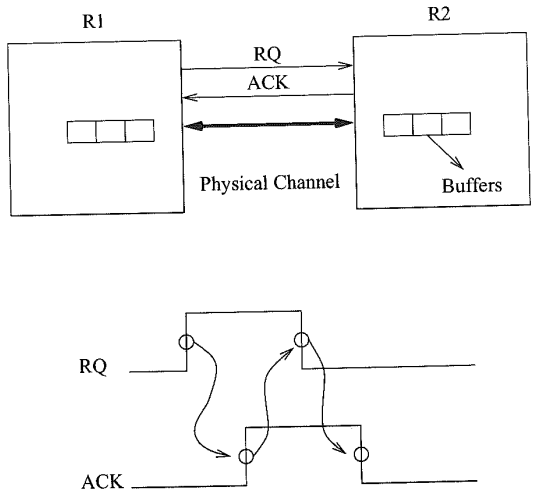


Figure 2.3. An example of asynchronous physical channel flow control.

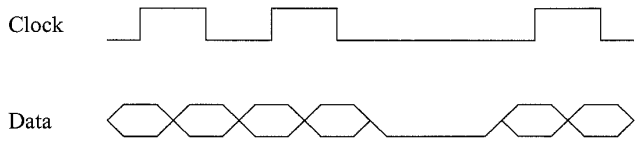


Figure 2.4. An example of synchronous physical channel flow control.

Another transfer may now be initiated. The ACK signal can serve to both acknowledge reception (rising edge) as well as the availability of buffer space (falling edge) for the transmission of the next unit of information. Thus, the flow of phits across the channel is synchronized to prevent buffer overflow in the receiving end of the channel. Note that higher-level message flow control mechanisms can ensure the availability of sufficient buffer space for each flit.

Example 2.2

Physical channel flow control can also be synchronous as shown in Figure 2.4. The clock signal is transmitted across the channel and both the rising and falling edges of the clock line validate the data on the data lines for the receiver. Such physical channel flow control mechanism can be found within the routers of the Intel iPSC/2 and iPSC 860 machines [257]. Figure 2.4 does not show the acknowledgment signals to indicate the availability of buffer space on the receiving node. Acknowledgment signals may be provided for the transfer of each data item, or the channel may utilize block acknowledgments, i.e., each acknowledgment signal indicates the availability of buffer space for some fixed number of data items. Such an approach both reduces the acknowledgment traffic as well as the signaling rate of acknowledgments. It also enables other optimizations for high-speed channel operation that are discussed in Chapter 7.

While interrouter transfers are necessarily constructed in terms of phits, the switching technique deals with flits (which could be defined to be the complete message packet!). The switching techniques set the internal switch to connect input buffers to output buffers, and forward flits along this path. These techniques are distinguished by the time at which they occur relative to the message flow control operation and the routing operation. For example, switching may take place after a flit has been received in its entirety. Alternatively, the transfer of a flit through the switch may begin as soon as the routing operation has been completed, but before the remainder of the flit has been received from the preceding router. In this case switching is overlapped with message-level flow control. In at least one proposed switching technique, switching begins after the first phit is received and even before the routing operation is complete! In general, high-performance switching techniques seek to overlap switching and message flow control as far as possible. While such an approach provides low-latency communication, it does complicate link-level diagnosis and error recovery.

This chapter describes the prevalent switching techniques that have been developed to date for use in current-generation multiprocessors. Switching layers can share the same physical channel flow control mechanism, but differ in the choice of message flow control. Unless otherwise stated, flow control will refer to message flow control.

2.3 Basic Switching Techniques

For the purposes of comparison, for each switching technique we will consider the computation of the base latency of an L-bit message in the absence of any traffic. The phit size and flit size are assumed to be equivalent and equal to the physical data channel width of W bits. The routing header is assumed to be 1 flit, thus the message size is $L + W$ bits. A router can make a routing decision in t_r seconds. The physical channel between two routers operates at B Hz, i.e., the physical channel bandwidth is BW bits per second. The propagation delay across this channel is denoted by $t_w = \frac{1}{B}$. Once a path has been set up through the router, the intrarouter delay or switching delay is denoted by t_s. The router internal data paths are assumed to be matched to the channel width of W bits. Thus, in t_s seconds a W-bit flit can be transferred from the input of the router to the output. The source and destination processors are assumed to be D links apart. The relationship between these components as they are used to compute the no-load message latency is shown in Figure 2.5.

2.3.1 Circuit Switching

In circuit switching, a physical path from the source to the destination is reserved prior to the transmission of the data. This is realized by injecting the routing header flit into the network. This *routing probe* contains the destination address and some additional control information. This routing probe progresses towards the destination reserving physical links as it is transmitted through intermediate routers. When the probe reaches the destination, a complete path has been set up and an acknowledgment is transmitted back to the source. The message contents may now be transmitted at the full bandwidth of the hardware path. The circuit may be released by the destination or by the last few bits of the message. In the Intel iPSC/2 routers [257], the acknowledgments are multiplexed in the reverse direction on the same physical line as the message. Alternatively, implementations may provide separate signal lines to transmit acknowledgment signals. A time-space diagram of the transmission of a message over three links is shown in Figure 2.6. The header probe is forwarded across three links followed by the return of the acknowledgment. The shaded boxes represent the times during which a link is busy. The space between these boxes represents the time to process the routing header, and the intrarouter propagation delays. The clear box represents the duration the links are busy transmitting data through the circuit. Note that the routing and intrarouter delays at the source router are not included and would precede the box corresponding to the first busy link.

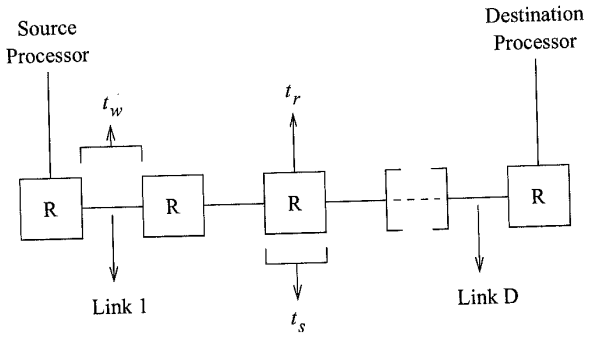


Figure 2.5. View of the network path for computing the no-load latency. (R = Router.)

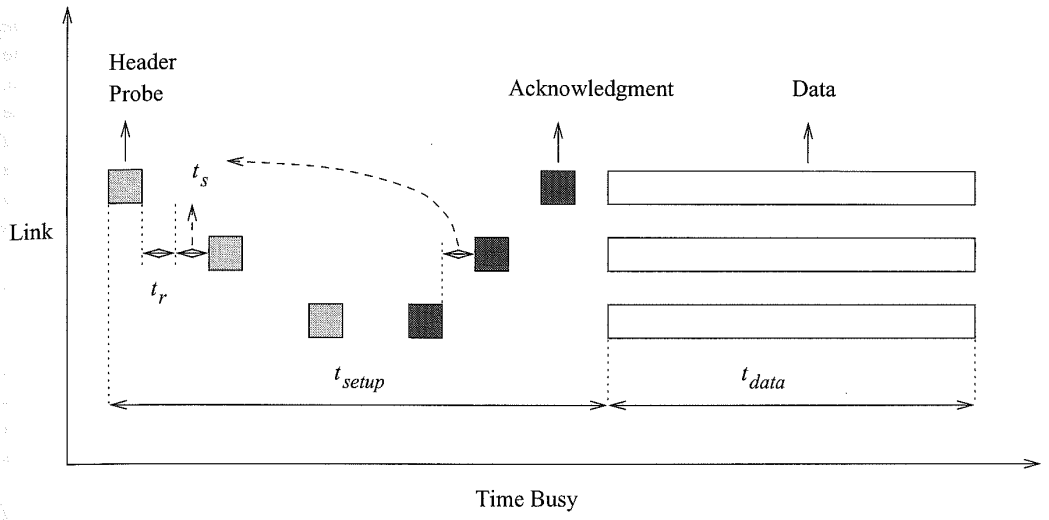


Figure 2.6. Time-space diagram of a circuit-switched message.

An example of a routing probe used in the JPL Mark III binary hypercube is shown in Figure 2.7. The network of the Mark III was quite flexible, supporting several distinct switching mechanisms in configurations up to 2,048 nodes. Bits 0 and 16 of the header define the switching technique being employed. The values shown in Figure 2.7 are for circuit switching. Bits 17–19 are unused while the destination address is provided in bits 1–11. The remaining 4-bit fields are used to address 1 of 11 output links at each individual router. There are 11 such fields supporting a 11-dimensional hypercube and requiring a two-word, 64-bit header. The path is computed at the source node. An alternative could have been to compute the value of the output port at each node rather than storing the addresses of all intermediate ports in the header. This would significantly reduce the size of the routing header probe. However, this scheme would require routing time and buffering logic within the router. In contrast, the format shown in Figure 2.7 enables a fast lookup using the header and simple processing within the router.

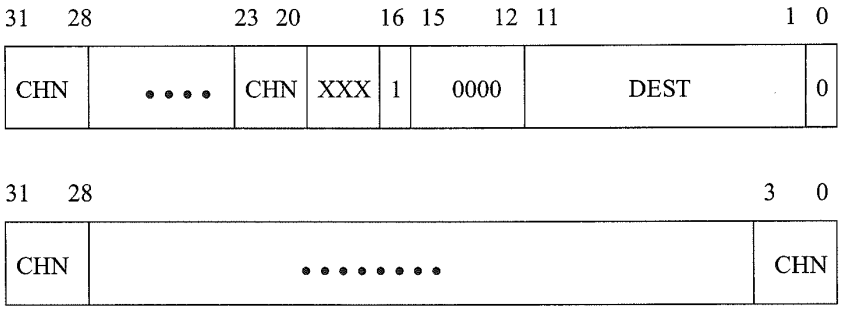


Figure 2.7. An example of the format of a circuit probe. (CHN = Channel number; DEST = Destination address; XXX = Not defined.)

Circuit switching is generally advantageous when messages are infrequent and long, i.e., the message transmission time is long compared to the path setup time. The disadvantage is that the physical path is reserved for the duration of the message and may block other messages. For example, consider the case where the probe is blocked waiting for a physical link to become free. All of the links reserved by the probe up to that point remain reserved, cannot be used by other circuits, and may be blocking other circuits preventing them from being set up. Thus, if the size of the message is not that much greater than the size of the probe, it would be advantageous to transmit the message along with the header and buffer the message within the routers while waiting for a free link. This alternative technique is referred to as *packet switching*, and will be studied in Section 2.3.2.

The base latency of a circuit-switched message is determined by the time to set up a path, and the subsequent time the path is busy transmitting data. The router operation differs a bit from that shown in Figure 2.1. While the routing probe is buffered at each router, data bits are not. There are no intervening data buffers in the circuit which operates effectively as a single wire from source to destination. This physical circuit may use asynchronous or synchronous flow control as shown in Figures 2.3 or 2.4. In this case the time for the transfer of each flit from source to destination is determined by the clock speed of the synchronous circuit or signaling speed of the asynchronous handshake lines. The signaling period or clock period must be greater than the propagation delay through this circuit. This places a practical limit on the speed of circuit switching as a function of system size. More recent techniques have begun to investigate the use of this delay as a form of storage. At very high signal speeds, multiple bits may be present on a wire concurrently, proceeding as *waves* of data. Such techniques have been referred to as *wave pipelining* [112]. Using such techniques the technological limits of router and network designs have been reexamined [102, 310] and it has been found that substantial improvements in wire bandwidth is possible. The challenges to widespread use remain the design of circuits that can employ wave pipelining with stable and predictable delays, while in large designs the signal skew remains particularly challenging.

Without wave pipelining, from Figure 2.6 we can write an expression for the base latency of a message as follows:

$$\begin{aligned} t_{circuit} &= t_{setup} + t_{data} \\ t_{setup} &= D[t_r + 2(t_s + t_w)] \\ t_{data} &= \frac{1}{B}\left\lceil \frac{L}{W} \right\rceil \end{aligned} \tag{2.1}$$

Actual latencies clearly depend on a myriad of implementation details. Figure 2.6 represents some simplifying assumptions about the time necessary for various events such as processing an acknowledgment, or initiating the transmission of the first data flit. The factor of 2 in the setup cost represents the time for the forward progress of the header and the return of the acknowledgment. The use of B Hz as the channel speed represents the transmission across hardwired path from source to destination.

2.3.2 Packet Switching

In circuit switching, the complete message is transmitted after the circuit has been set up. Alternatively, the message can be partitioned and transmitted as fixed-length packets, e.g., 128 bytes. The first few bytes of a packet contain routing and control information and are referred to as the *packet header*. Each packet is individually routed from source to destination. A packet is completely buffered at each intermediate node before it is forwarded to the next node. This is the reason why this switching technique is also referred to as *store-and-forward* (SAF) switching. The header information is extracted by the intermediate router and used to determine the output link over which

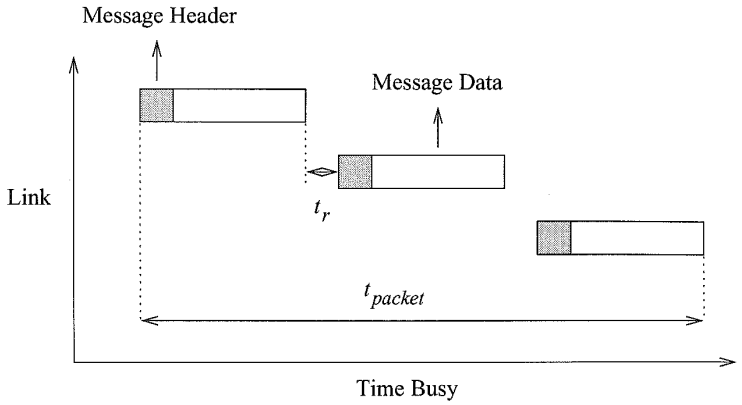


Figure 2.8. Time-space diagram of a packet-switched message.

the header is to be forwarded. A time-space diagram of the progress of a packet across three links is shown in Figure 2.8. From the figure we can see that the latency experienced by a packet is proportional to the distance between the source and destination nodes. Note that the figure has omitted the packet latency through the router.

Packet switching is advantageous when messages are short and frequent. Unlike circuit switching, where a segment of a reserved path may be idle for a significant period of time, a communication link is fully utilized when there are data to be transmitted. Many packets belonging to a message can be in the network simultaneously even if the first packet has not yet arrived at the destination. However, splitting a message into packets produces some overhead. In addition to the time required at source and destination nodes, every packet must be routed at each intermediate node. An example of the format of a data packet header is shown in Figure 2.9. This is the header format used in the JPL Hyperswitch. Since the hyperswitch can operate in one of many modes, bit field 12–16 and bit 0 collectively identify the switching technique being used: in this case it is packet switching using a fixed-path routing algorithm. Bits 1–11 identify the destination address, limiting the format to systems of 2,048 processors or less. The LEN field identifies the packet size in units of 192 bytes. For the current implementation packet size is limited to 384 bytes. If packets are routed adaptively through the network, packets from the same message may arrive at the destination out of order. In this case the packet headers must also contain sequencing information so that the messages can be reconstructed at the destination.

In multidimensional, point-to-point networks it is evident that the storage requirements at the individual router nodes can become extensive if packets can become large and multiple packets must be buffered at a node. In the JPL implementation, packets are not stored in the router,

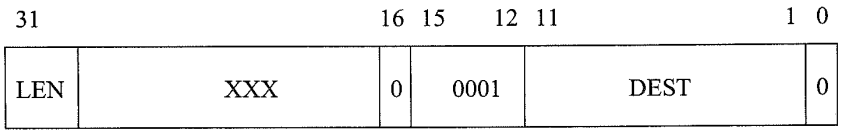


Figure 2.9. An example packet header format. (DEST = Destination address; LEN = Packet length in units of 192 bytes; XXX = Not defined.)

but are rather stored in the memory of the local node and a special-purpose message coprocessor is used to process the message, i.e., compute the address of an output channel and forward the message. Other multicomputers using packet switching also buffer packets in the memory of the local node (Cosmic Cube [313], Intel iPSC/1 [164]). This implementation is no doubt a carryover from implementations in local and wide area networks where packets are buffered in memory and special-purpose coprocessors and network interfaces have been dedicated to processing messages. In modern multiprocessors, the overhead and impact on message latency render such message processing impractical. To be viable, messages must be buffered and processed within the routers. Storage requirements can be reduced by using central queues in the router that are shared by all input channels rather than providing buffering at each input channel, output channel, or both. In this case, internal and external flow control delays will typically take many cycles.

The base latency of a packet-switched message can be computed as follows:

$$t_{packet} = D\left\{t_r + (t_s + t_w)\left\lceil\frac{L+W}{W}\right\rceil\right\} \tag{2.2}$$

This expression follows the router model in Figure 2.1, and as a result includes factors to represent the time for the transfer of packet of length $L + W$ bits across the channel (t_w) as well as from the input buffer of the router to the output buffer (t_s). However, in practice, the router could be only input-buffered, output-buffered, or use central queues. The above expression would be modified accordingly. The important point to note is that the latency is directly proportional to the distance between the source and destination nodes.

2.3.3 Virtual Cut-Through (VCT) Switching

Packet switching is based on the assumption that a packet must be received in its entirety before any routing decision can be made and the packet forwarded to the destination. This is not generally true. Consider a 128-byte packet and the router model shown in Figure 2.1. In the absence of 128-byte-wide physical channels, the transfer of the packet across the physical channel will take multiple cycles. However, the first few bytes will contain routing information that is typically available after the first few cycles. Rather than waiting for the entire packet to be received, the packet header can be examined as soon as it is received. The router can start forwarding the header and following data bytes as soon as routing decisions have been made and the output buffer is free. In fact, the message does not even have to be buffered at the output and can *cut through* to the input of the next router before the complete packet has been received at the current router. This switching technique is referred to as *virtual cut-through* (VCT) switching. In the absence of blocking, the latency experienced by the header at each node is the routing latency and propagation delay through the router and along the physical channels. The message is effectively pipelined through successive switches. If the header is blocked on a busy output channel, the complete message is buffered at the node. Thus, at high network loads, VCT switching behaves like packet switching.

Figure 2.10 illustrates a time-space diagram of a message transferred using VCT switching where the message is blocked after the first link waiting for an output channel to become free. In this case we see that the complete packet has to be transferred to the first router where it remains blocked waiting for a free output port. However, from the figure we can see that the message is successful in cutting through the second router and across the third link.

The base latency of a message that successfully cuts through each intermediate router can be computed as follows:

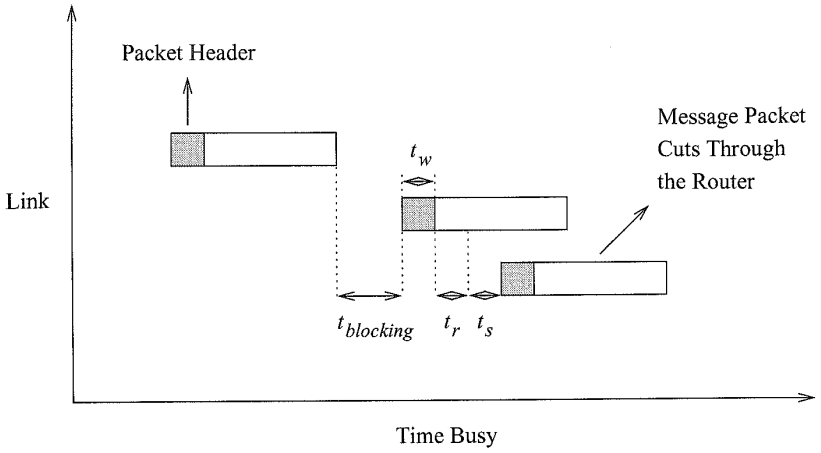


Figure 2.10. Time-space diagram of a virtual cut-through switched message. ($t_{blocking}$ = Waiting time for a free output link.)

$$t_{vct} = D(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{L}{W} \right\rceil \tag{2.3}$$

Cut-through routing is assumed to occur at the flit level with the routing information contained in 1 flit. This model assumes that there is no time penalty for cutting through a router if the output buffer and output channel are free. Depending on the speed of operation of the routers this may not be realistic. Note that only the header experiences routing delay, as well as the switching delay and wire delay at each router. This is because the transmission is pipelined and the switch is buffered at the input and output. Once the header flit reaches the destination, the cycle time of this message pipeline is determined by the maximum of the switch delay and wire delay between routers. If the switch had been buffered only at the input, then in one cycle of operation, a flit traverses the switch and channel between the routers. In this case the coefficient of the second term and the pipeline cycle time would be $(t_s + t_w)$. Note that the unit of message flow control is a packet. Therefore even though the message may cut through the router, sufficient buffer space must be allocated for a complete packet in case the header is blocked.

2.3.4 Wormhole Switching

The need to buffer complete packets within a router can make it difficult to construct small, compact, and fast routers. In wormhole switching, message packets are also pipelined through the network. However the buffer requirements within the routers are substantially reduced over the requirements for VCT switching. A message packet is broken up into flits. The flit is the unit of message flow control, and input and output buffers at a router are typically large enough to store a few flits. For example, the message buffers in the Cray T3D are 1 flit deep and each flit is comprised of eight 16-bit phits. The message is pipelined through the network at the flit level and is typically too large to be completely buffered within a router. Thus, at any instant in time a blocked message occupies buffers in several routers. The time-space diagram of a wormhole-switched message is shown in Figure 2.11. The clear rectangles illustrate the propagation of flits across the physical channel. The

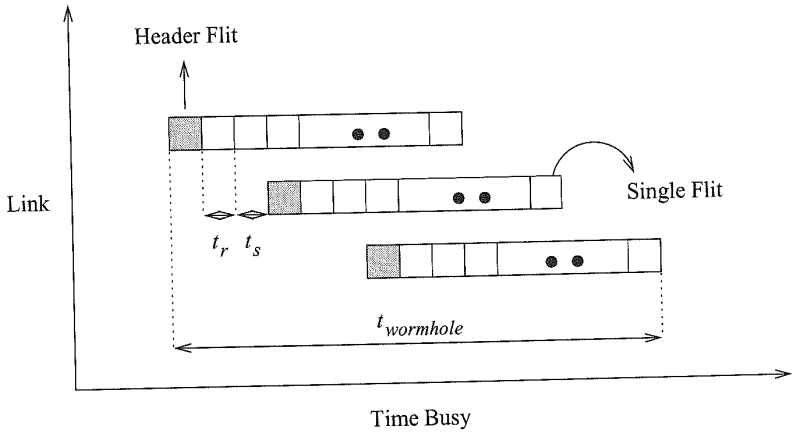


Figure 2.11. Time-space diagram of a wormhole-switched message.

shaded rectangles illustrate the propagation of header flits across the physical channels. Routing delays and intrarouter propagation of the header flits are also captured in this figure. The primary difference between wormhole switching and VCT switching is that the unit of message flow control is a single flit and, as a consequence, the use of small buffers. An entire message cannot be buffered at a router.

In the absence of blocking the message packet is pipelined through the network. However, the blocking characteristics are very different from that of VCT. If the required output channel is busy, the message is blocked "in place." For example, Figure 2.12 illustrates a snapshot of a message being transmitted through routers $R1$, $R2$, and $R3$. Input and output buffers are 2 flits deep and the routing header is 2 flits. At router $R3$, message A requires an output channel that is being used by message B. Therefore message A blocks in place. The small buffer sizes at each node ($<$ message size) causes the message to occupy buffers in multiple routers, similarly blocking other messages. In effect dependencies between buffers span multiple routers. This property complicates the issue of deadlock freedom. However, it is no longer necessary to use the local processor memory to buffer messages, significantly reducing average message latency. The small buffer requirements and message pipelining enable the construction of routers that are small, compact, and fast.

Examples of the format of wormhole-switched packets in the Cray T3D are shown in Figure 2.13. In this machine, a phit is 16 bits wide — the width of a T3D physical channel — and a flit is comprised of 8 phits. A word is 64 bits and thus 4 phits. A message is comprised of header phits and possibly data phits. The header phits contain the routing tag, destination node address, and control information. The routing tag identifies a fixed path through the network. The control information is interpreted by the receiving node to determine any local operations that may have to be performed, e.g., read and return a local datum. Depending on the type of packet, additional header information may include the source node address, and memory address at the receiving node. For example, in the figure, a read-request packet is comprised of only header phits while the read response packet contains four 64-bit words. Each word has an additional phit that contains 14 check bits for error correction and detection.

From the example in Figure 2.13 we note that routing information is associated *only* with the header phits (flits) and not with the data flits. As a result, each incoming data flit of a message packet is simply forwarded along the same output channel as the preceding data flit. As a result, the

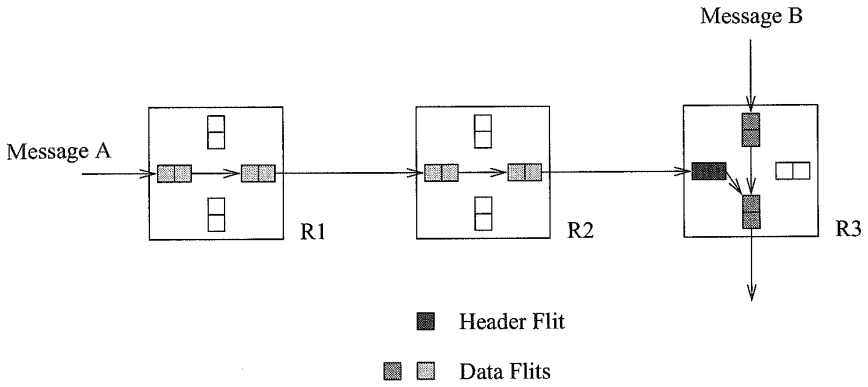


Figure 2.12. An example of a blocked wormhole-switched message.

transmission of distinct messages cannot be interleaved or multiplexed over a physical channel. The message must cross the channel in its entirety before the channel can be used by another message. This is why messages A and B in Figure 2.12 cannot be multiplexed over the physical channel without some additional architectural support.

The base latency of a wormhole-switched message can be computed as follows:

$$t_{wormhole} = D(t_r + t_s + t_w) + \max(t_s, t_w) \left\lceil \frac{L}{W} \right\rceil \tag{2.4}$$

This expression assumes flit buffers at the router inputs and outputs. Note that in the absence of contention, VCT and wormhole switching have the same latency. Once the header flit arrives at the destination, the message pipeline cycle time is determined by the maximum of the switch delay and wire delay. For an input-only, or output-only buffered switch this cycle time would be given by the sum of the switch and wire delays.

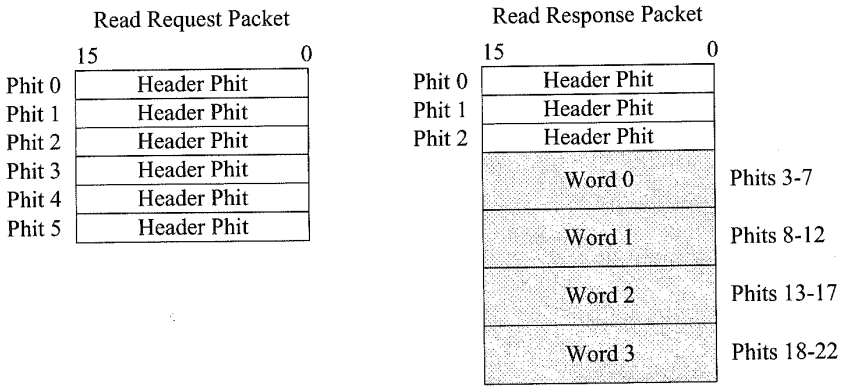


Figure 2.13. Format of wormhole-switched packets in the Cray T3D.

2.3.5 Mad Postman Switching

VCT switching improved the performance of packet switching by enabling pipelined message flow while retaining the ability to buffer complete message packets. Wormhole switching provided further reductions in latency by permitting small buffer VCT so that routing could be completely handled within single-chip routers, therefore providing low latency necessary for tightly coupled parallel processing. This trend toward increased message pipelining is continued with the development of the *mad postman switching* mechanism in an attempt to realize the minimal possible routing latency per node.

The technique is best understood in the context of bit-serial physical channels. Consider a 2-D mesh network with message packets that have a 2-flit header. Routing is dimension-order: messages are first routed completely along dimension 0 and then along dimension 1. The leading header flit contains the destination address of a node in dimension 0. When the message reaches this node, the message is forwarded along dimension 1. The second header flit contains the destination in dimension 1. In VCT and wormhole switching flits cannot be forwarded until the header flits have been received in their entirety at the router. If we had 8-bit flits, transmission of the header flits across a bit-serial physical channel will take 16 cycles. Assuming a 1-cycle delay to select the output channel at each intermediate router, the minimum latency for the header to reach a destination router three links away is 51 cycles. The mad postman attempts to reduce the per-node latency further by pipelining at the bit level. When a header flit starts arriving at a router, it is assumed that the message will be continuing along the same dimension. Therefore header bits are forwarded to the output link in the same dimension as soon as they are received (assuming that the output channel is free). Each bit of the header is also buffered locally. Once the last bit of the first flit of the header has been received, the router can examine this flit and determine if the message should indeed proceed further along this dimension. If it is to proceed along the second dimension, the remainder of the message starting with the second flit of the header is transmitted to the output along the second dimension. If the message has arrived at its destination, it is delivered to the local processor. In essence, the message is first delivered to an output channel and the address is checked later, hence the name of this switching technique. This strategy can work very well in 2-D networks

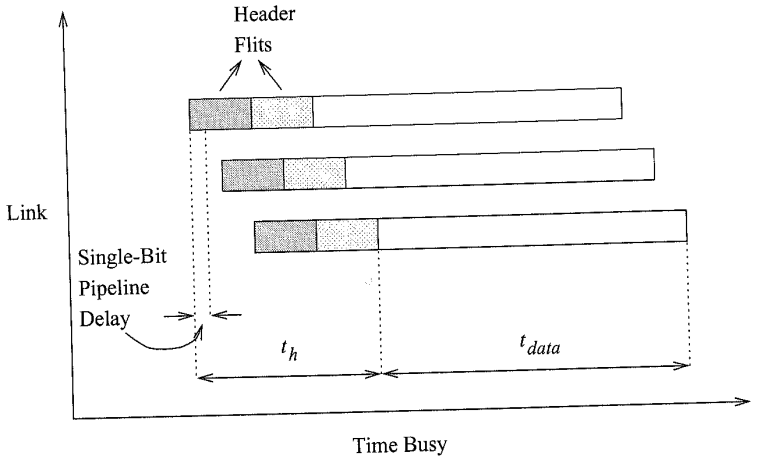


Figure 2.14. Time-space diagram for message transmission using mad postman switching.

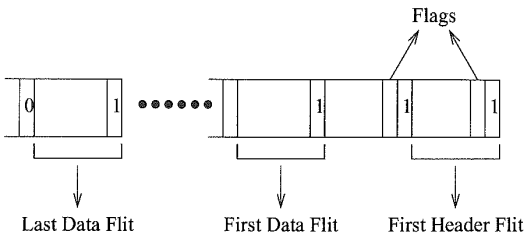


Figure 2.15. An example message format for the mad postman switching technique.

since a message will make at most one turn from one dimension to another and we can encode each dimension offset in 1 header flit. The common case of messages continuing along the same dimension is made very fast. A time-space diagram of a message transmitted over three links using the mad postman switching technique is illustrated in Figure 2.14.

Some constraints must be placed on the organization of the header. An example is shown in Figure 2.15, wherein each dimension offset is encoded in a header flit, and these flits are ordered according to the order of traversal. For example, when the message packet has completely traversed the first dimension the router can start transmitting in the second dimension with the start of the first bit of the second header flit. The first flit has effectively been stripped off the message, but continues to traverse the first dimension. Such a flit is referred to as a *dead address flit.* In a multidimensional network, each time a message changes to a new dimension, a dead flit is generated and the message becomes smaller. At any point if a dead flit is buffered, i.e., blocked by another message packet, it can be detected in the local router and removed.

Let us consider an example of routing in a 4×4, 2-D mesh. In this example the routing header is comprised of 2 flits. Each flit is 3 bits long: a special start bit and 2 bits to identify the destination node in each dimension. The message is pipelined through the network at the bit level. Each input and output buffer is 2 bits deep. Consider the case where a message is being transmitted from node 20 to node 32. Figure 2.16 illustrates the progress and location of the header flits. The message

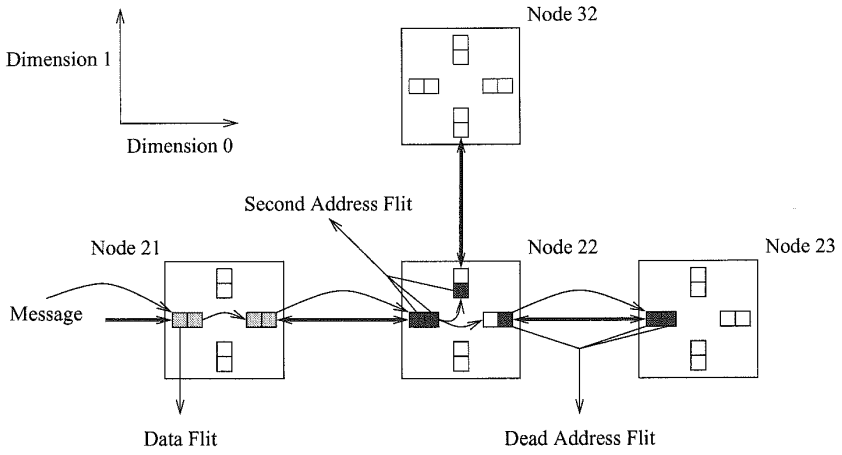


Figure 2.16. Example of routing with mad postman switching and generation of dead address flits.

is transmitted along dimension 0 to node 22 where it is transmitted along dimension 1 to node 32. At node 22, the first flit is pipelined through to the output as it is received. After receiving the third bit, it is determined the message must continue along dimension 1. The first bit of the second header flit is forwarded to the output in dimension 1 as shown in the figure. Note that header flits are stripped off as the message changes dimensions and the message becomes smaller. The dead address flit proceeds along dimension 0 until it can be detected and removed.

For a given number of processors the size of the dead address flit is determined by the number of processors in a dimension. Therefore it follows that for a given number of processors, low-dimension networks will introduce a smaller number of larger dead address flits while higher-dimension networks will introduce a larger number of smaller dead address flits. Initially it would appear that the dead address flits would adversely affect performance until they are removed from the network since they consume physical channel bandwidth. Since message packets will generally be larger than a dead address flit, the probability of a packet being blocked by a dead address flit is very small. It is more likely that a dead address flit will be blocked by a message packet. In this case the local router has an opportunity to detect the dead address flit and remove it from the network. At high loads, we are concerned with dead address flits consuming precious network bandwidth. It is interesting to note that increased blocking in the network will provide more opportunities for routers to remove dead address flits. The greater the congestion, the less likely that a packet will encounter a dead address flit!

By optimistically forwarding the message bit stream to an output channel the routing latency at a node is minimized and full bit-level pipelining can be achieved. Considering again a 2-D mesh with bit-serial physical channels and packets that have two 8-bit header flits, traversing three links, the minimum latency for the header to reach the destination is 18 rather than 51 cycles. In general, the mad postman strategy is useful when it takes multiple cycles for a header to cross a physical channel. In this case latency can be reduced by optimistically forwarding portions of the header onward before the correct output link is actually determined. However, the pin-out constraints of modern routers permit wider flits to be transmitted across a channel in a single cycle. If the header can be transmitted in one cycle, there is little if any advantage to be gained.

The base latency of a message routed using the mad postman switching technique can be computed as follows:

$$\begin{aligned} t_{madpostman} &= t_h + t_{data} \\ t_h &= (t_s + t_w)D + \max(t_s, t_w)W \\ t_{data} &= \max(t_s, t_w)L \end{aligned} \tag{2.5}$$

The above expression makes several assumptions. The first is the use of bit-serial channels which is the most favorable for the mad postman strategy. The routing time t_r is assumed to be equivalent to the switch delay and occurs concurrently with bit transmission, and therefore does not appear in the expression. The term t_h corresponds to the time taken to completely deliver the header.

Let us consider the general case where we do not have bit-serial channels, but rather C bit channels, where $1 < C < W$. Multiple cycles would be required to transfer the header flit across the physical channel. In this case the mad postman switching strategy would realize a base latency of

$$t_{madpostman} = D(t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{W}{C} \right\rceil + \max(t_s, t_w)\left\lceil \frac{L}{C} \right\rceil \tag{2.6}$$

For comparison purposes, in this case the expression for wormhole switching would have been

$$t_{wormhole} = D\left\{t_r + (t_s + t_w)\left\lceil\frac{W}{C}\right\rceil\right\} + \max(t_s, t_w)\left\lceil\frac{L}{C}\right\rceil \quad (2.7)$$

Assuming that the internal and external channel widths are C bits, a header flit (of width W bits) requires $\lceil\frac{W}{C}\rceil$ cycles to cross the channel and the router. This cost is incurred at each intermediate router. When $C = W$, the above expression reduces to the previous expression for wormhole switching with single-flit headers. As larger physical channel widths become feasible in practice, the advantage of the mad postman switching over wormhole switching will diminish.

2.4 Virtual Channels

The preceding switching techniques were described assuming that messages or parts of messages were buffered at the input and output of each physical channel. Buffers are commonly operated as FIFO queues. Therefore once a message occupies a buffer for a channel, no other message can access the physical channel, even if the message is blocked. Alternatively, a physical channel may support several *logical* or *virtual channels* multiplexed across the physical channel. Each unidirectional virtual channel is realized by an independently managed pair of message buffers as illustrated in Figure 2.17. This figure shows two unidirectional virtual channels in each direction across the physical channel. Consider wormhole switching with a message in each virtual channel. Each message can share the physical channel on a flit-by-flit basis. The physical channel protocol must be able to distinguish between the virtual channels using the physical channel. Logically, each virtual channel operates as if each were using a distinct physical channel operating at half the speed. Virtual channels were originally introduced to solve the problem of deadlock in wormhole-switched networks. Deadlock is a network state where no messages can advance because each message requires a channel occupied by another message. This issue is discussed in detail in Chapter 3.

Virtual channels can also be used to improve message latency and network throughput. By allowing messages to share a physical channel, messages can make progress rather than remain blocked. For example, Figure 2.18 shows two messages crossing the physical channel between routers $R1$ and $R2$. With no virtual channels message A will prevent message B from advancing until the transmission of message A has been completed. However, in the figure, there are two single-flit virtual channels multiplexed over each physical channel. By multiplexing the two messages on a flit-by-flit basis, both messages continue to make progress. The rate at which each message is forwarded

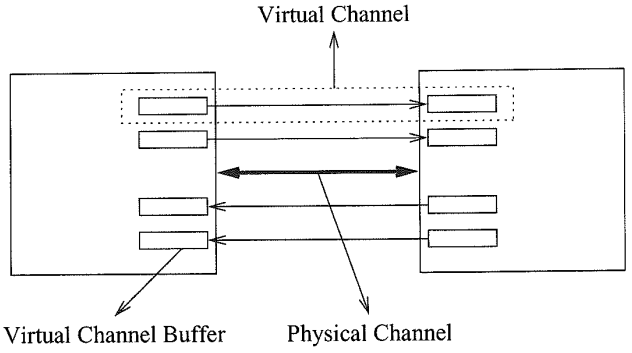


Figure 2.17. Virtual channels.

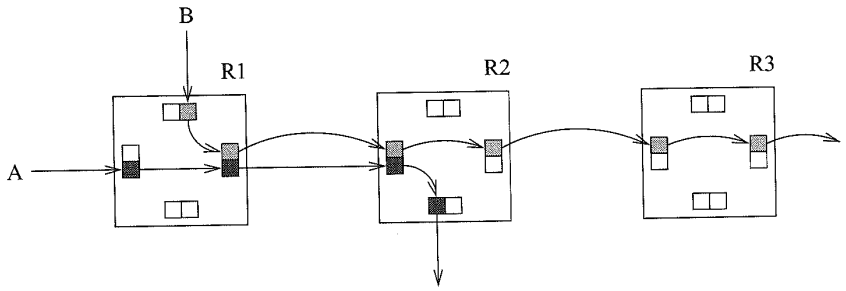


Figure 2.18. An example of the reduction in header blocking delay by using two virtual channels for each physical channel.

is nominally one half the rate achievable when the channel is not shared. In effect, the use of virtual channels decouples the physical channels from message buffers allowing multiple messages to share a physical channel in the same manner that multiple programs may share a central processing unit (CPU). The overall time a message spends blocked at a router waiting for a free channel is reduced leading to an overall reduction in individual message latency. There are two specific cases where such sharing of the physical link bandwidth is particularly beneficial. Consider the case where message A is temporarily blocked downstream from the current node. With an appropriate physical channel flow control protocol, message B can be make use of the full bandwidth of physical channel between the routers. Without virtual channels, both messages would be blocked. Alternatively, consider the case where message A is a very long message relative to message B. Message B can still make progress at half the link speed, and then message A can resume transmission at the full link speed. Studies have shown that message traffic in parallel programs is often bimodal comprised of short (cache lines, control messages) and long messages (data structures) [176].

The approach described in the preceding paragraph does not place any restrictions on the use of the virtual channels. Therefore, when used in this manner these buffers are referred to as *virtual lanes*. Virtual channels were originally introduced as a mechanism for deadlock avoidance in networks with physical cycles, and as such routing restrictions are placed on their use. For example, packets may be prohibited from being transferred between certain classes of virtual channels to prevent cyclic waiting dependencies for buffer space. Thus, in general we have virtual channels that may in turn be comprised of multiple lanes. While the choice of virtual channels at a router may be restricted, it does not matter which lane within a virtual channel is used by a message, although all of the flits within a message will use the same lane within a channel.

We have seen from Section 2.2 that acknowledgment traffic is necessary to regulate the flow of data and to ensure the availability of buffer space on the receiver. Acknowledgments are necessary for each virtual channel or lane, increasing the volume of such traffic across the physical channel. Furthermore, for a fixed amount of buffer space within a router, the size of each virtual channel or lane buffer is now smaller. Therefore the effect of optimizations such as the use of acknowledgments for a block of flits or phits is limited. If physical channel bandwidth is allocated in a demand-driven fashion, the operation of the physical channel now includes the transmission of the virtual channel address to correctly identify the receiving virtual channel, or to indicate which virtual channel has available message buffers.

We can envision continuing to add virtual channels to further reduce the blocking experienced by each message. The result is increased network throughput measured in flits/s, due to increased physical channel utilization. However, each additional virtual channel improves performance by a

smaller amount, and the increased channel multiplexing reduces the data rate of individual messages, increasing the message latency. This increase in latency due to data multiplexing will eventually overshadow the reduction in latency due to blocking leading to overall increasing average message latency. An analysis of this phenomena can be found in Chapter 9 which provides detailed performance data to quantify various aspects of network performance.

Increasing the number of virtual channels has a direct impact on router performance through their effect on the achievable hardware cycle time of the router. The link controllers now become more complex since they must support arbitration between multiple virtual channels/lanes for the physical channel, and this arbitration function can be on the critical path for internode delay. The number of inputs and outputs that must be switched at each node is increased, substantially increasing the switch complexity. For a fixed amount of buffer space in a node, how is this buffer space to be allocated among channels, and lanes within a channel? Further, the flow of messages through the router must be coordinated with the allocation of physical channel bandwidth. The increasing complexity of these functions can lead to net increases in internal and external flow control latencies. This increase affects all messages through the routers. Such trade-offs and related issues affecting the design of routers are discussed in detail in Chapter 7 and evaluated in Chapter 9.

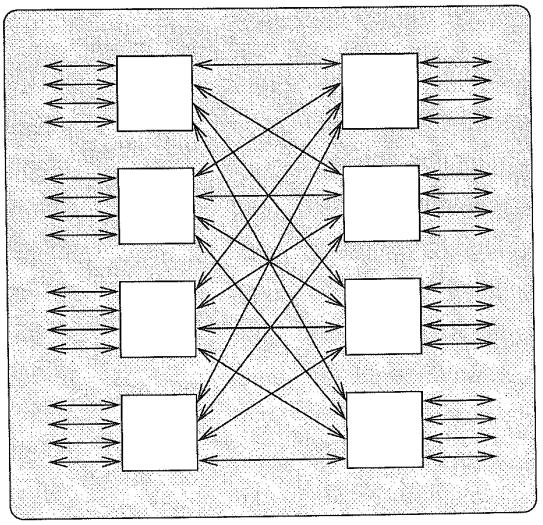
2.5 Hybrid Switching Techniques

The availability and flexibility of virtual channels have led to the development of several hybrid switching techniques. These techniques have been motivated by a desire to combine the advantages of several basic approaches, or have been motivated by the need to optimize performance metrics other than traditional latency and throughput, e.g., fault tolerance and reliability. Some common hybrid switching techniques are presented in this section.

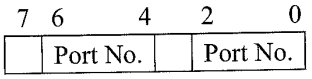
2.5.1 Buffered Wormhole Switching

A switching technique that combines aspects of wormhole switching and packet switching is *buffered wormhole switching* (BWS), proposed and utilized in IBM's Power Parallel SP systems. The switching technique and message formats have been motivated by the interconnection network utilized in the SP systems. This network is a multistage, generalized Omega network using 4×4 crossbar switches with bidirectional links. The system building block is a 16-processor system configured around the two-stage switch with eight crossbar switches as shown in Figure 2.19a. This module is referred to as a frame. Each 4×4 switch uses bidirectional links, and therefore can be viewed as an 8×8 implementation of the router organization shown in Figure 2.1 with the functionality described below.

The basic switching mechanism is wormhole switching. Message packets can be of variable length and up to 255 flits in length. Each flit is 1 byte and is equal to the physical channel width. The first flit of a message contains the length of the message while the following flits contain routing information. Routing is source-based where each routing flit contains the address of the output ports in intermediate switches. There is 1 routing flit for each frame, i.e., for each group of 16 processors. The format of the routing flit is shown in Figure 2.19b. Note that these 4×4 crossbar switches have bidirectional links, and therefore eight input ports and eight output ports. Bits 4–6 are used to select the output port of the first switch in the frame. Bits 0–2 are used to select the output port of the second switch in the frame. Bit 7 is used to determine which field is to be used. It is initially cleared and set by the first switch in the frame. Larger systems are built up as groups of frames. Every frame requires 1 routing flit.



(a)



(b)

Figure 2.19. (a) Organization of the switch used in the IBM Power Series parallel machines. (b) Routing flit format.

As the message is routed through the switches, the corresponding routing flit is discarded, shortening the message. This is similar to the mad postman switching strategy. As long as the path is conflict-free, the message progresses as in wormhole switching with interswitch flow control operating at the flit level. When output channels are available, data flow through the switch is through byte-wide paths through the internal crossbar and to the output port. When messages block, flow control within the switch is organized into 8-flit units referred to as *chunks*. When messages block, chunks are constructed at the input port of a switch, transmitted through 64-bit-wide paths to the a local memory. Subsequently, buffered chunks are transferred to an output port where they are converted to a flit stream for transmission across the physical channel.

When there is a conflict at the output of a routing node, flits are buffered within the switch as chunks. These chunks are buffered in a dynamically allocated central storage. The storage is organized as a linked list for each output, where each element of the list is a message to be transmitted on that output port. Since only the first chunk of a message contains routing information, each message is in turn organized as a list of chunks. Thus, ordering of flits within a message packet is preserved. This type of organization is depicted in Figure 2.20 where two messages are shown queued at an output port. The first message is comprised of 24 flits and the second message is 16 flits long. Messages waiting on each of the other output ports are similarly organized. When an

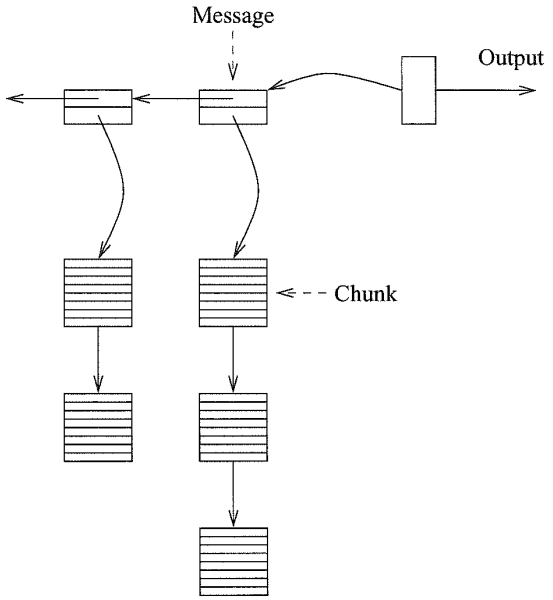


Figure 2.20. Logical organization of the message flit buffers.

output port becomes free, messages are transmitted to the output channel as 64-bit chunks in a single cycle: since the internal datapath and flow control to/from central memory is based on 8-flit chunks. The central storage is dual-ported and can support 128 chunks. A minimum of one chunk is made available for each output port. The remaining chunks are dynamically allocated as necessary. In a single cycle, one input port or one output port can be serviced from the central storage. Thus we see that short messages can be completely buffered.

BWS differs from wormhole switching in that flits are not buffered in place. Rather flits are aggregated and buffered in a local memory within the switch. If the message is small and space is available in the central queue, the input port is released for use by another message even though this message packet remains blocked. In this respect, BWS appears similar to packet switching. BWS differs from packet switching and VCT switching in that flow control is largely at the flit level and when messages are blocked, flow control (within the switch) is at the level of 8-flit chunks. If the central queue were made large enough to ensure that complete messages could always be buffered, the behavior of BWS would approach that of VCT switching.

The base latency of a message routed using BWS is identical to that of wormhole-switched messages.

2.5.2 Pipelined Circuit Switching

In many environments rather than minimizing message latency or maximizing network throughput, the overriding issue is the ability to tolerate the failure of network components such as routers and links. In wormhole switching, header flits containing routing information establish a path through the network from source to destination. Data flits are pipelined through the path immediately following the header flits. If the header cannot progress due to a faulty component, the message is

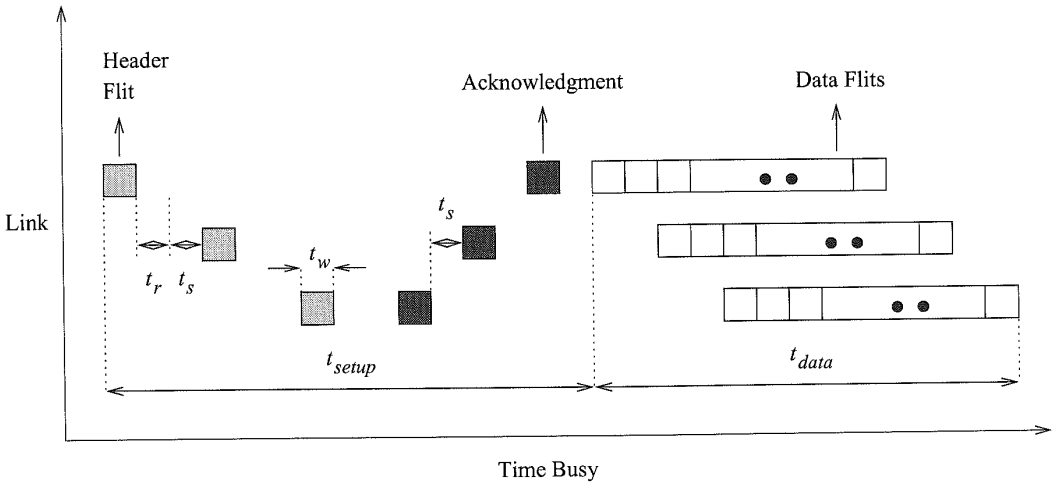
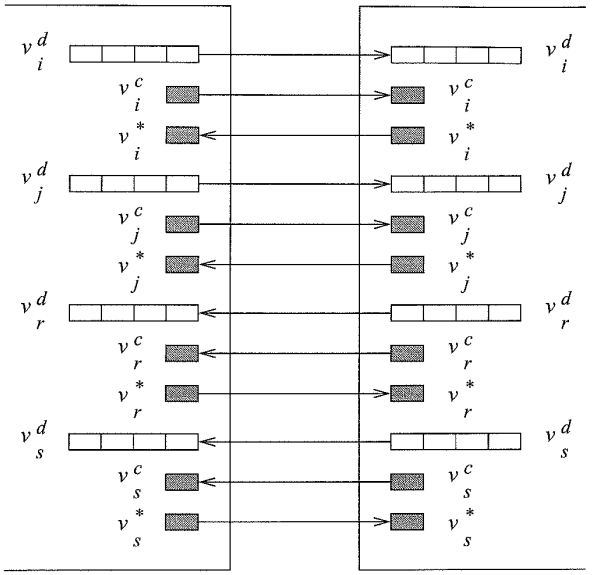


Figure 2.21. Time-space diagram of a message transmitted using PCS.

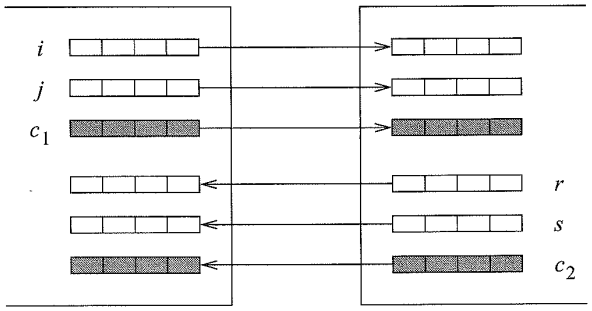
blocked in place indefinitely, holding buffer resources and blocking other messages. This situation can eventually result in a deadlocked configuration of messages. While techniques such as adaptive routing can alleviate the problem, it cannot by itself solve the problem. This has motivated the development of different switching techniques.

Pipelined circuit switching (PCS) combines aspects of circuit switching and wormhole switching. PCS sets up a path before starting data transmission as in circuit switching. Basically, PCS differs from circuit switching in that paths are formed by virtual channels instead of physical channels. In pipelined circuit switching, data flits do not immediately follow the header flits into the network as in wormhole switching. Consequently, increased flexibility is available in routing the header flit. For example, rather than blocking on a faulty output channel at an intermediate router, the header may backtrack to the preceding router and release the previously reserved channel. A new output channel may now be attempted at the preceding router in finding an alternative path to the destination. When the header finally reaches the destination node, an *acknowledgment flit* is transmitted back to the source node. Now data flits can be pipelined over the path just as in wormhole switching. The resilience to component failures is obtained at the expense of larger path setup times. This approach is flexible in that headers can perform a backtracking search of the network, reserving and releasing virtual channels in an attempt to establish a fault-free path to the destination. This technique combines message pipelining from wormhole switching with a more conservative path setup algorithm based on circuit switching techniques. A time-space diagram of a PCS message transmission over three links in the absence of any traffic or failures is shown in Figure 2.21.

Since headers do not block holding channel or buffer resources, routing restrictions are not necessary to avoid deadlock. This increases the probability of finding a path while still avoiding deadlocked configurations of messages. Moreover, reservation of virtual channels by the header does not by itself lead to use of physical channel bandwidth. Therefore, unlike circuit switching, path setup does not lead to excessive blocking of other messages. As a result, multipath networks in conjunction with the flexibility of PCS are good candidates for providing low-latency fault-tolerant performance. For purely performance-driven applications where fault tolerance is not a primary concern, the added overhead of PCS makes wormhole switching the mechanism of choice.



(a)



(b)

Figure 2.22. Virtual channel model for PCS.

In PCS, we distinguish between flits that carry control information, e.g., header flits and acknowledgment flits, and those that carry data. This distinction is supported in the virtual channel model that separates control flit traffic and data flit traffic. A unidirectional virtual channel v_i is composed of a *data channel*, a *corresponding channel*, and a *complementary channel* (v_i^d, v_i^c, v_i^*) and is referred to as a *virtual channel trio*. The router header will traverse v_i^c while subsequent data flits will traverse v_i^d. The complementary channel v_i^* is reserved for use by acknowledgment flits and backtracking header flits. The complementary channel of a trio traverses the physical channel in the direction opposite to that of its associated data channel. The channel model is illustrated in Figure 2.22. There are two virtual channels $v_i(v_r)$ and $v_j(v_s)$ from $R1$ ($R2$) to $R2$ ($R1$). Only

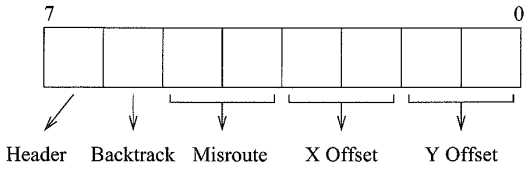


Figure 2.23. Example format of a PCS header.

one message can be in progress over a given data channel. Therefore, compared to existing channel models, this model requires exactly two extra flit buffers for each data channel — one each for the corresponding channel and the complementary channel, respectively. Since control flit traffic is a small percentage of the overall flit traffic, in practice all control channels across a physical link are multiplexed through a single virtual control channel as shown in Figure 2.22a. Thus, compared to the more common use of virtual channels, this model requires one extra virtual channel in each direction between a pair of adjacent routers. For example, channel c_1 in Figure 2.22b corresponds to flit buffers $v_i^c, v_j^c, v_r^*, v_s^*$. The implementation of PCS in the Ariadne router [7] utilized two data channels and one virtual control channel over each physical link.

This separation of control traffic and data traffic is useful in developing fault tolerant routing and distributed fault recovery mechanisms. Such mechanisms are discussed in greater detail in Chapter 6. The Ariadne router [7] is a single-chip PCS router with two virtual channel trios per physical channel. The prototype router had byte-wide physical channels and 8-bit flits. The format of the header flit is shown in Figure 2.23. In this design a single bit distinguished a control flit from a data flit (this only left 7-bit data flits!). A single bit distinguishes between backtracking flits and flits making forward progress. The misroute field keeps track of the number of misrouting steps the header has taken. The maximum number of misroutes that the header can take in this design is 3. Finally, two fields provide X and Y offsets for routing in a 2-D mesh.

The base latency of a pipelined circuit switched message can be computed as follows:

$$\begin{aligned} t_{pcs} &= t_{setup} + t_{data} \\ t_{setup} &= D(t_r + t_s + t_w) + D(t_s + t_w) \\ t_{data} &= D(t_s + t_w) + \max(t_s, t_w)\left(\lceil \tfrac{L}{W} \rceil - 1\right) \end{aligned} \tag{2.8}$$

The first term in t_{setup} is the time taken for the header flit to reach the destination. The second term is the time taken for the acknowledgment flit to reach the source node. We then have t_{data} as the time for pipelining the data flits into the destination network interface. The first term is the time for the first data flit to reach the destination. The second term is the time required to receive the remaining flits. The message pipeline cycle time is determined by the maximum of the switch delay and wire delay.

2.5.3 Scouting Switching

Scouting switching is a hybrid message flow control mechanism that can be dynamically configured to provide specific trade-offs between fault tolerance and performance. In PCS the first data flit is injected into the network only after the complete path has been set up. In an attempt to reduce PCS path setup time overhead, in scouting switching the first data flit is constrained to remain at least K links behind the routing header. When $K = 0$, the flow control is equivalent to wormhole switching, while large values can ensure path setup prior to data transmission (if a path exists).

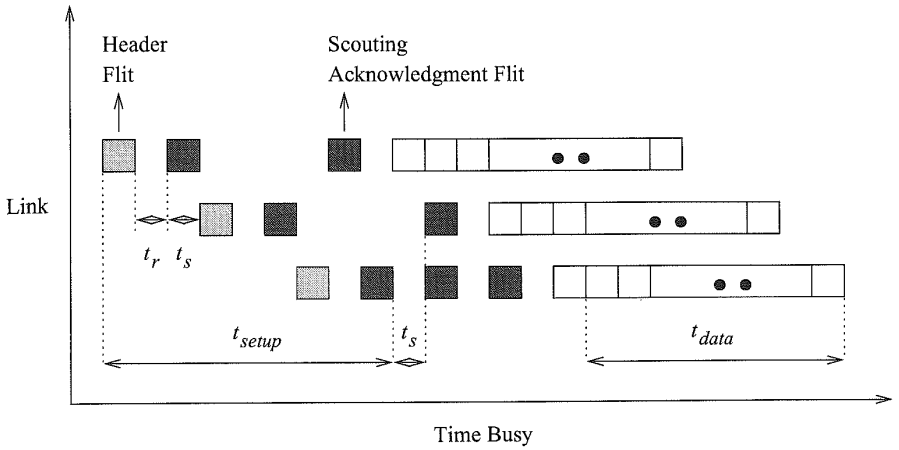


Figure 2.24. Time-space diagram of a message transmitted using scouting switching.

Intermediate values of K permit the data flits to follow the header at distance, while still allowing the header to backtrack if the need arises. Therefore when the header reaches the destination, the first data flit arrives shortly thereafter rather than immediately (as in wormhole switching). Figure 2.24 illustrates a time-space diagram for messages being pipelined over three links using scouting switching ($K = 2$). The parameter, K, is referred to as the *scouting distance* or *probe lead.* Every time a channel is successfully reserved by the routing header, a positive acknowledgment is returned in the opposite direction. As a particular case, positive acknowledgments are continuously transmitted when the routing header has reached the destination node. Associated with each virtual channel is a programmable counter. The counter associated with the virtual channel reserved by a header is incremented when a positive acknowledgment is received, and is decremented when a negative acknowledgment is received. When the value of the counter is equal to K, data flits are allowed to advance. As acknowledgments flow in the direction opposite to the routing header, the gap between the header and the first data flit can grow up to a maximum of $2K - 1$ links while the header is advancing. If the routing header backtracks, a negative acknowledgment is transmitted. For performance reasons, when $K = 0$ no acknowledgments are sent across the channels. In this case, data flits immediately follow the header flit and flow control is equivalent to wormhole switching.

For example, in Figure 2.25 a message is being transmitted between nodes A and G and $K = 2$. The initial path attempted by the header is row first. Data flits remain at least two links behind the header. On encountering faulty output link at node B, the header can backtrack over the previous link. Encountering another faulty link the header can still backtrack one more link to node C. During this time the first data flit remains blocked at node C. From node C it is possible to make progress towards the destination via node D. When the header reaches node F, it is $2K - 1 = 3$ links from the first data flit at node C, and data flits can begin to flow again.

By statically fixing the value of K, we fix the trade-off between network performance (overhead of positive and negative acknowledgment) and fault tolerance (the number of faults that must be tolerated). By dynamically modifying K, we can gain further improvement via run-time trade-offs between fault tolerance and performance. Such configurable flow control protocols are discussed in the context of fault tolerant and reliable routing in Chapter 6.

The base latency of scouting switching can be computed as follows:

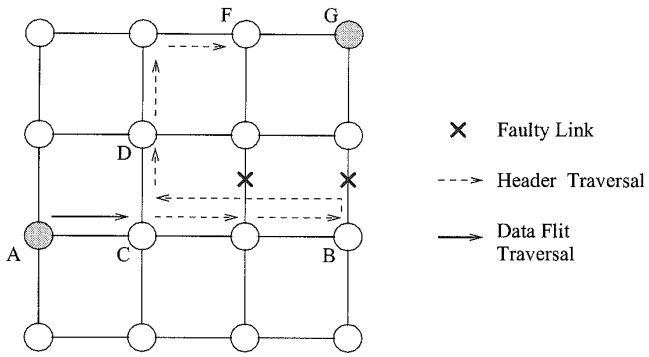


Figure 2.25. An example of fault-tolerant routing enabled by scouting switching.

$$\begin{aligned} t_{scouting} &= t_{setup} + (t_s + t_w)(2K-1) + t_{data} \\ t_{setup} &= D(t_r + t_s + t_w) \\ t_{data} &= \max(t_s, t_w)\left(\lceil \tfrac{L}{W} \rceil - 1\right) \end{aligned} \tag{2.9}$$

The first term is the time taken for the header flit to reach the destination. The first data flit can be at a maximum of $(2K-1)$ links behind the header. The second term is the time taken for the first data flit to reach the destination. The last term is the time for pipelining the remaining data flits into the destination network interface.

2.6 Optimizing Switching Techniques

The switching techniques described in this chapter are subject to application-specific optimizations to further improve performance and/or reliability. Such optimizations do not fundamentally alter the nature of these techniques but can lead to considerable improvements in performance in specific application domains. For example, consider the overheads experienced in transmitting and receiving a message. The programming interface may be a message-passing library comprised of various send and receive procedures. The concern at this level has been to provide consistent semantics for sending and receiving messages. The source program builds a message in local buffers and transfers control to the operating system via system calls. Data are copied into operating system space where protected device drivers construct message headers, packetize the data, and interact with special-purpose network interfaces to inject data into the network. This overhead is experienced with every message. When message sizes are small, the overhead on a per-message basis can be substantial. There have been several successful efforts to reduce this overhead. Often, hardware support is provided for packetization and network interfaces are becoming tightly coupled with memory and in some cases even the processor registers through memory-mapped techniques and existence on the processor memory bus rather than on the slower I/O buses. Copying to and from system buffers is also being eliminated through the use of message handlers that operate within the user address spaces. Modern machines are now beginning to focus on the design of the interface between the network and memory. These techniques are beneficial regardless of the switching techniques.

Similarly, optimizations within the low-level physical channel flow control protocols benefit most switching techniques. High-speed signaling mechanisms for physical channel flow control affect the

interrouter latency. As higher-dimensional networks and wider channels are employed the number of inputs and outputs at a router can become very large. Current packaging technology provides chip carriers with 300–400 pins. These pins can begin to become a scarce resource. One innovative technique to addressing this problem is the use of bidirectional signaling [75, 193]. This technique allows simultaneous signaling between two routers across a single signal line. Thus, full-duplex, bidirectional communication of a single bit between two routers can be realized with one pin (signal) rather than two signals. A logic 1 (0) is transmitted as positive (negative) current. The transmitted signal is the superposition of these two signals. Each transmitter generates a reference signal which is subtracted from the superimposed signal to generate the received signal. The result is a considerable savings over the number of input/output pins required, and consequent reduction in the packaging cost. Such optimizations at the physical level are also clearly independent of and benefit all switching techniques.

Application environments that exhibit locality in interprocessor communication are particularly good candidates for application-specific optimizations. For example, systolic computation makes use of fine-grained, pipelined, parallel transmission of multiple data streams through a fixed communication topology such as multidimensional mesh or hexagonal interconnection topologies. In such cases, it is beneficial to set up interprocessor communication paths once to be shared by many successive data elements (i.e., messages). The Intel iWarp chip was designed to support such systolic communication through *message pathways*: long-lived communication paths [39]. Rather than set up and remove network paths each time data are to be communicated, paths through the network persist for long periods of time. Special messages called *pathway begin markers* are used to reserve virtual channels (referred to as *logical channels* in iWarp) and set up interprocessor communication paths. Messages are periodically injected into the network to use these existing paths utilizing wormhole switching. On completion of the computation, the paths are explicitly removed by other control messages. Unlike conventional wormhole switching, the last flit of the message does not cause the routers to tear down the path. The overhead of preparing a node for message transmission is incurred once, and amortized over all messages that use the path. Such optimizations are possible due to the regular, predictable nature of systolic communication. The basic switching technique is wormhole, but it is applied in a manner to optimize the characteristics of the applications.

Not only fine-grained parallel algorithms exhibit communication locality that can be exploited by switching techniques. Studies of VLSI CAD programs and programs from linear algebra have shown that coarse-grained message-passing applications can also exhibit sufficient communication locality to benefit from long-lived communication paths and justify the design of an enhanced wormhole router [159]. As in the iWarp chip, paths are set up and shared by multiple messages until they are explicitly removed. The basic wormhole switching technique is modified to prevent the path from being removed when the first message has been successfully received at the destination. In this case interprocessor paths can even be shared between applications in a multiprogrammed parallel architecture. Message flow control must avoid deadlock as well as ensuring that reserved paths do not preclude new messages from being injected into the network.

The need for reliable message transmission has also led to proposals for enhancing switching techniques. For example, one way to ensure message delivery is to buffer the message at the source until it can be asserted that the message has been received at the destination. Receipt at the destination can be determined through the use of message acknowledgments. In this case, paths are removed by explicit control signals/messages generated by the source/destination rather than by a part of the message. Alternatively, consider the use of wormhole switching when the number of flits in the message exceeds the number of links, D, between the source and destination nodes. If each router only buffers a single flit, receipt of the header at the destination can be asserted at the source when flit $D+1$ is injected into the network. If messages are short, they can be padded with

empty flits so that the number of flits in the message exceeds the distance between the source and destination (padding must also account for buffer space within each router). By keeping track of the number of flits that have been injected into the network, the source router can determine if the header has been received at the destination. Moreover, the source node can determine if the whole message has been received at the destination by injecting $D + 1$ padding flits after the last data flit of the message. Such reliable switching techniques modify the basic wormhole switching mechanisms to include additional flits or control signals, e.g., acknowledgments or padding flits. This particular technique was proposed as *compressionless routing* by its developers [179].

The need to support distinct traffic types also leads to new optimizations of switching techniques [293]. Real-time communication traffic places distinct demands on the performance and behavior of network routers. Such traffic requires guaranteed bounds on latency and throughput. The manner in which messages are buffered and scheduled across physical channels must be able to provide such guarantees on a per-router basis. Such guarantees would be used by higher-level, real-time scheduling algorithms. Packet switching is attractive from this point of view since predictable demands are made on buffer requirements and channel bandwidth at each intermediate router. In contrast, the demands that will be placed on router resources by a message using VCT will vary depending on the load and communication pattern (i.e., is the output link free). Buffering of packets permits the application of priority-based scheduling algorithms and thus provide some control over packet latencies. Wormhole-switched messages use demand-driven scheduling disciplines for accessing physical channel bandwidth and may be blocked across multiple nodes. Demand-driven scheduling and very low buffer requirements work to provide low average latency but high variability and thus poor predictability. Priority-based scheduling of virtual channels is infeasible since a channel may have messages of multiple priorities, and messages may be blocked over multiple links. These properties make it difficult to utilize wormhole switching to support real-time traffic. Rexford and Shin [293] observed that packet switching and wormhole switching made demands on distinct router resources while sharing physical channel bandwidth. Thus, the authors proposed a scheme where virtual channels were partitioned to realize two distinct virtual networks: one packet-switched, and the other wormhole-switched. The two networks share the physical link bandwidth in a controlled manner, thus enabling the network to provide latency and throughput guarantees for real-time traffic, while standard traffic realized low average latency. The switching technique experienced by a message is determined at the source node, based on the traffic type.

We can envision other optimizations that deal with issues such as allocation/deallocation of buffer space within routers, allocation/deallocation of virtual channels, scheduling of virtual channels (equivalently messages) over the physical channel, etc. Some of these optimizations are examined in greater detail in the Exercises section at the end of this chapter.

2.7 A Comparison of Switching Techniques

The evolution of switching techniques was naturally influenced by the need for better performance. VCT switching introduced pipelined message transmission, and wormhole switching further contributed reduced buffer requirements in conjunction with fine-grained pipelining. The mad postman switching technique carried pipelining to the bit level to maximize performance. In packet switching and VCT messages are completely buffered at a node. As a result, the messages consume network bandwidth proportional to the network load. On the other hand, wormhole-switched messages may block occupying buffers and channels across multiple routers, precluding access to the network bandwidth by other messages. Thus, while average message latency can be low, the network saturates at a fraction of the maximum available bandwidth and the variance of message latency can be high.

The use of virtual channels decouples the physical channel from blocked messages, thus reducing the blocking delays experienced by messages and enabling a larger fraction of the available bandwidth to be utilized. However, the increasing multiplexing of multiple messages increases the delay experienced by data flits. Furthermore, multiple virtual channels can increase the flow control latency through the router and across the physical channel, producing upward pressure on average message latency.

The effects of wormhole switching on individual messages can be highly unpredictable. Since buffer requirements are low, contention in the network can substantially increase the latency of a message in parts of the network. Packet switching tends to have more predictable latency characteristics, particularly at low loads since messages are buffered at each node. VCT will operate like wormhole switching at low loads and approximate packet switching at high loads where link contention will force packets to be buffered at each node. Thus, at low loads we expect to see wormhole switching techniques providing superior latency/throughput relative to packet-switched networks, while at high loads we expect to see packet-switched schemes perform better. As expected, the performance of VCT approaches that of wormhole switching at low loads and that of packet switching at high loads. More detailed performance comparisons can be found in Chapter 9.

These switching techniques can be characterized as *optimistic* in the sense that buffer resources and links are allocated as soon as they become available, regardless of the state of progress of the remainder of the message. In contrast, pipelined circuit switching and scouting switching may be characterized as *conservative*. Data flits are transmitted only after it is clear that flits can make forward progress. These flow control protocols are motivated by fault tolerance concerns. BWS seeks to improve the fraction of available bandwidth that can be exploited by wormhole switching by buffering groups of flits.

In packet switching, error detection and retransmission can be performed on a link-by-link basis. Packets may be adaptively routed around faulty regions of the network. When messages are pipelined over several links, error recovery and control becomes complicated. Error detection and retransmission (if feasible) must be performed by higher-level protocols operating between the source and destination, rather than at the level of the physical link. If network routers or links have failed, message progress can be indefinitely halted, with messages occupying buffer and channel resources. This can lead to deadlocked configurations of messages and eventually failure of the network.

2.8 Engineering Issues

Switching techniques have a very strong impact on the performance and behavior of the interconnection network. Performance is more heavily influenced by the switching technique than by the topology or the routing algorithm. Furthermore, true tolerance to faulty network components can only be obtained by using a suitable switching technique. The use of topologies with multiple alternative paths between every source destination pair, and the availability of adaptive routing protocols simply reduces the probability of a message encountering a faulty component. The switching technique determines how messages may recover from faulty components.

Switching techniques also have a considerable influence on the architecture of the router, and as a result, the network performance. For example, consider the magnitude of the improvement in performance that wormhole switching provides over packet switching. First-generation multicomputers such as the Intel iPSC/1 utilized packet switching. The iPSC/1 network had routing times on the order of several milliseconds. In addition, message latency was proportional to the distance traveled by the message. In contrast, modern multicomputers such as the Intel Paragon and the Cray T3D have routing and switch delays on the order of several nanoseconds. Message latency has decreased

from a few tens of milliseconds to a few hundreds of nanoseconds. In one decade, latency improved by five orders of magnitude! Obviously, this improvement benefits from advances in VLSI technology. However, VLSI technology only improved performance by one order of magnitude. Network devices were clocked at 10 MHz in the Intel iPSC/1 while the Cray T3D is clocked at 150 MHz.

Pipelined message transfer alone cannot be responsible for this magnitude of performance improvement. What then is the source? An important difference between first-generation multicomputers and current multicomputers is that the routing algorithm is computed in hardware. However, packet switching would still be much slower than wormhole switching even if the routing algorithm were computed in hardware in both cases. Wormhole switching performs flow control at the flit level. This apparently unimportant change considerably reduces the need for buffering space. Small hardware buffers are enough to handle flit propagation across intermediate routers. As a consequence, wormhole routers are small, compact, and fast. Moreover, wormhole routers are able to handle messages of any length. However, packet-switched routers must provide buffering space for full packets, either limiting packet size or necessitating the use of local processor memory for storing packets. Access to local node memory is very expensive in terms of time. Storing packets in node memory not only consumes memory bandwidth, but also the network bandwidth of a node is reduced to a fraction of the memory bandwidth. However, wormhole routers do not store packets in memory. Memory is only accessed for injecting and delivering packets. As a consequence, channel bandwidth can be much higher than in packet-switched routers. Depending on the design of the network interface, channel bandwidth may even exceed memory bandwidth. This is the case for the iWarp chip, in which the processor directly accesses the network through special registers.

Even if the router is able to buffer full packets, the larger packet buffers are slower than flit buffers, increasing the flow control latency through the router and slowing down clock frequency. Furthermore, the use of hardware packet buffers implies a fixed packet size. Variable-sized messages must be partitioned into fixed-size packets. This increases message latency and percentage of network bandwidth devoted to overhead, e.g., processing and transmitting packet headers. The unit of flow control in VCT switching is also a packet. Thus, many of these design considerations are applicable to VCT switching. However, wormhole switching does not require messages to be split into packets. This is one of the reasons why VCT routers have not replaced wormhole routers.

We might expect that the mad postman switching technique may considerably increase performance over wormhole switching. However, mad postman switching can only improve performance if the default output channel selected at an intermediate router has a high probability of being the correct output channel. The highest probabilities occur in low-dimensional networks, e.g., 2-D meshes because messages turn only once. However, for a fixed pin-out on the router chips, low-dimensional networks allow the use of wider data channels. Consequently, a header can be transmitted across a physical channel in a single clock cycle, rendering finer-grained pipelining unnecessary and nullifying any advantage of using the mad postman switching.

In summary, we observe that wormhole switching owes its popularity in part to the fact that it performs flow control at the flit level, requiring only small flit buffers. Messages are not stored in memory when they block, but rather span multiple routers. However, the small buffers produce a short delay, and wormhole routers can be clocked at a very high frequency. The result is very high channel bandwidth, potentially higher than the bandwidth to local memory. Given the current state of the technology, we believe that the most promising approach to increase performance considerably with respect to current interconnection networks consists of defining new switching techniques that take advantage of communication locality, and optimize performance for groups of messages rather than individual messages. Similarly, we believe that the most effective way to offer an architectural support for collective communication, and for fault-tolerant communication, is by designing specific switching techniques. These issues will be explored in Chapters 5 and 6.

2.9 Commented references

Circuit switching has its origin in telephone networks. Packet switching has its origin in data networks for intercomputer communication. The first parallel machines were generally packet- or circuit-switched. The Intel iPSC/1 was packet-switched with message latencies on the order of milliseconds. The Direct Connect Module (DCM) introduced in the later-generation Intel iPSC/2 and iPSC/860 machines [257] employed circuit-switched communication with short messages being transmitted in a manner akin to wormhole switching. The GP 1000 from BBN employed a circuit-switched multistage network. The original Denelcor HEP [190], the MIT Tagged Token Dataflow Machine [12], and the Manchester Dynamic Dataflow Machine [144, 145], were all early machines that utilized a packet-switched interprocessor communication network.

Wormhole switching was introduced in the Torus Routing Chip [77, 78] and the performance for wormhole-switched multidimensional tori was examined in [71]. The latest in the line of machines from Intel, the Paragon [165], utilizes wormhole-switched communication. Other machines that utilize wormhole switching include the Cray T3D [258], the IBM Power Parallel SP series [326, 327], and the Meiko Computing Surface [22]. At the time of this writing, interconnection networks in current generation machines appear to be adopting wormhole switching as the mechanism of choice. While the introduction of VCT switching [172] predates wormhole switching, it is not yet in use in commercially available parallel architectures. The best known implementation of VCT is the Chaos router [188]. The use of virtual channel flow control was introduced in [73]. The Cray T3D [258] and the Cray T3E [312] utilize multiple virtual channels per physical channel.

Mad postman switching was introduced in [167] and has found its way into the implementation of low-cost asynchronous routers. However, with the increasing pin-out and channel width in modern routers, wormhole switching still appears to hold an advantage over the mad postman switching technique. More recently, pipelined circuit switching was proposed as a robust switching mechanism [127] and was subsequently realized in the Ariadne router [7]. Scouting switching [100] and the use of dynamically configurable switching techniques [80] was designed to improve the performance of pipelined circuit switching on message traffic that did not encounter any faulty channels.

A thorough study of router architectures and the development of a cost/performance model for router architectures can be found in [11, 57]. These models provide definitions of critical measures of router performance and enable assessment of the impact of virtual channels, channel signaling speed, message formats, etc. on the flow control latency and router delays.

EXERCISES

2.1 Modify the router model shown in Figure 2.1 to use input buffering only and no virtual channels. Rewrite the expressions for the base latency of wormhole switching and packet switching for this router model.

Solution Figure 2.26 illustrates an input-buffered version of Figure 2.1. In this case, in a single cycle a flit is routed through the switch across a physical channel, and into the input buffer of the next router. When a message arbitrates for the output of the router switch, it simultaneously acquires the output physical channel. In general, the duration of a cycle in an input-buffered switch will be longer than that of a switch that has both buffered inputs and buffered outputs. Similar observations can be made about output-buffered switches.

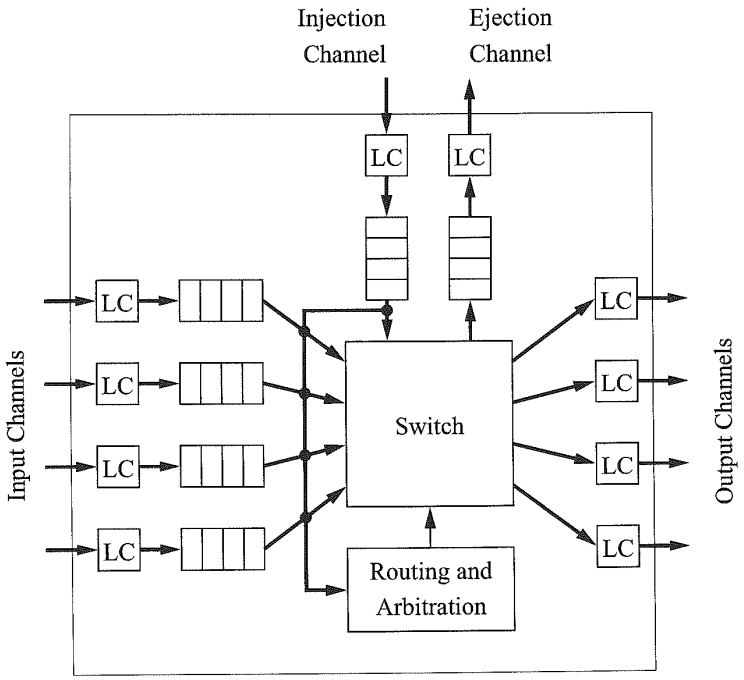


Figure 2.26. Architecture of an input-buffered router with no virtual channels. (LC = Link controller.)

The no-load latency for wormhole switching and packet switching can be rewritten as follows:

$$\begin{aligned} t_{wormhole} &= D(t_r + t_s + t_w) + (t_s + t_w)\left\lceil \frac{L}{W} \right\rceil \\ t_{packet} &= D\left\{t_r + (t_s + t_w)\left\lceil \frac{L+W}{W} \right\rceil\right\} \end{aligned} \tag{2.10}$$

2.2 Assume that the physical channel flow control protocol assigns bandwidth to virtual channels on a strict time-sliced basis rather than a demand-driven basis. Derive a expression for the base latency of a wormhole-switched message in the worst case as a function of the number of virtual channels. Assume that the routers are input-buffered.

Solution Assume that we have V virtual channels. Regardless of network traffic, each message will receive only $\frac{1}{V}$ of the physical channel bandwidth over every link. Therefore the ideal link speed seen by a message is Vt_w seconds. After routing a header, it takes a random number of cycles until the time slice is assigned to the selected virtual channel. In the worst case, it will take V cycles. Therefore the no-load latency becomes

$$t_{wormhole} = D[t_r + V(t_s + t_w)] + V(t_s + t_w)\left\lceil \frac{L}{W} \right\rceil \tag{2.11}$$

PROBLEMS

2.1 In wormhole switching, the last flit or tail flit causes the intermediate routers to release resources such as buffers or links. The time-space diagram used to describe wormhole switching adopts this view. Consider a modification to wormhole switching where the message path is not removed by the tail flit. Rather an acknowledgment is received from the destination, following which a tail flit is transmitted. Draw a time-space diagram of the transmission of a message over three links and write an expression for the number of cycles a link remains reserved for the transmission of this message. What percentage of this time is the link busy assuming no contention for any physical channel along the path? Assume that the routers are input-buffered only.

2.2 One of the advantages of virtual channels is the reduction of header blocking delay. However, this assumes that the physical channel bandwidth is allocated on a fair basis. Suppose we were to schedule virtual channels over the physical channel on a priority basis. Consider a design with two virtual channels. Show how priority scheduling of a low-priority message and a high-priority message over the same physical channel can result in deadlock.

2.3 Physical channel flow control synchronizes the transmission of phits across a channel. Assuming a synchronous channel, show how phits can be streamed across a channel, i.e., a sequence of K phits are transmitted before any acknowledgment is received from the receiver, so that we need only synchronize the transmission and receipt of K phits at a time. What are the requirements for buffer space on the receiver?

2.4 Consider a wormhole-switched network where virtual circuits persist until they are explicitly torn down by control signals or special messages. Draw a time-space diagram of the transmission of three messages over a path three links in length before it is removed by a special control flit injected into the path at the source node.

2.5 The IBM Power Parallel SP-2 represents a balanced design where the rate at which chunks can be transferred to switch memory is eight times the rate at which flits cross a physical channel. A similar observation can be made about the interface between switch memory and the output channels. The maximum message size is 8 chunks. Write an expression for the size of switch memory in flits in terms of the internal data path width (equal to one chunk), number of flits/chunk, and the number of input/output ports. Assume that one flit can be transmitted across the physical channel in one cycle. Validate this expression by instantiating with parameters from the SP-2.

2.6 The main advantage of packet switching over message switching is that several packets can be simultaneously in transit along the path from source to destination. Assuming that all the packets follow the same path, there is no need to add a sequence number to each packet. Draw a time-space diagram of the transmission of a message consisting of four packets over a path three links in length. Assuming that the routers are input-buffered only, compute the optimal number of packets that minimizes the base latency for the transmission of a message of L bits along D channels.

2.7 In the previous exercise, as all the packets follow the same path, assume that packet headers are stripped from all the packets but the first one. Assuming that the routers are input-buffered only, compute the optimal number of packets and the optimal packet size that minimizes the