UNIVERSITY OF CALIFORNIA

Los Angeles

Connection-Based Adaptive Routing Using Dynamic Virtual Circuits

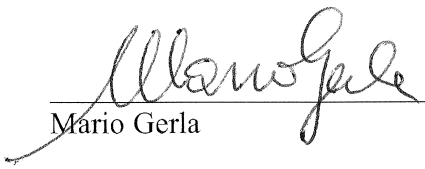
A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science

by

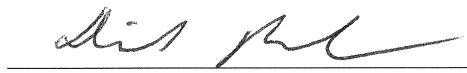
Yoshio F. Turner

2005

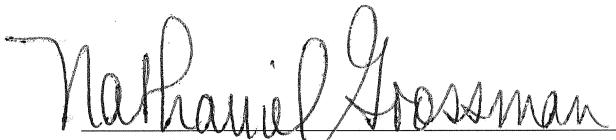
The dissertation of Yoshio F. Turner is approved.



Mario Gerla



David Rennels



Nathaniel Grossman

Yuval Tamir, Committee Chair

University of California, Los Angeles

2005

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

I am grateful to my thesis advisor, Professor Yuval Tamir, for his guidance and support. When I was a UCLA undergraduate, Professor Tamir introduced me to computer systems research and encouraged my interest in applying to graduate school. Through coursework and research discussions, he has shared his knowledge of and insight about computer systems. His guidance and feedback has helped to ensure and improve the quality of my research and its presentation. In short, he has been instrumental in helping me to build a solid foundation for my career.

I am grateful for useful technical discussions with many UCLA colleagues, especially G. (John) Janakiraman, Greg Frazier, Tiffany Frazier, Hsin-Chou Chi, and Professor David Rennels. I would like to thank Verra Morgan in the Computer Science Department Graduate Student Affairs Office for her consistent encouragement and outstanding administrative support,

I would also like to thank the Fannie and John Hertz Foundation, which awarded me a generous Hertz Fellowship. This fellowship provided the valuable opportunity to pursue my research interests with maximum freedom, unfettered by the need to consider funding sources.

Finally, I am grateful to my parents and my sister for their unconditional love and support, and for always encouraging me to pursue my goals.

VITA

October 19, 1966	Born in Los Angeles, CA
1988	B.S Computer Science and Engineering University of California, Los Angeles Los Angeles, CA
1988-1989	Chancellors Fellowship University of California, Los Angeles Los Angeles, CA
1989-1990	Teaching Assistant University of California, Los Angeles Los Angeles, CA
1990-1995	Graduate Fellowship Fannie and John Hertz Foundation
1991	M.S. Computer Science University of California, Los Angeles Los Angeles, California
1996-1998	Consultant Samsung Los Angeles Design Center Cypress, CA
1999-	Research Scientist Hewlett Packard Laboratories Palo Alto, CA

PUBLICATIONS AND PRESENTATIONS

J. Gummaraju and Y. Turner, ‘‘Hidra: History-Based Dynamic Resource Allocation for Server Clusters,’’ *First International Conference on Internet Technologies and Applications (ITA-05)*, Wrexham, UK (September 2005).

G. Janakiraman, J. R. Santos, D. Subhraveti, and Y. Turner, ‘‘Cruz: Application Transparent Distributed Checkpoint-Restart on Standard Operating Systems,’’ *International Conference on Dependable Systems and Networks (DSN)*, Yokohama, Japan (June-July 2005).

G. Janakiraman, J. R. Santos, and Y. Turner, ‘‘Automated System Design for Availability,’’ *International Conference on Dependable Systems and Networks (DSN)*, Florence, Italy (June-July 2004).

G. Janakiraman, J. R. Santos, and Y. Turner, ‘‘Automated Multi-Tier System Design for Service Availability,’’ *First Workshop on the Design of Self-Managing Systems (at DSN-2003)*, San Francisco, CA (June 2003).

A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel, ‘‘Diagnosing Performance Overheads in the Xen Virtual Machine Environment,’’ *First ACM/USENIX Conference on Virtual Execution Environments (VEE05)*, Chicago, IL (June 2005).

M. Mesarina and Y. Turner, ‘‘Reduced Energy Decoding of MPEG Streams,’’ *ACM/SPIE Multimedia Computing and Networking (MMCN)*, San Jose, CA (January 2002).

M. Mesarina and Y. Turner, ‘‘Reduced Energy Decoding of MPEG Streams,’’ *ACM/Springer Multimedia Systems Journal (special issue)* **9**(2) (August 2003).

J. R. Santos, Y. Turner, and G. Janakiraman, ‘‘End-to-End Congestion Control for InfiniBand,’’ *IEEE INFOCOM*, San Francisco, CA (April 2003).

J. R. Santos, K. Dasgupta, G. Janakiraman, and Y. Turner, ‘‘Understanding Service Demand for Adaptive Allocation of Distributed Resources,’’ *IEEE Global Internet Symposium*, Taipei, Taiwan (November 2002).

J. R. Santos, Y. Turner, and G. Janakiraman, ‘‘Evaluation of Congestion Detection Mechanisms for InfiniBand Switches,’’ *IEEE Globecom (High-Speed Networks Symposium)*, Taipei, Taiwan (November 2002).

Y. Tamir and Y. F. Turner, ‘‘High-Performance Adaptive Routing in Multicomputers Using Dynamic Virtual Circuits,’’ *6th Distributed Memory Computing*

Conference, Portland, OR, pp. 404-411 (April 1991).

Y. Tamir and Y. F. Turner, ‘‘High-Performance Adaptive Routing in Multicomputers Using Dynamic Virtual Circuits,’’ Computer Science Department Technical Report CSD-900026, University of California, Los Angeles, CA (September 1990).

Y. F. Turner and Y. Tamir, ‘‘Deadlock resolution in networks employing connection-based adaptive routing,’’ Computer Science Department Technical Report CSD-960032, University of California, Los Angeles, CA (August 1996).

Y. F. Turner and Y. Tamir, ‘‘Connection-Based Adaptive Routing Using Dynamic Virtual Circuits,’’ *International Conference on Parallel and Distributed Computing and Systems*, Las Vegas, NV, pp. 379-384 (October 1998).

Y. Turner, T. Brecht, G. Regnier, V. Saletore, G. Janakiraman, and B. Lynn, ‘‘Scalable Networking for Next-Generation Computing Platforms,’’ *3rd Annual Workshop on System Area Networks (SAN-3)*, Madrid, Spain (February 2004).

Y. Turner, J. R. Santos, and G. Janakiraman, ‘‘An Approach for Congestion Control in InfiniBand,’’ HP Laboratories Technical Report HPL-2001-277, Hewlett Packard Laboratories, Palo Alto, CA (October 2001).

ABSTRACT OF THE DISSERTATION

Connection-Based Adaptive Routing Using

Dynamic Virtual Circuits

by

Yoshio F. Turner

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2005

Professor Yuval Tamir, Chair

High-bandwidth, low-latency communication is the key to delivering high performance in scalable multicomputers and clusters, which consist of compute nodes that include communication switches interconnected via point-to-point links. With advancing semiconductor technology, the rate at which processors generate and consume traffic is increasing. Future designs must exploit the increasing number of transistors per chip to also achieve higher performance in the communication subsystem, matching the needs of faster CPUs.

Two techniques that can use additional hardware resources to improve network performance are adaptive routing and connection-based routing. Adaptive routing may change traffic routes to avoid congestion or faults. Connection-based routing can reduce packet processing overhead by reserving resources prior to communication. Most connection-based routing schemes, such as traditional virtual circuits, are static — once a connection is established, it remains fixed for its lifetime.

This thesis presents a new mechanism that combines the advantages of adaptive routing and connection-based routing for arbitrary topologies. The proposed *Dynamic Virtual Circuit* (DVC) mechanism allows existing connections to be quickly torn down in order to release resources needed for others or to be re-established along routes that are better for current network conditions.

The challenges for the DVC mechanism are to efficiently retain connection semantics despite rerouting, and to avoid or resolve protocol or packet buffer deadlocks. A previous approach for deadlock resolution is extended for use in DVC networks, and its performance limitations are identified. Subsequently, a competing deadlock avoidance technique is developed. It is based on unconstrained routing of DVCs combined with a deadlock-free virtual network. Simulations show that with DVCs, global routing optimization is possible and provides performance superior to fixed routing. The use of deadlock-free escape paths is sufficiently infrequent to preserve the bandwidth efficiencies of the DVC mechanism.

Connection-based adaptive routing may be beneficial for different networking environments. We evaluate the potential of such routing to reduce packet loss in large multi-path ATM or IP switches. With the class of traffic patterns considered, the results show that routing of the heaviest flows can lead to significantly lower cell loss than oblivious distribution.

Chapter One
Introduction

Advances in VLSI technology have enabled the development and widespread availability of high-performance microprocessors, high capacity memory chips, and single-chip switches supporting high-speed communication links. Microprocessor and memory components can be used to construct inexpensive and physically compact compute nodes which can be combined to form a *parallel system* in which multiple processors operate concurrently to perform parallel and distributed computations or transactions. Potential advantages of parallel systems include incremental system expandability, fault tolerance, and the ability to exploit concurrent execution to improve performance [Atha88, Noak93, Ande95, Stun95, Scot96b, Agar99, Duni05].

To realize performance advantages for a wide variety of applications, a parallel system must support high throughput, low latency communication among the compute nodes. At small system sizes, high performance communication can be provided by connecting each pair of nodes with a high speed communication link without intervening switches. For larger systems, this approach has prohibitive cost because the number of links would grow quadratically with the number of nodes. To achieve high performance at an acceptable cost, larger systems use a *scalable interconnection network* in which switches are used to route data between pairs of nodes that are not directly connected [Laud97, Gall97, Scot96a, Stun95, Stun94a].

Nodes communicate by exchanging *messages* across the network. With

packet switching, the most commonly used basic switching technique, messages are divided into fixed or bounded size *packets* which are individually handled and forwarded by switches along a path of links between the source node and the destination node.

A *multicomputer* is a type of parallel system in which several compute nodes (up to hundreds or thousands) are packaged together as a single system and interconnected with a scalable interconnection network. Each node has a processor and memory, and some nodes may have specialized functionality (e.g., disk, video). The nodes are interconnected via a network of switches with point-to-point links. If each switch is connected to a node, the network is called *direct* [Scot96a]. An example of a multicomputer that has a direct network with an arbitrary, irregular topology is shown in Figure 1.1. If nodes are connected only to a subset of the switches, the network is called *indirect*. An example of an indirect network is a multistage interconnection network in which switches are arranged in a series of connected stages, and nodes are connected only to the first and last stages of the network [Stun95].

A *cluster* is another type of parallel system in which several separate computers are interconnected via a high-performance local area network. The bandwidth capacities of local-area network communication links and the speed of switching logic are currently high enough to support a wide variety of distributed computing applications [Ande95, Part94, Petr02, Bode95].

The key to high performance for a multicomputer or high-end cluster interconnection network is to make efficient use of network resources such as link bandwidth and the buffers at each switch that are used to store packets. In

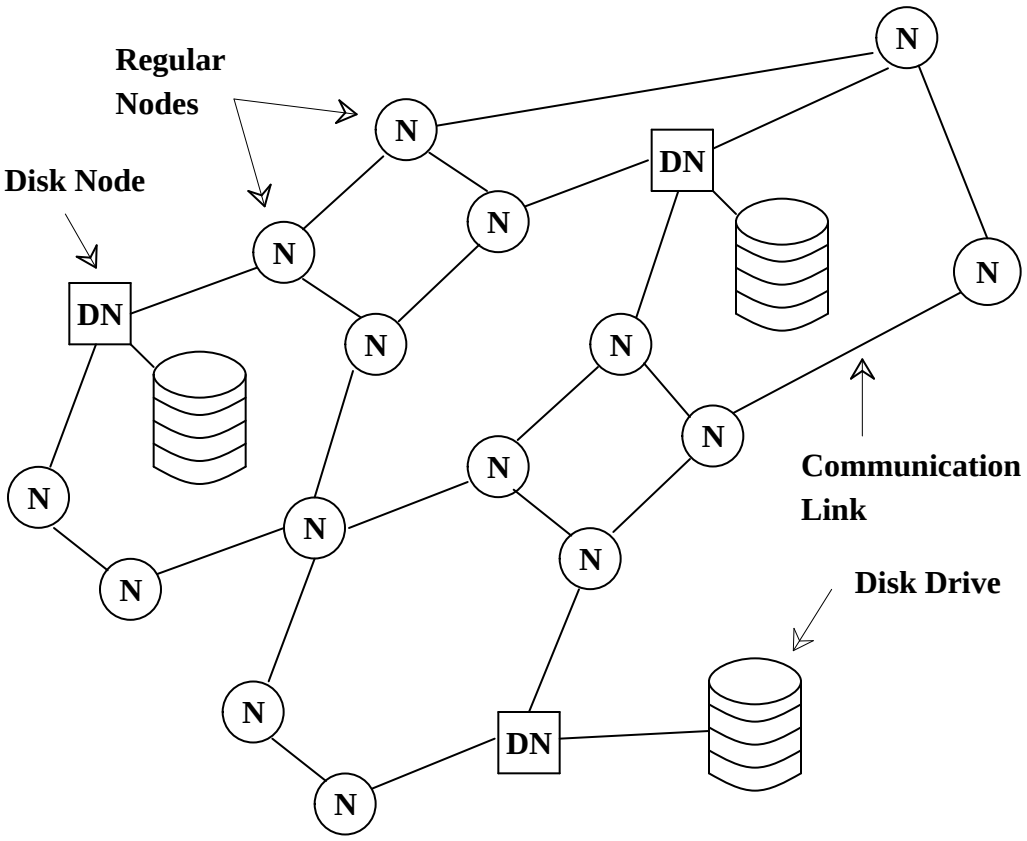


Figure 1.1: Multicomputer with arbitrary topology

particular, traffic load should be balanced across network resources to avoid overloading some resources while other resources which could perform useful work are under-utilized. High contention for resources increases packet queuing and communication latency, while under-utilization limits the network's throughput capacity. A second requirement for efficient use of network resources is to maximize the utilization of these resources for transferring *application* data. In particular, the goal should be to minimize the use of these resources to transfer or process packet addressing and sequencing information.

The *routing scheme* used in the network is the primary factor that determines

how efficiently network resources are used. The routing scheme determines the network path taken by each packet from its source node to its destination node. The routing scheme should direct packets through the lowest latency paths from source to destination. It should take into account the topology of the network and *adapt* to the current workload and resource availability to route packets around congested or faulty areas [Bold97, Dall93, Duat98, Glas94, Kim97, Taji77, Venk96]. By avoiding congested network resources, *adaptive routing* schemes have the potential to effectively balance load across the network resources.

A critical requirement for an adaptive routing scheme is to avoid *packet buffer deadlock*. Multicomputers and high-end clusters perform distributed computations that generate and consume packets at very high rates and require low message latency to avoid impeding performance by stalling applications. Applications would pay prohibitive performance penalties if the network were to discard packets which would subsequently have to be re-transmitted. Therefore, multicomputer and high-end cluster networks use backpressure flow control mechanisms in which packets that encounter congestion are blocked rather than discarded. As a result, unless sufficient constraints are imposed on packet routes and/or buffer usage, a sequence of blocked packets can form a *deadlock cycle* in which no packet in the cycle can advance forever. The routing scheme should therefore minimize routing constraints while preserving deadlock-freedom.

In addition to providing adaptive routing, the routing scheme should minimize the overhead for routing and forwarding packets. The addressing and control information that is sent with each packet should be minimized to enable efficient utilization of link bandwidth. In addition, the processing required to interpret and

route each packet at each hop should be minimized to provide low latency communication.

These overheads can be high with conventional *packet switching*, in which each packet is routed independently through the network. With packet switching, each packet must contain routing information (e.g., the packet's source and destination node identifiers) to enable switches to route the packet properly toward the destination. At each hop, a switch delays each arriving packet while processing the routing information in the packet header to determine which output port to use to forward the packet. In addition, if packets injected into the network by a node can take different paths to reach a common destination, then ordering information (e.g., a sequence number) must be transmitted with each packet to enable the data to be delivered in-order to the application at the destination node. For workloads where most packets have small data payloads, routing and sequencing information can consume a high fraction of the network link bandwidth.

Bandwidth and processing overheads can be minimized by using *connection-based routing* [Chow88, Bork90, De 96, Dao97, Hsu92, Tyme81], in which resources are reserved in advance of communication at each switch along the path from source node to destination node. With this approach, addressing and routing information for a connection is recorded at the switch at connection setup time. As a result, the only information a switch needs in order to route an arriving packet is an identification of the connection used by the packet. Thus connection-based routing reduces the processing required for routing each packet compared to packet switching, and enables reducing the amount of addressing information that must be transmitted with each packet. Finally, since all packets that use a connection take

the same path from source to destination, with connection-based routing packet headers do not require sequencing information for packet reordering at the destination.

Connection-based routing reduces the overhead for forwarding each packet but introduces connection setup/teardown operations. Thus this approach can improve overall network efficiency if these new operations are fast or occur infrequently, e.g., for workloads in which each connection is used by several packets [Hsu92].

In this dissertation, we present new communication techniques that combine the benefits of connection-based routing and adaptive routing. In the following sections, we outline the main contributions of the dissertation. Section 1.1 gives a brief overview of the primary contribution, a routing scheme called *Dynamic Virtual Circuits*, or *DVCs*, which provides fast mechanisms for dynamically changing the routes of established connections while avoiding packet buffer deadlock. This enables each connection to be routed or rerouted onto a low latency path through the network from source to destination to provide high performance. While most of our design and evaluation of the DVC scheme is in the context of interconnection networks for multicomputers or clusters, we have also investigated the potential benefits from using a similar approach in the context of switching fabrics used to implement network switches. Specifically, we present an evaluation of the potential performance benefits of explicitly routing selected flows through IP or ATM multistage network switches. Section 1.2 outlines our evaluation of different configurations for providing alternate paths through the network switch, and our evaluation of the benefits of different techniques for

routing traffic through the switch. Our evaluation shows that routing some flows on selected paths has the potential to significantly improve switch performance by reducing contention between different flows.

1.1. Dynamic Virtual Circuits

The primary contribution of this dissertation is the design and evaluation of a novel packet routing mechanism called *Dynamic Virtual Circuits*, or DVCs [Tami91, Turn98, Turn05]. DVCs add adaptive routing capabilities to an existing connection-based routing technique called *virtual circuit switching* [Bork90, Dao97, Hsu92, Tyme81]. Virtual circuit switching provides end-to-end connections called *virtual circuits*. The network's physical resources (packet buffers, link bandwidth, etc.) are multiplexed among active virtual circuits. Once a virtual circuit is established, packets can be sent without the addressing and sequencing information needed in pure packet switched networks. Packets are quickly forwarded along the virtual circuit's network path by using a single lookup in a small virtual channel table at each hop. This mechanism enables low overhead packet forwarding even in networks with irregular topologies [Flin00]. In these topologies, intermediate nodes between a source and destination cannot use simple algorithmic routing. Instead, routing is based on large tables at each node which are constructed at system initialization and may be changed over time to adapt to changing traffic conditions. With virtual circuits, the routing tables can be used during virtual circuit establishment to determine a virtual circuit's network path. Subsequently, packets are forwarded along the path using the more efficient virtual circuit forwarding mechanism.

Most virtual circuit schemes have the limitation that each established circuit's route cannot be changed until the connection is no longer needed and the circuit is torn down. This prevents adaptation to changes in traffic patterns, and it prevents establishment of new virtual circuits once all the required resources are assigned to existing circuits.

To solve this problem we have proposed the *Dynamic Virtual Circuits* (DVC) mechanism [Tami91, Turn98] which combines adaptive routing and connection-based routing. With DVCs, a portion of a virtual circuit can be torn down from an intermediate node on the circuit's path and later be re-established, possibly along a different route, while maintaining the virtual circuit semantics of reliable in-order packet delivery. The DVC mechanism provides fast circuit establishment and re-routing by using only local operations at each node. This enables packets to proceed on new circuit paths without waiting for end-to-end handshakes.

In this dissertation, we present the DVC algorithm and propose a hardware/firmware architecture for its implementation. The challenge in devising the algorithm is to simultaneously provide fully adaptive routing, deadlock-freedom, and the low per-hop overhead of static virtual circuit switching. Compared to pure packet switching networks, DVC networks pose a more complicated deadlock avoidance problem because DVCs introduce dependencies among circuit establishment and teardown operations and also dependencies between these operations and packet buffer resources.

We present two contrasting approaches to eliminate packet buffer deadlocks in DVC networks. The first approach is based on deadlock detection and resolution in which deadlocks may occur in the network and then later be detected

and resolved. The attraction of this approach is that it minimizes the restrictions that are placed on the use of buffer resources since no resources must be reserved to prevent deadlocks. Through detailed experimental evaluation using simulation, we find that this approach works well in scenarios where deadlock conditions are infrequent. However, performance can suffer in scenarios where deadlock events have high probability because it may not be possible to remove packets from deadlock cycles as rapidly as new packets arrive to refill buffers in the cycles.

We present an alternative deadlock avoidance approach for DVC networks which completely eliminates the possibility of deadlock. The main challenge in devising this scheme is to retain the total freedom to establish virtual circuits on any network path, while eliminating the possibility of deadlocks involving packet routing buffer dependencies or the additional dependencies introduced by circuit manipulation operations. The approach we take is to avoid packet buffer deadlock by providing a dependency cycle-free virtual network for use as an escape path for any packets that encounter dependency cycles. In addition, we provide a second virtual network which decouples packet routing dependencies from the dependencies introduced by circuit manipulation operations. This enables us to avoid complex deadlock cycles that involve both types of dependencies.

In addition to presenting the algorithm used to realize the DVC mechanism with deadlock avoidance, we describe and evaluate the hardware resources and mechanisms needed to implement this algorithm. Our analysis shows that the scheme is simple to implement, with only modest hardware requirements. We also present a correctness argument, based on an analysis of the system's state space, to prove that each packet injected into the network using a DVC is delivered to the

DVC destination in order. We evaluate the performance of adaptive routing with DVCs using a detailed simulation model and show that DVCs have the potential to enable higher performance compared to non-adaptive routing. Compared to the deadlock detection and resolution scheme, our deadlock avoidance scheme imposes more restrictions on buffer usage but provides much better performance at high load levels since deadlock never occurs.

1.2. Reducing Packet Discarding in ATM and IP Switches

The DVC mechanism provides high performance by enabling each connection to be routed onto a low latency network path from source to destination. To investigate the potential value of using similar approaches in the context of switching fabrics used to implement network switches, we have performed a limit study evaluating the potential performance benefits of explicitly routing selected flows through IP or ATM multistage network switches.

Modern network switches enable the creation of high-speed multiservice networks that simultaneously support real-time (deadline sensitive) and best-effort traffic [Turn86, Part94]. A large-scale network switch uses a multistage switching fabric to direct packets, or cells, from the switch inputs to the switch outputs. By adding extra stages to the switching fabric, alternate paths can be provided between each input and output. These alternate paths can be exploited to reduce traffic contention that can lead to packet discarding.

Our focus is on the effective management of a low-cost subset of multi-path switches, those with only a single extra stage for providing alternate paths. The performance of such a switch is determined by the degree of contention of different

flows for switch resources. The potential for contention between traffic flows depends on the precise set of alternate paths through the switch, which is determined by the manner in which the extra stage is attached to the remaining stages. Therefore, we evaluate the performance that results with different configurations for the connection of the extra stage. Specifically, we compare a standard butterfly configuration with an alternative configuration that minimizes worst case contention. Our results show that for some traffic patterns the alternative interconnection results in a slightly higher probability of meeting performance (cell loss rate) requirements. To enable effective exploitation of the alternate paths that are provided by the extra stage, we also investigate traffic management policies that control the routes that cells use to traverse the switch. We compare policies that, for select flows, may explicitly route each flow along one of the alternate paths it can use to a policy that distributes all traffic evenly among all available paths. Our results show that using the former policies has the potential of resulting in significantly improved performance compared to the latter policy.

1.3. Organization

The rest of this dissertation is organized as follows. Chapter 2 summarizes related work in three areas: routing algorithms for multicomputer and cluster networks, connection-based routing techniques, and techniques for reducing packet discarding in large-scale switches used for general-purpose networks. Chapter 3 presents an overview of our proposed DVC technique. Chapter 4 describes a deadlock detection and correction technique for use with DVCs and evaluates the

performance behavior of this approach. Chapter 5 describes a deadlock avoidance technique for use with DVCs which overcomes the performance limitations identified for the deadlock detection and resolution approach. Chapter 6 presents an evaluation of the potential for using adaptive routing and connection-based routing in the context of large-scale switches for general-purpose networks which can discard packets in congestion scenarios. Finally, we present conclusions and suggest directions for future work in Chapter 7.

Chapter Two
Background and Related Work

This chapter provides necessary background information and surveys previous work that is related to this dissertation. Our proposed Dynamic Virtual Circuits mechanism combines the benefits of adaptive routing and connection-based routing for multicomputer and cluster interconnection networks. The techniques we use to provide deadlock-free adaptive routing in DVC networks build on previously proposed routing algorithms. in Section 2.1 we survey a broad range of these previously proposed routing algorithms for packet switching multicomputer and high-end cluster networks. The routing algorithms that we describe vary in degree of adaptivity, the approaches taken to eliminate packet buffer deadlocks, and the network topologies for which they can be applied. In Section 2.2, we present overviews of example switch implementations to illustrate how these routing algorithms and deadlock elimination techniques have been used in practice. In Section 2.3, we describe connection-based routing techniques including circuit switching and traditional static virtual circuits. In addition, Section 2.3 surveys previous proposals that had similar goals to those of the DVC mechanism — combining the advantages of adaptive routing and connection-based routing. We describe the advantages of the DVC mechanism compared to these prior approaches.

In Chapter 6, we propose and evaluate techniques for exploiting alternate network paths and adaptive routing to improve performance in a different networking environment, the design and management of large-scale ATM and IP

switches which discard packets when packet buffers overflow. In Section 2.4, we review previous techniques for reducing the incidence of packet discarding in these large-scale switches.

2.1. Routing in Packet Switched Multicomputer Networks

In this section we describe representative routing algorithms for multicomputer and high-end cluster interconnection networks. We focus on networks that use *packet switching* in which data transmitted between a source and destination are partitioned into packets which are individually routed through the network as directed by the routing algorithm. At each switch, the routing algorithm determines for each arriving packet the set of output ports which are appropriate to forward the packet.

2.1.1. Overview of Routing Algorithm Design Issues

The primary goals of routing algorithm design are to maximize the network's capacity in terms of throughput and to minimize packet latency at each level of throughput, and to achieve these goals for a wide variety of traffic patterns. An additional goal can be to support Quality of Service (QoS) requirements, which for example may reserve link bandwidth and buffer space for packets that support high-priority applications or system management functions. Finally, for multicomputers and high-end cluster networks, the switch must be able to execute the routing algorithm for each arriving packet quickly, on the order of a few clock cycles, to ensure the network retains the low latency packet forwarding provided by high speed switching hardware.

The most distinguishing feature of a routing algorithm is its degree of adaptivity. With *fixed-path* routing, also sometimes called *oblivious*, *deterministic*, or *non-adaptive* routing, the path a packet takes through the network depends only on the addresses of the packet's source and destination nodes. A fixed-path routing algorithm can be described by a *routing function* which takes as input a packet's source and destination IDs and current location in the network and returns the unique output port that the packet must use for its next hop. Although fixed-path routing is simple, it can give rise to network regions with high traffic congestion which limits performance.

With *adaptive* routing algorithms, a packet can take any of multiple paths. In a packet switching network, at each switch a packet visits, the routing algorithm can use information in the packet along with the current state of the switch to determine the appropriate next hop for the packet. The packet's header typically includes addressing information such as the destination node ID and the source node ID. The switch state includes the contents of its packet buffers and the status of its output links, and may additionally include other state such as the contents of routing tables or information relayed from other switches about current or recent traffic congestion in the network. The *routing function* that describes an adaptive routing algorithm returns the set of output ports a packet can choose from for the next hop. An additional *selector function* can be defined which describes the selection of exactly one of these output ports which will be used to forward a particular packet. By allowing packets to select from among multiple output ports, adaptive routing algorithms can enable reduced network congestion by routing packets around congested regions, thereby balancing the load on various network

resources and reducing packet latency compared to using fixed-path routing.

Both fixed-path and adaptive routing must address the problem of *packet buffer deadlock*, a situation that arises when there is a set of resources (e.g., packet buffers) in which each resource is held by a client (e.g., a packet) that cannot release the resource until the client first acquires another resource in the set. Since each client is blocked waiting for a resource to become unblocked, no resource can be released, and the clients never progress. This problem can spread throughout the network until all clients in the network join the deadlock and cannot make further progress.

Figure 2.1 shows an example of a deadlock in which packets have filled the buffers that are shown. Each packet waits for the next buffer in the cycle to become free before it can be forwarded. A *resource dependency* exists from a first buffer to a second buffer if a packet in the first buffer can be blocked waiting for the second buffer to become free. In general, every deadlock contains a subset of resources whose resource dependencies form a cycle.

To eliminate deadlocks, there are two basic approaches: deadlock avoidance, and deadlock detection and resolution. Deadlock avoidance is based on restricting the use of resources. For example, packet routing can be restricted such that there are no resource dependency cycles in the network [Dall87]. As we describe later in Section 2.1.4.2, these restrictions can be relaxed to allow dependency cycles so long as packets that become caught in such cycles have an alternative ‘‘escape path’’ that they can use to leave the cycle [Duat96]. The alternative to deadlock avoidance is deadlock detection and resolution. As a simple example, some networks resolve deadlocks by simply discarding packets that experience long

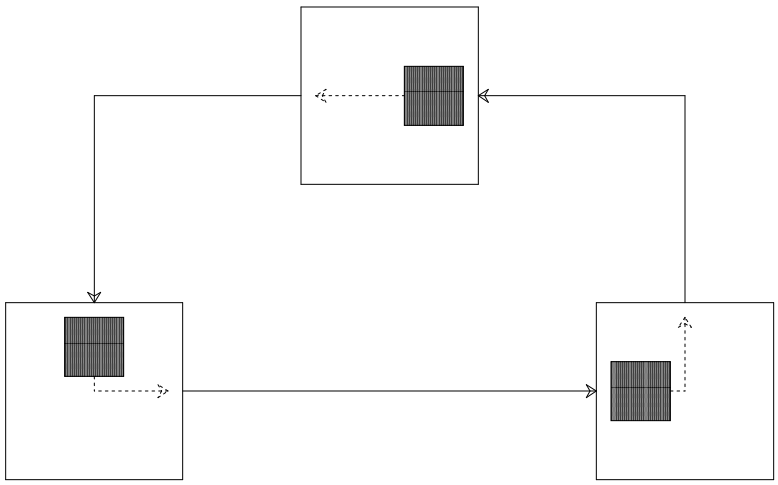


Figure 2.1: Deadlock involving three switches. Each switch has a packet buffer which is full of packets that are blocked waiting for free space in the packet buffer at the next switch.

delays which may arise either from deadlock or from traffic congestion [Metc76, Chow88, Kim97]. As we describe later in Section 2.1.5.1, for multicomputer and high-end cluster networks which do not discard packets, deadlock resolution requires temporarily adding reserved resources to the network to enable packets that are stuck in deadlocks to advance [Jaff89].

In general, other problems besides deadlock must be addressed in the design of routing algorithms and switches. *Livelock* occurs if a packet continues to move about in the network (i.e., it does not enter a deadlock) but never reaches its destination. For example, livelock can occur if a packet is repeatedly discarded from the network and re-transmitted, never reaching its destination. As another example, if the path a packet takes through the network contains a loop that does not include the destination node, the packet may never exit the loop. In this case livelock can be avoided by ensuring that routes are loop-free or that any routing

loops are only followed temporarily. *Starvation* occurs if a packet becomes blocked forever at a switch or at a host interface because the packet scheduling policy always favors other packets for advancement. Even though the packet is blocked forever, this is not a deadlock situation because there is no cycle of blocked resources. Starvation can be avoided by using a starvation-free packet scheduling policy, for example by giving higher priority to older packets.

In networks with regular topologies (e.g., trees, meshes, or ring networks), a switch can determine which output ports to use to forward a packet by performing a simple arithmetic calculation based on the coordinates of the current switch and the packet's destination node. In contrast, for routing in networks with arbitrary topologies, routing algorithms cannot use simple arithmetic calculations to route packets. Instead, for these networks routing tables are provided at each network node to store the routing function. The contents of the routing tables may be computed continuously using a distributed algorithm that monitors the costs of traversing each link of the network and sends cost updates to neighboring nodes [Taji77, Bert87]. To prevent inconsistent table entries as the distributed algorithm proceeds, some algorithms require each switch to periodically broadcast cost information about its adjoining links to every other switch (this is called a *link-state* approach) [McQu80]. Each switch uses the broadcast information from other switches to construct an estimate of the state of the entire network topology. Since all switches receive the same link state data from the broadcasts, the switches use the same data to calculate a consistent routing function that, for example, avoids routing loops. Alternative techniques called *distance vector* algorithms require switches to diffuse link state information through the network by sending

state information only to immediate neighbor switches. Compared to link state approaches, distance vector approaches are more susceptible to the problem of inconsistent table entries at different switches, which can lead to routing loops and potential livelock. However, distance vector algorithms do not require the global broadcasts required for link state approaches and can be devised to minimize undesirable looping [Shin87]. For very large networks, it may be desirable to employ hierarchical network routing to keep the size of the routing tables from growing too large [Awer89].

The rest of this section is organized as follows. Section 2.1.2 describes the pipelined ‘‘cut-through’’ packet forwarding mechanisms used in modern switches to enable very low end-to-end packet latency compared to naive store-and-forward packet switching. In order to take full advantage of these fast forwarding mechanisms, each switch must execute the routing algorithm for each arriving packet quickly, on the order of a few clock cycles. Next, Section 2.1.3 describes representative fixed-path routing techniques that have been proposed for early multicomputers. These techniques are designed to be free of packet buffer deadlocks and to require only simple logic for implementation. Section 2.1.4 describes several adaptive routing techniques which also avoid deadlocks, and Section 2.1.5 describes generalized results in the theory of packet buffer deadlocks in interconnection networks. Two contrasting general approaches for eliminating packet buffer deadlocks are outlined: deadlock detection and correction, and deadlock avoidance. Finally, Section 2.2 presents examples of real implementations of switches for multicomputer interconnection networks.

2.1.2. Cut-Through Switching

For parallel systems, the methods used at each switch to interpret, route, and forward each packet must be fast in order to minimize the end-to-end latency of packets sent through the network as well as to maximize throughput. In the simplest form of packet switching, called store-and-forward routing, each packet that arrives at a switch must be stored completely at the switch before being forwarded to the next switch. This results in high end-to-end packet latencies due to queuing delays at each hop. If each link operates at rate **r**, a packet of length **L** which traverses **n** hops through the network has a minimum end-to-end delay of **n*L/r**. Thus with store-and-forward routing, end-to-end latencies increase proportionally with network diameter.

Current switches avoid such high latencies by using forwarding techniques that transmit packets through the network in a pipelined fashion in which a switch can begin forwarding an arriving packet to the next switch before the entire body of the packet arrives to the switch. At each hop, a very short delay **t** is incurred for routing and forwarding the initial portion (of size **b** < **L**) of the packet, leading to a delay of **n*t** for the initial portion of the packet to reach the destination. The remaining portion of the packet (length **L**–**b**) arrives at rate **r**, leading to a total end-to-end latency of **n*t**+**(L**–**b)/r**. For long packets, this delay increases slowly with network diameter since **t** is small. Two variations of this basic technique are *virtual cut-through* [Kerm79] and *wormhole routing* [Dall87].

With wormhole routing, the contents of each message are handled as a series of contiguous data blocks called flow control units, or *flits*. A flit is composed of a fixed number of physical transmission units, or *phits*. A switch may not begin

forwarding a message until it has first reserved at the next switch a free buffer that can store at least one flit. Once the buffer is reserved, each flit of the message may be transmitted to the buffer whenever it has free space for the flit. Using this flow control policy, a message can be spread across multiple switches with different flits of the message stored at buffers at each switch, forming a worm-like path through the network. If the flit at the front of the message encounters congestion, this flit blocks and can result in all the subsequent flits of the message also being blocked at multiple switches along the path. This behavior is the main disadvantage of wormhole routing since a message which is blocked across multiple switches consumes the buffers and potentially also consumes link bandwidth for extended periods while the message remains blocked. Holding these scarce resources for extended periods prevents other messages from making progress and reduces overall performance.

With virtual cut-through, messages are partitioned into packets with bounded length. As with wormhole routing, when the initial portion of a packet arrives at a switch input port, the switch examines the packet header, determines which output to use, and if the output is free the switch can begin forwarding the packet immediately. Unlike wormhole routing, with virtual cut-through a packet that blocks in the network only consumes resources at a single switch, similar to store-and-forward routing. This is accomplished using a flow control policy which ensures that two conditions are satisfied before a switch is allowed to begin forwarding a packet to the next switch: 1) the output link is idle, and 2) the next switch is able to receive the *entire packet* without blocking. The second condition is satisfied if the next switch has enough free buffer space to accommodate the

packet, or if the next switch is currently transmitting a packet and releasing buffer space at a rate fast enough to guarantee that an arriving packet will not overflow the buffer. If the head of the packet encounters congestion in the network and becomes blocked, the second flow control condition described above guarantees that all the subsequent phits of the packet can accumulate at the buffer at the head of the path instead of being blocked at multiple switches as occurs with wormhole routing. Thus with virtual cut-through, a blocked packet only consumes a buffer at a single switch, whereas resources that span multiple switches are held by the packet only long enough to forward the packet at full link speed.

There are a number of additional trade-offs between virtual cut-through and wormhole routing. With virtual cut-through, each buffer must have capacity for at least one complete packet. With wormhole routing, even buffers with capacity of only one or two flits suffice, although performance may be improved by providing larger buffers. The smaller buffer requirements of wormhole routing are becoming unimportant as improvements in fabrication technology continue to reduce transistor size, lowering the chip area cost for deep buffers. A second possible advantage of wormhole routing is that messages need not be chopped into packets as with virtual cut-through, although breaking long messages into smaller packets can improve performance by allowing messages to multiplex the use of buffers and link bandwidth at the granularity of packets.

2.1.3. Fixed-Path Routing

With fixed-path routing, the hardware logic required for a switch to determine where to forward a packet can be very simple and fast, at least for networks with regular topologies. However, the use of fixed-path routing algorithms may lead to poor performance for some traffic patterns and does not tolerate node or link failures which break some of the fixed paths.

Fixed-path routing algorithms that avoid deadlock have been devised for regular topologies commonly used in multicomputers such as the **k**-ary **n**-mesh and the **k**-ary **n**-cube. In those topologies, each node/switch can be labeled with an ID consisting of **n** radix-**k** digits $\mathbf{S_0 S_1 \cdots S_{n-1}}$ where each $\mathbf{S_i}$ is a value in $\mathbf{0, 1, \cdots, k-1}$. The ID gives the coordinates of the node/switch in an imaginary **n**-dimensional space. In the **k**-ary **n**-mesh, each switch has links to all the switches whose IDs differ in exactly one dimension by plus or minus one. In the **k**-ary **n**-cube, there are additionally links between the switches at coordinates **0** and **k–1** in each dimension, thus forming rings in each dimension. The **k**-ary **n**-cube is also called a **n**-dimensional *torus* network.

A special case is when **k**=**2**. The **2**-ary **n**-mesh and **2**-ary **n**-cube are identical and are also called *binary* **n**-cube, or *hypercube*. Each switch ID in the hypercube can be represented by a vector of **n** radix-2 digits (i.e., **n** bits). The dimensions that must be traversed along any *minimal* (i.e., shortest-hop) path from a source node to a destination node are determined by taking the bitwise XOR of the two bit vectors used as their node IDs. The positions of the ‘‘1’’ bits in the result identify the dimensions that must be traversed, and the number of ‘‘1’’ bits is the number of hops in the minimal paths.

In **k**-ary **n**-meshes in which each switch has a separate packet buffer for packets that arrive on each dimension, deadlock-free fixed-path routing can be achieved by ordering the dimensions and restricting packets to minimal paths that traverse the dimensions in order. This approach, called *Dimension Order Routing* (DOR), and sometimes called *e-cube* routing in the hypercube [Sull77, Lang82], eliminates any dependency cycles that include buffers belonging to different dimensions. Since physical cycles do not exist within a single dimension in the **k**-ary **n**-mesh, there is no way a dependency cycle can form within a dimension. Thus with no dependency cycles within a dimension and no dependency cycles between dimensions, DOR in the **k**-ary **n**-mesh is deadlock-free [Dall87].

In contrast to the **k**-ary **n**-mesh, the **k**-ary **n**-cube with $\mathbf{k \neq 2}$ has a physical cycle within each of the **n** dimensions. Buffer dependency cycles within a dimension can be removed by introducing a second buffer at each switch input port that is used exclusively by any packets from the point they traverse an ‘‘end-around link’’ (i.e., the link between coordinate positions **0** and $\mathbf{k-1}$) in a dimension until they reach the destination or change dimensions. Using this second buffer, which has its own flow control, eliminates the cycle of buffer dependencies within each dimension.

In general, fixed-path deadlock-free routing for any topology can be devised using the methodology presented in [Dall87]. In this methodology, each link is conceptually divided into multiple *buffering virtual channels*, or *BVCs*. Note that BVCs are usually called ‘‘virtual channels’’, but we use the term BVC since the term ‘‘virtual channel’’ is also commonly used to express a completely different concept (which has to do with virtual circuits, as we introduce later in

Section 2.3.2). A BVC is a packet buffer for which the switch provides separate flow control. The routing algorithm determines which link and BVC a packet will use on each hop. A buffer dependency graph for a routing algorithm is defined as a graph in which the vertices are BVCs and there is an edge from BVC $\mathbf{c_i}$ to BVC $\mathbf{c_j}$ if and only if the routing algorithm can direct a packet which is stored in $\mathbf{c_i}$ to be forwarded in one hop to $\mathbf{c_j}$. For fixed-path routing, this buffer dependency graph must be acyclic to guarantee deadlock-freedom. It can be shown that the buffer dependency graph is acyclic if and only if the routing algorithm assigns a partial ordering to the BVCs and requires packets to use the BVCs according to that ordering.

2.1.4. Adaptive Routing

Fixed-path routing cannot adapt to changing traffic patterns and cannot tolerate faults in network switches or links which break some of the fixed paths and disconnect some sources and destinations. In contrast, *adaptive routing* algorithms can adapt to changing traffic patterns, and some adaptive routing algorithms can route packets around failed components, ensuring no loss of connectivity.

Adaptive routing algorithms can be classified as *fully adaptive* or *partially adaptive*. Fully adaptive routing algorithms allow all *minimal* paths (i.e., all shortest hop paths) to be used between each source and destination node, whereas partially adaptive algorithms allow only a strict subset of the minimal paths to be used. Some routing algorithms also allow *non-minimal* paths that are longer than shortest hop paths. Routing algorithms that tolerate the loss of any single link or switch usually must be non-minimal, since some source-destination pairs are

connected by only a single minimal path. With non-minimal routing, packets can travel away from their destinations as well as closer to them. This may introduce the possibility of livelock, in which some packet forever loops in the network without reaching its destination. Thus non-minimal routing algorithms should be free of both deadlocks and livelocks.

To avoid deadlock, adaptive routing algorithms can take the same approach as used with fixed-path routing algorithms in which the network's buffer dependency graph is constrained to be acyclic. We describe schemes that take this approach in Section 2.1.4.1, and alternative approaches that provide greater routing flexibility by allowing dependency cycles in Section 2.1.4.2.

2.1.4.1. Adaptive Routing without Buffer Dependency Cycles

Most early attempts to provide fully-adaptive deadlock-free routing were based on ensuring acyclic buffer dependency graphs, the same basic approach which is required for deadlock-free fixed-path routing. Some of these early approaches took straightforward approaches which amounted to allocating a BVC at each link for each distinct network path that all packets could take through the link [Hilb89, Lind91]. The resulting solutions are extremely hardware-intensive as the number of BVCs grows exponentially with the network size.

One way to reduce the hardware costs is to give up fully adaptive routing in favor of providing only partially adaptive routing. For example, Konstantinidou proposed a partially-adaptive routing algorithm for the hypercube [Kons90a]. In this algorithm, the only paths that can be taken are minimal paths that consist of two series of hops. In the first series of hops, links are traversed for all the

dimensions in which the source coordinate is zero and the destination coordinate is one. These zero-to-one transitions may be taken in any order. In the second series of hops, links are traversed for all the dimensions in which the source coordinate is one and the destination coordinate is zero. These one-to-zero transitions may be taken in any order. At the end of these series of hops, the packet has reached its destination. The scheme is adaptive since the dimensions are not required to be traversed in a fixed order, but it is not fully-adaptive since no one-to-zero transitions are allowed to be taken before any zero-to-one transitions. Deadlock-freedom is preserved by using two buffers at each switch, one buffer for packets that are taking the first series of hops, and a second buffer for packets that are taking the second series of hops. Packets in the first buffer wait only for buffers at switches with more ‘‘1’’s in their addresses or for the second buffer at the same switch. Packets in the second buffer wait only for buffers with fewer ‘‘1’’s in their switch addresses. As a result, there are no cycles in the buffer dependency graph, and the network is deadlock-free. To avoid the bottleneck this approach incurs near switches with many ‘‘1’’s in their address, an additional buffer can be added to each switch for use by a subset of the packets which take the two series of hops described above in the opposite order.

A similar approach called *planar-adaptive* routing has been proposed for non-minimal partially-adaptive fault-tolerant routing for k-ary n-meshes and k-ary n-cubes [Chie92, Chie95]. In this approach, each 2-D plane (or m-D sub-mesh) is provided enough BVCs to enable dependency cycle-free fully-adaptive routing within that plane (or sub-mesh). Packets are restricted to traverse the planes (or sub-meshes) in a particular order, thus ensuring there are no buffer dependency

cycles that span different planes (or sub-meshes), and therefore the entire network is deadlock-free. For fault-tolerance, a mechanism is provided for detecting faulty switches or links and for deactivating neighboring healthy switches/links in order to form convex regions of faults in the topology. After the formation of convex fault regions, packets that reach these regions switch to a new routing mode in which the packets take hops along the periphery of the fault region until reaching a switch from which they can continue to their destination node using the normal routing mode.

The Turn Model [Glas92, Glas94] is a different approach to deadlock-free adaptive routing that does not rely on multiple BVCs per link and which, depending on the topology, may or may not allow adaptive, fully-adaptive or non-minimal routing. In this approach, which works with topologies that have multiple dimensions (e.g., **k**-ary **n**-mesh), the ‘‘turns’’ of the network are analyzed, where a turn is a possible change of dimensions or directions within a network path. For example, in a 2-D mesh, a path can change from the X dimension to the Y dimension, and can do so going in various directions within each dimension (the full set of possible turns for the 2-D mesh is +X+Y, +X-Y, -X-Y, -X+Y). A sufficient condition for deadlock-freedom is that the paths taken by packets in the network use a set of turns which cannot be arranged to form a cycle. In the Turn Model methodology for designing routing algorithms, all possible turns are analyzed, and the minimum number of turns is prohibited such that the remaining turns cannot form a cycle. The remaining turns determine the routing algorithm and its adaptivity. Often the resulting algorithms allow non-minimal paths to be taken, which can be used to avoid faulty components or congestion.

2.1.4.2. Adaptive Routing with Buffer Dependency Cycles

The routing algorithms described in the previous section rely on the fact that an acyclic buffer dependency graph is a *sufficient* condition for deadlock avoidance. However, acyclic buffer dependency graphs are not *necessary* for avoiding deadlock with adaptive routing.

For example, in the Compressionless Routing scheme, packets can take any routes which may incur buffer dependency cycles, but any packets that block in the network are removed from the network and re-transmitted from the source node [Kim97]. Fault-tolerant and deadlock-free routing are provided by adding to the tails of messages sufficient padding flits to ensure that the head of the message arrives at the destination before the last padding flit is transmitted by the source. In this way, a packet that becomes blocked at a faulty switch or enters a deadlock is able to use the backpressure flow control signaling mechanism of wormhole routing to signal to the source node that there is a problem. The signaling causes the packet to be removed from the network and for the source node to retransmit the message. Any minimal path may be chosen for the packet to traverse. The number of required padding flits is determined by the number of pipeline stages (transmission stages plus buffering stages) that the packet must traverse on a minimal path from source to destination. Compressionless routing can use non-minimal paths by adding additional padding flits beyond what is needed for minimal paths. A potential problem with Compressionless routing is that retransmissions are not guaranteed to eventually succeed in delivering a message to the destination. Thus without introducing some further mechanism, the scheme is susceptible to livelock.

A different approach to non-minimal fully-adaptive routing that does not rely on retransmissions is *misrouting*. In one example of this approach, a central packet buffer is provided at each switch. Whenever the buffer is full, the switch must transmit a packet on each output link, even if doing so would require one or more of the packets to be transmitted in an ‘‘unprofitable’’ direction that takes it one hop further away from its destination [Ngai89a, Ngai89b, Ngai87]. This rule guarantees that the switch frees buffer space that can be used to receive packets from the neighbors, and therefore the network is deadlock-free. Any buffer dependency cycles are broken dynamically by misrouting packets to outputs that the packets were not waiting for. Livelocks are avoided by assigning unique priorities to packets such that during a packet’s lifetime in the network, at most a bounded number of other packets will be assigned higher priorities. One policy that satisfies this requirement defines priority as a combination of a packet’s age and distance in hops from its destination, where older packets have highest priority, and packets closer to their destinations have higher priority among equally aged packets. Only the lowest priority packets in the buffer are routed in unprofitable directions. Misrouting has the advantage of sending packets away from regions of high traffic congestion. This increases traffic at the periphery of a congested region, but decreases congestion in the interior of the region. This scheme is applicable to any network topology that has a path from each switch to every other switch and in which all switches have output bandwidth that equals or exceeds the input bandwidth. The primary disadvantage of this approach is the hardware complexity required to select which packets must be misrouted.

Chaos routing is a similar misrouting routing method that has reduced

hardware complexity [Kons91, Kons90b]. In this scheme, switches have small packet buffers at each input port and one larger central buffer. Packets that arrive to a switch input buffer and do not make progress after a timeout are placed in the central queue. Packets in the central queue have priority over packets in the input buffers for profitable transmission. If the queue becomes full, however, then one of the packets in the queue is picked at random and misrouted. The misrouting behavior guarantees deadlock-freedom, and the randomness of the selection from the central queue guarantees livelock-freedom in a probabilistic sense rather than in a deterministic sense. It has been shown that with this technique, the probability that a packet will require **t** hops to be delivered tends to zero in the limit as **t** tends to infinity.

Several approaches have been devised for deadlock-free adaptive routing that allow buffer dependency cycles and do not rely on re-transmissions or packet misrouting. One of the first examples of these approaches was a technique presented in [Pifa91]. In this approach, a routing function is identified which leads to an acyclic buffer dependency graph. However, the routing function is then extended such that packets can take hops that do not correspond to edges in the acyclic buffer dependency graph as long as after doing so, the packet would be able to reach its destination by taking *only* hops that correspond to edges in the acyclic buffer dependency graph. The network remains deadlock-free because packets that encounter buffer dependency cycles always have the option to follow the basic routing algorithm which is dependency cycle-free. The routing algorithms that result from this approach improve on the algorithms in [Kons90a], by providing fully-adaptive routing with the use of only two BVCs per switch and without

introducing bottlenecks at switches that have many ‘‘1’’s in their address. Packets are routed along all the zero-to-one transitions using the first BVC. In addition, packets can also take one-to-zero transitions using this BVC, so long as the packet requires an additional zero-to-one transition to reach its destination. Once all the zero-to-one transitions are done, then the packet is placed in the second BVC and all necessary remaining one-to-zero transitions are then taken in any order. Since every packet in the network has the option to be routed according to the acyclic channel dependency graph of [Kons90a], the scheme preserves deadlock freedom even though the routing function does not restrict packets to take hops that correspond to edges in this graph.

A similar approach which provides deadlock-free non-minimal fully-adaptive routing for k-ary n-cubes and k-ary n-meshes is presented in [Dall93]. The basic idea is to provide escape paths for packets that encounter buffer dependency cycles. Two specific algorithms are presented, one termed ‘‘static’’ and the other ‘‘dynamic’’.

In the ‘‘static’’ approach of [Dall93], the **n** dimensions of the network are ordered, and packets initially try to traverse the dimensions in increasing order. A packet that encounters blocking in this first phase may then proceed to take a path that deviates from the dimension ordering by taking a hop on a lower dimension than it has previously traversed. Such a packet is said to have undergone a *Dimension Reversal*, and the number of times a packet has done this is the packet’s *Dimension Reversal Number* (DRN). The buffers (BVCs) at each switch are partitioned into classes in which the packets of a single class all have the same DRN. As a packet traverses the network, it changes class upon each Dimension

Reversal it undergoes. If the packet enters the final class, it is not permitted to undergo further Dimension Reversals and must reach its destination using fixed Dimension Order Routing (DOR). The algorithm allows packets that have not run out of Dimension Reversals to take non-minimal paths to avoid faulty switches or links. The routing algorithm is deadlock-free since packets can always escape from buffer dependency cycles within a buffer class by transitioning to the next higher class through a Dimension Reversal.

In the ‘‘dynamic’’ approach of [Dall93], the BVCs are divided into two classes: adaptive and deterministic. Packets start out in the adaptive channel class. Each packet is tagged with its DRN. A packet with DRN **p** is not allowed to wait for a packet at the next switch with DRN **q** if $\mathbf{p} \geq \mathbf{q}$. If there is no other choice, however, then the packet with DRN **p** transitions from the adaptive class into the deterministic class, where it continues to the destination using DOR.

Finally, the Disha non-minimal fully-adaptive routing algorithm takes an approach which is similar to but simpler than the approach just described [Anja95]. To eliminate deadlocks, this scheme provides an escape path for any packet that becomes blocked for a timeout period. A token is cycled through the network that can be grabbed by any switch that has a blocked packet. The switch that grabs the token forwards the packet onto a dedicated virtual network that is empty except for the packet. Once the packet is delivered, the token is released back into the network. The original Disha technique was later generalized to avoid the need for the token and exclusive access to the escape path virtual network [Venk96]. The resulting approach inspired the basic deadlock avoidance mechanism for data packets which is presented in Chapter 5 for Dynamic Virtual Circuits.

2.1.5. Packet Buffer Deadlock: Theory

In this section we examine further the two basic approaches for eliminating deadlocks: deadlock detection and resolution, and deadlock avoidance. Deadlock detection and resolution provides maximum flexibility in the use of network resources, but can suffer from occasional deadlocks that must be resolved. Deadlock avoidance imposes some restrictions on the use of network resources to eliminate the possibility of deadlock.

2.1.5.1. Deadlock Detection and Resolution

In this section we describe an approach that has been proposed for *deadlock detection and resolution* in communication networks [Jaff89]. This approach is attractive because it does not restrict routing in any way. In general, routing restrictions can result in inefficient use of system resources. With this deadlock detection and resolution approach, the network can freely utilize most of its resources to improve performance rather than constraining its use of resources in order to avoid deadlocks. As a result, it is possible for deadlock to occur, but the deadlock detection and resolution scheme removes any deadlocks that arise. The scheme can be effective in environments where deadlocks are rare events. This can be the case in networks that use adaptive routing techniques that are effective at relieving severe network congestion which could be a precursor to deadlocks. In Chapter 4 we extend this scheme to be applied to Dynamic Virtual Circuits networks.

The deadlock detection algorithm is initiated when a buffer containing packets fails to make progress in a specified timeout period. The algorithm

identifies a cycle of similarly blocked buffers as a *potential* deadlock. Deadlock resolution is accomplished by introducing an additional buffer into each switch in the potential deadlock cycle. The additional buffer allows a single packet in each switch of the cycle to advance by one hop. Once the hop is taken, the extra buffer is removed. Through this mechanism, packets involved in deadlock cycles make progress and eventually escape from the cycles.

The algorithm detects and corrects deadlocks in packet switching networks whose switches have the following properties: each switch has a central buffer instead of using input or output buffering; links are bidirectional; inputs and outputs can all operate simultaneously; and the routing algorithm directs packets eventually to their destinations. It is shown in Chapter 4 how this algorithm can be extended for use in a network with input-buffered switches and with Dynamic Virtual Circuit switching.

In the original algorithm [Jaff89], whenever a central buffer becomes full and remains so for some time without transmitting any packets, a deadlock is suspected. When a switch decides it *may* be in a deadlock, it first attempts to determine if there is a cycle of switches with full buffers which may form a *potential* deadlock. That is, all actual deadlock cycles are found by the algorithm, but the algorithm may find other transient cycles as well, if the buffers along those cycles are advancing very slowly. A cycle is found by a distributed technique that requires each blocked switch to query a neighbor to which it has packets. Switches that participate in cycle detection and have full buffers are not allowed to inject any new packets into the network until detection is complete. In response to a cycle detection query, a switch either replies that there is no cycle, causing the algorithm

to terminate, or it sends back a number as a reply. The number is the maximum of the switch's node ID and the largest number the switch has received in the current search for a cycle. If there is a cycle, one switch in the cycle has the highest node ID. Eventually, this switch will transmit its node ID to answer a query, and this node ID will loop around the entire cycle and be received by the same switch. Such a switch that receives its own ID declares itself the *leader* of the cycle. The leader sends a notification around the cycle to commit each member to perform the deadlock recovery step, a procedure called *rotation*. Any member of the cycle which has discovered in the meantime that it is not part of a deadlock may, until it commits to rotation, cause all the switches in the cycle to cancel rotation.

If all switches in a cycle commit to rotation, then one packet in each buffer in the cycle is rotated one hop along the cycle (without violating the routing function by misrouting packets). This is accomplished as follows. A special ''Auxiliary Buffer'' is reserved for use only in rotation. Upon starting rotation, the Auxiliary Buffer is activated in each switch in the cycle. One packet at each switch in the cycle is forwarded one hop along the cycle path and is stored in the Auxiliary Buffer at the next switch. The rotation causes each switch in the cycle to transmit exactly one packet, freeing a slot in the normal buffer. The rotation also causes each switch in the cycle to receive exactly one packet, which arrives to the Auxiliary Buffer and then is transferred to the empty slot in the normal buffer. Finally, to complete the rotation process, the now empty Auxiliary Buffer is deactivated at each switch in the cycle.

The rotation process or cancelling of cycle detection completes one iteration of the algorithm. After one or more iterations, a packet may reach a switch where

it can be delivered or transmitted to a switch that is not part of the deadlock cycle. This frees a slot in one of the buffers of the deadlock cycle, thus resolving the deadlock. However, in the worst case the deadlock can persist because new packets are injected into the cycle from switches outside the cycle. Even in this worst case, all packets are guaranteed to be delivered eventually to their destinations by repeated iterations of the rotation procedure. For consistency, switches distinguish between different iterations of the deadlock detection and resolution algorithm. Each switch maintains a sequence number, which it stamps on outgoing requests and replies. Any requests that arrive stamped with higher numbers than the switch's current sequence number are queued and handled by the switch on its subsequent iterations.

The cycle detection and deadlock resolution algorithm requires, at each switch, a fixed amount of storage (the Auxiliary Buffer, equal to the size of one packet) that is available even when the normal buffer is full. Further details of the algorithm can be found in [Jaff89].

2.1.5.2. Deadlock Avoidance

In this section we summarize previous results that generalize the theory of *deadlock avoidance* and subsume many of the deadlock avoidance approaches presented for the routing algorithms described in the previous sections. These results enable developing a wide range of adaptive routing algorithms that avoid deadlocks by selectively restricting packet routes or the use of buffers (BVCs) at each switch. We make use of the results presented in this section in Chapter 5 where we present an approach for deadlock avoidance in DVC networks.

We describe in an informal fashion the necessary and sufficient conditions for deadlock avoidance which were formulated by Duato for virtual cut-through networks [Duat96]. The key idea is that the use of BVCs must be restricted such that each packet in the network can use at least one of the BVCs at a neighboring switch as an escape route from any dependency cycles. The escape route guarantees the packet can eventually advance.

Duato also proved similar results for wormhole networks [Duat95]. With wormhole routing a packet may become blocked across multiple switches simultaneously. As a result, the set of buffer dependencies is larger with wormhole routing than with virtual cut-through. Therefore, some routing algorithms that are deadlock-free in virtual cut-through networks can be prone to deadlock in wormhole routing networks. We do not discuss any further the conditions for deadlock-freedom in wormhole routing networks but instead focus on virtual cut-through networks.

A routing algorithm can be described by a *routing function* denoted as **R** which takes as input the current BVC that is storing a packet and the packet's destination node ID, and returns as output the set of BVCs at neighboring switches to which the packet may be forwarded in one hop. In a virtual cut-through network, Duato has shown the necessary and sufficient condition that routing functions of this form must satisfy to guarantee deadlock-freedom. Duato introduces the notion of a restricted *routing subfunction* $\mathbf{R_1}$, a mathematical device used to analyze a network's routing function **R**. When given the same inputs (current BVC, destination node ID) as routing function **R**, a routing subfunction $\mathbf{R_1}$ returns a subset of the BVCs returned by **R**. Duato showed that a routing function

R is deadlock-free if and only if a routing subfunction $\mathbf{R_1}$ can be found that is fully-connected (i.e., can route packets from any node to any other node) and that acts as an escape path for packets that enter dependency cycles caused by the full routing function **R**.

We denote by **C** the set of all BVCs that are returned by routing subfunction $\mathbf{R_1}$. These BVCs are used as escape resources from dependency cycles. A BVC can be used as an escape resource for all packets, or alternatively it can be used as an escape resource only for packets with particular destinations.

Duato defines the *extended resource dependency graph* for a routing subfunction $\mathbf{R_1}$ to be a graph where the vertices are the BVCs in **C** and the edges are the dependencies between these BVCs that arise from using the routing subfunction $\mathbf{R_1}$. In particular, there is a dependency from BVC $\mathbf{q_i}$ to BVC $\mathbf{q_j}$, where both $\mathbf{q_i}$ and $\mathbf{q_j}$ are members of **C**, if and only if a packet can arrive to $\mathbf{q_i}$ as a result of using the full routing function **R** and then subsequently can be routed in one hop using the routing subfunction $\mathbf{R_1}$ to BVC $\mathbf{q_j}$.

There are two types of such dependencies. A *direct dependency* exists from $\mathbf{q_i}$ to $\mathbf{q_j}$ if it is possible for a packet to use $\mathbf{q_i}$ as an escape resource (i.e., if the packet can arrive at $\mathbf{q_i}$ as a result of using routing subfunction $\mathbf{R_1}$), and if such a packet can subsequently be routed using $\mathbf{R_1}$ to $\mathbf{q_j}$ in one hop. A *direct cross dependency* exists from $\mathbf{q_i}$ to $\mathbf{q_j}$ if it is possible for a packet to arrive to $\mathbf{q_i}$ by using the full routing function **R** but not by using the routing subfunction $\mathbf{R_1}$, and if such a packet can subsequently be routed using $\mathbf{R_1}$ to $\mathbf{q_j}$ in one hop. Direct cross dependencies cover the case where a BVC $\mathbf{q_i}$ is an escape resource for packets with one set of destinations, but these packets are not routed to $\mathbf{q_j}$ according to $\mathbf{R_1}$,

hence there is no direct dependency. However, it is possible that packets with other destinations can be routed to $\mathbf{q_i}$ using the full routing function **R**, and from there if the routing subfunction $\mathbf{R_1}$ can route these packets to $\mathbf{q_j}$, then there is a dependency arising from $\mathbf{R_1}$ from $\mathbf{q_i}$ to $\mathbf{q_j}$.

Duato proved that ‘‘A connected and adaptive routing function **R** ... is deadlock-free iff there exists a routing subfunction $\mathbf{R_1}$ that is connected and has no cycles in its extended resource dependency graph.’’ Duato derived additional results that are specific to a subclass of the routing functions considered above. Routing functions in the subclass take as input the packet destination and the current node ID (as opposed to the current BVC). In this case, for deadlock avoidance it is sufficient to ensure that the set of BVCs supplied by the routing subfunction $\mathbf{R_1}$ are always supplied by $\mathbf{R_1}$ whenever the routing function **R** also supplies them. This leads to a routing algorithm design methodology that first allocates a set **C** of BVCs for a deadlock-free routing subfunction $\mathbf{R_1}$, and then adds other BVCs for use only by the routing function **R** and not $\mathbf{R_1}$. The routing function **R** may use the new BVCs in any manner that directs the packets eventually to their destinations, so long as all packets in the new BVCs can be routed in one hop to a BVC in set **C**.

2.2. Switch Implementations

This section briefly describes representative examples of switch implementations for multicomputers or as standalone proof-of-concept switch prototypes. The switches employ various routing techniques that were outlined in the previous sections.

One of the earliest multicomputer switches was the Torus Routing Chip developed at Caltech in the mid-1980s [Dall86]. This design employs wormhole routing for 2D torus networks with Dimension Order Routing. Two BVCs per link are provided to prevent deadlock [Dall87].

The Chaos Router Chip was developed at the University of Washington [Kons91]. This device uses virtual cut-through and non-minimal fully-adaptive routing. A central queue buffers packets. Packets in the central queue are pseudo-randomly chosen for misrouting in order to prevent deadlock.

The IBM SP2 switch was designed to connect RISC System/6000 processors together via a multistage interconnection network [Stun95]. The SP2 switch employs *buffered wormhole* switching. Buffered wormhole switching is very similar to wormhole routing. The only distinction is that each arriving message uses a switch input buffer until the message blocks, in which case the message is transferred to a larger central queue at the switch that is storing the head of the message. In many cases the central queue has sufficient free space to buffer the entire message, as with virtual cut-through. However, if the central queue has insufficient free space for the entire packet, then the packet blocks in the multiple switches in the network just as in wormhole routing. The network employs source routing, in which the source node places routing flits at the beginning of each packet to specify the entire path the packet will use to reach the destination. Switches along the path interpret the first routing flit in the packet to determine where to forward the packet on the next hop. A switch that uses up the information in the first routing flit deletes that flit from the packet, in which case the following routing flit becomes the new first routing flit and will be processed by the switch at

the next hop.

The SGI Spider chip was the switch used in the SGI Origin multiprocessor [Gall97]. The switch has 4 BVCs per link, where each BVC uses a 256-byte buffer organized as Dynamically-Allocated Multi-Queue (DAMQ) buffer [Tami88a]. The switch uses routing table lookup to route packets. The routing tables are software-programmable and must be assigned values that avoid deadlock. The routing tables at each switch are hierarchical: a ‘‘metatable’’ specifies routing for the high-order 5 bits of a packet’s destination address while a ‘‘local table’’ specifies routing for the low-order 4 bits of a packet’s destination address once the packet reaches a switch with a matching high-order 5 bits. The use of hierarchical routing reduces the size of each routing table from 512 entries (one per destination) to 48 entries. An interesting optimization provided by the Spider chip is that the routing tables indicate the output link a packet will use at the next switch rather than at the current switch. This output port ID is transmitted with the packet, allowing a switch that receives the packet to immediately initiate crossbar arbitration for the appropriate output link and in parallel perform routing table lookup to determine the output link the packet will use at the next switch. Another interesting feature of the Spider chip is that it can recover from physical transmission errors at the link level transparently to the routing algorithm and to the packet source and destination nodes. This capability is enabled in the Spider chip by employing a go-back-*n* ARQ protocol [Bert87] across each link to quickly detect and retransmit corrupted data phits between the two endpoints of each point-to-point link.

The final example we present is the Cray T3E switch. The Cray T3E network

topology is a 3-D torus. The network provides fault-tolerant minimal fully-adaptive routing and support for fast distributed eureka and barrier operations [Scot96b]. Each switch has 5 BVCs per link, plus one unbuffered BVC dedicated for high-priority synchronization flits (for barriers and eurekas). Two of the BVCs are used for deterministic routing of remote read/write requests, and two other BVCs are used for deterministic routing of the replies. The fifth BVC is used for minimal fully-adaptive routing with buffer dependency cycles. A packet using a deterministic BVC may, on its next hop, transmit to another deterministic BVC or may alternatively transmit in any profitable direction to an adaptive BVC that is free. The routing algorithm used by the four deterministic BVCs is *direction-order routing*, which derives from the Turn Model [Glas94]. Direction-order routing orders the directions in the network (as opposed to the dimensions of the network). The order chosen for the Cray T3E network is +X, +Y, +Z, -X, -Y, and -Z. This ordering of directions removes enough turns to prevent dependency cycles. In addition to the path specified by direction-order routing, a packet may take one initial hop in any positive direction before following the direction-order path to the destination. A final hop in the -Z direction may also be taken. These initial and final hops do not introduce dependency cycles. Furthermore, they can be used to route around faulty or missing nodes or links within a two-dimensional plane of the network. The network uses wormhole routing except for the adaptive BVC, which employs virtual cut-through. Using virtual cut-through for the adaptive BVC eliminates some dependencies that would be present with pure wormhole routing and would complicate deadlock avoidance [Duat95].

2.3. Connection-Based Routing

In this section we review related work on *connection-based routing*, in which resources are reserved in advance of communication at each switch along the path from source to destination. Compared to pure packet switching, connection-based routing reduces the overhead for forwarding each packet but introduces connection setup/teardown operations. Connection-based routing can improve network efficiency so long as these new operations are fast or occur infrequently, e.g., for workloads in which each connection is used by several packets [Hsu92].

In Section 2.3.1 we describe *circuit switching*, the most elementary form of connection-based routing in which connection establishment reserves all the link bandwidth along a path from a source to a destination. In Section 2.3.2 we describe *virtual circuit switching*, which combines some of the advantages of circuit switching and packet switching. In particular, virtual circuit switching allows data transmission on an established circuit to have low routing and sequencing overheads, similar to circuit switching, while retaining the advantage of packet switching that each link can be time-shared at packet granularity by traffic with different source nodes and destination nodes.

2.3.1. Circuit Switching

With *circuit switching*, each connection, or *circuit*, reserves all the bandwidth on a network path from the source node to the destination node. Once the circuit is established, data can be transmitted at full link bandwidth without the need to transmit control information for routing or sequencing [Chow88]. However, the physical links on a circuit's path are statically allocated to the circuit throughout its

lifetime. Even when no information is being sent through the established circuit, the links cannot be used for other circuits.

Circuit switching was used by the JPL Hyperswitch binary n-cube multicomputer [Chow87, Pete88, Chow88]. In this network, each circuit is established to transmit exactly one message of arbitrary length, and then the circuit is torn down. The Hyperswitch uses an adaptive routing approach based on *header backtracking* to establish new circuits. Similar backtracking adaptive routing algorithms have been proposed for wormhole networks [Alle94]. With header backtracking in the JPL Hyperswitch circuit switching network, to transmit a message a source node injects a circuit establishing header which traverses the network and allocates links to the new circuit until the header reaches a link which is already allocated to another circuit. Instead of waiting for the link to be released, the header may backtrack by one hop to the last switch it visited and try another path. The algorithm attempts to avoid congested regions (subcubes of the hypercube) by not retrying the dimension that was blocked until all other dimensions have been attempted without success. The scheme provides minimal adaptive routing in the sense that the final circuit, once it is established, is guaranteed to use a minimal path to the destination. However, since the header may backtrack, more switches than necessary may be visited than in a non-backtracking minimal routing scheme. Once a circuit is established, the message is guaranteed to reach its destination since it can use all the bandwidth on the circuit's path. However, there is no guarantee that circuit establishment will succeed in finding a free path to the destination, and the header may backtrack all the way back to the source node. Therefore, livelock is possible in which a sender may re-

inject the same header infinitely many times, each time unsuccessful in setting up a path to the destination.

2.3.2. Virtual Circuit Switching

Virtual circuit switching [Bork90, Dao97, Hsu92, Tyme81] is a form of connection-based routing which provides end-to-end connections called *virtual circuits*. The network's physical resources (packet buffers, link bandwidth, etc.) are multiplexed among active virtual circuits, unlike with circuit switching. Once a virtual circuit is established, packets can be sent without the addressing and sequencing information needed in pure packet switched networks. Packets are quickly forwarded along the virtual circuit's network path by using a single lookup in a small virtual channel table at each hop. This mechanism enables low overhead packet forwarding even in networks with irregular topologies [Flin00]. In these topologies, intermediate nodes between a source and destination cannot use simple algorithmic routing. Instead, routing is based on large tables at each node which are constructed at system initialization and may be changed over time to adapt to changing traffic conditions. With virtual circuits, the routing tables can be used during virtual circuit establishment to determine a virtual circuit's network path. Subsequently, packets are forwarded along the path using the more efficient virtual circuit forwarding mechanism.

A traditional virtual circuit is static; it holds an unchanging set of resources throughout its lifetime. Section 2.3.2.1 describes static virtual circuits in more detail. Section 2.3.2.2 then discusses previously proposed or implemented approaches for removing the static restrictions of virtual circuits. We also discuss

the advantages of Dynamic Virtual Circuits compared to these prior approaches.

2.3.2.1. Static Virtual Circuits

With virtual circuits, a source node initiates communication with a destination by establishing a new virtual circuit on some path to the destination. The source node then transmits one or more data packets over the new virtual circuit. The data packets are forwarded at each intermediate switch with very little processing and carry only a few bits of overhead (packet header) information. Finally, the source terminates the virtual circuit. Each source and each destination may have many virtual circuits established at the same time.

We next describe the basic steps of virtual circuit establishment and disestablishment. Consider a virtual cut-through network composed of **n x n** switches interconnected with bidirectional point-to-point links. At each switch, one or more ports may connect the switch to one or more hosts. The switch is input buffered with an **n x n** crossbar connecting the **n** input ports to the **n** output ports.

The source host establishes a new virtual circuit by injecting a Circuit Establishment Packet (CEP) into the network. The CEP records the virtual circuit's source and destination addresses, which are used to adaptively route the CEP. For example, CEP routing may be accomplished through the use of routing tables maintained at each switch, such as in the SGI Spider chip [Gall97].

A CEP allocates for a new virtual circuit one *Routing Virtual Channel* (RVC) on each link it traverses on its path from source to destination (including the source and destination host interface links). An RVC is an entry in a table at the switch

input port that is connected to the link. Each physical link is logically subdivided into multiple RVCs. Each packet header has a field that identifies the RVC used by the packet. At each switch, the mapping from input RVC to output RVC is recorded in an ‘‘Input Mapping Table’’ (IMT) at the input port. The IMT is indexed by the RVC value in the header of an arriving packet. An IMT entry records the next hop routing information for the virtual circuit that has allocated the RVC. This routing information consists of the switch output port and the output RVC value to use to forward packets that are using the virtual circuit. An example of an established virtual circuit is shown in Figure 2.2.

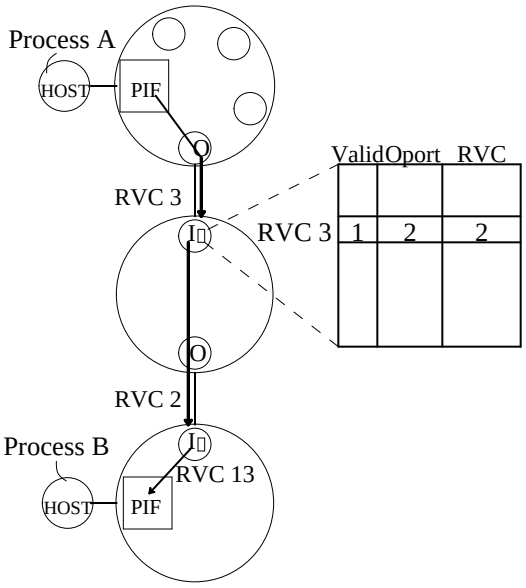


Figure 2.2: A virtual circuit directed from process A to process B and traversing three switches. The circuit enters the network using the processor interface (PIF) at the topmost switch. At the input port of the next switch, the Input Mapping Table (IMT) specifies the output port (Oport) and Routing Virtual Channel (RVC) ID to use for the next hop. The circuit uses RVC 3 of the first link and RVC 2 of the second.

Note that we use the term ‘‘RVC’’ instead of the more standard ‘‘virtual

channel'' to distinguish it from the same term commonly used to refer to flow-controlled buffers that prevent deadlock and increase performance [Dall87]. As described in Section 2.1.3, we call the latter *Buffering Virtual Channels*, or ''BVCs''. In contrast, ''RVCs'' simply identify virtual circuits, and they do not require separate flow-controlled buffers.

The source may transmit one or more data packets over a new virtual circuit. Each data packet is quickly routed at each switch, by accessing the IMT entry with the RVC value in the packet header. The RVC value in each packet's header is overwritten with the output RVC value recorded in the IMT entry, and the packet is enqueued for transmission to the next switch. The RVC number is the only addressing information that must be transmitted with the packet. Since all packets that use the circuit take the same path through the network, the packets arrive at the destination in FIFO order, thus no additional sequencing information must be transmitted with the packet. Therefore, the overhead for routing data packets with virtual circuits is significantly reduced compared to a pure packet switching mechanism, particularly for packets with small data payloads.

The source host terminates the virtual circuit by injecting a Circuit Destruction Packet (CDP) into the network. The CDP traverses the circuit path, releasing at each hop the RVC that is allocated to the virtual circuit after the data packets that use the virtual circuit have departed the switch.

An example of a multicomputer that used virtual circuits is iWarp [Bork90]. The iWarp multicomputer was designed especially for ''systolic'' applications in which elements of data streams are processed in a pipelined fashion across multiple processors. The use of virtual circuits in this system was motivated by the high

degree of temporal locality in the traffic patterns that result from systolic applications. That is, the traffic from each source node is directed to a small, unchanging set of destinations. In particular, each processor in a systolic application communicates only with processors that are immediately downstream in the systolic pipeline, and therefore can be supported with a small number of virtual circuits originating from each node. To support real-time requirements of some systolic applications, iWarp provides means for each virtual circuit in iWarp to reserve a fraction of the link bandwidth on the virtual circuit path.

An approach to connection-based routing that combines static virtual circuits (for traffic exhibiting locality) and conventional wormhole routing (for other traffic) was proposed by Dao, Yalamanchili, and Duato [Dao97]. In this proposal, a source node tries to establish a static virtual circuit by injecting into the network a circuit establishment probe. The probe is adaptively routed toward the destination; along the way the probe may backtrack as it searches for a path with free resources on each link. If the probe backtracks to the source node, the source either re-injects the probe for another try or gives up on establishing a circuit, in which case it uses conventional wormhole routing for traffic it sends to the destination. Existing circuits are not torn down to free resources for new circuits. Nor are circuits rerouted to adjust to congestion or faults.

2.3.2.2. Relaxing the Restrictions of Static Virtual Circuits

Most virtual circuit schemes have the limitation that each established circuit's route cannot be changed until the connection is no longer needed and the circuit is torn down. This prevents adaptation to changes in traffic patterns, and it prevents establishment of new virtual circuits once all the required resources are assigned to existing circuits. To solve this problem we have proposed the *Dynamic Virtual Circuits* (DVC) mechanism [Tami91, Turn98] which combines adaptive routing and connection-based routing. In this section we present prior approaches for combining adaptive routing and virtual circuits.

In the Codex network, each node can reroute established virtual circuits for which the node is the virtual circuit destination [Bert87]. The policy a node uses to determine which virtual circuits to reroute is based on global information about link utilization and other monitored statistics. In particular, each switch continually monitors various statistics about its communication links (e.g., utilization, and processing delays) and periodically broadcasts this information to every node in the network. In this manner, each node periodically obtains information about all the links in the network. Each node uses this information to periodically select (pseudorandomly) some of the virtual circuits for which the node is the destination. The node evaluates each of these selected virtual circuits to determine whether to reroute it onto a new network path. To make this decision, the node uses an algorithm that finds the shortest path through a weighted-edge graph that models the network's behavior. In this graph, each vertex corresponds to a switch, and each edge corresponds to a network link. The weight of an edge is computed according to a function that attempts to characterize the delay that would

be experienced by packets using the virtual circuit if the virtual circuit were using a path that included the corresponding link. This function takes into account the flow rate of the virtual circuit and also the monitoring information that is provided periodically to the node about the corresponding link. The shortest path search algorithm finds the path in the network that would experience the smallest increase in delay if the virtual circuit were routed onto this path. The node reroutes the virtual circuit onto this path if it differs from the current path used by the virtual circuit. Rerouting of an existing virtual circuit can involve considerable delay as the destination node sends a message to the sender node to establish the new path for the virtual circuit. Based on the description in [Bert87], it appears that this delay for rerouting could be eliminated by having the sender node instead of the destination node make rerouting decisions. Either way, the scheme has the disadvantages of not being able to reroute packets after they have left the sender node, and the possibility of incurring long delays to release resources held by an existing virtual circuit.

A different approach that relaxes the static nature of virtual circuits was proposed by Hsu and Banerjee [Hsu90]. Unlike with the Codex network, in which only destination nodes can reroute virtual circuits, this approach allows intermediate switches on a virtual circuit path to trigger teardown of the circuit to free resources for new virtual circuits or to re-route the circuit. This approach proposes to add logic to support on each switch a small number of virtual circuits, called *cached circuits*. A cached circuit relaxes the static restrictions of virtual circuits; a cached circuit may be torn down in order to free up resources at an intermediate switch. The resources may be needed, for example, to establish a new

cached circuit. The intermediate switch selects an existing cached circuit to be a victim, and it sends a request to the source node of the victim circuit to tear it down. The packets in transit from the victim's source must progress past the intermediate switch before the resources held by the victim circuit can be released. Therefore, new circuit establishment may be blocked for an extended period while packets on the victim circuit are being flushed out. In addition, if a circuit is rerouted onto a more desirable path, the packets that are flushed out are forced to use the original path.

Virtual circuits are used in Asynchronous Transfer Mode (ATM) networks [De 96]. ATM supports two types of connections: virtual circuits (VCs) and virtual paths (VPs). A VC is composed of a sequence of VPs from source to destination. Each VP can support 2^{16} VCs. A VC can be rerouted to improve quality of service via a round trip exchange of control messages between the source and destination [Cohe94]. A VP that is used by many VCs can be rerouted when a link on its route fails. Rerouting a VP is transparent to the VCs that use it. A VP can be rerouted onto a backup route that is either pre-defined or is selected after a failure is detected [Kawa99, Gers99]. VP rerouting is accomplished through a round trip exchange of control messages on the backup path between the VP endpoints [Kawa99]. Alternatives to end-to-end VP rerouting include rerouting only the portion of the VP between two switches that are adjacent to the failure, and rerouting the portion of a VP from a switch that is upstream from the failure and the VP destination. These strategies differ in the time required to reroute a VP and in the spare bandwidth that is needed to guarantee that all VPs can be rerouted and meet their bandwidth requirements after any single failure [Mura97, Ande94].

Our Dynamic Virtual Circuits (DVCs) [Tami91] proposal, described in Chapter 3, differs from the above proposals by allowing virtual channel resources to be quickly reallocated through local operations at a switch, avoiding the long delays of schemes that require interactions with faraway nodes before releasing local resources. Resource reallocation avoids blocking circuit establishment, and it enables adaptive circuit rerouting.

2.4. ATM and IP Switch Cell Loss Reduction Techniques

In this section we review previous work in the problem area of reducing data discarding caused by congestion in large-scale switches that support Asynchronous Transfer Mode (ATM [De 96]) networks or IP networks. Switches in these networks, which are intended for general-purpose networking applications, typically do not use the backpressure flow control techniques used by the more specialized multicomputer networks to avoid discarding packets, although it has been suggested that similar flow control features should be provided (at least for controlling best-effort traffic which has no performance guarantees) [Kung94]. Instead, in typical ATM or IP networks, data can continue arriving to a switch even when the switch lacks adequate free buffer space to store the data. Data that arrives when the switch is in this state is discarded. In Chapter 6 we propose new techniques for reducing data loss in such switches.

An $N \times N$ ATM or IP switch consists of N input ports, N output ports, and a switching fabric that forwards packets (called *cells* in ATM terminology) from input ports to output ports. Small switches can be built with a crossbar or shared bus organization. For cost effectiveness and high performance, large switches are

typically implemented with an internal switching fabric which has a multistage indirect network topology, in which each stage consists of a number of small switching elements (SEs) and adjacent stages are interconnected with point-to-point links between SEs at the adjacent stages [Part94].

A primary goal in the design of such switches is to minimize the loss of cells or packets that can result from congestion. Cell loss degrades the quality of real-time traffic and necessitates costly retransmission of best-effort traffic. Cell loss is measured by the Cell Loss Ratio (CLR), the ratio of the number of cells dropped by a switch to the number of cells injected to the inputs of the switch.

Cell loss reduction techniques can be classified into two categories: techniques that limit the traffic entering the switch, and techniques that reduce cell losses within the switching fabric.

Techniques in the first category require cooperation between the switches and the end points of the communication. With some techniques, the switch prevents entry of cells (or packets, or traffic flows) based on the current state of the switch. With *connection admission control*, the sender provides the switch with the Quality of Service (QoS) requirements of a proposed new traffic flow. The QoS requirement can specify properties such as the minimum throughput or the maximum cell loss ratio that the switch must provide for the flow. The switch accepts or rejects the request for establishment of the new traffic flow through the switch on the basis of an estimate of the switch's ability to satisfy the new connection's QoS requirements without violating the QoS requirements of existing connections [Jami97]. An alternative approach that relies on implicit cooperation with the traffic sources is Random Early Detection [Floy93], in which the switch

probabilistically tags or drops packets when its buffer occupancy exceeds some threshold. For this scheme to work in preventing the switch buffers from filling and causing much higher levels of cell loss, the source or destination nodes must employ a congestion control mechanism such as TCP that reduces the rate of traffic injection for a traffic flow whenever the source node detects that one of its packets was lost or tagged in the network.

In the second category of techniques, cell losses within the multistage switching fabric are reduced without limiting the arrival of traffic to the inputs of the switch. These techniques include: inter-stage flow control, cell sorting and recirculation, providing large or shared buffers in SEs, and providing alternate paths.

Within a switching fabric, inter-stage flow control can be provided to completely prevent the buffers of individual SEs from ever overflowing. However, this approach simply shifts the problem of cell loss away from the switching fabric and back to the inputs of the switch.

Unbuffered fabrics (in which SEs do not contain buffers), such as the Starlite [Huan84] and Sunshine [Giac91] switches, reduce cell loss by adding several extra stages, located immediately after the switch input ports, to sort and recirculate cells. The extra stages form a Batcher sorting network [Batc68], which bitonically sorts and recirculates cells. Cells routed to the same output port of the switch conflict in the sorting network and are recirculated back to the inputs for another pass if a recirculation path is available, or else are dropped. The sorted cells are guaranteed to proceed without conflict or discarding through a multistage banyan routing network until they reach their outputs.

In buffered fabrics in which each SE contains packet buffers, cell loss in an SE can be reduced by increasing the size of internal buffers. The buffers may be dedicated to particular SE inputs or outputs, or they may be shared. Shared buffers utilize buffer storage more efficiently than partitioned buffers, reducing the buffer capacity needed to achieve an acceptable cell loss ratio. This is particularly important where the amount of buffer storage must be sufficient for handling occasional bursts of traffic without (or with minimal) cell loss. Examples of switching elements with shared buffers include the IBM Prizma [Denz95, Denz92], Alcatel MPSR [Henr93, Boet90, Bann91], and, for multiprocessor interconnect, the IBM SP1 [Stun94b, Stun94a] switch. The benefit of shared buffers comes at the cost of increased implementation complexity. The buffer's bandwidth must be at least equal to the sum of the bandwidths of the SE's links. That is typically accomplished by reading and writing the buffer through a wide bus, which requires input link interfaces to accumulate up to several bytes of an arriving cell before writing, and output link interfaces to serialize the wide data read from the buffer. Several linked lists must be managed within the shared buffer as cells are read and written in an interleaved fashion. Virtual cut-through [Kerm79] can be implemented to avoid store-and-forward delays under light load by bypassing the shared buffer, at the expense of an additional crossbar and bypass logic [Stun94b].

Cell loss within the switching fabric may also be reduced by providing alternate paths for each source-destination pair. Alternate paths balance load, thereby decreasing the probability that traffic bursts overflow SE buffers. Alternate paths can be provided by adding extra switching stages to a single-path switch. In some cases this is done without changing the raw internal bandwidth of

the switch. For example, the Beneš topology [Bene64] has twice the number of stages as a single-path banyan topology, providing enough alternate paths to route simultaneous flows from all the inputs through an arbitrary permutation without conflict. Alternate paths can also be provided through internal bandwidth expansion, which adds extra links or SEs to switching stages. That approach reduces cell loss by further reducing the peak load per SE output. For example, Dilated Butterflies or Multibutterflies add extra links to a standard butterfly banyan topology to provide alternate paths [Leig92]. The Alcatel Multi-Path Self-Routing Switch combines both approaches by adding both an extra stage and also extra SEs to each stage, creating a large number of alternate paths with low peak load per SE [Henr93].

Finally, some techniques mitigate the negative impact of cell loss by using higher-level protocols. For example, cells marked as low priority can be chosen for discard ahead of higher priority cells. For TCP/IP traffic over ATM networks, Early Packet Discard [Roma95] can be used to reduce packet loss.

2.5. Summary

Several routing schemes have been developed for multicomputer and cluster interconnection networks that use packet switching, in which data is partitioned into bounded sized packets which are routed individually through the network. Routing schemes can be fixed-path or adaptive, in which case packets have a choice of multiple paths to reach their destinations. With the use of backpressure flow control that prevents packets from being discarded as a result of congestion, fixed-path and adaptive routing algorithms may be susceptible to packet buffer

deadlocks. Several approaches have been developed to eliminate deadlocks. One approach, deadlock detection and resolution, imposes minimal restrictions on packet routing and packet buffer usage at the expense of incurring occasional deadlocks which trigger a separate mechanism for detecting the deadlocks and removing them. An alternative approach is deadlock avoidance in which some restrictions are placed on routing and buffer usage to eliminate the possibility that deadlocks can arise. Several schemes have used variations of packet buffer deadlock avoidance. This approach was generalized by Duato [Duat96, Duat95] who identified that deadlocks are avoidable so long as any packets that encounter a resource dependency cycle are guaranteed to have an escape path from the cycle.

Connection-based routing, in which resources are reserved at each switch along a path from source to destination prior to communication, can be more efficient than packet switching in terms of the overhead for routing and processing packets at each hop. The most basic type of connection-based routing is circuit switching which reserves the entire link bandwidth along the path for a connection to provide extremely low overhead for data forwarding at the expense of high resource allocation costs. Static virtual circuits combine the advantages of circuit switching and packet switching. However, most virtual circuit schemes are static, holding an unchanging set of resources throughout their lifetime, thereby reducing potential throughput and precluding the use of adaptive routing for a circuit after it is established. Some proposed mechanisms allow the endpoints of virtual circuits to re-route the virtual circuits upon failure or congestion, at the expense of time-consuming and bandwidth-consuming end-to-end coordination. The new mechanism described in Chapter 3, *Dynamic Virtual Circuits*, removes the static

restrictions of virtual circuits by allowing any node in the middle of a circuit to make a local decision to break it and later re-establish it as needed.

In general-purpose ATM and IP networks composed of large-scale switches, a key challenge is reducing the loss of data because of buffer overflows within the switch. Techniques to reduce cell or packet loss either limit the traffic entering a switch, or reduce cell losses within the scalable switching fabric that is built into the switch. Chapter 6 investigates the use of adaptive routing and connection-based routing techniques within a large-scale switch to reduce the incidence of cell discarding.

Chapter Three
Dynamic Virtual Circuits: Overview

In this chapter, we present a high-level overview of our proposed Dynamic Virtual Circuits (DVC) mechanism. The DVC mechanism enables multicomputer and cluster interconnection networks to combine the benefits of connection-based routing and adaptive routing. Unlike with traditional static virtual circuits, the establishment of each new Dynamic Virtual Circuit is guaranteed, even when there are no free Routing Virtual Channels (RVCs) on a desired link. Resources allocated to idle DVCs are reassigned to active DVCs as needed.

The DVC mechanism overcomes the limitations of prior approaches for combining adaptive routing and connection-based routing (these approaches are described earlier in Section 2.3.2.2). In particular, with the DVC mechanism, any switch along the path of an existing virtual circuit can tear down the circuit and re-establish it on a new path by using fast, local operations that do not involve costly delays for coordination with remote nodes. Thus DVCs avoid delays caused by waiting for disestablishment requests to be processed through a circuit source node or destination node. With DVCs, an intermediate switch can tear down the portion of a circuit from itself to the destination node while leaving intact the portion of the circuit from the source node to the intermediate switch. Any packet that is using the torn down circuit and subsequently arrives to the intermediate switch causes the switch to initiate re-establishment of the torn circuit in order to forward the arriving packet.

With the DVC mechanism, if a particular circuit becomes slow or blocked,

due to congestion or failure, a node can make a *local* decision to break the circuit and re-establish it using an operational, less congested route. Adaptation can occur quickly since: (1) it is not necessary to wait for the source node to reroute the circuit, and (2) as the circuit is being re-established, busy RVCs that are needed along the new path are released quickly.

The following sections describe the basic techniques and switch hardware support for establishing, disestablishing, and rerouting DVCs. We significantly extend these techniques in the subsequent chapters in order to eliminate deadlocks that can arise in DVC networks. Deadlocks can arise both from packet buffer dependencies and from communication dependencies caused by the introduction of virtual circuit manipulation operations.

3.1. Tearing Down and Re-Establishing Circuits

In the simplest case, the Dynamic Virtual Circuits mechanism is identical to the static virtual circuits mechanism which was described in Section 2.3.2.1. When a Circuit Establishment Packet (CEP) arrives at a switch input port, the switch invokes a routing algorithm to determine the desired output port to use to forward the packet. In networks that have a regular network topology (e.g., k-ary n-cube), the routing algorithm can use simple arithmetic operations to determine an appropriate output port [Chen90]. In contrast, for irregular networks, a more complex scheme based on large routing tables may be used [Taji77]. If there is a free RVC on the desired output port, this RVC can be allocated to the new circuit. The Input Mapping Table (IMT) at the switch input port is modified to indicate a new mapping from the input RVC used by the arriving packet to the desired output

port and the newly allocated output RVC. After a switch establishes a mapping for a new DVC, the switch will correctly route any arriving data packets that use the new DVC. Once the circuit is no longer needed, the source node injects a Circuit Destruction Packet (CDP) which traverses the circuit path and releases all the RVC resources that were allocated to the circuit.

With traditional static virtual circuits mechanisms, establishment of a new virtual circuit will fail if a CEP arrives at a switch and is routed to a link which does not have any free RVCs (i.e., if all the RVCs of the desired output link are already allocated to existing circuits). In contrast, the Dynamic Virtual Circuits mechanism is able to successfully establish a connection even in this case. With the DVC mechanism, the switch selects one of the established DVCs on the desired link as a victim for temporary destruction in order to release and reassign the RVC resource that is currently held by the victim. Ideally, the victim DVC is the circuit whose next arriving packet will arrive at the switch at the furthest time in the future. Once a victim is selected, the switch generates and sends a Circuit Destruction Packet (CDP) for the victim circuit. The CDP is marked as *nonterminal* to inform the destination node that the circuit is being torn down temporarily from an intermediate node, as opposed to permanently from the source node. After the generated CDP is transmitted on the output link, the CEP can be transmitted to establish the new DVC.

When a CEP arrives at a switch to establish a DVC, the switch records the addressing information for the new circuit (e.g., it records the destination node ID) for use in case the circuit must be re-established at some later time. When a data packet arrives at a switch using a DVC that the switch has previously torn down,

the switch retrieves the destination node ID which was originally recorded for the torn circuit and initiates re-establishment of the circuit as follows. The switch performs routing to determine an appropriate output port to use to re-establish the torn circuit. Then the switch creates a CEP, updates the mapping tables, and transmits the CEP to re-establish the circuit on a new path to the destination. Once the CEP is transmitted to the next switch, the data packet that triggered the circuit re-establishment, and any future data packets that use the same circuit, can be transmitted along the new path without waiting for any coordination.

Figure 3.1 shows an example of DVC establishment. The figure shows a single switch as it progresses through four steps of circuit establishment.

In Step 1 of Figure 3.1, a CEP arrives to the left input port to establish a new circuit. The switch processes the addressing information in the CEP and determines the CEP should be routed to the top output port. Suppose that all the RVCs of the top output port are already reserved for currently established DVCs. In this case, the switch selects one of these DVCs as a victim for teardown. The selected victim is shown in the figure as the circuit traversing the switch from the bottom input port to the top output port. A data packet (D) which is using the victim circuit is shown queued at the bottom link's input buffer waiting for transmission through the top output port.

In Step 2 of Figure 3.1, the switch creates a CDP to tear down the victim circuit. The CDP is also used to flush from the switch any data packets that are using the victim circuit. The CDP is thus enqueued in the input buffer at the bottom input port behind any data packets that are already enqueued. The switch marks as ‘‘unmapped’’ the IMT entry for the victim circuit's RVC at the bottom

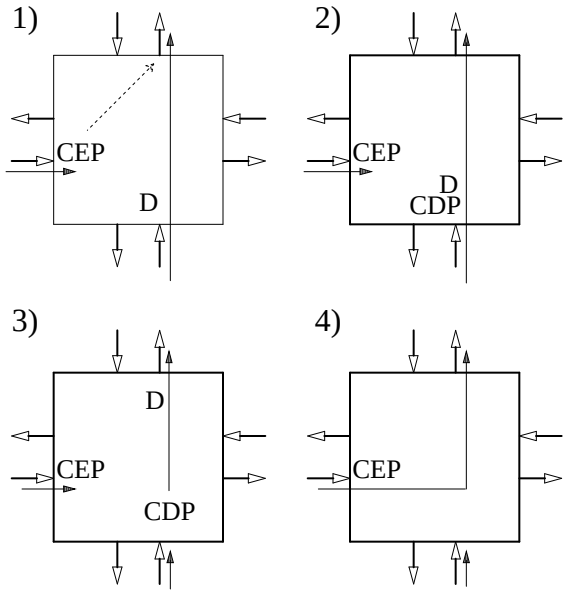


Figure 3.1: Example: DVC Establishment (no free RVCs).
CEP = Circuit Establishment Packet
CDP = Circuit Destruction Packet
D = Data Packet

input port. This ensures that future data packets that arrive on this RVC will trigger the switch to initiate re-establishment of the victim circuit.

In Step 3 of Figure 3.1, the victim circuit's data packet is flushed from the switch by transmitting it to the neighbor. After that occurs, the CDP can be transmitted. As the CDP leaves the switch, the RVC at the output port that was held by the victim circuit is released.

In Step 4 of Figure 3.1, the switch allocates the released RVC to the new circuit by recording a new mapping in the IMT entry at the left input port. The new mapping indicates that the RVC used by the CEP at the left input port is mapped to the top output port and the just-released RVC of the top output link. Next, the CEP that is establishing the new circuit can be transmitted using the top

output port and the newly-allocated RVC.

The subsequent arrival at the switch of any data packet that is using the torn victim will result in an IMT lookup that returns the ‘‘unmapped’’ status for the input RVC. This causes the switch to initiate circuit re-establishment, a process similar to normal circuit establishment except that the switch must generate a new CEP using the addressing information that was recorded upon the previous establishment of the victim circuit.

The above description of the DVC mechanism is accurate at a high level of abstraction, but it is also a simplified description. In reality, multiple input ports and output ports operate and interact concurrently at each switch. Care must be taken to prevent components of the switch from entering inconsistent states due to improper ordering of events or from becoming stuck forever waiting for each other to complete some operation. We discuss these issues further in Section 3.3.

3.2. Maintaining Packet Ordering with DVCs

Although DVCs provide most of the advantages and overcome the difficulties of static virtual circuits, they do not guarantee physical FIFO packet arrival at the destination as with static virtual circuits. For example, a circuit that has been torn down from an intermediate switch may be re-established on a different path before the nonterminal CDP reaches the destination. In that case, the packets on the re-established branch of the circuit arrive at the destination before all the packets on the torn-down branch arrive. Since proper packet ordering is vital, some mechanism must be provided to allow the destination host to determine the order in which packets were sent by the source host. We describe two mechanisms that can

be used for this purpose. One mechanism is based on recording packet sequence numbers for each circuit at each switch. The second mechanism is based on using logical timestamps to distinguish different branches of the same circuit.

With the packet sequence number mechanism, for each circuit that is established at a switch, a packet count register records how many packets have arrived at the switch using the circuit. A sliding window protocol can be used to bound the maximum value of the packet counters. Also stored at each intermediate switch is other information needed to uniquely identify the particular DVC, such as source and destination node and process IDs. When a circuit is re-established, the packet count at the switch that is initiating the re-establishment is attached to the CEP along with the information that was originally used to establish the circuit (source process ID, destination process ID, etc.). The destination host accepts packets on re-established circuits only after its local packet counter indicates that all previous packets have already been received. Any packets that cannot be accepted are buffered at the destination host interface until they can be accepted. The main disadvantage of this scheme is that it requires hardware at each input port to record and increment the packet counter values.

The alternative timestamp mechanism avoids the need to update a packet counter for each arriving packet. The mechanism requires only a small amount of information to uniquely identify branches, and it updates circuit information at the intermediate switch only when a circuit is re-established or disestablished from that intermediate switch. Because the circuit information is rarely updated, it is not necessary to store it on-chip or provide dedicated hardware for updating. Each switch maintains a count of the number of nonterminal circuit destructions it has

initiated. This count serves as a logical timestamp that, in conjunction with the identifier of the switch where the circuit was broken, uniquely identifies the circuit destruction event. The counter has to be sufficiently large (40-64 bits) so that there would be no danger of the counter ‘‘wrapping around’’ leading to possible incorrect packet ordering. When the torn down DVC is to be re-established, the stored destruction timestamp and the switch identifier are attached to the generated CEP. At the destination, whenever a circuit branch CEP arrives, a matching circuit branch CDP must be found with the same timestamp and switch identifier values. The only such CDP is the one terminating the branch to be ordered just prior to the CEP’s branch.

3.3. DVC Implementation: Overview

In this section we describe the basic hardware/firmware architecture that can be used to implement the DVC mechanism. This description includes the sequencing of low-level operations that are needed for the architecture to support establishing DVCs, forwarding packets on established DVCs, tearing down DVCs, and re-establishing torn DVCs. In Chapters 4 and 5, we describe further hardware requirements that are needed to eliminate deadlocks in DVC networks.

Figure 3.2 shows a multicomputer node with a host processor, local memory used by the host processor, a switch, and a special routing processor with its memory. The routing processor is a general-purpose processor which is used as a dedicated controller to perform some of the infrequent but complex operations that are needed to support DVCs. Frequent operations, such as routing and forwarding of a data packet on an established circuit, are handled entirely within the switch,

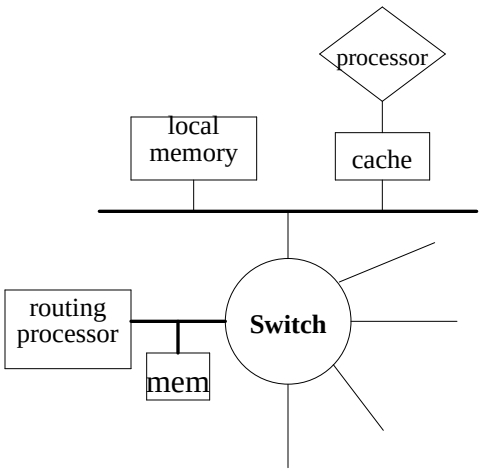


Figure 3.2: A multicomputer node.

using dedicated hardware. The routing processor handles tasks such as initiating circuit destruction, re-establishing a circuit, updating of global routing tables [Taji77], and resolution of deadlocks (Chapter 4). With current generation VLSI technology, it is possible to integrate the routing processor and its memory on the same chip with the communication switch.

To support DVCs, information which must be accessed for each data packet as it arrives or departs the switch is stored in lookup tables that are provided at switch input ports and output ports (see Figures 3.3 and 3.4 below). Less frequently accessed information can be stored in the private memory of the routing processor. The tables in the routing processor's memory may include the following:

- *Circuit Information Table (CIT)*: Contains one entry per input RVC at the switch. The entry identifies all the routing information required to re-establish a torn DVC. The information in a table entry is attached to a CEP that is generated to re-establish a torn circuit. If the timestamp mechanism is used to

handle re-ordering at the destination, then the timestamp value for a circuit can be stored in the corresponding entry.

- *Inverse Output Mapping Table* (INV): Contains one entry per output RVC from the switch. The table provides the inverse of the mapping provided by the Input Mapping Tables. Each entry maps an output RVC to the corresponding input port and input RVC. The mappings are recorded upon circuit establishment. Each entry also contains a single *reserved* bit, which the Routing Processor sets when the circuit is selected as a victim and resets when the teardown is completed. A circuit for which the *reserved* bit is set cannot be selected again as a victim. This rule is used to prevent redundant teardowns of a single circuit.
- *Routing Table*: Describes the routing function. Many organizations are possible. For example, the routing table may contains one entry for each possible destination node ID. Each entry contains the output port on the estimated shortest delay path to a destination node. This table is accessed during circuit establishment and can be updated dynamically [Taji77].

In addition to the tables above, for DVCs that originate at a node, the node maintains a single *Source Table*, which maps logical DVC identifiers to RVC IDs for the first hop of the DVCs (i.e., from the host to the local switch). At each switch, for DVCs whose destination is the node, there is a single *Destination Table*, which maintains the information (CEP and CDP timestamps) necessary for ordering packets arriving over different paths of the same DVC. Each packet originating from a node requires one access to the node's Source Table. Each packet delivered to the node requires one access to the Destination Table. Hence,

the switch's processor interface must support fast access to these tables.

As a concrete example, suppose a switch is implemented on a single chip and consists of four input ports, four output ports, a central crossbar, a Processor Interface (PIF) to the host processor, and the Routing Processor Interface (RPI). Each input or output port consists of a set of data signals (e.g., a data bus that is eight-bits wide) and one or more flow control signals. The input port uses a flow control signal to stop the output port from sending data when, for example, the buffer at the input port becomes full. The RPI translates read and write requests by the routing processor to both read/write operations on storage elements inside the switch and commands affecting the behavior of its modules. The RPI also fields interrupt requests raised by modules and passes them on to the routing processor.

3.3.1. Input Port Hardware and Operation

When a packet arrives to a switch input port, it is placed in the input buffer and routed to determine the desired output port. Once the packet reaches the head of the buffer, the buffer makes a request to the switch crossbar for a connection to the desired output port. After the request is granted, the packet is removed from the buffer and sent through the crossbar and the output port to the neighboring switch.

Figure 3.3 shows a possible block diagram of the hardware located at each input port of a hypothetical switch that supports the DVC mechanism. There are four main components in the figure: a Synchronizer [Tami88b], a Dynamically-Allocated Multi-Queue (DAMQ) input buffer to store arriving packets [Tami88a], an Auxiliary Buffer, and an Input Mapping Table (IMT). The DAMQ buffer has

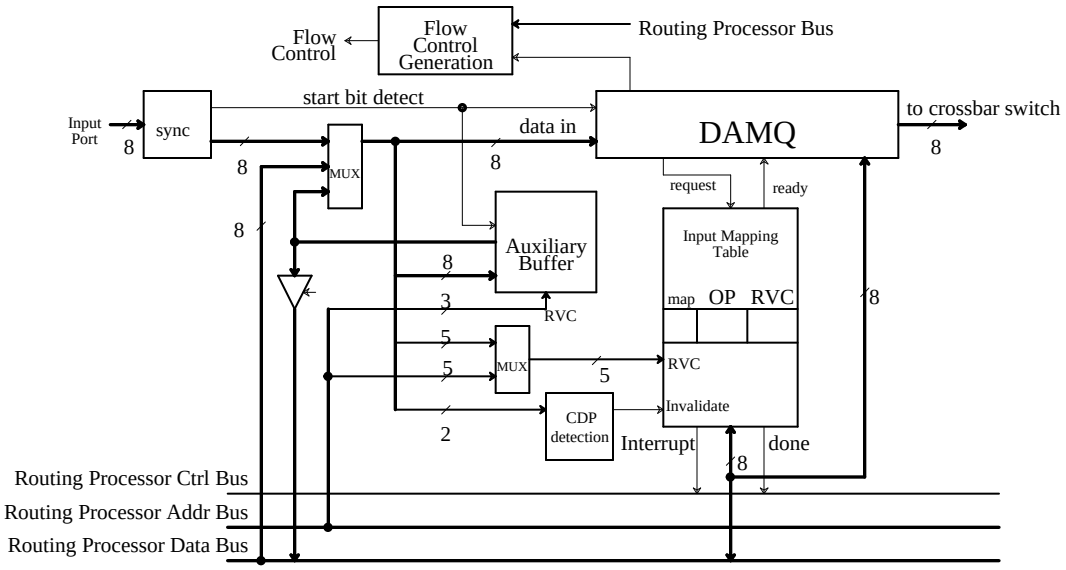


Figure 3.3: Input port routing hardware.

internal linked lists which allow separate forwarding for packets that are waiting for different output ports. This feature allows DAMQ buffers to avoid head-of-line blocking which limits network performance with the use of FIFO buffers. The Synchronizer produces data (8-bits wide in this example) synchronized to the local clock signal. These signals are input to both the DAMQ buffer, which is the main packet buffer at the input port, and to the Auxiliary Buffer, which is a much smaller FIFO buffer and normally stores a copy of the first several bytes of the most recent packet arriving through the input port. In normal operation, as packets arrive they are placed in both the DAMQ buffer and the Auxiliary Buffer. In addition, the RVC value in the header of each incoming packet is forwarded to the Input Mapping Table for lookup. If the lookup references a mapped entry, the header is modified to contain the output RVC ID, and the new header is latched into the DAMQ buffer. If the access references an unmapped entry, the Input

Mapping Table raises an interrupt for the routing processor and causes the DAMQ buffer control to use the flow control line to stop traffic into the input port. A packet arriving on an input RVC that has no mapping is either a Circuit Establishment Packet or a packet arriving on a circuit that has been disestablished from this switch. In either case, routing processor intervention is required and incoming packet flow must be stopped.

The DAMQ buffer normally takes its input from the output of the Synchronizer, but it can also take its input either from the Auxiliary Buffer or from a packet buffer located at the RPI via the routing processor data bus. The DAMQ input comes from the RPI when, for example, the routing processor needs to insert a CDP into the circuit. The DAMQ input comes from the Auxiliary Buffer in order to accept a CEP that was held in the buffer while the corresponding IMT entry was being set up. Flow from the input port is halted when the input to the DAMQ buffer is taken from one of the other sources. Packets arriving on the input port would otherwise be lost.

In some cases, the routing processor requires information contained in the body of an arriving packet. For example, when a CEP arrives, the header byte indicates the input RVC, and subsequent bytes of the packet indicate the packet's destination. To access these bytes, the routing processor can read the Auxiliary Buffer whenever it is not being written by the input port.

To tear down a victim circuit, the routing processor generates a CDP and enqueues it in the DAMQ buffer at the input port used by the victim circuit. In contrast, a CDP that arrives to an input port from a remote node can be handled without the intervention of the routing processor. To support fast handling of

CDPs at the input port, an *Invalidate* input signal is added to the Input Mapping Table. This signal causes the table lookup to mark the entry referenced as unmapped. Arrival of a CDP automatically triggers this operation.

3.3.2. Output Port Hardware and Operation

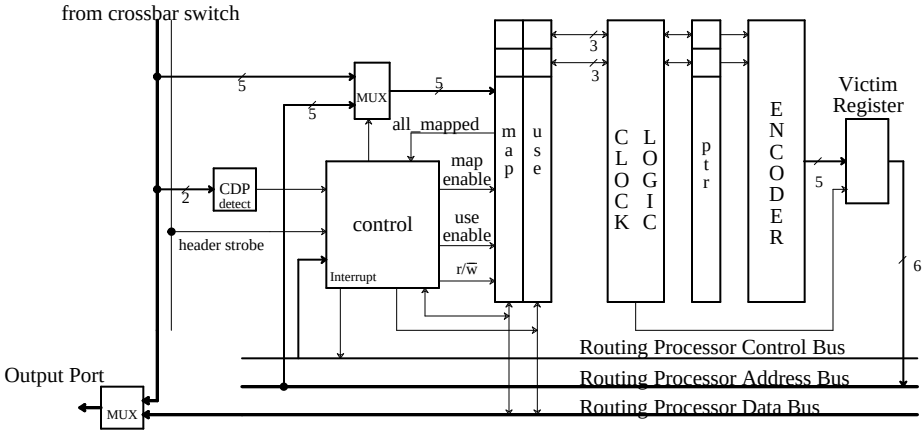


Figure 3.4: Output port logic. Invalidates circuits and picks victim output RVCs.

Figure 3.4 shows a block diagram of the hardware at each output port. The hardware consists of a table and logic for picking victim RVCs. The Output Port Table (OPT) is used to keep track of mapped and unmapped output RVCs and to maintain output RVC use information. The figure shows an example with 32 entries in the OPT, one per output RVC. The OPT is normally accessed when packets arrive at the output port from the crossbar — the entry corresponding to the RVC number of the packet is updated. In addition, the table is accessed by the routing processor when a DVC is established. Each table entry consists of two bits: *map* and *use*. The *map* bit specifies whether the output RVC is allocated to an established circuit. The *use* bit indicates whether a packet has been sent on the

corresponding output port recently.

The information in the Output Port Table drives the circuit that selects a victim when there is a need to find a free RVC for use in establishing or re-establishing a DVC through the output port. The victim selection module is a combinational circuit that continuously computes the victim output RVC ID. The victim RVC ID is placed in the Victim Register, which can be read by the routing processor. If there are any unmapped output RVCs (i.e., RVCs not allocated to established circuits), these are picked by the victim selection logic. If all the output RVCs are allocated to established DVCs, one of those RVCs is picked and the corresponding DVC is disestablished, starting from this switch. The ‘‘clock’’ replacement algorithm, commonly used for page replacement in virtual memory [Corb68], is used to pick the victim established DVC.

3.3.3. Sequencing of Switch Operations

Some of the operations performed by the switch involve several sequential steps. Proper ordering of events is crucial for avoiding inconsistent states. For example, in order to establish a new DVC, it is sometimes necessary to tear down an existing DVC to free an output RVC. A straightforward but incorrect procedure for performing this operation would have the routing processor reset the *map* bit for the victim RVC at both the IMT and at the OPT as soon as a victim is selected. Then, the routing processor would create a CDP and send it directly out the output port. Once the CDP is sent, the pending circuit establishment request would be fulfilled.

The procedure above is wrong because there may be packets that are using the

victim circuit and that are enqueued in the DAMQ buffer at the input port used by the victim circuit. If the mapping tables are changed and the CDP is sent before these enqueued packets exit the switch, the packets will be forwarded out the same output port and output RVC as the packets on the new circuit being established, thus mixing packets of different circuits. Also, one of the enqueued packets may be a CDP, rendering the creation of a CDP by the routing processor redundant.

The correct procedure incorporating these considerations is shown in Figure 3.5. This figure shows the sequence of low-level operations required when a data packet arrives. If the IMT entry is unmapped, the routing processor must re-establish the DVC to the circuit's destination. As shown in step 2 of Figure 3.5, the routing processor accesses the Circuit Information Table (Section 3.3) to determine the destination node ID for the data packet. Next, in steps 3 and 4, the routing processor picks a victim output RVC for use in the new DVC. To do this, it reads the Victim Register, which contains the selected RVC ID as well as the corresponding *map* bit. The routing processor then checks the Inverse Output Mapping Table, stored in its private memory, to ensure that this RVC is not already reserved for tear down. If the Inverse Mapping Table indicates that the selected RVC is reserved, the routing processor reads the Victim Register again to get a different victim.

Assuming that the output port table entry for the chosen victim is valid (i.e., the test in step 5 succeeds), the following procedure correctly tears down the victim circuit and re-establishes the circuit for the data packet:

1. The victim input port, IP′, and its input RVC ID, IC′, are read from the Inverse Output Mapping Table. The routing processor writes a command to the switch

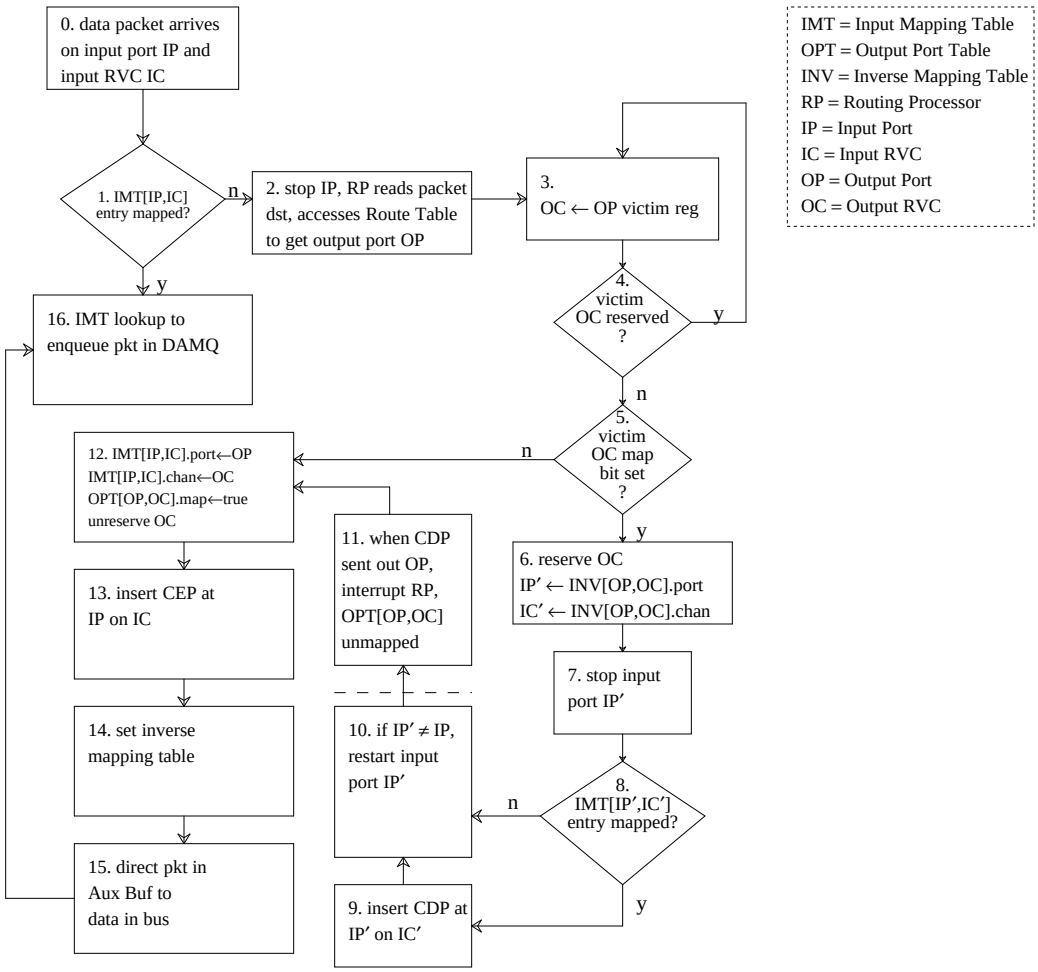


Figure 3.5: Handling of data packets arriving at an input port.

which causes it to assert the flow control signal to stop packet flow from the neighbor switch to the victim input port (steps 6 and 7).

2. If the IMT entry corresponding to the victim circuit is valid, this means that no disestablishment of the victim circuit is in progress at the local switch. If this is the case, the routing processor creates and enqueues a CDP at the victim input port DAMQ buffer and restarts packet flow (steps 8 through 10);
3. At this point (between steps 10 and 11), while the CDP is waiting for its turn in

the DAMQ buffer, the routing processor returns to its normal mode of waiting for interrupts from the switch. When the CDP is finally sent from the DAMQ buffer through the crossbar to an output port, the routing processor is interrupted again by the output port logic (step 11);

4. At this point the victim circuit has been torn down, freeing the victim output RVC for use by the DVC being re-established. The routing processor sets up the mapping tables for this DVC, creates a CEP, and inserts the CEP at the data packet's input port. Once this is done, the data packet can be directed to the DAMQ buffer (steps 12 through 16).

3.4. Summary

The Dynamic Virtual Circuits mechanism combines the benefits of adaptive routing and traditional static virtual circuits. The DVC mechanism minimizes the addressing and control information sent with each packet and the latency for packet forwarding at each switch. Unlike static virtual circuits, DVCs can adapt to congestion or failures in the system by allowing switches to make local decisions regarding the possible need to reroute the circuit through a different physical path. This chapter described the basic techniques for allowing circuits to be broken and reestablished while maintaining the semantics of traditional virtual circuits.

An overview was presented of the hardware necessary to support DVCs in the context of a complete communication coprocessor for multicomputer nodes. Dedicated hardware is used on-chip to handle the critical frequent case, while a routing processor is used for more complex but less frequent tasks.

The next two chapters describe techniques for eliminating deadlocks in DVC

networks. Chapter 4 describes a technique that uses deadlock detection and resolution, and Chapter 5 presents an alternative technique that uses deadlock avoidance. These two chapters additionally describe extensions to the hardware/firmware architecture that was described in this chapter that are needed in order to eliminate deadlocks in DVC networks.

Chapter Four

Deadlock Detection and Resolution in DVC Networks

In this chapter, we present a scheme for deadlock detection and resolution in multicomputer or cluster interconnection networks that use the DVC mechanism. The proposed scheme enables virtual circuits to be established on any network path from source to destination without imposing constraints for deadlock avoidance. In this scheme, a switch initiates a distributed deadlock detection procedure whenever packets stored in buffers at the switch fail to make progress in a timeout period, indicating the possibility of deadlock. If the deadlock detection procedure can identify a cycle of blocked resources in the network that constitutes a potential deadlock, then a deadlock resolution procedure is initiated that enables blocked packets in the cycle to make progress.

The scheme proposed in this chapter is an extension of a scheme developed by Jaffe and Sidi for deadlock detection and recovery in packet switching networks [Jaff89, Cido87]. In particular, we extend the original scheme to enable deadlock resolution in networks where switches use input buffers. In the original scheme, each switch is assumed to have a central buffer which it uses to store each arriving packet. This system model is not appropriate for high-performance multicomputer and cluster interconnection networks, for which it is generally more efficient to use separate buffers at each switch input port [Tami92].

We provide additional extensions to the original scheme to handle new types of dependencies which are introduced by the circuit manipulation operations in

DVC networks. The original scheme was not designed to take into account these new dependencies.

In Section 4.1, we briefly review the original deadlock detection and resolution scheme of Jaffe and Sidi for centrally-buffered packet switching networks. Section 4.2 presents our extensions to the original scheme for networks composed of switches that use packet buffers at each input port instead of central buffers. Section 4.3 describes the new types of dependencies which are introduced by the DVC mechanism. In addition, it presents a complete scheme for detecting and resolving deadlocks in a network with DVCs. In Section 4.4, we present a performance evaluation of the scheme. A detailed cycle-by-cycle simulation model of a network employing our deadlock detection and resolution scheme was implemented on a multi-threaded, object-oriented, event-driven simulation environment. We have used this simulation model to validate the correctness of the scheme and to evaluate its performance.

4.1. Deadlock Resolution in Centrally Buffered Packet Switching Networks

The original scheme proposed by Jaffe and Sidi in [Jaff89] can be used to detect and resolve deadlocks in packet switching networks that have the following properties: each switch uses a central buffer to store in-transit packets (as opposed to using input or output buffers); all network links are bidirectional; and the routing algorithm used in the network leads packets eventually to their destinations for delivery. When a central buffer becomes full and remains so for some time without transmitting any packets, it is possible there is a deadlock. When a switch decides it *may* be in a deadlock, it initiates execution of a distributed algorithm

which determines if there is a cycle of switches with full buffers, constituting a *potential* deadlock. In addition to finding all actual deadlock cycles, the algorithm may find cycles that are only temporary, in which the buffers are advancing slowly. Once the algorithm finds a cycle, the switches in the cycle initiate a deadlock resolution procedure called *rotation*. In rotation, one dedicated buffer at each switch in the cycle is introduced for temporary use. Each switch in the cycle uses the new buffer to forward exactly one packet one hop along the cycle. After a rotation is complete, the dedicated buffers are removed from use. Each rotation guarantees packet progress, and it may break a deadlock. If packets remain blocked in buffers after rotation, a new iteration of the cycle detection algorithm begins. The algorithm has low overhead, as it uses link bandwidth primarily by sending control messages over links that are otherwise blocked and unable to be used by normal packets.

Further details of this deadlock detection and resolution algorithm are presented in Section 2.1.5.1 and in reference [Jaff89]. In the remainder of this chapter we show how this algorithm can be modified for use in a network with an input-buffered switch model and with the Dynamic Virtual Circuits mechanism.

4.2. Deadlock Resolution in Input-Buffered Packet Switching Networks

We extend the original algorithm to enable deadlock detection and resolution in networks where each switch uses input buffers instead of the central buffers assumed for the original algorithm [Jaff89]. Our approach is to derive from the physical network an equivalent virtual network which has all the properties required by the original algorithm. The switches in the physical network execute a

procedure that emulates use of the original deadlock detection and resolution algorithm in the equivalent virtual network. All deadlocks in the physical network are detected and resolved, because this procedure detects and resolves the corresponding deadlocks in the equivalent virtual network.

The virtual network is described as a graph. Each node in the graph represents a virtual switch which uses a central buffer to store packets. Each virtual switch corresponds to a distinct switch input buffer in the physical network. The central buffer of a virtual switch is assigned the same capacity as the corresponding input buffer in the physical network. Each edge in the virtual network describes a virtual link between two virtual switches. An edge corresponds to a unique hop in the physical network that can be taken between an input buffer at one physical switch to an input buffer at a neighboring physical switch. Specifically, a directed edge is provided in the virtual network from a first virtual switch to a second virtual switch if and only if a packet in the input buffer corresponding to the first virtual switch can take a single hop to reach the input buffer corresponding to the second virtual switch.

This construction of the virtual network ensures that each hop taken by a packet in the physical network (virtual network) corresponds to exactly one hop taken by the packet in the counterpart virtual network (physical network). Therefore, input buffers in the physical network are in a deadlock if and only if the central buffers of the corresponding virtual switches in the virtual network are in a deadlock. Furthermore, each step taken by packets when resolving the deadlock in one network corresponds to exactly one step in the other network. We guarantee that all deadlocks in the physical network are detected and resolved by executing a

procedure that emulates use of the original deadlock detection and resolution algorithm in the equivalent virtual network.

Formally, the physical network can be described as an undirected graph $G = (V, E)$. $V = \{v_i\}$, i=0,1,2,...,$N-1$ represents the set of N physical switches. E is a set of undirected edges, where each edge represents a physical bidirectional link between two neighboring switches. The link between switches v_i and v_j is represented as the edge $e_{ij} = e_{ji} = \{v_i , v_j\} \in E$. At switch v_i, the input buffer for link e_{ij} has capacity C_{ij} for storing packets.

The virtual network $G' = (V', E')$ is constructed from the physical network G as follows. Each virtual switch in V' corresponds to a unique input buffer in physical network G. Virtual switch v'_{ij} corresponds to the input port, at switch v_i, for link e_{ji} from neighbor switch v_j. Therefore,

$$V' = \{v'_{ij}\}, \forall\, i,j \text{ such that } e_{ji} \in E.$$

Virtual switch v'_{ij} uses a central packet buffer with capacity C_{ij}, matching the capacity of the corresponding input buffer in the physical network.

Each edge in E' is a directed edge represented as an ordered pair of virtual switches. Each edge in E' corresponds to a unique hop in the physical network from an input buffer at one switch to an input buffer at an adjacent switch. If v_i and v_j are neighbor switches, i.e., $e_{ij} = e_{ji} \in E$, then each input buffer at v_i (except the input buffer for link e_{ji}) can send a packet in one hop to the input buffer for link e_{ij} at switch v_j. Therefore,

$$E' = \{(v'_{im} , v'_{ji})\}, \forall i,j,m \text{ such that } e_{ij} \in E, e_{im} \in E, \text{and } m \neq j.$$

For a simple example of constructing the virtual network from a physical

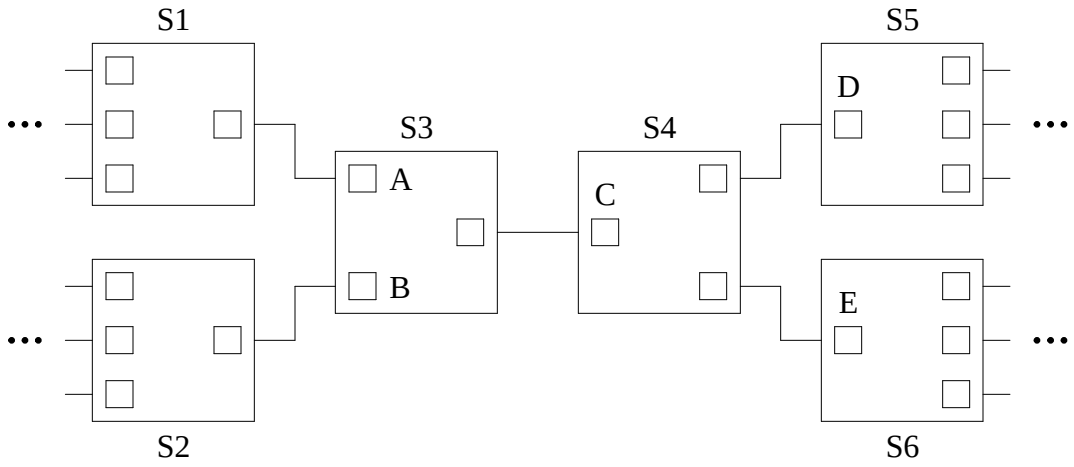


Figure 4.1: Partial physical network.

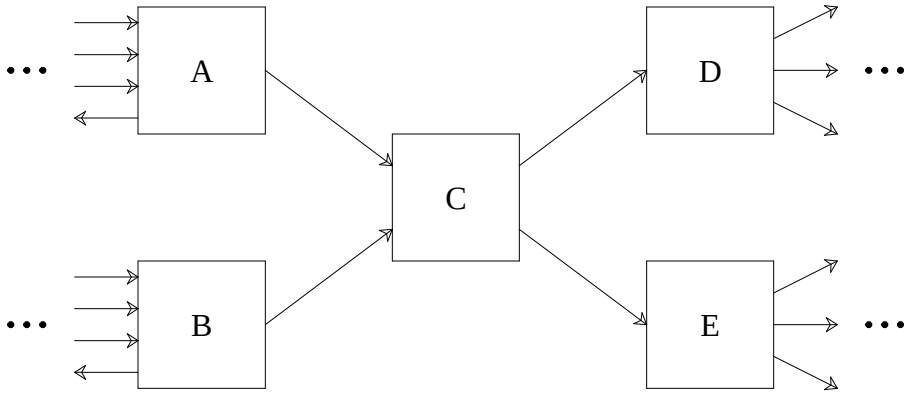


Figure 4.2: Partial virtual network.

network, consider the partial physical network shown in Figure 4.1. In this physical network, some of the input buffers are labeled with capital letters. Consider input buffer C at physical switch S4. This input buffer can receive only packets that are sent from switch S3. Furthermore, input buffers A and B are the only buffers in S3 that can buffer packets to be sent through S3's crossbar switch to buffer C. Once a packet is buffered at C, it may be sent through switch S4's crossbar switch to either switch S5 or switch S6. If sent to S5, the packet is

buffered at D. If sent to S6, the packet is buffered at E. Part of the virtual network corresponding to the physical network of Figure 4.1 is shown in Figure 4.2. Here, each virtual switch is labeled with the same capital letter used to identify the corresponding physical input port buffer in Figure 4.1. All the inputs and outputs of virtual switch C are shown, along with the fan-in and fan-out of virtual switches A, B, D, and E. Virtual switches A and B can send packets to virtual switch C, which in turn can send to virtual switches D and E, which is the same behavior as the corresponding physical input port buffers. Note that, although not shown in Figure 4.2, virtual switch A can also send to the virtual switch corresponding to a physical input buffer in switch S2, and similarly virtual switch B can send to a virtual switch corresponding to an input buffer in S1. The key point is that since the virtual network consists of switches that use central buffering, it can be used for deadlock resolution as long as it conforms in all other ways to the original algorithm's system model.

The only difference between the two models is that the deadlock resolution algorithm requires control messages to be able to travel in either direction over a link, whereas the edges in the virtual network graph are unidirectional. To correct for this difference, we add the following edges to the virtual network graph for the exclusive use of control messages that travel opposite to the direction of packet flow:

$$\{(v'_{ji}, v'_{im})\}, \forall\, i, j, m \text{ such that } e_{ij} \in E,\ e_{im} \in E, \text{ and } m \neq j.$$

In effect, this makes all the links in the virtual network bidirectional, at least for control packets. These reverse edges can be added to the virtual network since each reverse edge connects two virtual switches that correspond to input ports at

adjacent switches in the physical network. Sending control packets on additional edge (v'_{ji}, v'_{im}) in the virtual network graph corresponds in the physical network to sending the message on link $e_{ij} \in E$ in the direction from switch v_j to switch v_i (the physical link is bidirectional). For example, referring to Figures 4.1 and 4.2, one of the reverse virtual edges that we add is from virtual switch C to virtual switch A. To send control packets along this new virtual edge, in the physical network a control message will be sent on the bidirectional link from switch S4 to switch S3. The message will contain information that identifies the message source as virtual switch C and the message destination as virtual switch A. Switch S3 will use this information to determine how to interpret the control message in the context of the deadlock detection and resolution algorithm.

It is straightforward to execute the original resolution algorithm as though it were running on the virtual network instead of directly on the physical network. The original deadlock resolution algorithm uses sequence numbers on each control message to distinguish different iterations of the algorithm. Each switch maintains a sequence number that indicates its count of which iteration it is performing. With input buffering, it is necessary to maintain multiple independent sequence numbers, one for each virtual switch mapped to the physical switch (i.e., a sequence number is needed for each input port). Also, it is necessary to send along with each control message enough information to identify for the receiver which virtual switch created the message. This is simply an input port number in the physical network. In the case of control messages that traverse the reverse direction of a link in the virtual network, the destination virtual switch must be identified as well, since the destination could correspond to any of the input ports

at the destination physical switch. Whether this information is present depends on the type of the control message, since all messages of a single type travel in the same direction along virtual network links. Finally, the set of state variables stored at each switch in the original algorithm must be allocated for each virtual switch mapped to a single physical switch (i.e., for each input port).

This section dealt with one fundamental difference between Dynamic Virtual Circuit networks and the network model assumed by the original deadlock resolution algorithm. We showed how to derive from the physical network an equivalent virtual network which is structurally equivalent to the network model of the original deadlock resolution algorithm. By running the algorithm on each virtual switch instead of on each physical switch, the model differences are reconciled. These extensions are sufficient to detect and resolve deadlocks in networks that use pure packet switching. For networks that use the DVC mechanism, however, additional dependencies must be handled, as we describe in the next section.

4.3. Deadlock Resolution With Dynamic Virtual Circuits

The use of Dynamic Virtual Circuits instead of pure packet switching introduces new dependencies which can cause deadlocks that do not correspond to cycles in the virtual network graph. As a result, these deadlocks cannot be handled by simple cycle detection and rotation. In this section we describe the new dependencies introduced by the DVC mechanism and present extensions to the deadlock detection and resolution algorithm to handle the new dependencies.

In a pure packet switching system, a packet can be blocked because the packet

buffer at the next switch is full. If FIFO packet buffers are used, ‘‘head of line’’ (HOL) blocking can cause a packet directed to one output port to be blocked waiting for a packet ahead of it in the queue which is directed to a different output port. In a DVC system, a packet can additionally be blocked because it is unmapped and requires an output Routing Virtual Channel (RVC) to be allocated to it before it can proceed. In order to allocate an RVC, the switch may have to tear down another virtual circuit from a different input port than the unmapped packet. In this case, there is a new dependency from the input port that stores the unmapped packet to a second input port (at the same physical switch) which stores the Circuit Destruction Packet (CDP) that is introduced to tear the victim circuit. Since the virtual network graph has no edges between virtual switches corresponding to different input ports at the same physical switch, the new dependency maps to a dependency in the virtual network between virtual switches that are not neighbors. As a result, there can be a deadlock cycle that does not involve a cycle of virtual switches in the virtual network’s topology. Thus, this dependency introduced by the DVC mechanism violates the deadlock model assumed for the cycle detection and rotation procedures in the original algorithm of Jaffe and Sidi [Jaff89].

The following sections present a solution for detecting and resolving deadlocks in a DVC environment. Section 4.3.1 describes in detail the new dependencies introduced by the DVC mechanism. In addition, Section 4.3.1 presents a technique for avoiding deadlocks in DVC networks for the special case in which the underlying routing algorithm does not have buffer dependency cycles. For the general case of a DVC network in which the routing algorithm can

introduce buffer dependency cycles, Section 4.3.2 presents our extensions to the deadlock detection mechanism, and Section 4.3.3 presents our extensions to the deadlock resolution mechanism.

4.3.1. Deadlock Avoidance in DVC Networks with Dependency Cycle-Free Routing

In a pure packet switching network, if the routing algorithm does not introduce buffer dependency cycles, then the network is deadlock-free. DVC networks should also have this property, because it allows dispensing with deadlock detection and resolution operations when dependency cycle-free routing algorithms are used, and because it reduces the number and probability of deadlock scenarios when the routing algorithm does have dependency cycles. However, new dependencies introduced by unmapped data packets create the danger of deadlocks in DVC networks even with dependency cycle-free routing policies such as dimension-order routing in a mesh. To completely avoid deadlocks in such an environment, we must guarantee that the new dependencies cannot persist indefinitely. As described below, this can be done by guaranteeing that the CDPs created to tear victim circuits will be sent eventually.

For a specific example of the problem, consider a DVC network in which switches have FIFO input port buffers. Figure 4.3 shows an example deadlock scenario with two switches in a DVC network. In this figure, two unmapped data packets, U_1 and U_2, are waiting in Auxiliary Buffers for RVCs to be allocated to them. As described earlier in Section 3.3.1, the Auxiliary Buffer is a buffer which holds at most one packet and is used for holding an unmapped data packet until a

valid circuit can be established for it. When the Auxiliary Buffer is occupied, other packets are prevented from arriving at the input port from the physical neighbor switch. Once unmapped data packets are given valid circuits, they can be enqueued in the input port FIFO buffer. CDP_1 is created to free an RVC for U_1. U_1 cannot proceed until CDP_1 proceeds; this is the new dependency introduced by the DVC mechanism. Similarly, at the other switch, U_2 waits for CDP_2.

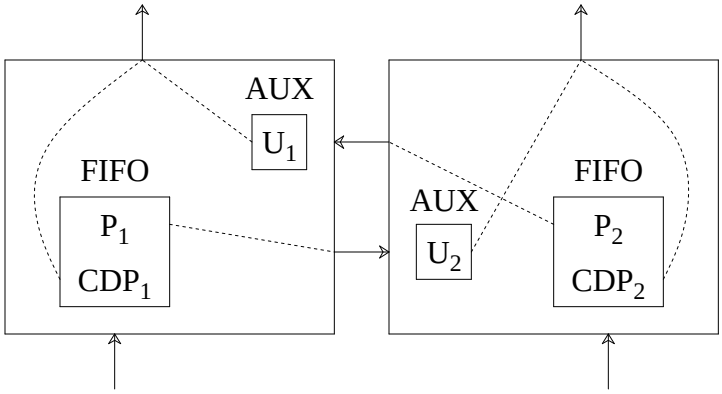


Figure 4.3: Deadlock with FIFO buffers

CDP_1 is blocked by P_1 ahead of it in the FIFO buffer. Similarly, CDP_2 is blocked by P_2. P_1 and P_2 are blocked by U_2 and U_1, respectively. Because of the ‘‘head of line’’ (HOL) blocking of the FIFO buffers, no packet shown in the figure can advance, and there is a deadlock in the physical network that does not correspond to a cycle in the virtual network.

One way to avoid this problem is to ensure that the new dependency never arises. A mechanism requiring hardware support could be employed which ensures that the circuit chosen for destruction is not being used by any packets currently in the switch. This removes the requirement that the CDP created to tear down the circuit needs to be appended to an input buffer at a different input port. Instead,

the CDP can be forwarded directly out the appropriate output port without waiting for other packets at the switch. By eliminating the new dependency between non-neighbor virtual switches, the network remains deadlock-free. The drawback of this approach is that it requires extensive hardware support for keeping track of how many packets at the switch are using each virtual circuit, and for ensuring that victim selection is restricted to selecting virtual circuits for which no packets are present at the switch. In general, such a virtual circuit may not exist, if the number of RVCs on a link is less than the number of packets that can be buffered at a time at a switch. Therefore, this approach requires disallowing such configurations.

We take a different approach that avoids this hardware complexity and allows any circuit to be chosen as victim (hence, the new dependency may exist). In this approach, we replace the FIFO buffers with Dynamically Allocated Multi-Queue Buffers (DAMQ Buffers) [Tami92], which use multiple internal queues in each buffer to eliminate HOL blocking. The use of DAMQ buffers renders the new dependency merely temporary. For example, if the FIFO buffers in Figure 4.3 were replaced with DAMQ buffers, then the CDPs in the figure would not be blocked by packets P_1 and P_2 (i.e., there would be no HOL blocking). The unmapped packets would still have to wait for the CDPs at different input ports to be transmitted, but with DAMQ buffers, that is equivalent to waiting for the buffer at the next switch to have free space, which is no different than in the case of pure packet switching. Thus, deadlock is avoided when dependency cycle-free routing is used as long as the CDP can be placed into a free slot in the the DAMQ buffer at the victim circuit's input port.

If the DAMQ buffer is full when the CDP is created, there are two

possibilities. If the DAMQ buffer has no packets for the CDP's desired output port, then no packets in the DAMQ buffer are using the victim circuit. Therefore, in this case the CDP bypasses the DAMQ buffer and is transmitted directly to the output link, eliminating the new dependency between the two input buffers at the same switch. The other case is where the DAMQ buffer is full and does have packets for the same output port that is desired by the CDP. In this case, the CDP must wait for all of these packets to be transmitted, because some of these packets may be using the victim circuit. Thus, the CDP is buffered in fixed-sized queues of the Routing Processor's private memory that are reserved for extending the storage of the DAMQ buffers. One Routing Processor queue is maintained for each internal queue of the DAMQ buffer. The packets in the Routing Processor's queue wait to be inserted at the end of the DAMQ buffer when it frees space. Before the CDP (or any other packet) is placed in the appropriate Routing Processor queue, the corresponding input port is blocked from receiving additional packets from the neighbor switch. Flow from the neighbor is resumed only after all the Routing Processor's queues for the input port are empty and the DAMQ buffer has free space. With this mechanism, transmission of the CDP depends once again only on the eventual availability of the buffer at the next switch. Therefore, the dependency between the two input buffers at the same switch is only temporary, and deadlock is avoided.

The storage required for the Routing Processor's queues is determined by the worst case scenario in which all n input ports of a switch have unmapped packets to send. Then, there are at most n CDPs created by the local switch to release RVCs, and there are at most n CEPs created to establish or re-establish the

mappings for the unmapped packets. If the routing algorithm is dependency cycle-free, then the only packets that will be stored in the Routing Processor's queues are these locally created CDPs and CEPs. As a result, these queues must have sufficient capacity to store $2n$ packets.

The techniques described in this section eliminates all deadlocks if the routing algorithm does not have cyclic dependencies. With more general routing algorithms, deadlocks involving DVC dependencies can still occur. The following sections describe techniques for detecting deadlocks and for resolving them by breaking the extra DVC dependencies and then performing cycle rotation.

4.3.2. Deadlock Detection Phase

Deadlock detection is similar for networks with DVCs and for networks with pure packet switching. With packet switching, the deadlock detection phase is triggered when buffers are full and block for some time. With DVCs, the buffers need not be full since deadlocks can occur due to DVC dependencies. Therefore, deadlock detection begins when a buffer is found whose packets have not made progress recently. The second step is to discover a cycle of blocked buffers whose packets are waiting for one another to advance. That is accomplished by executing a distributed algorithm in which switches send control messages to their immediate neighbors. Hardware support is necessary to send control messages even when packet buffers are full.

Identification of a blocked buffer, the first step of deadlock detection, is accomplished by having the switch periodically check whether some buffer in the switch has become blocked since the last check. A status bit is associated with

each buffer in the switch. Periodically, the switch sets the status bit for each buffer that contains a packet and then allows a waiting period to elapse. During the waiting period, whenever a packet is moved from a buffer, the status bit associated with the buffer is cleared, to indicate that the buffer is not blocked. Once the waiting period elapses, the switch checks the status bits. Any that are still set must belong to buffers whose packets have not been able to move during the waiting interval. The first such buffer found is declared to be ‘‘Blocked.’’

Once a blocked buffer is found, the deadlock detection algorithm is triggered. A blocked buffer may be caused by a deadlock or by ordinary processing or queuing delays. An output port for which a packet in the blocked buffer is waiting is chosen as the next hop along a suspected deadlock cycle. The algorithm proceeds with cycle detection by sending a TEST control message out that port. A separate flow control mechanism is required to send control messages, since often when control messages are sent the receiving input ports are unable to receive normal traffic (perhaps because they are in a deadlock). One way to accommodate the control messages is to use the Auxiliary Buffer to store them during execution of deadlock detection. A switch that detects its input port is blocked causes any packet in the Auxiliary Buffer to be brought into the Routing Processor’s private memory, thereby freeing space in the Auxiliary Buffer for control messages. To regulate the flow of control messages, two flow control signals could be used on each communication link. They indicate to the neighbor whether regular packets or control messages can be transmitted. Alternatively, dedicated flow control signals can be omitted by transmitting flow control information on the data lines as piggybacked information on packets going in the opposite direction. For this to

work it must be possible to transmit the flow control information even when the receiving input port is unable to receive and buffer packets.

One possible outcome of cycle detection is the failure to locate a cycle, in which case there is no deadlock and the algorithm terminates. The other possible outcome is that a cycle is found, in which case cycle rotation is committed to by all members of the cycle, and rotation can then proceed.

4.3.3. Cycle Rotation: Handling DVC Dependencies

In this section, we present two approaches to cycle rotation in a DVC environment. In the first approach, described in Section 4.3.3.1, some specific RVCs are reserved for breaking DVC dependencies for deadlock rotation. Packets arriving on these reserved RVCs are handled specially in a relatively slow procedure. The advantages of this scheme are that it can be implemented mostly in software by a Routing Processor attached to each switch in the network that also controls the complex circuit manipulation operations. Also, the scheme does not require any additional functionality at the receiving host interface over what is already necessary for DVC processing. In the second approach, described in Section 4.3.3.2, unmapped packets that are rotated have additional bytes attached to them (growing their length) to provide: 1) addressing information allowing the packet to be forwarded via pure packet switching to the destination, and 2) sequencing information that allows the endpoint receiving node to properly order the packet within the stream of packets on its virtual circuit. Compared to the first approach, this alternative scheme has the advantages of simplicity and speed but requires extra storage in the input buffers of each switch and extra

complexity at the receiving host interface because the scheme introduces a new packet type that the interface must process.

4.3.3.1. Rotating Unmapped Packets Using a Dedicated RVC

In the first approach we present for cycle rotation in DVC networks, each switch maintains one dedicated RVC at each output port which is used during rotation to break the DVC dependencies, which would otherwise block rotation. In cycle rotation, each unmapped packet that needs to be rotated can allocate the dedicated RVC instead of waiting for circuit teardown to free up a normal RVC. After rotation, the dedicated RVC is freed so that it can be used in a subsequent rotation of some other unmapped packet. In the following discussion of this technique, the terms ‘‘virtual switch’’ and ‘‘input port’’ are used interchangeably.

When a virtual switch detects that it is the leader of a deadlock cycle, it executes the procedure in Figure 4.4 to prepare to perform rotation. The procedure sets a virtual switch state variable, *mode*, to indicate that the virtual switch is preparing for rotation. The procedure disables normal flow of packets from the physical neighbor switch to the input port, and disables circuit manipulation operations that would interfere with the state of the input port. These are left disabled until cycle rotation is aborted or complete. If there is a packet in the input port Auxiliary Buffer, it is transferred to the ‘‘Alternate Auxiliary Buffer’’ in the Routing Processor’s private memory. Finally, the virtual switch sends a CYCLE control message to the next virtual switch in the cycle. Each other virtual switch along the detected deadlock cycle executes the same procedure of Figure 4.4 upon receiving a CYCLE control message.

```
mode := CYCLE_MODE
disable packet flow from physical neighbor into the input port
active := active + 1  ; disables circuit operations
if currently unmapped packet is in the Auxiliary Buffer {
    transfer packet to Alternate Auxiliary Buffer
}
send a CYCLE control message to successor switch
in the detected deadlock cycle
```

Figure 4.4: Procedure executed by a virtual switch to prepare for rotation

CYCLE messages traverse the entire ring. When the leader switch receives a CYCLE message, all switches in the cycle have prepared for rotation. The leader then proceeds to select and forward a packet to the next switch in the cycle.

The packet chosen by the virtual switch for forwarding is the one that is first in line to be sent to the successor from the input port. The buffers for an input port are checked for packets in the following order: the DAMQ buffer, the queue in Routing Processor private memory that extends the DAMQ buffer, and the Alternate Auxiliary Buffer. If no packet is found, then a ‘‘dummy’’ packet is rotated to the next switch, where it is discarded.

If the switch finds a packet to rotate, it executes the procedure shown in Figure 4.5. A mapped packet is rotated using the normal packet forwarding mechanism. The procedure for rotating an unmapped packet is more complicated. An unmapped packet is assigned to use the dedicated RVC on the output port before it is rotated. A CEP is created to allocate the RVC for the packet, and a CDP is allocated to free it. If the unmapped packet is itself a CEP, then only the CDP is created; likewise if the unmapped packet is a CDP, then only the CEP is created. The created CEP and CDP each contain the current switch ID and a ‘‘timestamp’’. As described in Section 3.2, this information allows the destination

node to sequence packets that belong to the same flow in FIFO order. Each switch maintains a local timestamp value, which is incremented after each circuit destruction event and after each conversion of an unmapped packet to a sequence of packets that uses the dedicated RVC. Incrementing the timestamp value in this manner ensures that these are uniquely identifiable events; a CEP and CDP with matching teardown ID and timestamp values must be consecutive packets in the circuit stream, even if they do not arrive consecutively or even in the logically correct order [Tami91]. The resulting group of packets (CEP, data packet, and CDP) is then forwarded.

```
if packet is mapped {
    rotate it to successor switch
} else {
    allocate the dedicated RVC to the packet
    if packet is a CEP initializing a new circuit {
        record address information in Circuit Destruction Table
    if packet is not a CEP {
        create a CEP to establish mapping
        rotate CEP
    }
    rotate the packet
    if packet is not a CDP {
        create a CDP to release mapping
        rotate the CDP
    }
    timestamp := timestamp + 1
}
```

Figure 4.5: Packet rotation procedure using dedicated RVC

Creating and forwarding a CEP and a CDP to rotate an unmapped data packet introduces the possibility of unbounded growth in the number of packets in the network. To prevent this, the network treats the three packets as an atomic group of packets. The group travels uninterrupted on each hop toward the destination. If the group becomes part of a subsequent deadlock, it will be rotated as a single unit, without triggering the creation of new CEPs or CDPs.

On each rotation a switch participates in, a group of up to three new packets enters the switch, and a group of up to three packets leaves the switch. The total number of such packet groups that are present in the switch cannot increase in a series of rotations. At worst, each rotation results in no change in the total number of packet groups present, and on some rotations the number of groups may go down if a dummy packet is rotated in. Since the number of packet groups present in the switch before deadlock first occurs is bounded, and on each rotation the number of packet groups does not increase, the storage requirement for this mechanism is finite and equal to the capacity needed for normal operation plus the extra capacity needed to hold a CEP and a CDP for each packet when the storage for normal operation is depleted.

To treat rotated groups of up to three packets as a single unit, each switch that receives such a group forwards it on the dedicated RVC of the appropriate output port. The packet scheduling mechanism in the switch (i.e., the crossbar arbitration) must ensure that once a packet is transmitted on the dedicated RVC, the same input is granted exclusive access to the output until it transmits a CDP on the dedicated RVC. During normal operation, whenever a packet is received on the dedicated RVC, the Routing Processor is interrupted and the packet is placed in the Routing Processor private memory allocated for that virtual channel. Once a CDP arrives using the dedicated RVC, all the packets stored for the RVC are routed to an output port, are allocated the dedicated RVC on that output port, and then enqueued for transmission. The procedure for handling the arrival of each rotated packet P is summarized in Figure 4.6.

When a virtual switch that has committed to perform rotation receives a

```
if P is a dummy packet {
    discard P
} else if P uses the dedicated RVC {
    transfer P to Routing Processor private memory
    while P is not a CDP {
        wait for a new packet P to rotate in
        transfer new P to Routing Processor private memory
    }
    determine output port OP for the saved packets
    map saved packets to the OP's dedicated RVC
    enqueue packets in DAMQ buffer or in DAMQ buffer extension
} else if P is mapped or there is a CEP waiting to map P {
    if P and the Alternate Auxiliary Buffer packet AAB are
    of the same virtual circuit {
        enqueue AAB packet in DAMQ buffer or extension
    }
    enqueue P in DAMQ buffer or extension
} else {
    if P and the AAB packet are of the same virtual circuit {
        allocate the dedicated RVC to AAB
        create CEP and/or CDP for AAB
        enqueue CEP, AAB, and CDP in DAMQ buffer or extension
    }
    allocate the dedicated RVC to P
    create CEP and/or CDP for P
    enqueue CEP, P, and CDP in DAMQ buffer or extension
}
active := active - 1   ; normal operation if active=0
```

Figure 4.6: Handling rotated packet arrival with the dedicated RVC approach

rotated packet from the predecessor virtual switch in the cycle, it checks whether it is the leader of the cycle. If so, then the rotation will be complete after the virtual switch handles the rotated packet or packet group according to the procedure in Figure 4.6. Otherwise, the virtual switch is not the leader and must rotate one of its packets or packet groups to the successor virtual switch in the cycle by executing the procedure in Figure 4.5. To maximize concurrency in the distributed rotation procedure, the virtual switch executes the procedure in Figure 4.5 before it invokes the procedure in Figure 4.6 to handle the arriving rotated packets.

4.3.3.2. Rotating Unmapped Packets Using a New Packet Type

An efficient alternative to the dedicated RVC technique described in the previous section is to introduce a new packet type that is routed using pure packet switching and which is used to break DVC dependencies upon rotation. An unmapped packet that needs to be rotated is first converted into this new packet type by adding a few bytes to the header. The new bytes identify the packet as being of the new type and also contain the same information (circuit addressing information, teardown switch ID, logical timestamp) that is present in CEPs created in the dedicated RVC technique. The Auxiliary Buffer needs to be made large enough to hold the maximum sized packet that is modified in this manner. In addition, in normal operation the flow control signaling must not allow a packet to be sent to the neighbor until that neighbor's DAMQ buffer has sufficient space to store a maximum sized packet that has been converted to this new type. The procedure for handling a rotated packet P with this technique is shown in Figure 4.7.

With this technique, there is no need for a special RVC, there is no need to create new CEPs and CDPs to handle unmapped packets, and the procedure for handling rotated packets is simplified compared to the procedure in Figure 4.6. However, hardware support is required to handle the new packet type.

```
if P is a dummy packet {
    discard P
} else if P is of the new packet type {
    determine output port needed by P
    enqueue P in DAMQ buffer or extension
} else if P is mapped or there is a CEP waiting to map P {
    if P and the Alternate Auxiliary Buffer packet AAB are
     of the same virtual circuit {
        enqueue AAB in DAMQ buffer or extension
    }
    enqueue P in DAMQ buffer or in DAMQ buffer extension
} else {
    ;; P is unmapped and not of the new packet type
    if P and AAB are of the same virtual circuit {
        convert AAB to the new packet type
        enqueue AAB in DAMQ buffer or extension
    }
    convert P to the new packet type
    enqueue P in DAMQ buffer or DAMQ buffer extension
}
active := active - 1   ; enables normal operation if active=0
```

Figure 4.7: Handling rotated packet arrival with the new packet type approach

4.4. Performance Evaluation

To investigate the behavior of our deadlock detection and resolution scheme for DVC networks, we developed a simulation testbed which provides cycle-level emulation of the algorithm described in this chapter. The simulator models the details of packet forwarding and circuit manipulation when deadlocks do not occur, but it makes the simplifying assumption that control messages sent by the switches executing the deadlock detection and resolution algorithm do not interfere with ordinary traffic. Each control message traverses a physical link in a fixed 16 clock cycles. This assumption is justified since control messages in a reasonable system are at most a small fraction of the overall traffic, and furthermore many control messages are sent over links that are not usable by ordinary packets because of the

congestion that triggers the deadlock detection algorithm. For the latter control messages, the simplifying assumption made for the simulator has no effect on the measured performance.

The particular network simulated is a two dimensional square mesh of 36 nodes. Extensive simulation revealed that with fixed-path routing policies that exhibit many dependency cycles, deadlocks are extremely rare under light load, but under heavier load deadlocks are plentiful. Under heavy load, the deadlock resolution algorithm is unable to resolve deadlocks faster than new packets are injected into the network to cause new deadlocks. Therefore, once the first deadlock occurs, and cycle rotation begins, the network remains forever in ‘‘deadlock mode,’’ continually rotating packets around detected deadlock cycles. Since deadlock detection and rotation is relatively slow compared to normal operation, the resulting network latency is high and throughput is very low. The conclusion of these investigations is that for deadlock resolution to be effective, the traffic conditions that give rise to deadlocks must be infrequent.

To further investigate the behavior of this deadlock resolution technique, a traffic pattern was devised to cause deadlocks only occasionally. In these experiments, the routing algorithm is set so that all destination nodes except two are reached via row-first routing. The two remaining nodes are reached via column-first routing. The two remaining nodes are at opposite corners of the mesh, with one node at row 2 column 5, and the other node at row 5 column 2. Most of the time, the destination distribution is uniform, and the load is light (using only 10% of the mesh’s bisection bandwidth). However, periodically, a burst of traffic lasting for 10000 clock cycles at high load (60% of bisection bandwidth) and a

non-uniform destination distribution occurs. The destination distribution is chosen so as to make deadlocks very likely. In particular, the two nodes at the opposite corners that are routed column-first send all their injected packets to each other (column-first), and the nodes at the other two corners (at row 2 column 2, and at row 5 column 5) send all of their injected packets to each other. As shown in Figure 4.8, this results in a cyclic traffic pattern that makes deadlock very likely during the burst. Note that since all nodes route to the two corner nodes column-first, other deadlocks besides the cycle shown in the figure can occur in this system.

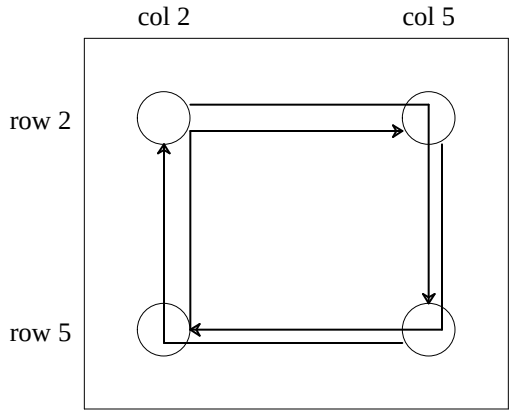


Figure 4.8: Mesh with cyclic routing pattern

Once the burst of heavy deadlock-prone traffic is over, deadlocks persist in the network until the deadlock resolution algorithm can clear the network of enough packets that normal operation can proceed. Figure 4.9 shows the time required after the burst period ends to completely recover from the effects of the burst, versus the number of clock cycles between subsequent checks by each routing processor for input ports that are not making progress.

The graph shows that for both very large and very small timeout periods,

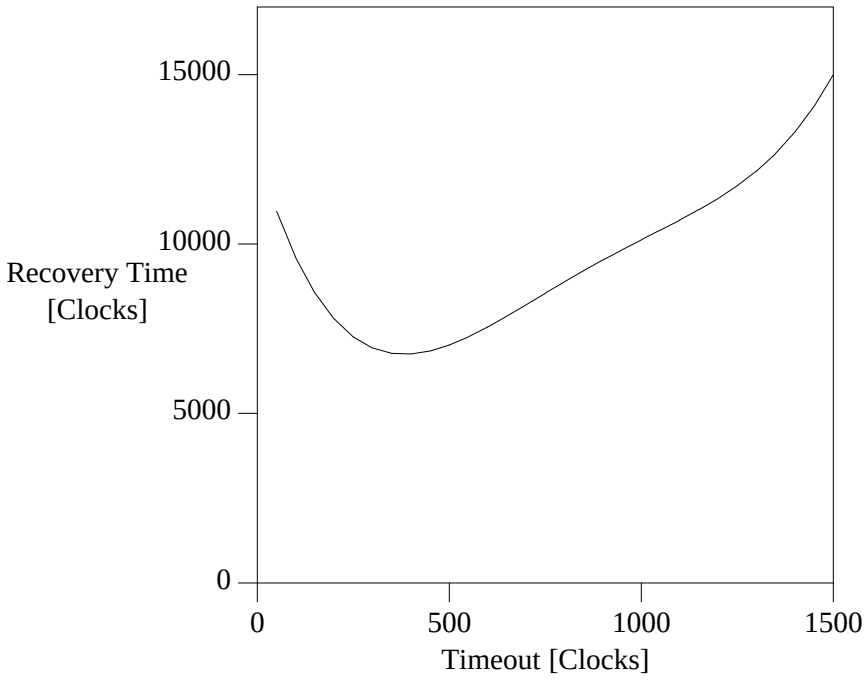


Figure 4.9: Recovery time versus timeout period

recovery time is long. With a long timeout period, recovery time is long because deadlocks can persist for a long time before the detection algorithm is triggered by a timeout. Packets can be blocked in deadlock for a long time without the system making any attempt to force them to advance through cycle rotation. Understanding why small timeout values cause long recovery times requires more explanation.

Figure 4.10 shows the average number of times over ten runs of ten bursts each that switches declare themselves blocked and later return to an unblocked state without first participating in a cycle rotation. These switches declare they are blocked, participate in the deadlock detection algorithm, and then find that they are not in a deadlock because they are able to send a packet before identifying a

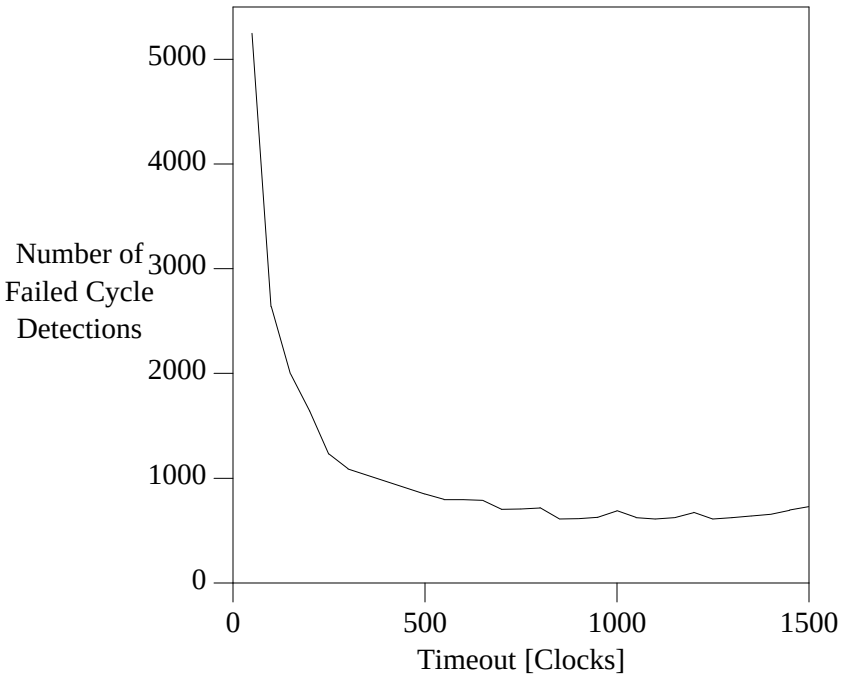


Figure 4.10: Number of failed cycle detection attempts versus timeout

deadlock cycle.

The frequency of such failed detection attempts has an effect on the time needed to discover deadlocks because of its impact on the range of ‘‘sequence numbers’’ in simultaneous use in the network by switches participating in the detection and resolution algorithm. The algorithm uses sequence numbers to distinguish between different iterations. This prevents deadlock cycles from being missed due to the arrival of obsolete information [Jaff89]. This means that for a deadlock cycle to be detected, all the switches in the deadlock cycle must agree on the current sequence number. As the graph shows, when the timeout value is very low, switches time out frequently without detecting a deadlock. This happens because transient congestion that does not constitute deadlock is likely to cause

timeouts when the timeout value is small. Each timeout causes the node that timed out to increment its local sequence number by one. Because switches are timing out at random times and incrementing their sequence numbers at different rates, once a deadlock happens, the switches in the deadlock cycle can have significantly different sequence numbers and therefore require a time-consuming synchronization. When the timeout period is long, many fewer failed deadlock cycle detections occur. Those that do occur are mostly at switches that are in deadlock but are not in the deadlock cycles at the ‘‘core’’ of the deadlocks. The switches must wait for the deadlock cycles to be rotated before they can advance. With larger timeouts, since failed attempts are rare, the sequence numbers of the switches in actual deadlocks are relatively close together when deadlock detection begins, resulting in lower synchronization costs.

4.5. Summary

Schemes that prevent deadlocks in buffered packet networks restrict the use either of buffer resources or of network routes, whereas deadlock detection and recovery enables maximum flexibility in the use of network resources. Based on a scheme for recovering from deadlocks in centrally-buffered packet switching networks, this chapter presented a technique for deadlock detection and resolution in a network using input-buffered switches and the Dynamic Virtual Circuits mechanism. Input buffers are accommodated by mapping the physical network to an equivalent virtual network that uses central buffering.

Dynamic Virtual Circuits support efficient adaptive routing in networks of arbitrary topology. However, the DVC mechanism introduces a new class of

buffer dependencies not present with conventional packet switching. These additional buffer dependencies have the potential for causing deadlocks. We have shown how to design the mechanism such that when the underlying packet routing algorithm does not exhibit buffer dependency cycles, the new DVC dependencies do not cause deadlocks. Furthermore, with general routing, modifications to the deadlock resolution algorithm and packet handling procedures were presented that remove the new dependencies that would prevent cycle rotation.

We investigated the performance of deadlock resolution. As expected, networks tend not to deadlock under light load even when there is the potential for deadlocks. However, under constant heavy load with cyclic routing dependencies, deadlocks can be frequent. In this case, deadlock resolution cannot remove packets from deadlock cycles faster than they are replenished by the introduction of new packets, and the resulting performance is poor. If the traffic injected into the network gives rise to deadlocks only occasionally, recovery occurs and normal operation can proceed. For a scenario with only occasional deadlocks, we investigated the effect on recovery time of varying how long a node is blocked before it initiates detection. Both very small and very large timeouts were found to cause the longest recovery times, because of synchronization overhead with small timeouts and long periods of no activity with large timeouts.

The conclusion from the performance evaluation is that deadlocks must be rare in order for this deadlock detection and resolution mechanism to have acceptable performance. If traffic patterns and loads are such that deadlocks are frequent, then the mechanism is not able to clear the deadlocks quickly enough to maintain high throughput. In contrast, if deadlocks are rare events (for example,

caused by a link failure or by intermittent large packet storms), then the mechanism can recover from deadlock in about ten thousand clock cycles. The mechanism is sensitive to the timeout value used to trigger deadlock detection. In our experiments, recovery time varied by up to a factor of two for reasonable timeouts less than 1000 clock cycles.

Chapter Five
Deadlock Avoidance in DVC Networks

In this chapter, we present and evaluate a deadlock avoidance scheme for networks that use the Dynamic Virtual Circuits mechanism. The deadlock avoidance scheme enables virtual circuits to be established (or re-established) along any network path from source to destination, without the possibility of deadlock. Compared to the deadlock detection and resolution scheme for DVC networks that was presented in Chapter 4, the deadlock avoidance scheme provides better performance at high load because deadlock cannot occur. The scheme places minimal restrictions on path and buffer usage to avoid deadlock. Only a small fraction of the packets in the network are subject to these restrictions, while the vast majority of packets are forwarded along the unconstrained paths of virtual circuits. With this approach, the DVC mechanism retains the advantages of routing flexibility and efficient utilization of bandwidth and packet buffer resources.

In Section 5.1, we present our proposed deadlock avoidance scheme for DVC networks. The approach uses a dependency cycle-free virtual network to avoid packet routing deadlocks. In addition, to avoid more complex deadlocks that could arise from the dependencies introduced by circuit establishment and teardown operations, the scheme uses a second virtual network to decouple circuit manipulation and packet routing operations. In Section 5.2, we present a correctness proof of the DVC algorithm with deadlock avoidance. The proof analyzes the system's state space to show that each packet injected into the network using a DVC is delivered to the DVC destination in order. In Section 5.3,

we present a description and evaluation of the hardware resources and mechanisms needed to implement the DVC algorithm with deadlock avoidance and show that the scheme is simple to implement with modest hardware requirements. In Section 5.4, we present simulation results that explore the potential performance benefits of DVCs using the deadlock avoidance scheme. This is done by considering limit cases, where the more sophisticated (complex) routing possible with DVCs leads to significantly higher performance than can be achieved with conventional packet switched networks, which typically must use simple (e.g., algorithmic) routing.

5.1. Scheme for Avoiding Deadlocks in DVC Networks

The DVC mechanism supports adaptive routing by imposing no restrictions on the choice of path for any circuit, and by enabling circuits to be rerouted adaptively during their lifetimes onto new paths to minimize latency and maximize throughput. The flexibility of adaptive routing comes at the cost of possible deadlocks that involve the packet buffers. Deadlock cycles may also involve RVCs, since those resources are contended for by the circuit establishment and disestablishment operations at a switch.

This section shows how the mechanism for tearing down and re-establishing Dynamic Virtual Circuits can be combined with a deadlock avoidance scheme that provides packets with an escape path from potential deadlock cycles. Data packets that take the escape path are routed individually to their destinations, independently of the virtual circuit paths.

In this chapter, we assume that packet ordering is maintained for adaptively

rerouted DVCs with the use of the packet sequence number mechanism described earlier in Section 3.2. With this mechanism, some packets are stamped with sequence numbers for reordering at the destination. Specifically, each CEP is stamped with the sequence number for the next data packet of the circuit. Each switch along the path records the sequence number in the CEP as a circuit is established or re-established. The switch increments the sequence number for each data packet that subsequently arrives on the circuit.

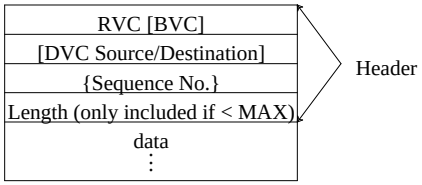


Figure 5.1: Packet Format. Fields in ‘‘[]’’ have the stated use only for diverted data packets (described in Section 5.1.1). The sequence number field is used only for CEPs, diverted data packets, and the next non-diverted data packet.

The general packet format is shown in Figure 5.1. A packet consists of a header followed by data phits. The first phit of the header records the RVC value. An additional four bits of the RVC field indicate packet type and whether the packet is of maximum length. If not, a length field is present in the header. For most packets, the header consists only of the RVC field and possibly the length field. For a minority of packets, the header includes additional fields. These additional fields are required for data packets that are diverted onto deadlock escape paths, as we describe next.

5.1.1. Avoiding Deadlocks Involving Data Packets

To avoid deadlocks arising from the unrestricted paths of DVCs, we embed in the physical network two virtual networks: the *primary network* and the *diversion network* [Duat96]. Each virtual network is composed of one *Buffering Virtual Channel* (BVC) per switch input port (i.e. per link). We name the two BVCs the *primary BVC* and the *diversion BVC*. Each is associated with a buffer (the ‘‘primary’’ and ‘‘diversion’’ buffers), which may be a FIFO buffer, or a more efficient Dynamically Allocated Multi-Queue (DAMQ) buffer [Tami92], or a buffer with any other organization.

The primary network supports fully-adaptive routing. This allows data packets to follow the unconstrained paths that are assigned to virtual circuits, but it also creates dependency cycles among packet buffers. In contrast, routing in the diversion network is constrained such that dependency cycles do not exist (e.g., in a mesh topology, Dimension-Order Routing (DOR) could be used in the diversion network). In addition, the diversion network can accept blocked packets from the primary network to provide a deadlock-free escape path [Duat96]. We say that a data packet is *diverted* if it takes a hop from the primary network into the diversion network.

Whereas virtual circuit forwarding is used to route data packets in the primary virtual network, traditional packet routing is used in the diversion network. Hence, while data packet headers in the primary network consist of only an RVC number, headers of packets in the diversion network must include source, destination, and sequencing information (Figure 5.1). When a data packet in the primary network becomes eligible to be diverted, the switch obtains this information from the IMT

entry where this required information is recorded. To enable locating the correct IMT entry, the primary buffer retains each packet's input RVC value until the packet is forwarded on the output RVC. As long as diversions are rare, the slower forwarding of diverted packets at intermediate switches and the overhead of transmitting the larger headers of diverted packets will not significantly impact network performance. Furthermore, if diversions are rare, small diversion buffers are sufficient (e.g., capacity of one packet).

Data packets that become blocked in the primary network can time out and become eligible to enter the diversion network on the next hop. The routing function used for the diversion network determines which output links the blocked packet in the primary network can take to enter the diversion network. The packet eventually advances, either by taking one hop on its virtual circuit path within the primary network, or by taking one hop into the diversion network on an adjacent switch. To enable switches to identify which virtual network an arriving packet is using, one RVC is designated as special. A data packet arriving on this special RVC uses the diversion BVC, else it uses the primary BVC.

Since diverted data packets may arrive out of order at the destination, each diverted data packet is stamped with a packet sequence number for use by the packet reordering at the destination. After a packet is diverted, the next data packet on the same circuit is also stamped with a sequence number. At each subsequent switch the non-diverted data packet visits along the circuit path, the switch reads the sequence number and locally updates its record to account for the diverted data packets. As long as only a minority of packets require the escape path, most of the advantages of static virtual circuits are maintained with DVCs.

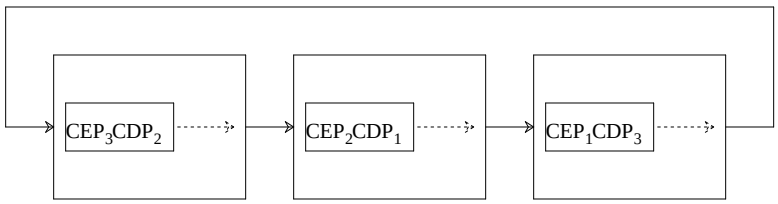


Figure 5.2: Deadlock involving only control packets in three switches. Packet buffer capacity is two packets. Each CEP_i establishes a unique virtual circuit. The matching CDP_i will disestablish the circuit set up by CEP_i.

5.1.2. Avoiding Deadlocks Involving Control Packets

Whereas data packets are diverted from their virtual circuit paths to avoid deadlock, control packets (CEPs and CDPs) cannot deviate from virtual circuit paths. Instead, each CEP must traverse the path that is selected for it by the network's fully adaptive routing function, and each CDP must traverse the path used by the virtual circuit it is disestablishing. A deadlock forms when a cycle of packet buffers fills with CEPs and CDPs, as shown by example in Figure 5.2.

To avoid such deadlocks we develop a mechanism which prevents any buffer from filling with control packets and blocking. The mechanism is derived by analyzing all possible sequences of arrivals of control packets on a single RVC to identify the storage required for each arrival sequence. Deadlocks are prevented by providing switches with packet buffers that are large enough to store the control packets of the worst case sequence without filling and blocking.

We initially examine arrival sequences consisting only of control packets on a single RVC to find the per-RVC storage requirement. Control packets arrive in alternating order (CDP, CEP, CDP, etc.) on an RVC. If a CDP arrives and the CEP

that matches it is present at the switch without an intervening data packet, then the CEP and CDP are deleted from the network (freeing buffer slots). If they were not deleted, the circuit establishment and disestablishment would be wasted operations since the circuit is not used by any data packet. Thus if the first packet in an arrival sequence is a CEP, then the CDP that follows it causes both packets to be deleted. One other case of packet deletion is possible; a CDP is deleted if it arrives at a switch where its circuit is already torn down as a victim. If the first packet of a sequence is a CDP that tears down an existing circuit, then the second packet is a CEP that establishes a new circuit. The third packet is a CDP, which matches the CEP and causes both packets to be deleted. Therefore, in the worst case storage is required for one CDP and one CEP for each RVC. One additional buffer slot is needed that is common to all the RVCs. This slot accommodates the arrival on any RVC of a CDP that will be deleted along with its matching CEP. Hence an input port that supports R RVCs requires storage for R CEPs plus $(R+1)$ CDPs to handle arrival sequences without data packets.

We next consider arrival sequences that include data packets. A data packet that uses a circuit that is currently allocated an output RVC at the switch is called ‘‘mapped’’. A mapped data packet eventually frees its buffer slot via normal forwarding or via diversion. Therefore, including a mapped data packet in an arrival sequence does not increase the storage required to prevent deadlock. A data packet that uses a circuit that is not currently allocated an output RVC is called ‘‘unmapped’’. It is complex to divert an unmapped data packet because its circuit identification information is not recorded in the IMT entry, as with mapped data packets, but rather in a CEP that is somewhere in the packet buffer.

To avoid this complexity, we prohibit diverting unmapped data packets. However, this causes the storage requirement to become unbounded because an infinite repeating pattern of packet arrivals of the following form may arrive to a single RVC: CEP, followed by unmapped data packets, followed by CDP. Matching CEPs and CDPs cannot be deleted because unmapped data packets intervene.

To restore a bound on the storage requirement, we restrict each input to store at most one unmapped data packet at a time (having a larger limit on the number of unmapped packets would also work but would increase the storage requirements). If a data packet arrives and the Input Mapping Table lookup reveals that the RVC is free or allocated to a different circuit, then the primary buffer asserts flow control to block subsequent packet arrivals.

Blocking flow when an unmapped data packet arrives prevents control packets from arriving in addition to data packets. Blocking control packets can cause deadlocks, as shown in Figure 5.3.

We prevent such deadlocks by introducing a new Buffering Virtual Channel (BVC) for control packets. We call the new BVC the *Control BVC*. The Control BVC is the third and final BVC, in addition to the Primary BVC and the Diversion BVC. The buffering associated with the Control BVC is dedicated to storing control packets. Introducing the Control BVC allows control packets to arrive even when the Primary buffer is full with data packets. The deadlock of Figure 5.3 does not occur since the CEPs in the figure can all advance using the Control BVC.

Prohibiting unmapped data packets from being diverted and restricting each input port to store at most one unmapped data packet cause the storage requirement

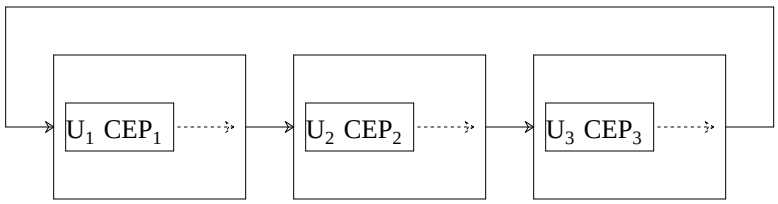


Figure 5.3: Example deadlock involving three switches. Each U_i is an unmapped data packet that is waiting for CEP_i to establish an RVC mapping. When a CEP transmits to the next switch, the unmapped data packet is converted into a mapped data packet. So long as the unmapped data packets are present at the input ports, subsequent arrivals on the primary Buffering Virtual Channel (BVC) are blocked. Therefore, the CEPs cannot make progress, and a deadlock results.

for control packets to increase slightly over the case with no data packets. The additional storage is common to all the RVCs at a switch input port and can store one CDP and one CEP. This increases the storage requirement to $(R+1)$ CEPs plus $(R+2)$ CDPs for an input port that supports R RVCs. The two additional control packets arrive after the unmapped data packet and on its RVC. Sections 5.1.2.1 and 5.1.2.2 present an exhaustive examination of the storage requirements for all possible packet arrival sequences. The worst case sequence of packet arrivals on the RVC that is used by the unmapped data packet is as follows (in order of arrival): mapped data packets, CDP_1 (will release the RVC), CEP_2 (will allocate the RVC for the next data packet), unmapped data packet, CDP_2 (will release the RVC), and CEP_3 (will allocate the RVC for yet another circuit). Since the unmapped data packet cannot be diverted, all four control packets in the sequence are necessary and cannot be deleted. After this sequence of arrivals, CDP_3 may arrive which matches CEP_3. In this case, both CDP_3 and CEP_3 are unnecessary and are deleted.

The control packets created by source hosts (as opposed to switches along the path) trigger critical operations at the destination host. The initial CEP injected by a source to establish a new circuit causes the destination host to allocate memory for delivering packets that use the circuit. The final CDP injected by the source to disestablish the circuit triggers release of resources at the destination host. These control packets may be deleted by a switch if all the data packets that use the circuit are diverted, in which case the final CDP may catch up to the initial CEP at some switch, with no intervening data packet. To resolve this potential problem, the first data packet that uses a circuit replicates the information in the initial CEP, and the final data packet of a circuit consists only of a header (no data payload). Since data packets are never deleted, they are guaranteed to reach the destination and trigger the actions.

In order to demonstrate in a rigorous fashion how much buffer space is required for the control BVC to avoid deadlock cycles involving control packets, the following subsections present an enumeration of all possible sequences of packet arrivals on one RVC of a switch input port. We assume in this analysis that for an extended period, competition with data packets for link bandwidth prevents any of the control packets that arrive from being transmitted to the next switch. The control packets that arrive to the switch must be accommodated by the switch input buffers. Otherwise, the switch would block the arrival of subsequent control packets, a condition we wish to avoid in order to guarantee that inter-switch deadlocks involving control packets cannot form. The arrival sequences are partitioned into two major classes: sequences that contain no data packets, and sequences that contain data packets.

5.1.2.1. Cases Without Data Packets

We first consider all arrival sequences (on a single RVC) that lack data packets. The empty sequence (no arrivals at all) is a trivial example. In that case, no buffer space is required to accommodate the sequence. Non-empty arrival sequences without data packets begin with the arrival of either a CEP or a CDP.

If a CEP arrives first, then the next arriving control packet on the same RVC can only be a CDP that matches the CEP. When the CDP arrives, the switch identifies both the CDP and the CEP ahead of it as unnecessary, and both packets are deleted. We write the arrival sequence as follows (from first arrival to last): CEP [CDP]. The square brackets mean the arrival of the packet inside the brackets causes both that packet and the last packet in the sequence to be dropped. Only buffer storage for the CEP is required in this case, since the CDP is dropped as it arrives.

If a CDP (call it CDP_1) arrives first in the sequence, then the next arriving control packet on the same RVC is CEP_2 for a new circuit. CDP_1 will release the RVC upon transmission, and then CEP_2 will try to allocate it and set up a new mapping. After CEP_2 arrives, the next arrival would be CDP_2, which matches CEP_2. Both packets are unnecessary and are dropped. The sequence is thus the following: CDP_1 CEP_2 [CDP_2]. The maximum buffer requirement in this case is storage for the first CDP and the CEP.

To summarize, the maximum control packet storage required for arrival sequences without data packets is 1 CDP and 1 CEP for each RVC. Next we examine the arrival sequences that include data packets.

5.1.2.2. Cases With Data Packets

There are three categories of arrival sequences (on a single RVC) that include data packets: sequences with data packets that are all mapped; sequences with data packets that are all unmapped; and sequences with a mixture of mapped and unmapped packets. We show that the sequences with unmapped data packets pose the greatest demand on control packet buffer storage.

Consider the arrival sequences with data packets, all of which are mapped. A data packet is mapped if it is on a circuit that has allocated the RVC. The RVC is allocated by the CEP that precedes the data packet. The RVC is allocated when the CEP transmits to the next switch. Therefore, if a data packet is mapped, the CEP ahead of it is stored at some different switch or has been delivered to the destination. Since by our assumption control packets are unable to transmit downstream for a long time, if a control packet arrives in the sequence ahead of a data packet, then the control packet is present when the data packet arrives. Therefore, the data packet is unmapped, not mapped. That implies that each arrival sequence in this category begins with a sequence of data packets that are followed by a sequence of control packets. Having data packets strictly at the front of the arrival sequence does not change the storage requirement for the control packets at the tail of the sequence. Hence again storage is required for one CDP and CEP on each RVC. The worst case arrival sequence is the following, where $M_1 \cdots M_n$ are mapped data packets: $M_1 \cdots M_n \; CDP_1 \; CEP_2 \; [CDP_2]$.

We now consider arrival sequences on one RVC that include data packets, all unmapped. When the first unmapped data packet arrives, flow on the primary BVC is blocked, preventing subsequent arrivals of data packets. Control packets

can still arrive after the data packet.

Suppose the first packet that arrives in the sequence is a CEP. Then, if a CDP follows it, both are deleted. Otherwise, the data packet would follow the CEP. In that case, the sequence that requires the most buffering is the following: $CEP_1\ U\ CDP_1\ CEP_2\ [CDP_2]$. U in the sequence is the unmapped data packet. The control packet storage required is a single CEP in addition to the usual per-RVC requirement of one CEP and one CDP.

If the first packet in the sequence were a data packet instead of a CEP, then the switch would generate a CEP and insert it ahead of the data packet. Thus this case is equivalent to the previous case and has the same storage requirement.

If a CDP (call it CDP_0) were the first packet in the sequence, then the worst case sequence is CDP_0 followed by the sequence from the previous case. That is, the sequence is the following: $CDP_0\ CEP_1\ U\ CDP_1\ CEP_2\ [CDP_2]$. The storage requirement is one CDP and one CEP in addition to the usual per-RVC storage requirement.

Finally, we consider arrival sequences that include both mapped and unmapped data packets. By the same reasoning as for sequences with only mapped data packets, the mapped data packets in the sequences in the current category must precede all other packets in the sequence. Again, the presence of the mapped data packets does not affect the storage required for control packets. The worst case sequence is the following: $M_1 \cdots M_n\ CDP_0\ CEP_1\ U\ CDP_1\ CEP_2\ [CDP_2]$.

5.1.3. Algorithm for Handling Control Packets with DVCs

As described in the previous sections, deadlock-freedom is maintained using data packet diversion through a deadlock-free virtual network. The DVC mechanism imposes two restrictions that simplify the algorithms and their implementations: each input port accommodates at most one unmapped data packet at a time, and unmapped data packets cannot be diverted. Dedicated control packet buffers are used to prevent deadlocks that include control packets. These control packet buffers are large enough to ensure that they can never fill up and cause blocking.

The control packet buffer is organized as follows. For each RVC i, a logical queue is maintained of the control packets in the order they will be transmitted to the next switch. The head of the logical queue for RVC i is comprised of a queue H_i with space for one CEP and one CDP. Whenever an unmapped data packet is present on RVC i, the tail of the logical queue is extended by a queue T_* with space for one CEP and one CDP. Queue T_* is shared by all RVCs at an input port but used by only one RVC at a time.

The algorithm for managing DVCs and control packets is listed in Figure 5.4. Details of data packet forwarding are not shown because the focus is on circuit manipulation. The algorithm listing shows the key DVC control events that occur at a switch and the actions the switch takes in response. For example, the first event shown is ‘‘Output port Z becomes free’’. The following lines 1 through 15 show how the switch decides which next packet to transmit on output Z and the actions that are taken if the transmitted packet is a CDP or a CEP.

The algorithm listing uses the convention that i and j refer to input RVCs, k is

an output RVC, X is an input port, and Z is an output port. Also, N_X is a primary input buffer, D_X is a diversion buffer, and H_i and T_* are control packet buffers.

Conditions/Events and Actions (atomic):

1. Output port Z becomes free

```
1       Arbitrate access (assume RVC i
        of input port X maps to RVC k
        of output port Z), but also
        obey BVC flow control.
        Arbitration priority:

2          A. a packet in some D_X
              buffer waiting for port Z
              or a mapped packet on
              RVC k from buffer N_X
3          B. a CEP/CDP on RVC k from
              head of H_i

4       when a CDP from RVC i is
        transmitted on RVC k:
5          delete mapping from i to k
6          if a CEP is waiting at the
           head of some H_j for RVC j {
7            set up new mapping from
             RVC j to RVC k
8          }
9          if required, transfer head
           of T_* to H_i

10      when a CEP from input RVC i is
        transmitted on output RVC k:
11         if an unmapped packet on
           RVC i exists {
12            convert unmapped packet
              to a mapped packet
13            unblock flow to primary
              input buffer N_X
14         }
15         if required, transfer head
           of T_* to H_i
```

2. CEP arrives on RVC i of input port X

```
16      if H_i is not full {
17         enqueue CEP at tail of H_i
18      } else {
19         place CEP at tail of T_*
20      }
```

3. CEP at head of H_i, and RVC i does not map to an output RVC

```
21      record SRC, DST in Input
        Mapping Table entry for
        RVC i

22      output port Z <-
        route(CEP's destination field)

23      if free RVC k exists on output
        port Z {
24         set up mapping from i to k
25      } else if the number of RVCs on Z
        for which there are CDPs at the
        switch is less than the number of
        CEPs that are on unmapped RVCs
        and are routed to output port Z {

26         // it is necessary to select and
           // teardown a victim

27         select RVC k (that reverse maps to
           input RVC j != i) such that no CDP
           resides at H_j

28         create CDP, place at tail of H_j

29         (note: the next data packet to
           arrive on RVC j must be considered
           unmapped even though RVC j stays
           mapped to RVC k until the
           transmission of the CDP in H_j)
30      }
```

4. Unmapped packet arrives on RVC i of input port X

```
31      block flow to primary input buffer N_X
32      if H_i has no CEP {
33         create CEP using circuit
           info in RVC i's Input Mapping
           Table (IMT) entry

34         place CEP at tail of H_i
35      }
```

5. CDP arrives on RVC i of input port X

```
36      if CDP is redundant {
37         delete arriving CDP
38      } else if CDP matches a CEP not
        associated with an unmapped packet {
39         delete both the CEP and the
           arriving CDP
40      } else if H_i not full {
41         place arriving CDP at tail of H_i
42      } else {
43         place arriving CDP at tail of T_*
44      }
```

Figure 5.4: Algorithm for Handling Control Packets with DVCs

5.2. Algorithm Correctness

We consider the algorithm to be *correct* if the following statement is true: *All data packets injected into the network on a DVC are delivered eventually to the DVC destination in the order injected*. Eventual delivery to the DVC destination is guaranteed if packets make progress (the network is deadlock-free) and data packets are never delivered to incorrect destinations. In-order delivery is guaranteed by attaching a sequence number to each packet that may arrive at the destination out of order.

Section 5.2.1 below proves that network deadlocks are impossible. Section 5.2.2 proves that data packets are associated with the same DVC from injection to delivery. Together with the FIFO delivery mechanism described in Section 5.1, these results show the algorithm satisfies the statement of correctness.

5.2.1. Proof of Deadlock-Freedom

To prove the network never enters a deadlocked configuration, we examine data packets and control packets separately. Theorem 1 below shows that diverted and mapped data packets cannot be part of a deadlock. Theorems 2 and 3 prove that control packets make progress which in turn guarantees that mappings are eventually provided for unmapped data packets. Together, the theorems show that every packet, regardless of its type, is guaranteed to make progress.

Theorem 1: Every mapped data packet in a primary buffer N_X and every data packet in a diversion buffer D_X is eventually forwarded.

Proof: This follows from Duato's sufficient condition for deadlock freedom in a virtual cut-through packet-switching network [Duat96]. Deadlock is avoided if

there exists a set C of BVCs such that all packets can reach set C in one hop, routing in set C reaches all destinations from all switches, and the set C is free of buffer dependency cycles. The set of diversion buffers D_X meets the definition of set C. A packet in any primary buffer N_X can enter the diversion network by taking one hop and is routed to its destination with a routing policy that is free of dependency cycles (Section 5.1.1). □

Theorem 2: Control packets do not block in deadlock cycles that involve multiple switches.

Proof: Assume a deadlock involves a control packet and an entity at another switch. By examining all possible chains of dependencies from a control packet to entities at neighboring switches, we show that each chain includes an entity that cannot be in a deadlock cycle, contradicting the assumption.

A control packet may wait directly for one of the following five entities: a buffer slot at the next switch, the output link, a mapped data packet at the same switch, a control packet at the same switch, or an unmapped data packet at the same switch. We examine each entity in turn. First, dedicated control packet buffers at the neighbor cannot be part of a multiple switch deadlock cycle because they have sufficient capacity to avoid filling and blocking. Second, eventual access to the output link is guaranteed for bounded length packets with virtual cut-through forwarding and starvation-free switch crossbar scheduling. Third, mapped data packets cannot participate in deadlock cycles (Theorem 1). Fourth, all control packets have the same set of possible dependencies to other entities, hence a dependence of one control packet on another control packet at the same switch does not introduce the possibility for a deadlock cycle that spans multiple switches.

Fifth, a CDP may directly wait for an unmapped data packet at the same switch and on the same RVC. In turn, the unmapped data packet waits for a CEP at the same switch to transmit, which will cause the unmapped data packet to become mapped. The chain of dependencies (from the CDP to the unmapped data packet to the CEP at the same switch) is equivalent to case four above (direct dependence from one control packet to another at the same switch). □

Theorem 3: Control packets do not enter intra-switch deadlocks.

Proof: Such deadlocks arise from dependency cycles within a single switch. We construct a graph of all the dependencies at a switch that involve control packet buffers. We show that the resulting graph is acyclic. Therefore, intra-switch deadlock is impossible.

From the algorithm listing and the enumerations of buffer states in Section 5.1.2.1 and Section 5.1.2.2, we can construct the dependency graph shown in Figure 5.5. The graph shows all possible dependencies for the buffers associated with two input RVCs: a mapped RVC i at input port X, and an unmapped RVC j at input port Y. Input ports X and Y are at the same switch. RVC i is mapped to RVC k of output port Z.

RVCs i and j are arbitrary representatives of their classes: mapped and unmapped RVCs, respectively. The set of all possible dependencies associated with these representative RVCs completely characterizes all the dependencies associated with all mapped RVCs and all unmapped RVCs. That is because it turns out that mapped RVCs have local dependencies only to packets associated with unmapped RVCs, and unmapped RVCs have local dependencies only to packets associated with mapped RVCs.

Each vertex of the dependency graph is labeled with a 3-tuple (α, β, γ). Component α specifies a buffer slot. Component β specifies the type of packet occupying the buffer slot. The packet may be a CDP, a CEP, a mapped data packet (denoted ‘‘M’’), or an unmapped data packet (denoted ‘‘U’’). Component $\gamma = TRUE$ if the input RVC used by the packet is mapped to an output RVC. Otherwise, $\gamma = FALSE$.

A directed arc from vertex $(\alpha_1, \beta_1, \gamma_1)$ to $(\alpha_2, \beta_2, \gamma_2)$ represents a potential dependency. The packet in buffer slot α_1 of type β_1 whose RVC mapping status is γ_1 may be blocked by the packet in buffer slot α_2 of type β_2 whose RVC mapping status is γ_2.

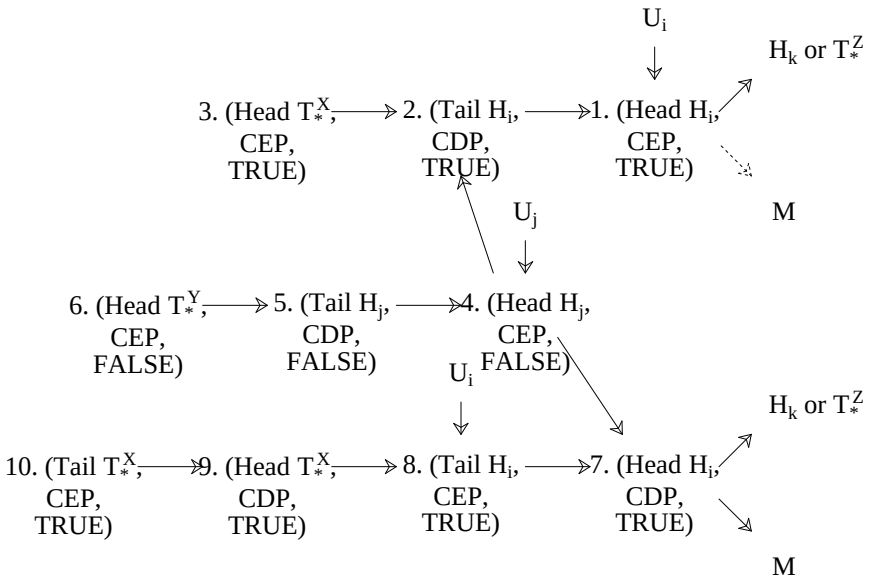


Figure 5.5: Buffer dependency graph for mapped RVC i and unmapped RVC k.

Figure 5.5 shows buffers H_i and T_* at input port X (vertices 1–3 and 7–10), and buffer H_j at input port Y (vertices 4–6). Output RVC k is on output port Z, and

buffers H_k and T_*^Z are at the neighbor switch.

The symbol U_i represents an unmapped data packet on RVC i that resides in primary buffer N_X. Similarly, U_j represents an unmapped data packet on RVC j at input port Y. Each unmapped packet waits for the transmission of a CEP in order to be converted into a mapped data packet. The CEP resides in the H buffer for the unmapped packet's RVC. Note that U_i is unmapped even though RVC i is mapped to RVC k; according to lines 10–12 of the algorithm listing in Figure 5.4, U_i will be converted to a mapped data packet simultaneously with the next CEP transmission on output RVC k from buffer H_i.

To construct the graph, consider the packet at the head of H_i. It may be either a CEP or a CDP. Recall that H_i is the head of the logical queue of control packets for RVC i. Vertices 1–3 show the logical queue for RVC i if a CEP is at the head of H_i. If a CDP is at the head, then vertices 7–10 apply. In both cases, each packet not at the head of H_i depends only on the packet immediately ahead in the logical queue. Thus each of vertices 2, 3, and 8–10 has only one outgoing arc directed to the packet immediately ahead in the logical queue.

For the unmapped RVC j, the only packet that can be at the head of H_j is a CEP to establish a new mapping. The logical queue for RVC j is shown in vertices 4–6. Again, each packet not at the head of the queue depends on the packet immediately ahead.

To complete construction of the graph, we examine the dependencies for each packet at the head of a logical queue (vertices 1, 4, and 7). First we examine vertices 1 and 7. Since RVC i is mapped to RVC k, the packet at the head of H_i blocks waiting for all mapped data packets on RVC i to be flushed out of the

switch. That dependency corresponds to the arrows from vertices 1 and 7 to the symbol M. By theorem 1, mapped packets eventually make progress. Hence the arrows to symbol M cannot be part of a deadlock cycle. Once all the mapped data packets have left, the packet at the head of H_i waits for access to output port Z for transmission. At the next switch, the packet will be either discarded or else deposited in H_k or T_*^Z. We represent that in the graph by the arrows from vertices 1 and 7 to the symbols H_k and T_*^Z. Since those buffers are at a different switch, that dependency cannot cause intra-switch deadlock.

Finally, consider vertex 4. The CEP at the head of H_j is blocked from transmission until a mapping can be established from RVC j to an output RVC. Establishing a mapping may first require that an existing mapping be disestablished (algorithm lines 26–30). In that case, the CEP blocks waiting for a CDP on a mapped RVC to transmit and free up an output RVC. This explains the potential dependencies from vertex 4 to vertices 2 and 7.

By inspection, the buffer dependency graph of Figure 5.5 is acyclic. Therefore, there can be no intra-switch deadlocks involving control packets. □

Theorem 4: The network is deadlock-free.

Proof: By Theorem 1, mapped data packets and diverted data packets always make progress. By Theorems 2 and 3, control packets always make progress, hence if a switch creates a CEP to establish a mapping for an unmapped data packet, the CEP is guaranteed to establish the mapping and transmit to the next switch. Once that occurs, the unmapped data packet becomes a mapped data packet. Therefore, all types of packets make progress, and the network is deadlock free. □

5.2.2. Correct Delivery

To complete the correctness discussion, we show that the deadlock-free network delivers each data packet to the destination of its virtual circuit (instead of some other destination). We also show that the destination can associate each data packet with its virtual circuit. This association is needed to deliver the contents of a data packet to the host application to which the packet's circuit is assigned.

The proof is based on a state transition table derived from an elaboration of the possible states of a switch. From the perspective of some RVC i of input port X of a switch, the switch state is the combination of the states of the buffers H_i, T_*, the primary buffer N_X, and the record in the IMT entry of a mapping to an output RVC. For our purpose and from the perspective of RVC i, the state of buffers at other input ports does not matter: the state of those buffers does not determine the destination to which a data packet on RVC i is forwarded. The state of the diversion buffer D_X is also irrelevant because diverted data packets do not participate in circuit manipulation operations.

To derive the state transition table, we first elaborate the state space of a switch as a list of sets of states, where each set of states is classified as either ''reachable'' or ''unreachable''. Reachable states arise through error-free operation of the switch, and the switch never enters unreachable states. A set of states is labeled unreachable if it contradicts steps of the control packet handling algorithm (Figure 5.4) or basic properties of DVCs such as the alternating arrival of CEPs and CDPs on an RVC.

Table 5.1 describes the state space of a switch. In the table, each row identifies a set of states. Each state is represented by the contents of H_i, T_*, and

two more fields that capture the relevant state of primary input buffer N_X and the IMT entry for RVC i. H_i can hold one CEP and one CDP on RVC i. T_* can hold one CEP and one CDP on any RVC. The state of N_X and the IMT entry for RVC i are too numerous to list exhaustively, but the relevant states are determined by the mapping status of RVC i, and whether an unmapped packet is present at input port X. That information is represented by using two symbols: map_i and U_X. The state notation used in Table 5.1 is described in Figure 5.6 which lists the possible states of a slot for H_i and T_* and defines the symbols used in columns U_X and map_i. In addition, the symbol ‘‘d’’ in Table 5.1 means ‘‘don’t care’’.

Each set of states in Table 5.1 is labeled as either ‘‘reachable’’ or ‘‘unreachable’’. Reachable states may arise through error-free operation of the switch. In contrast, the switch never enters unreachable states.

The ‘‘comment’’ column explains the unreachable states. ‘‘Non-FIFO’’ means the state corresponds to a non-FIFO configuration of the buffers (e.g. the head slot of H_i is empty but the tail slot is full). ‘‘T/U inconsistent’’ means the unmapped packet and the contents of buffer T_* are not using the same RVC (Section 5.1.3). ‘‘Consecutive CEPs’’ and ‘‘consecutive CDPs’’ refers to states that violate the alternating order of CEPs and CDPs on a single RVC. ‘‘U implies CEP’’ refers to states in which an unmapped data packet is present but there is no CEP to establish a mapping for it. That condition is impossible because of lines 10–15 and 32–35 of Figure 5.4. Note that there is a small time window in which an unmapped packet can be present before the CEP for it has been created, but this is a transient state that quickly resolves. We ignore such transient states in our analysis.

N	Reach	H_i head	H_i tail	T_* head	T_* tail	U_X	map_i	comment
1	F	–	CEP	d	d	d	d	non-FIFO
2	F	–	CDP	d	d	d	d	non-FIFO
3	F	d	d	–	CEP ∨ CDP	d	d	non-FIFO
4	F	d	d	–	CEP′ ∨ CDP′	d	d	non-FIFO
5	F	d	–	CEP ∨ CDP	d	d	d	non-FIFO
6	F	d	d	CEP ∨ CDP	d	i′ ∨ –	d	T/U inconsistent
7	F	d	d	d	CEP ∨ CDP	i′ ∨ –	d	T/U inconsistent
8	F	d	d	CEP′ ∨ CDP′	d	i ∨ –	d	T/U inconsistent
9	F	d	d	d	CEP′ ∨ CDP′	i ∨ –	d	T/U inconsistent
10	F	CEP	CEP	d	d	d	d	consecutive CEPs
11	F	d	d	CEP	CEP	d	d	consecutive CEPs
12	F	d	CEP	CEP	d	d	d	consecutive CEPs
13	F	d	CDP	CDP	d	d	d	consecutive CDPs
14	F	d	d	CDP	CDP	d	d	consecutive CDPs
15	F	CDP	CDP	d	d	d	d	consecutive CDPs
16	F	d	d	CDP′	CDP′	d	d	consecutive CDPs
17	**T**	–	–	–	–	– ∨ i′	d	
18	F	– ∨ CDP	– ∨ CDP	d	d	i	d	U implies CEP
19	**T**	–	–	CEP′	–	i′	d	
20	**T**	–	–	CDP′	–	i′	d	
21	**T**	–	–	CDP′	CEP′	i′	d	
22	**T**	CEP	–	–	–	– ∨ i	d	
23	**T**	CEP	–	–	–	i′	d	
24	**T**	CEP	–	CEP′	–	i′	d	
25	**T**	CEP	–	CDP′	–	i′	d	
26	**T**	CEP	–	CDP′	CEP′	i′	d	
27	F	CEP	CDP	d	d	– ∨ i′	d	CDP cancels CEP
28	**T**	CEP	CDP	– ∨ CEP	–	i	d	
29	F	CEP	CDP	CEP	CDP	d	d	CDP cancels CEP
30	F	CDP	d	d	d	d	F	CDP must be mapped
31	**T**	CDP	–	–	–	– ∨ i′	T	
32	**T**	CDP	–	CEP′ ∨ CDP′	–	i′	T	
33	**T**	CDP	–	CDP′	CEP′	i′	T	
34	**T**	CDP	CEP	–	–	d	T	
35	**T**	CDP	CEP	CEP′ ∨ CDP′	–	i′	T	
36	**T**	CDP	CEP	CDP′	CEP′	i′	T	
37	**T**	CDP	CEP	CDP	– ∨ CEP	i	T	
38	F	d	d	CEP′	CEP′ ∨ CDP′	i′	d	symmetry w/RVC i

Table 5.1: Switch State Space

To verify that the manual elaboration of the state space is complete and consistent, we wrote a program that exhaustively generates all combinations of all values each buffer may hold. The program verifies that each generated state matches at least one set of states in the list and that all matching sets are labeled

H_i and T_*		T_* only		U_X		map_i	
-	empty	CDP′	CDP on RVC $i' \neq i$	-	no unmapped packet	T	RVC i is mapped to an output RVC k, which also means that N_X may store mapped data packets from RVC i.
CDP	CDP on RVC i	CEP′	CEP on RVC $i' \neq i$	i	one unmapped packet on RVC i is in N_X (note: N_X may hold at most one unmapped packet)		
				i′	the unmapped packet is on some RVC $i' \neq i$	F	RVC i is unmapped, which means any mapped packets in N_X did not arrive on RVC i.

Figure 5.6: State Notation for Table 5.1 and Table 5.2

identically (all reachable or all unreachable).

Using the set of reachable states and the control packet handling algorithm, we can derive all the possible state transitions. Table 5.2 shows the state transitions for all reachable states. The actions that can trigger a transition are as follows: k (RVC i acquires mapping to output RVC k), D (CDP arrives on RVC i), E (CEP arrives on RVC i), U (unmapped packet arrives on RVC i), T (transmission of packet at head of H_i), D' (CDP arrives on RVC $i' \neq i$), E' (CEP arrives on RVC $i' \neq i$), $U_{i'}$ (arrival of unmapped packet on some RVC $i' \neq i$), $TE_{i'}$ (transmission of CEP at head of $H_{i'}$), and $TD_{i'}$ (transmission of CDP at head of $H_{i'}$). Each entry of the table is derived by tracing through the control packet handling algorithm to determine what the next state would be given each possible action. Note that each empty entry in Table 5.2 means the column's action cannot occur in the present state for the row. For example, for rows 0–9, there is no packet in buffer H_i. Therefore, action T, transmission of the packet at the head of H_i, cannot happen.

In some cases, two next states are indicated for some transitions, indicating that either of the two next states may become the new present state. This is

Present State						Action/Next State									
ID	H_i		T_*		$U_X map_i$	k	D	E	U	T	D′	E′	$U_{i'}$	$TE_{i'}$	$TD_{i'}$
0	–	–	–	–	– F		0	10	12		0	0	2	0	0
1	–	–	–	–	– T		26				1	1	3	1	1
2	–	–	–	–	i′ F		2	14			2,6	4		0	2
3	–	–	–	–	i′ T		27				3,7	5		1	3
4	–	–	CEP′	–	i′ F		4	16			2			0	
5	–	–	CEP′	–	i′ T		28				3			1	
6	–	–	CDP′	–	i′ F		6	18			6	8			2
7	–	–	CDP′	–	i′ T		29				7	9			3
8	–	–	CDP′	CEP′	i′ F		8	20			6				4
9	–	–	CDP′	CEP′	i′ T		30				7				5
10	CEP	–	–	–	– F	11	0		12		10	10	14	10	10
11	CEP	–	–	–	– T		0		13	1	11	11	15	11	11
12	CEP	–	–	–	i F	13	22				12	12		12	12
13	CEP	–	–	–	i T		23			1	13	13		13	13
14	CEP	–	–	–	i′ F	15	2				14	16		10	14
15	CEP	–	–	–	i′ T		3			3	15	17		11	15
16	CEP	–	CEP′	–	i′ F	17	4				14			10	
17	CEP	–	CEP′	–	i′ T		5			5	15			11	
18	CEP	–	CDP′	–	i′ F	19	6				18	20			14
19	CEP	–	CDP′	–	i′ T		7			7	19	21			15
20	CEP	–	CDP′	CEP′	i′ F	21	8				18				16
21	CEP	–	CDP′	CEP′	i′ T		9			9	19				17
22	CEP	CDP	–	–	i F	23	22	24			22	22		22	22
23	CEP	CDP	–	–	i T		23	25		26	23	23		23	23
24	CEP	CDP	CEP	–	i F	25	22				24	22		24	24
25	CEP	CDP	CEP	–	i T		23			31	25	23		25	25
26	CDP	–	–	–	– T		26	31	32	0	26	26	27	26	26
27	CDP	–	–	–	i′ T		27	33		2	27,29	28		26	27
28	CDP	–	CEP′	–	i′ T		28	34		4	27			26	
29	CDP	–	CDP′	–	i′ T		29	35		6	29	30			27
30	CDP	–	CDP′	CEP′	i′ T		30	36		8	29				28
31	CDP	CEP	–	–	– T		26		32	10	31	31	33	31	31
32	CDP	CEP	–	–	i T		37			12	32	32		32	32
33	CDP	CEP	–	–	i′ T		27			14	33,35	34		31	33
34	CDP	CEP	CEP′	–	i′ T		28			16	33			31	
35	CDP	CEP	CDP′	–	i′ T		29			18	35	36			33
36	CDP	CEP	CDP′	CEP′	i′ T		30			20	35				34
37	CDP	CEP	CDP	–	i T		37	38		22	37	37		37	37
38	CDP	CEP	CDP	CEP	i T		37			24	38	38		38	38

Table 5.2: State Transition Table

because the next state depends on the present state of $H_{i'}$, which is not displayed in Table 5.2 because it is not associated with RVC i.

Table 5.2 is used below to show that a packet P, which is injected on virtual circuit V, reaches the destination of circuit V, and the host interface at the destination associates packet P with circuit V.

Theorem 5: Whenever packet P is mapped, its input RVC identifies virtual circuit V.

Proof: The proof is by induction on the distance from the virtual circuit's source.

Basis: The theorem is true at the injecting host interface because the host transmits only one CEP for the circuit until it is disestablished.

Induction Step: Assume the theorem is true for P at switch or host S_n (n hops from the source). We now show the theorem is also true at the host or switch that is $n+1$ hops from the source (at host/switch S_{n+1}). Suppose packet P is about to transmit to host/switch S_{n+1}'s primary BVC on RVC k. Since P is about to transmit, P must be a mapped data packet at host/switch S_n. Therefore, the previous control packet on RVC k must have been a CEP that identified V.

If S_{n+1} is a host instead of a switch, then the host will correctly associate packet P with virtual circuit V. However, if S_{n+1} is a switch, then there may be a danger that the CEP will be deleted before it reaches the head of H_k. Only if the CEP reaches the head of H_k will the switch S_{n+1} record the circuit identification information for V in the IMT entry for RVC k (algorithm line 21). We will now prove that the danger is unwarranted. The CEP will successfully reach the head of H_k at switch S_{n+1}.

Switch S_{n+1} may delete a CEP if a CDP arrives from S_n or is generated locally. In our scenario, a CDP may not arrive at S_{n+1} from S_n, since packet P is about to transmit from S_n. It is impossible for a CDP to transmit ahead of a mapped data packet such as P (see arbitration priority rules in lines 1–3 of Figure 5.4). Therefore, only a CDP that is generated locally at S_{n+1} could trigger the deletion of the CEP.

From Table 5.2, we can identify all the state transitions in which a CEP is deleted upon introduction of a CDP. That is, we can identify all present states in the table such that the column D transition takes the switch to a next state that has one less CEP than the present state. For some of those states (10, 11, and 14 through 21), the CEP that identifies virtual circuit V is already at the head of the logical queue (i.e. H_k for switch S_{n+1}), hence the IMT entry identifies V. Even if that CEP is deleted, the information is available when packet P arrives at switch S_{n+1}. The remaining qualifying states are 24, 25, 31, and 33 through 36. In those states, a CDP is present in the buffer. According to line 27 of Figure 5.4, the presence of the CDP prevents the introduction of a locally-generated CDP. Therefore we conclude that it is not possible for a locally-generated CDP to cancel the CEP before the IMT is written to identify circuit V. □

Theorem 6: If packet P is a diverted packet, then P's header identifies circuit V.

Proof: By theorem 5, at the time P is diverted, the IMT entry identifies circuit V. The procedure for diversion attaches that information to packet P's header. After diversion, the packet header is not altered until P is delivered. Therefore, packet P is always associated with circuit V. □

5.3. Implementation Issues

Efficient implementation of the DVC algorithms presented earlier requires specialized hardware support. To demonstrate the feasibility of using DVCs, this section presents key components of a possible implementation.

5.3.1. Switch Organization

We assume an $n \times n$ switch is a single-chip device. The switch includes a processor that performs the relatively complex yet infrequent circuit manipulation operations of the DVC algorithm. This processor also executes higher-level routing protocols that identify low latency paths [Taji77]. The switch has packet buffers at each input port. Each input port X has primary buffer N_X, diversion buffer D_X, and control packet buffer T_*^X. The input buffers are connected to output ports through an $n \times n$ crossbar.

Each input port also has an Input Mapping Table (IMT) that records RVC mappings (Figure 5.7).

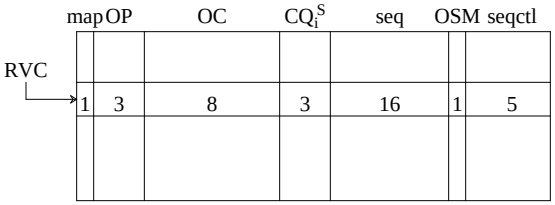


Figure 5.7: Input Mapping Table (one at each input port). Field widths in bits are shown.

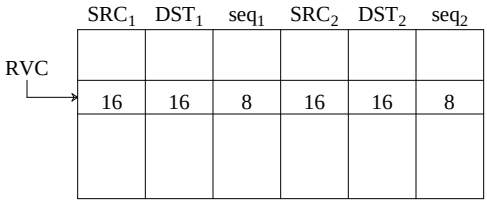


Figure 5.8: Circuit Information Table (one for each input port). Holds information about mapped and torn DVCs, and CEP information from H_i.

When a data packet arrives to an input port, its RVC identifier is used to access an

IMT entry. The entry's ‘‘map’’ field indicates whether the input RVC is mapped to an output RVC (i.e., whether a circuit is established). If so, the ‘‘OP’’ field specifies the output port to forward the arriving packet, and the ‘‘OC’’ field specifies the output RVC value which replaces the value in the packet's header. The input RVC value is saved in N_X with the data packet and is discarded when the packet is dequeued upon transmission. The remaining fields of the IMT are used to keep track of control packet buffering (Section 5.3.2) and FIFO sequencing (Section 5.3.3).

Each output port has an Output Mapping Table (OMT). The OMT is indexed by output RVC value and indicates the status of the output RVC. Each OMT entry has 3 bits. The ‘‘map’’ bit is set when the output RVC becomes mapped to an input RVC. It is cleared when a CDP transmits to the next switch using that RVC, releasing the output RVC for use by a new circuit. The ‘‘victim’’ bit is set when the circuit mapped to the output RVC is chosen as a victim for teardown. It too is cleared when the CDP transmits to the next switch. At that point, the RVC can be mapped to an input RVC that has a CEP waiting to establish a circuit. The ‘‘active’’ bit indicates whether the RVC has been used by any recently transmitted packet. The switch uses the ‘‘active’’ bit to choose a victim. In particular, the ‘‘clock’’ algorithm [Corb68], which approximates LRU, can be used to select a victim.

The IMT and OMT are accessed for each packet and must therefore be stored in high-speed (SRAM) memory. Other information is accessed only when packets are diverted or when circuits are manipulated and can therefore be stored in lower speed dense (DRAM) memory. This latter information consists of tables for

routing CEPs, tables that store DVC identification and routing information (these tables are updated upon DVC establishment and read when DVCs are rerouted), and tables used to implement the H_i control buffers.

5.3.2. Control Buffer Implementation

At each input port, the algorithm in Section 5.1 requires the ability to store up to four control packets (CEPs, CDPs) for one RVC (in H_i and T_*) and up to two control packets for the rest of the RVCs (in H_j). To minimize the cost of this storage, the storage for each RVC is split into two components: CQ_i^S — a frequently-accessed component stored as part of the IMT in dedicated SRAM at each port, and CQ_i^{CEP} — an infrequently-accessed component stored in the DRAM available on the switch. The CQ_i^S component consists of a compact representation of the state of the logical queue of control packets for input RVC i. From Table 5.2, the logical queue of control packets for an RVC i can at any time be in one of eight states: empty, CEP, CDP, CEP CDP, CDP CEP, CEP CDP CEP, CDP CEP CDP, CDP CEP CDP CEP. The CQ_i^S field of the IMT specifies the current state as a 3-bit value. Since CDPs carry no information other than the RVC value, the 3-bit state encoding represents all the CDPs in the queue.

The infrequently-accessed CQ_i^{CEP} component is maintained in DRAM in the Circuit Information Table (CIT) (Figure 5.8). For each input RVC, there is an entry in the table with space to record the information of two CEPs (DVC source, destination, and sequencing information). The first set of fields (SRC_1, DST_1, and seq_1) reflects the current status of the DVC and is set from the contents of the CEP that established the current DVC. The second set of fields (SRC_2, DST_2, and seq_2), if

valid, corresponds to the DVC that will replace the current DVC. This second set of fields is the storage of CQ_i^{CEP} — the CEP that is part of H_i (Section 5.1.3). Storage for information from two CEPs is needed since the switch may contain data packets on the previous DVC when the CEP for a new DVC arrives. If one or more of these data packets needs to be diverted, the information for the previous DVC will still be needed.

Finally, a dedicated buffer for each input port provides storage for the one CEP that is part of T_*.

Performance can be optimized by adding a small, fast write buffer for arriving CEPs. The switch can access the buffer as a cache to forward CEPs quickly without incurring the DRAM access time for writing to the CIT. A write buffer only benefits CEPs that arrive to empty H_i queues. This should be the common case since control packets present only a light load with reasonable traffic patterns.

The switch must transmit the control packets and data packets in a logical queue in FIFO order, even though the packets are stored physically at the switch in various memory buffers (i.e., the primary buffer for data packets, and the control buffers described above for control packets). Each buffer preserves the partial order of the packets it stores from the logical queue. The partial order of mapped data packets of a logical queue is preserved because the primary buffer maintains in FIFO order its data packets that are destined to a single output port. The partial order of control packets in the logical queue is recorded and maintained by using the IMT CQ_i^S field. Although these buffers preserve the correct partial order of transmission, a mechanism is needed to control the total interleaved order of transmissions from these buffers.

The total order of a logical queue is preserved by transmitting all of its mapped data packets before any of its control packets. The mapped data packets must precede the control packets because a CDP that precedes a mapped data packet would delete the current RVC mapping, and a preceding CEP would establish a new mapping before the data packet is transmitted. Transmission of mapped data packets ahead of control packets is enforced through the use of the ‘‘seqctl’’ field in the IMT (Figure 5.7). The seqctl field has initial value zero and records the number of mapped data packets that are present in the logical queue for an RVC. Control packets in the logical queue are allowed to transmit only when the value in ‘‘seqctl’’ is zero. Mapped data packets can transmit anytime. The ‘‘seqctl’’ value is incremented when a mapped data packet arrives at a switch on the RVC or when an unmapped data packet on the RVC becomes mapped as a result of transmitting a CEP. The ‘‘seqctl’’ field is decremented when a data packet that previously arrived on the RVC is either transmitted to the next switch along its circuit or is diverted.

5.3.3. Sequencing and Diversion

As explained in Section 5.1, to support FIFO delivery, one sequence number per input RVC is maintained in the IMT. The IMT entry ‘‘seq’’ (sequence number) field (Figure 5.7) is incremented whenever a packet is transmitted whose header has no sequence number field (Figure 5.1). The index used to access the IMT entry is the value of the input RVC on which the packet arrived at the switch. As discussed in Section 5.3.1, this value is saved in the main data packet buffer N_x. Whenever a packet with a sequence number is transmitted from the primary buffer,

that sequence number replaces the value in the IMT ‘‘seq’’ field.

After a data packet is diverted, the next data packet that is transmitted normally on the same circuit is stamped with a sequence number (Section 5.1.1). The IMT entry’s single-bit ‘‘OSM’’ (Out of Sequence Mode) field is used as a flag to determine whether a data packet that is transmitted normally should be stamped. The flag is set when a data packet is diverted. Before a data packet begins transmission to the next switch, the OSM flag for the packet’s input RVC is read. If the flag is set, then it is cleared and the sequence number is added to the data packet header on-the-fly as it is transmitted to the next switch. With virtual cut-through, a packet is transmitted only if the next switch has sufficient buffer space for a maximum size packet. Hence, lengthening the packet, following diversion, as it is transmitted, will not cause the buffer at the next switch to overflow.

Packet diversion requires forwarding a timed-out packet to an output other than the one associated with the logical queue storing the packet. Hence, when a packet times out, its request from the crossbar arbiter is modified to include access to the switch output that was its original destination as well as access to the switch output(s) on the diversion path(s). If access to the diversion network is granted first, the packet’s RVC field is changed to indicate use of the diversion BVC, and DVC information from the IMT and CIT is added to the header.

5.4. Performance Evaluation

One key advantage of DVCs is the potential for reducing overall network contention by establishing circuits or rerouting existing circuits onto low latency paths. In most systems, traffic patterns change dynamically, and circuits require time to adjust their paths to compensate. For a first-order evaluation of the performance *potential* for DVCs with adaptive routing, we consider simpler limit cases with stable traffic patterns and circuit placements. The actual performance of a system with DVCs will depend on the routing and rerouting algorithms.

We consider Uniform, Transpose and Bit-Reversal traffic patterns. In all cases, we precompute the paths used by DVCs in order to simulate ideal conditions where circuits have settled into a steady-state, low contention configuration. We compare the performance of the resulting configuration against a packet switched network using Dimension Order Routing (DOR), which is known to perform well for Uniform and poorly for Transpose and Bit-Reversal patterns [Glas94]. Transpose and Bit-Reversal traffic patterns are shown in a small mesh in Figure 5.9.

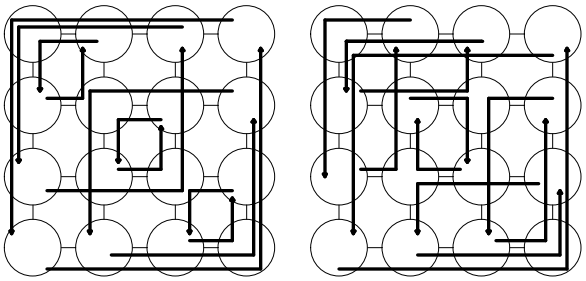


Figure 5.9: Non-uniform traffic patterns

For our simulation experiments, packet size is 32 phits, and switches have

DAMQ primary input buffers. For DVC simulations, there is also a diversion buffer of capacity 32 phits. The results for DOR and routed DVCs are shown for equal total input buffer capacity, which for DVCs is the sum of the primary input buffer capacity plus 32 phits for the diversion buffer. The interval between packet creations has a geometric distribution. At each switch, the crossbar arbitration policy gives priority to the packet that has resided longest at the switch. The performance metrics are the average latency, the average node throughput and the normalized throughput. Packet latency is defined as the number of cycles from when the first byte of a packet is generated at a sender to when it leaves the network. Normalized throughput expresses throughput as a fraction of the network bisection bandwidth.

With DVCs, a packet is diverted if it is blocked at the head of the queue at a switch and the timeout expires. If the timeout interval is too short, packets will be diverted unnecessarily. If the timeout interval is too long, it will take longer to resolve deadlocks. To evaluate the impact of this effect on performance, our evaluation includes simulations for a variety of timeout values.

5.4.1. Intelligent Flow-Based Routing

To precompute the paths used by DVCs, we use a simple heuristic placement algorithm which produces a set of virtual circuit routes that results in low contention, subject to the constraint that all virtual circuits follow paths of minimum hop count. Our heuristic technique places virtual circuits sequentially; the order of virtual circuit placement affects the final outcome. For each circuit, the heuristic selects a path using the well-known Bellman-Ford algorithm for

```
foreach virtual circuit {
    start at virtual circuit's DST
    search outward from DST until reaching SRC
    PlaceFlow(SRC, DST);
}

PlaceFlow(SRC,DST):
    minD[I] <- infinity for all input ports I in network
    unmark all nodes
    execute_list <- {DST}
    while SRC not at head of execute_list {
        NODE <- head of execute_list
        remove head of execute_list from execute_list
        PROC(NODE);
    }
    PROC(SRC);
END PlaceFlow.

PROC(NODE):
    foreach input port I of NODE {
        MinOut[I] <- k, where output port k minimizes
                     (MinD[neighborIP_k] + Delay[I,k]),
                     neighborIP_k is the input port
                     at NODE's neighbor connected to output port k,
                     and Delay[I,k] is the delay estimated from
                     the queuing model if the new flow were
                     placed from input port I to output port k
        minD[I] <- MinD[neighborIP_MinOut[I]] + Delay[I,k]
    }
    mark NODE
    Append unmarked neighbors of NODE to execute_list
END PROC.
```

Figure 5.10: Routing Procedure

finding paths with minimal cost. The cost of each link is set to an estimate of the delay that would be experienced by packets waiting to use the link.

The delay for a link is estimated by modeling the link as a Geo(N)/D/1 queuing system fed by packet arrivals from all DVCs whose routes include the link. Each $n \times n$ switch is modeled by n Geo(N)/D/1 queues, one for each output link. Feeding the jth Geo(N)/D/1 queue are $n-1$ traffic sources, representing the traffic from $n-1$ input ports through output port j. The traffic from the ith input port to output port j is represented by a flow $f_{(i,j)}$ with geometric interarrival time distribution and rate equal to the sum of the rates of all placed virtual circuits traversing from input port i to output port j. We use a numerical software package

to evaluate the mean occupancy of each queue. We use this value as the link cost, since queue occupancy is proportional to the average waiting time for the link (assuming non-blocking links).

To ensure that the resulting paths have minimum hop count, the path finding algorithm starts at the virtual circuit destination and searches toward the source on all paths, one hop at a time. When the source is reached, the minimum cost path of minimum hop count is located. The overall procedure is detailed in Figure 5.10.

5.4.2. Transpose Traffic Pattern

The transpose traffic pattern sends all packets across the diagonal of the mesh; packets from the source at row i column j are sent to the destination at row j column i. Nodes along the diagonal only forward packets; they do not produce or consume any packets of their own. For this traffic pattern, Figure 5.11 shows the latency versus throughput for an 8×8 mesh, with input buffer capacity 64, 96, and 288 phits.

These results show that for all levels of buffer capacity, the maximum throughput achieved with DVCs is about twice that achieved with DOR. With DOR, only the horizontal incoming links of the switches along the diagonal are utilized. With routed DVCs, both the vertical and horizontal incoming links of the diagonal are utilized with approximately equal traffic loads assigned to each link. The results also show that the impact of increasing the buffer capacity is higher latency, not higher throughput. Throughput does not increase for either DOR or routed DVCs because it is limited by the bandwidth of saturated links along the mesh diagonal. Finally, the results show that using DVCs with long timeouts for

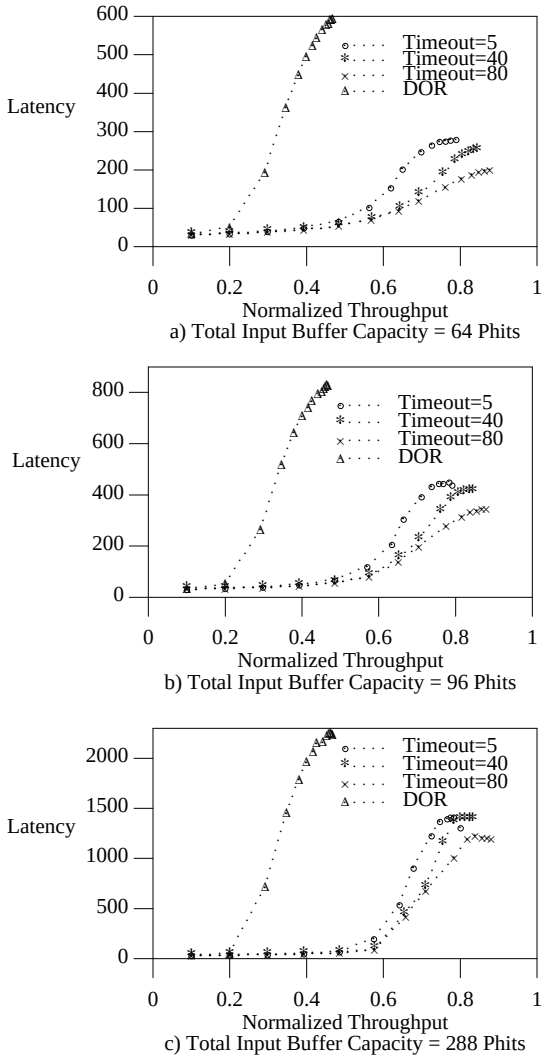


Figure 5.11: Transpose traffic: Latency vs. Normalized Throughput.

packet diversion results in higher maximum throughput than using short timeouts. Short timeouts increase the frequency of packet diversions, which for the transpose traffic pattern occur before the packet crosses the diagonal, the congested point in the network. Since diverted packets use DOR, they can only use the horizontal incoming links of switches on the diagonal. Hence packet diversions shift traffic from the vertical to the horizontal incoming links at the diagonal. These traffic

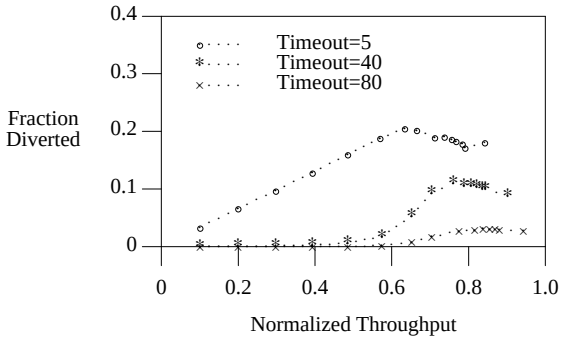


Figure 5.12: Transpose traffic: Fraction of Traffic Diverted vs. Normalized Throughput. Total input buffer capacity = 96 phits.

imbalances reduce performance, thus in this case longer timeouts which minimize packet diversions are better.

Figure 5.12 shows the fraction of traffic diverted versus normalized throughput for the DVC network with input buffer capacity of 96 phits. For low and medium network loads, as the load increases, the fraction of diverted packets increases. However, past a certain point, the fraction of diverted packets decreases as the load increases. The reason for this is that at these high loads the increase in network throughput is mostly for the subset of circuits using low-contention routes. Other circuits and their diversion paths are saturated and their throughput does not increase as the applied load increases. For the low-contention circuits no diversion occurs so more packets get through the network without a corresponding increase in the number of diverted packets.

The performance of a distributed application is often limited by the performance of its slowest member rather than by the aggregate throughput available to the application. For example, an application whose nodes communicate via the transpose traffic pattern may occasionally need to

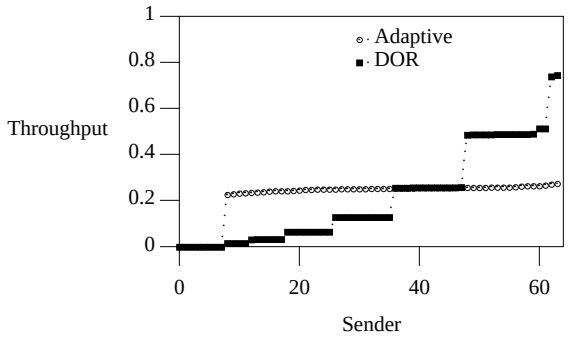


Figure 5.13: Transpose traffic: Throughput fairness. Throughput vs. sender, sorted. Total input buffer capacity = 64 phits. Average node throughput = 0.241 for DOR (48.2% normalized), 0.242 for routed DVCs (48.4% normalized).

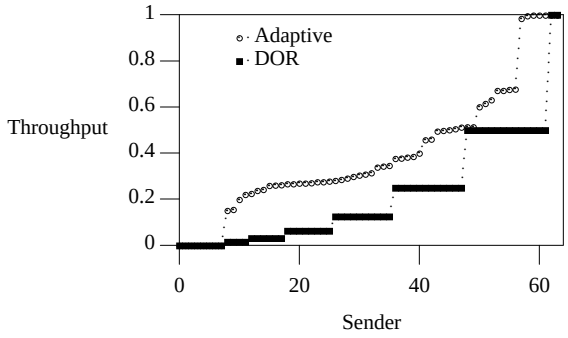


Figure 5.14: Transpose traffic: Throughput fairness at saturation. Throughput vs. Sender, sorted. Total input buffer capacity = 288 phits. Average node throughput = 0.24 for DOR (48% normalized), 0.47 for routed DVCs (94% normalized)

synchronize to ensure that all sending nodes are in a known state. If some flow is particularly slow, progress of nodes associated with this flow will be impeded and the progress of all other nodes will be throttled upon synchronization. Hence, it is useful to evaluate the fairness of the system by comparing the throughputs achieved by individual senders.

Figure 5.13 shows the raw (not normalized) throughput achieved by each

source node in the 8×8 mesh using the transpose traffic pattern. The throughputs from each sender are displayed, sorted to be monotonic (the first eight sources are along the diagonal and do not generate packets). Throughputs for routed DVCs and DOR are displayed as separate curves. Since fairness in a network decreases as the load increases, comparison of the fairness of the two policies should be done at the same average node throughput. In Figure 5.13, average node throughput for DOR is at its maximum, 0.233, and average node throughput for routed DVCs is 0.242. Since unfairness increases with average node throughput, the result in Figure 5.13 is biased slightly in favor of DOR, yet the routed DVCs achieve far greater uniformity of sender throughput than does DOR. As we increase applied load further, the routed DVCs policy also becomes unfair, but only at much higher levels of average node throughput than can be achieved with DOR. This is demonstrated in Figure 5.14, which shows throughput fairness at saturation, in which each source constantly tries to inject packets.

5.4.3. Bit-Reversal Traffic Pattern

The bit-reversal traffic pattern sends messages from each source $x_{n-1}x_{n-2}\cdots x_0 y_{n-1}y_{n-2}\cdots y_0$ to destination $y_0 y_1 \cdots y_{n-1} x_0 x_1 \cdots x_{n-1}$. Figure 5.15 shows latency versus throughput on an 8×8 mesh, for total input buffer capacity 64, 96 and 288 phits. The reported throughput is normalized to the bisection bandwidth, the upper bound on throughput for the bit-reversal traffic pattern.

The results for bit-reversal show that, as with transpose traffic, routed DVCs significantly outperforms DOR and there is no advantage to increasing the buffer size. Unlike with transpose traffic the latency-throughput results with bit-reversal

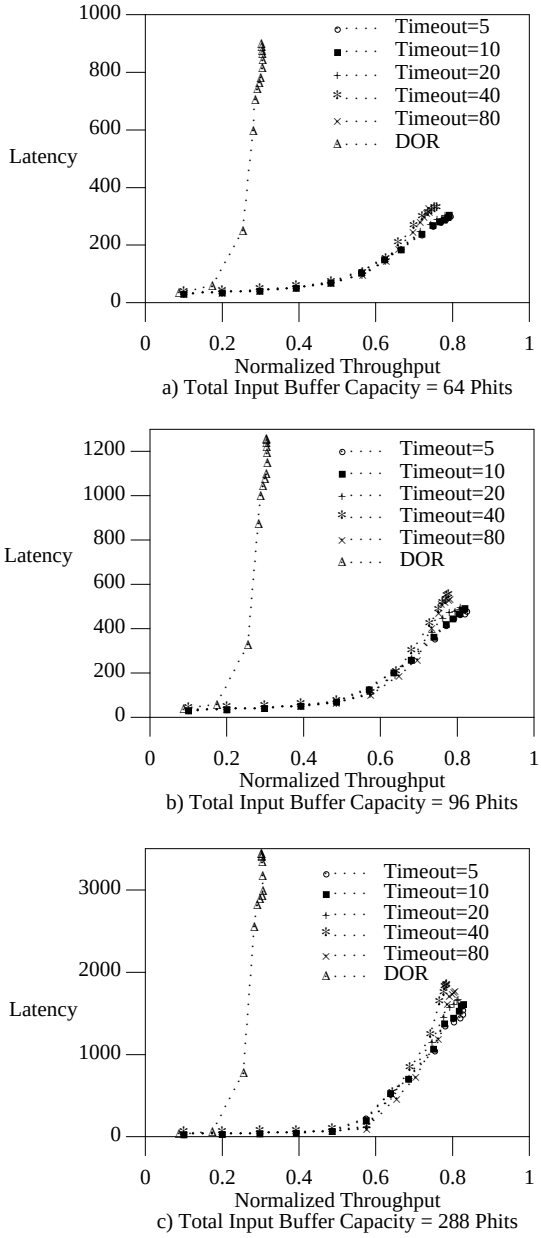


Figure 5.15: Bit-Reversal traffic: Latency vs. Normalized Throughput.

traffic are nearly independent of the diversion timeout value. With bit-reversal traffic, diverted packets do not necessarily follow poor paths that increase congestion.

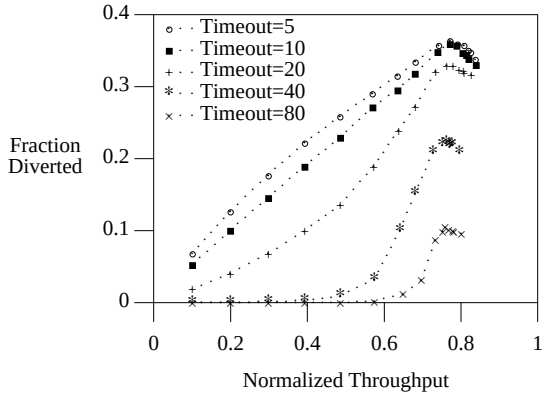


Figure 5.16: Bit-Reversal traffic: Fraction of Traffic Diverted vs. Normalized Throughput. Total input buffer capacity = 96 phits.

Figure 5.16 shows the fraction of traffic diverted versus throughput with total input buffer capacity 96 phits. These results show, as with transpose, that increasing timeout values greatly reduce the fraction of traffic diverted. Since diverted packets are handled less efficiently than packets on DVCs, these results and the transpose traffic results indicate that long timeout values should be used.

5.4.4. Uniform Traffic Pattern

For uniform traffic in a square mesh, DOR distributes load evenly across the links that comprise the network bisection and thus should perform well. In contrast, some adaptive routing schemes tend to steer more traffic toward the center of the network, causing congestion [Pert92].

Figure 5.17 shows latency versus throughput for uniform traffic with total input buffer capacity 64 and 288 phits. The results show that the performance of routed DVCs is close to that of DOR. Unlike transpose and bit-reversal traffic, with uniform traffic the use of larger buffers improves performance. Increasing the

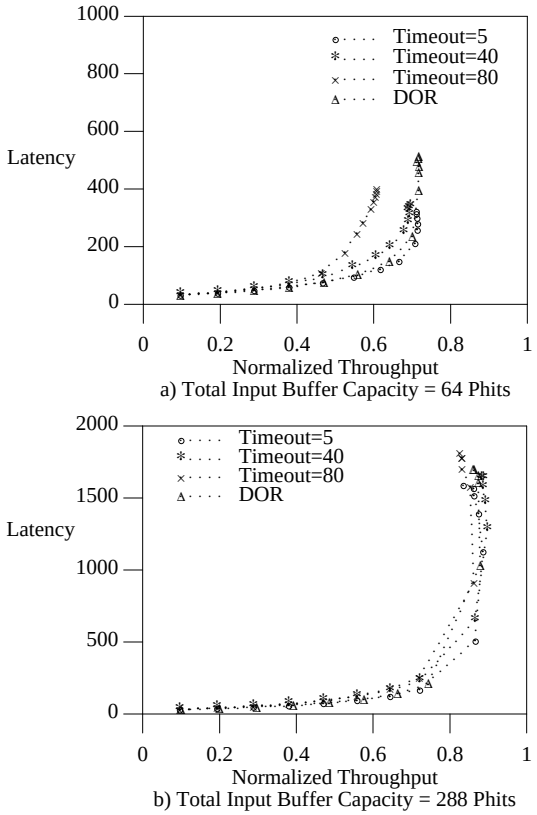


Figure 5.17: Uniform traffic: Latency vs. Normalized Throughput.

buffer capacity increases the number of flows that can be represented at any instant by the packets that are present at a switch. Larger buffers are therefore more helpful for uniform traffic with $O(N^2)$ flows than for the previous traffic patterns which have only $O(N)$ flows (one from each source node). Performance also improves with the use of smaller timeout values which effectively increase the useful buffer capacity by enabling more packets to take advantage of the 32 phit diversion buffers. With large buffers (288 phits), routed DVCs and DOR have nearly identical performance.

For routed DVCs with short timeouts, as the applied load increases beyond saturation the network throughput decreases slightly. This may occur because

congestion in the primary virtual network causes a larger number of packets to enter the diversion virtual network which has limited buffering and therefore limited throughput.

5.4.5. The Impact of Network Size

For a larger mesh network of size 16×16 and the same traffic patterns as used previously, Figures 5.18, 5.19 and 5.20 show latency versus throughput, and Figures 5.21, 5.22 and 5.23 show the fraction of traffic diverted versus normalized throughput. For non-uniform traffic, the results show that routed DVCs significantly outperform DOR, but the performance difference is smaller than on the 8×8 mesh. With the larger network, packets travel longer distances, and there are more opportunities for delays and deadlocks. Hence, the fraction of packets diverted tends to be larger than on the 8×8 mesh, resulting in more of the traffic facing congestion as with DOR.

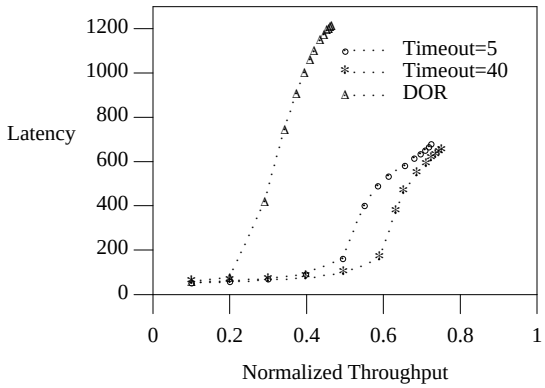


Figure 5.18: 16×16 Transpose traffic: Latency vs. Normalized Throughput. Total input buffer capacity = 64 phits.

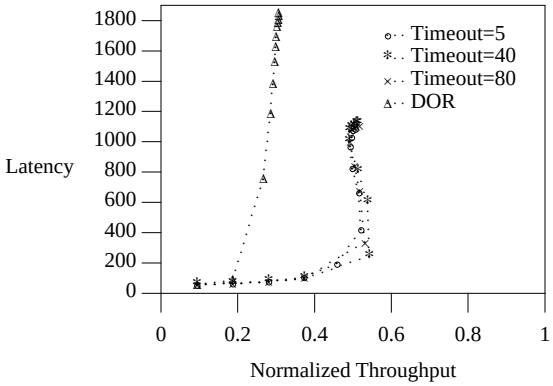


Figure 5.19: 16×16 Bit-Reversal traffic: Latency vs. Normalized Throughput. Total input buffer capacity = 64 phits.

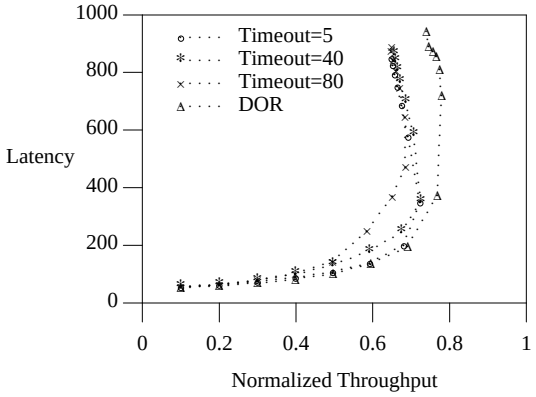


Figure 5.20: 16×16 Uniform traffic: Latency vs. Normalized Throughput. Total input buffer capacity = 64 phits.

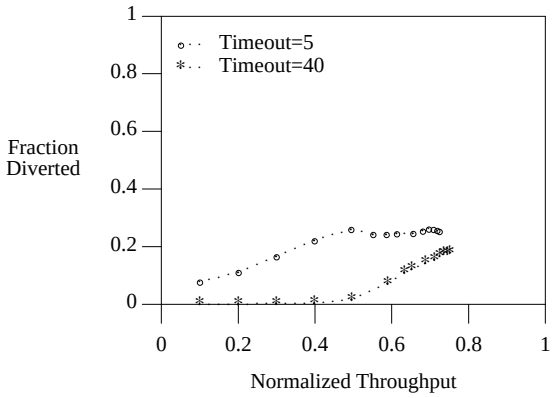


Figure 5.21: 16×16 Transpose traffic: Fraction of Traffic Diverted vs. Normalized Throughput. Total input buffer capacity = 64 phits.

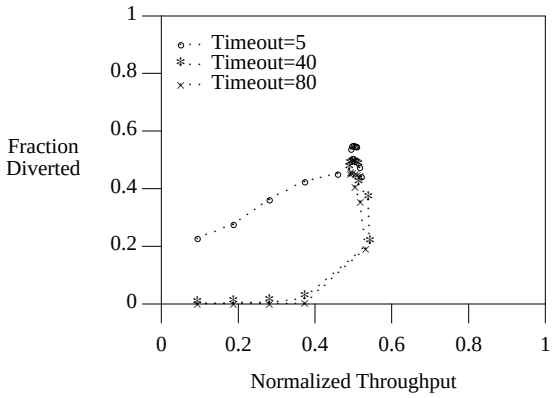


Figure 5.22: 16×16 Bit-Reversal traffic: Fraction of Traffic Diverted vs. Normalized Throughput. Total buffer capacity = 64 phits.

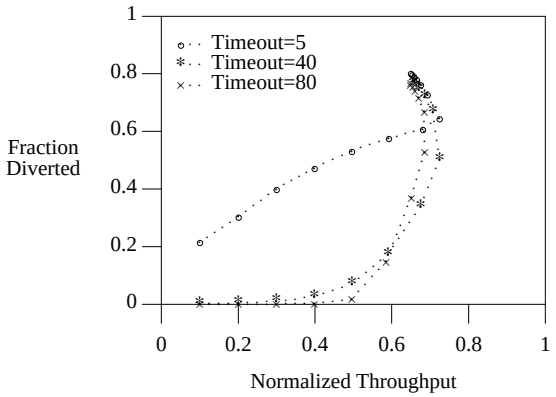


Figure 5.23: 16×16 Uniform traffic: Fraction of Traffic Diverted vs. Normalized Throughput. Total input buffer capacity = 64 phits.

5.5. Summary

In this chapter, we presented a deadlock avoidance scheme for DVC networks. The scheme enables each Dynamic Virtual Circuit to be established on any path from source to destination without the possibility of deadlocks involving packet buffer resources or virtual circuit manipulation operations. To guarantee deadlock-freedom in DVC networks, our solution decouples data packet routing and circuit manipulation operations. To enable data packets to use unconstrained virtual circuit paths, we leverage the existing approach from packet switching networks of allowing data packets that encounter buffer dependency cycles to transition to a dependency cycle-free virtual network, the *diversion network*. To avoid deadlocks involving circuit manipulation operations, we develop a new approach, based on an analysis of control packet arrival sequences, which guarantees that control packets cannot experience blocking across switches.

We presented correctness arguments showing that the DVC algorithms ensure the network is deadlock-free and that data packets are delivered to correct

destinations. The arguments are based on an elaboration of the state space of a switch and the possible state transitions that are allowed by the DVC algorithm.

We presented a hardware/firmware architecture for the implementation of the DVC mechanism with deadlock avoidance. Our analysis shows that the hardware requirements to implement this scheme are modest, enabling practical implementation. Our performance evaluation results show that with virtual circuits, global routing optimization is possible and provides performance superior to fixed routing. Furthermore, the results show that the use of deadlock-free escape paths is sufficiently infrequent to preserve the bandwidth efficiencies of the DVC mechanism.

Chapter Six

Reducing Cell Loss in Low Complexity Multi-Path Multistage Switching Fabrics

As described in the previous chapters, the DVC mechanism enables efficient communication by establishing a connection which is routed onto a low latency network path from a source to a destination. To investigate whether similar approaches can be useful in the context of a network switch, we evaluate in this chapter the potential performance benefits of explicitly routing selected flows through a large-scale ATM or IP switch. These benefits may motivate the development of mechanisms that enable some flows to use efficient connections, such as virtual circuits, established on selected paths through a network switch.

Modern network switches enable the creation of high-speed multiservice networks that simultaneously support real-time (deadline sensitive) and best-effort traffic [Turn86, Part94]. Multiservice networks eliminate redundant costs of building and operating multiple single-service networks, and large, scalable switches provide low cost per port and convenient central management.

An $N \times N$ ATM or IP switch consists of N input ports, N output ports, and a switching fabric that forwards packets or cells from input ports to output ports. Small switches can be built with a crossbar or shared bus organization. For cost effectiveness and high performance, large switches are typically implemented with a multistage organization, in which each stage consists of a number of switching elements (SEs) [Part94].

A key goal in the design of such switches is to minimize the loss of cells or

packets. Cell loss degrades the quality of real-time traffic and necessitates costly retransmission and congestion control throttling of best-effort traffic [Jaco88]. Cell loss is the result of conflicting demands by multiple cells for access to resources, such as communication links or buffer space.

The existing techniques for reducing cell loss, which we surveyed earlier in Section 2.4, fall into two categories. Some techniques limit the arrival of traffic from other switches or sources to the switch inputs. Other techniques reduce cell loss within the switching fabric without limiting the arrival of traffic to the switch.

An important sub-category of the second category is the use of multi-path switches, in which multiple paths are provided in the switching fabric for each source-destination pair. The use of multi-path switches enables load balancing and the reduction of the intensity of cell bursts that can cause buffer overrun and hence cell loss. Multiple paths can be provided by adding extra switching stages, sometimes called ‘‘distribution’’ stages, to a minimal banyan network that alone provides only one route from each input to each output. We refer to this minimal banyan network as the *routing network* since it can route cells from any input to any output.

The focus of this chapter is on the effective management of a low-cost subset of multi-path switches, those with only a single additional stage for providing alternate paths. Specifically, we evaluate the potential benefits of alternate switch configuration and traffic management schemes for reducing cell loss in these switches processing a mix of real-time and best-effort traffic. The switching fabric configurations we consider differ in the manner in which the single extra stage is attached to the routing stages. The management policies we consider differ in the

control of the routes that cells use to traverse the switch. We measure the relative performance of policies that explicitly route some flows onto single paths and policies that uniformly distribute all traffic over all available paths.

The chapter is organized as follows. Section 6.1 gives an overview of the switch design issues that we investigate in this chapter. Section 6.2 characterizes the impact on CLR of traffic load, buffer capacity, and the location of conflicts. Section 6.3 uses the characterization for the evaluation of switch configurations and routing policies.

6.1. Switch Design Considerations and Performance Metrics

The multistage switching fabrics under consideration are composed of small $n \times n$, output-queued switching elements (SEs), such as the AT&T T7650 Phoenix ATM Switch Element [Low97] or the Fujitsu MB86680B switching element [Fuji94]. In keeping with our focus on low complexity switches, we assume the switching elements do not distinguish between real-time and best-effort traffic in a multiservice network. Hence, all traffic contends for switch resources (buffer space and link bandwidth) with equal priority. The output buffers are first-in first-out queues with random tie-breaking for simultaneous arrivals. The topology is a butterfly network consisting of $\log_n N$ stages of N/n SEs per stage, plus one additional *distribution* stage before the $\log_n N$ routing stages. The distribution stage provides n paths through the fabric for each input-output pair (Figures 6.10 and 6.11 in Section 6.3.1 show two examples of switching fabrics that satisfy our topology requirements). The link bandwidth between adjacent stages of the switching fabric equals the combined bandwidth of the inputs to the switch, i.e., we

assume the switching fabric does not provide *bandwidth expansion* in which the switching fabric would have higher bandwidth than the switch provides externally.

A key performance metric for a switch is the *Cell Loss Ratio* (*CLR*), which is the ratio of the number of cells dropped by the switch to the number of cells injected into the switch. Cell loss is caused by bursts of arrivals of cells to a buffer within the switching fabric, causing the buffer to become full. Cells that subsequently arrive to the full buffer are discarded.

One way to reduce cell loss is to provide multiple paths through the switch. Multiple paths can be used to reduce the number and severity of flow conflicts which can cause buffer overflow and cell loss. Alternate paths are provided by adding distribution stages to the switching fabric.

For switching fabrics that have only a single distribution stage, a design issue that affects the CLR is the manner in which the distribution stage is connected to the first stage of the minimal banyan network. The configuration of this interconnection determines the precise set of alternate paths through the switching fabric. Therefore, this configuration also determines the locations in the switching fabric where flows can encounter resource contention that can lead to cell loss. For example, a distribution stage using the standard butterfly configuration has the property that if any of the alternate paths for two flows conflict, then all their alternate paths conflict. This can potentially lead to a high CLR. In Section 6.3.1, we investigate whether different configurations can be used which result in fewer conflicts between alternate paths, leading to a lower CLR.

Another key design issue that affects CLR is the routing policy that is used to exploit the alternate paths through the switching fabric. There are two basic

approaches: *oblivious distribution*, and *explicit routing* of flows onto single paths.

With oblivious distribution, the distribution stage uniformly distributes the cells of each flow across all the alternate paths. To implement this policy, the distribution stage simply cycles through the set of all possible switch settings, causing cells to be distributed onto the alternate paths without being examined and routed in the distribution stage. Oblivious distribution reduces the magnitude of traffic bursts that a single heavy flow can apply to any buffer in the routing network. However, a disadvantage of oblivious distribution is that alternate paths for two different flows can conflict in the routing network, leading to cell loss.

The alternative to oblivious distribution is to route flows onto individual paths. In many cases, routing can be used to avoid conflicts by directing flows onto paths that are not used by other flows, thus potentially reducing CLR compared to the use of oblivious distribution. To determine the potential benefits of this approach, in Section 6.3.2 we evaluate and compare oblivious distribution to an idealized policy that routes some flows onto single paths.

6.2. Characterizing Cell Losses in Multistage Switches

Effective configuration and management of multistage switching fabrics require an understanding of the key factors that determine the CLR. In Section 6.2.1 we use simulations to characterize the impact of three such factors: traffic load, buffer size, and the location of conflicts for resources between traffic flows. Since the result of interest is the rate of *rare events* (cell losses), simulations that produce statistically significant results are time-consuming — some taking several days on a modern workstation. This makes it impractical to evaluate a

wide range of switch configurations, routing policies, and traffic patterns. In order to overcome this limitation, we use the characterization from Section 6.2.1 to approximate CLR for numerous configurations that have not been directly simulated. The approximation technique is presented in Section 6.2.2.

We adopt the common modeling assumption that traffic arriving at the first stage of the multistage switch has memoryless, Bernoulli arrival [Four97, Desm91]. Even with the assumption of Bernoulli arrival to the switch inputs, non-initial stages have different arrival distributions (determined by the departure process of cells from the previous switching stage). Hence, a simple Geo/D/1 Markov model of a single queue can be applied to evaluate cell loss in the initial stage, but not in subsequent stages. Therefore, we take the approach of employing discrete-event simulation to obtain cell loss statistics in the multistage case. Each simulation run covers 10^{10} simulated cycles, where a cell entirely arrives to or departs from an SE in a single cycle. Therefore, the simulation provides meaningful CLR values down to about 10^{-7} or 10^{-8}.

6.2.1. Simulation-Based Characterization of Factors Determining the CLR

The key factors that determine the CLR are traffic load, buffer size, and the locations where traffic flows contend for resources in the switching fabric. We use simulation to measure cell loss in each stage of a fabric under uniform traffic, and under non-uniform traffic that consists of a mix of real-time and best-effort flows (appropriate for multiservice networks).

To characterize the impact of the three factors listed above on cell loss, we evaluated two switches under a uniform traffic workload: a 2×2 switch composed

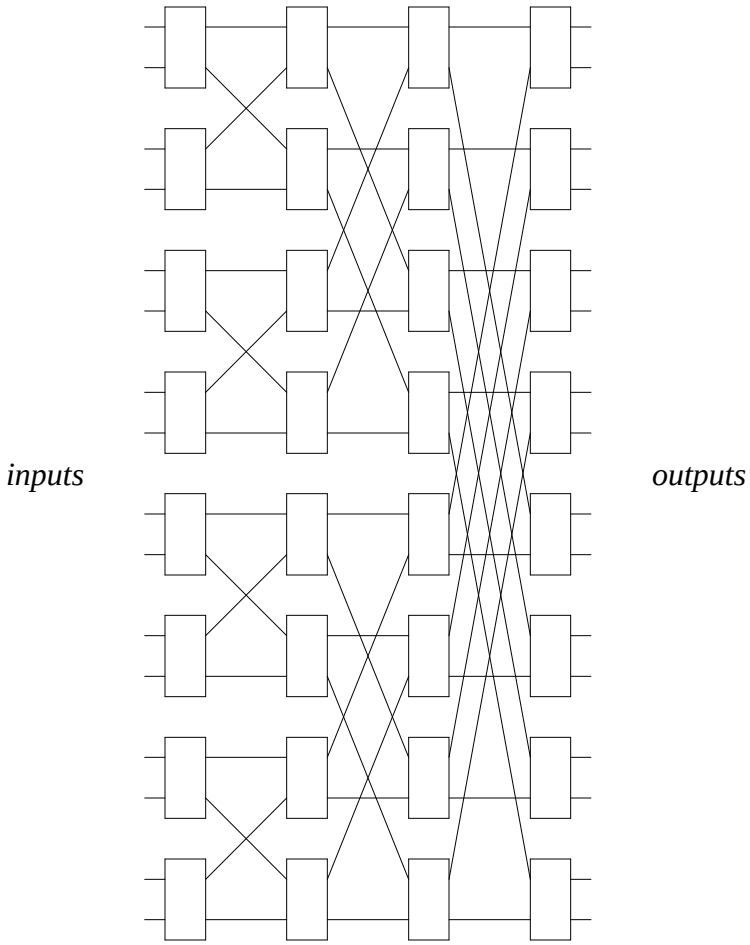


Figure 6.1: 16×16 butterfly switching fabric, 2×2 SEs, no distribution stage. Provides exactly one path between each switch input and each switch output.

of a single 2×2 SE, and a 16×16 switch composed of four stages of 2×2 SEs interconnected in a butterfly topology without a distribution stage (Figure 6.1).

For the 2×2 switch, the simulation results in Figure 6.2 show how CLR varies with the capacity of each output buffer and with the applied load. The applied load is the utilization of the external inputs to the switch. The results show that CLR increases rapidly with applied load, since at high load more cells are present in the

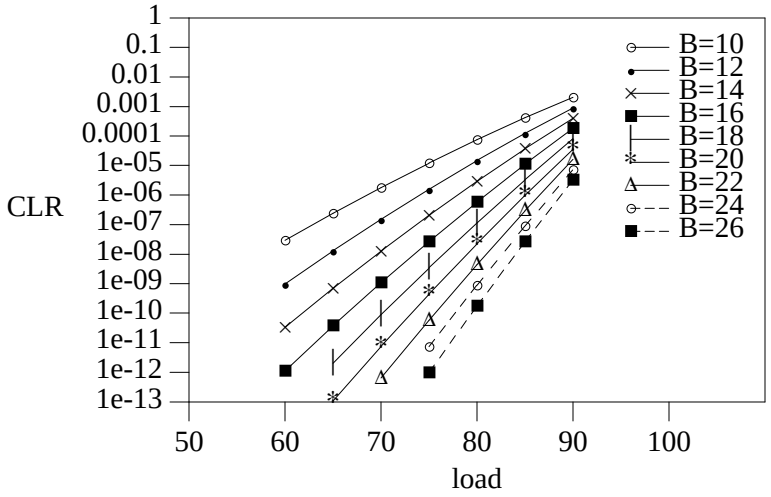


Figure 6.2: CLR versus load, 2×2 SE, uniform traffic, B is output buffer capacity in cells.

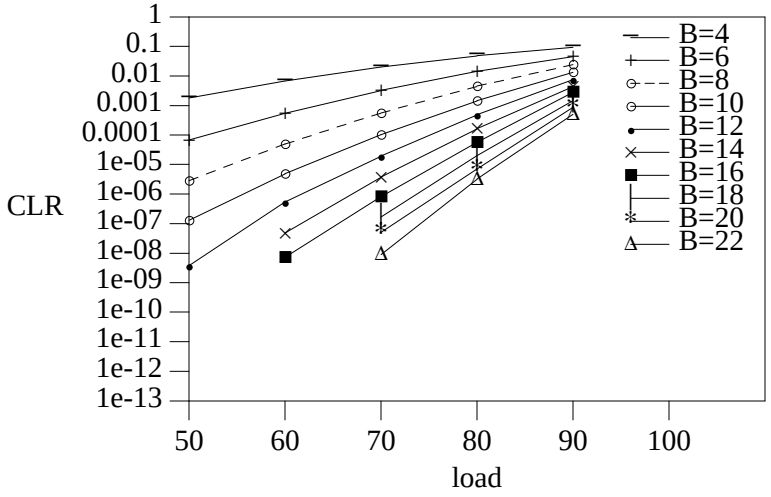


Figure 6.3: CLR versus load, 16×16 switch, 2×2 SEs, no distribution stage, uniform traffic. B is output buffer capacity in cells.

network contending for link and buffer resources. The results also show that CLR decreases as buffer capacity is increased. With larger buffers, more intense bursts of arriving cells are required before a buffer will overflow and drop subsequent

cells.

Figure 6.3 presents simulation results for a 16×16 multistage switch. With such a switch, the CLR is much higher than for a single 2×2 switching element (Figure 6.2). For the same buffer capacity per switching element and the same applied load, the 16×16 switch might be expected to have a cell loss ratio that is a factor of four higher than for the 2×2 switch, since in the 16×16 switch each cell traverses four 2×2 switching elements (one in each stage). However, by comparing Figures 6.3 and 6.2, we observe that the cell loss ratios differ by a much larger factor, as much as two orders of magnitude. This difference cannot be caused by increased losses in the initial stage of the 16×16 switch, since the SEs in the initial stage and the 2×2 switch experience statistically identical traffic (Bernoulli arrival with uniform source and destination distribution). Therefore, the huge difference in overall cell loss is caused by higher cell loss in the non-initial stages of the 16×16 switch.

Figure 6.4 breaks down the cell loss ratio in the 16×16 switch by stage number. The graph shows that compared to the initial stage (stage number 0), subsequent stages experience significantly higher cell loss, especially in the operating region of practical interest, where the CLR is below 10^{-6} [Suzu92, Henr93]. Since buffer overflow leading to cell loss is caused by traffic burstiness (when applied load is less than 100%), we conclude that the statistical arrival process of cells to non-initial stages has higher burstiness than the Bernoulli arrival process to the initial stage. From Figure 6.4, we observe also that the largest increase in cell loss is from the initial stage to the next stage, whereas all subsequent stages have much smaller increases. Therefore, most of the increase in

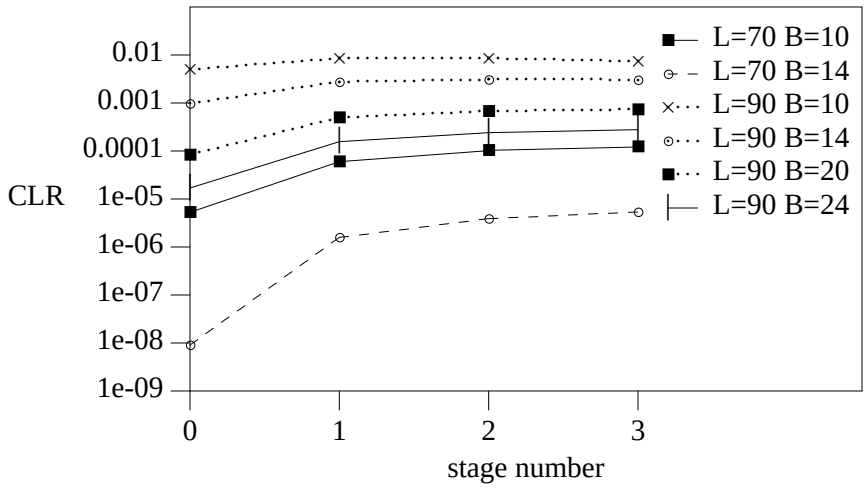


Figure 6.4: CLR versus stage number, uniform traffic, 16×16 switch, 2×2 SEs, no distribution stage, L=traffic load, B is output buffer capacity in cells.

burstiness is contributed by the handling of cells in the initial stage.

In addition to considering cell loss with uniform traffic, we have also examined non-uniform traffic scenarios and obtained similar results. The non-uniform traffic patterns considered consist of a few heavy real-time flows combined with a relatively light load of uniformly distributed best-effort traffic. These scenarios may be reasonable for a workgroup or department sharing a switch in which most of the traffic consists of delay insensitive traffic (e.g., file transfers, e-mail), while for example a few heavy real-time flows deliver high-resolution digital video streams to a few desktops. We select traffic load levels such that the combined applied load does not overload the capacity of any internal link of the switching fabric. This constraint is consistent with the use of end-to-end congestion control which would throttle best-effort traffic in scenarios where links are overloaded [Jaco88].

To characterize the impact on CLR of the location of conflicts, suppose that two real-time flows of equal load (30% each, or 25% each) contend for a single output of an SE in the *initial* stage of a multistage switching fabric. In addition, suppose this traffic is combined with uniformly distributed background traffic contributing an applied load of 30%. Figure 6.5 shows the results of the simulation of a single SE under such non-uniform traffic to analyze the cell losses that occur at the site of contention. The results show that CLR increases with applied load, and decreases as buffer capacity is increased, similar to the CLR behavior under pure uniform traffic.

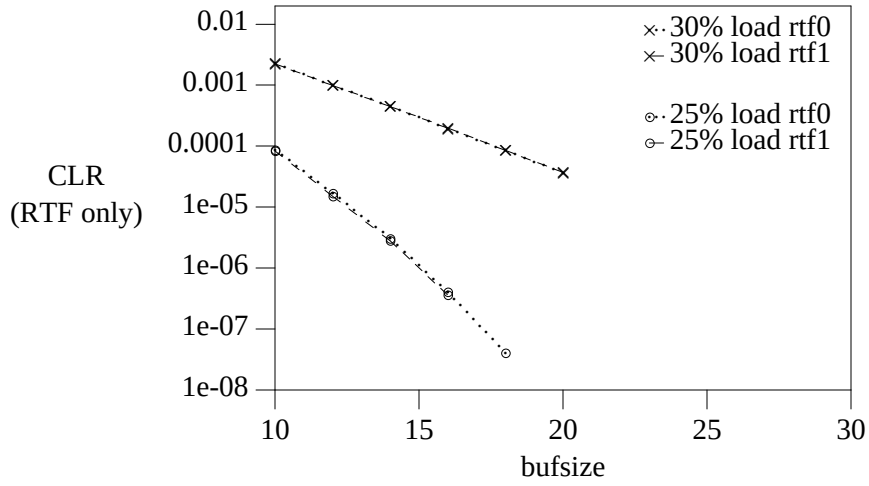


Figure 6.5: CLR versus buffer capacity, 2×2 switch (one 2×2 SE), two conflicting independent equally loaded real-time flows, rtf0 and rtf1. Results are shown for real-time flows with load 30% each, and for real-time flows with load 25% each. Uniform background traffic load = 30%.

Figure 6.6 shows a similar scenario for a multistage switch. Two independent real-time flows of equal load (30%, 25%, 20%, or 15% each) conflict in the *second* stage of a multistage switching fabric, along with a uniform background traffic of

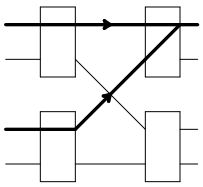


Figure 6.6: Traffic pattern for Figure 6.7. 4×4 switch, 2×2 SEs, no distribution stage. Two independent, equally loaded real-time flows conflict in the second stage.

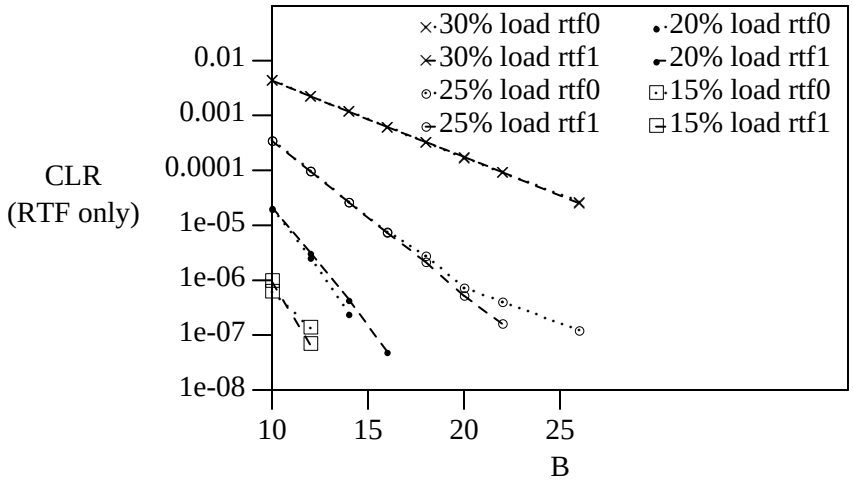


Figure 6.7: CLR versus output buffer capacity B, 4×4 switch, 2×2 SEs, no distribution stage, two equally loaded independent real-time flows conflict in the second stage. Uniform background traffic load = 30%.

load 30%. The cell loss ratio experienced by the two real-time flows when there is a 30% uniform background traffic is shown in Figure 6.7. All the cell loss events that occurred in these simulations were in the second stage where the real-time flows conflict.

Comparing Figures 6.7 and 6.5 reveals that for any particular buffer size and real-time flow load (with 30% background traffic), there are always fewer losses when the real-time flows conflict in the first stage. The difference is most

significant (more than an order of magnitude) when loss ratios for both scenarios are low and therefore within the range of practical interest. For example, when each real-time flow has load 25% and the output buffer capacity is 18 cells, the single-stage switch has a loss ratio of about 4×10^{-8} while the two-stage switch has a loss ratio of about 3×10^{-6}.

6.2.2. Extrapolation Technique

The results above can be used to quickly estimate CLR in multistage switches for numerous configurations that have not been directly simulated. The approach is based on the observation that when a traffic pattern is applied to each stage, the result is that losses from one stage to the next increase the most from the initial stage to the second stage. We collect, in advance, a set of simulation results for different patterns to determine what is the cell loss ratio when conflicts occurs in the second stage. We also use the Markov model to determine what the cell loss ratio would be for similar patterns of flow contention in the initial stage. If we plot the multiplicative factor by which CLR increases from the first to the second stage against the CLR in the first stage, we obtain a curve that can be interpolated to provide a first-order estimate of cell loss for a variety of traffic conditions.

Figure 6.8 plots the scale-up factor determined through simulation for a 4×4 two-stage butterfly switch with 2×2 SEs that have 8 buffer slots per output port. The scale factor is plotted against CLR in the first stage. Figure 6.9 shows the same results as Figure 6.8 except for a two-stage 16×16 switch consisting of 4×4 SEs. In each figure, one curve shows how the scale-up factor for uniform traffic grows, as applied load is varied. The other two curves show scale-up factor for

non-uniform traffic patterns with fixed uniform background traffic rate of 30% and 10%, respectively. For each curve, several non-uniform traffic scenarios were simulated and plotted onto one curve. The simulated scenarios differed not only in the applied loads of the real-time flows, but also in the number of real-time flows contending for a single output queue. Scenarios with from 1 to 4 contending flows were simulated for each curve. The smoothness of the non-uniform traffic curves in Figures 6.8 and 6.9 suggests that the scale-up factor is somewhat insensitive to the exact number of flows and their individual applied load levels.

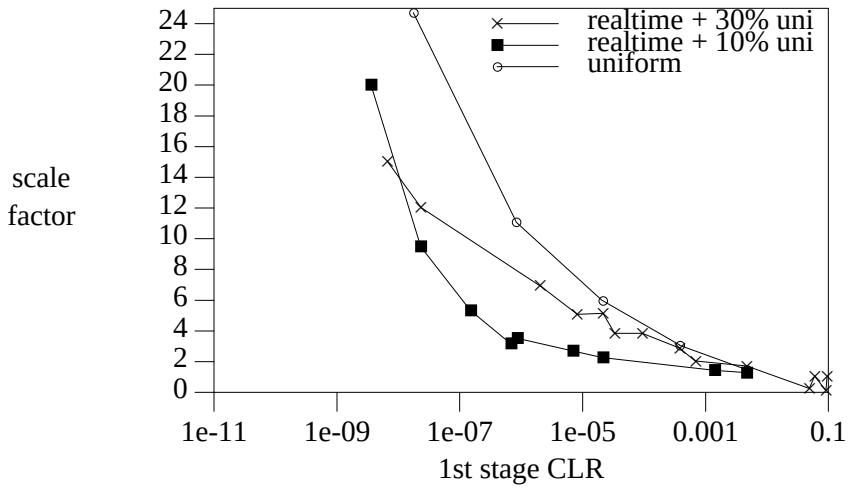


Figure 6.8: Scale factor (ratio of CLR in 2nd stage to CLR in 1st stage) vs. 1st stage CLR, 4×4 switch, 2×2 SEs, no distribution stage, output buffer capacity = 8.

From these results we observe that the increase in CLR from first to second stage is greater for switches with 2×2 SEs than for switches with 4×4 SEs. This is because an output queue of a first stage 4×4 SE experiences more arrival burstiness than does an output queue of a first stage 2×2 SE. In particular, with 4×4 SEs, as many as four cells may arrive each cycle to an output queue, whereas only two

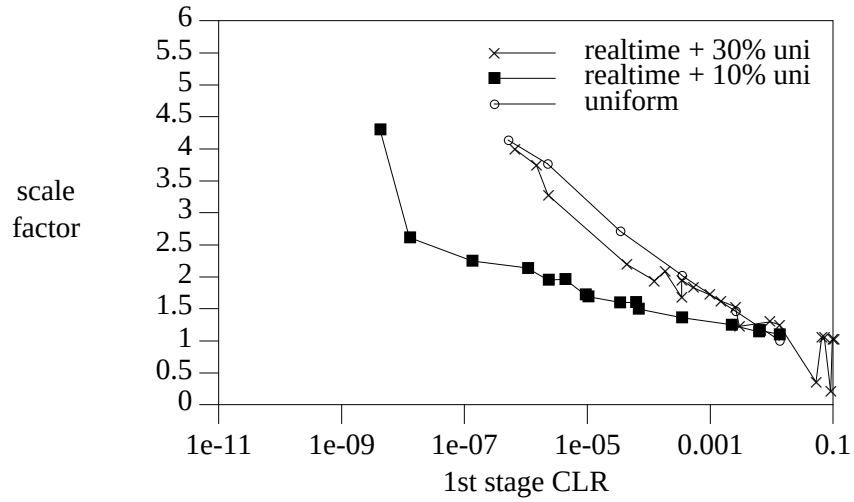


Figure 6.9: Scale factor (ratio of CLR in 2nd stage to CLR in 1st stage) vs. 1st stage CLR, 16×16 switch, 4×4 SEs, no distribution stage, output buffer capacity = 8.

cells may arrive to an output queue of the 2×2 SE. Given the higher burstiness in the traffic to the first stage 4×4 SE output queue, the relative increase in CLR from first to second stage due to the burstiness of the departure process from the first stage is more pronounced in the 2×2 SE case.

The data in Figure 6.8 and 6.9 can be used to estimate loss ratios under a wide variety of traffic conditions without performing time-consuming simulations. The technique estimates CLR in a SE of a non-initial stage of a multistage switch in a three step process. The first step is to invoke a Markov model of an initial stage SE output queue, using as inputs the flow rates of the traffic flows routed to the non-initial stage SE output queue under study. The second step is to correct for the fact that the Markov model does not give the correct CLR, since the traffic to the non-initial stage does not have Bernoulli arrival. To make the correction, we select the curve from Figure 6.9 and 6.8 with the same background uniform load as the actual

traffic and interpolate along that curve to get a scaling factor for the CLR value returned by the analytical model. Finally, we multiply that CLR value by the scaling factor to obtain an estimate of the actual CLR experienced by traffic at the SE under study.

Using several example cases, we have compared the CLR results obtained with this extrapolation technique to the CLR results obtained through simulation. We used this comparison to verify that the extrapolation technique is accurate to within a factor of two for the traffic patterns considered, which is sufficient accuracy for our purposes.

6.3. Evaluation of Multistage Switch Design Options

There are two basic approaches to using the capabilities of a multi-path switch: distributing the cells of each flow among the available paths into multiple subflows or using one of the available paths for all the cells of a particular flow. For both approaches a key factor in determining switch performance is the number of alternate paths for each source-destination pair. The specific routes of the alternate paths are another key factor since they can impact the probability of resource conflicts between cells of different paths.

As discussed earlier, an extra distribution stage can be added to a multistage switch in order to provide multiple paths between every input and output. One of the ways of using the multiple paths to reduce cell loss is to distribute heavy real-time flows into two or more subflows, thus reducing contention in the routing stages prior to the final stage. At the final stage, the split subflows merge back together and exit the switch. In large switches with many routing stages, or in

switches with heavy background traffic, losses in the output stage due to merging subflows of a single flow can overwhelm any advantage due to supporting multiple paths. Therefore, we exclude that possibility by assuming in the following that the SEs in the output switching stage are special parts with buffers that are sufficiently larger than the standard SEs comprising the other stages so that any contribution of the output stage to the overall CLR is negligible.

6.3.1. Evaluation of Distribution Stage Interconnection Alternatives

The interconnection of the distribution stage to the initial routing stage determines the specific routes of alternate paths through the multistage switching fabric. These routes determine the locations in the switching fabric where flows will conflict and cause cell loss. This fact motivates an evaluation of whether some interconnections between the distribution stage and the routing stages may be preferable to others, specifically, preferable to the standard butterfly interconnection. For this evaluation, we use the routing policy of oblivious distribution to exploit the alternate paths. With this policy, each flow is uniformly divided into lighter weight ‘‘subflows’’ across all the alternate paths.

We compare the expected cell loss with two distribution stage configurations: a butterfly configuration and an alternative configuration that reduces cell losses when subflows contend in certain routing stages. For particular example settings of traffic and switch parameters, we also examine the probability that arriving flows satisfy CLR requirements.

Suppose an $N \times N$ switching fabric has $\log_n N$ stages of $n \times n$ SEs connected in a butterfly configuration. Then the $\log_n N$ stages provide exactly one route for each

source-destination pair. If we prepend a single distribution stage to this fabric, then each source-destination pair will have alternate paths through the switch.

The SEs and links in routing stages prior to the output switching stage partition into n distinct subnetworks, as shown in Figure 6.10 (for $n=2$). For each source-destination pair, providing exactly one alternate path through each subnetwork guarantees that the n subflows of a common flow do not conflict before reaching the output switching stage. Distribution stage configurations with this property connect the n outputs of each SE to all n subnetworks.

One such distribution stage configuration is the butterfly interconnection that is identical to the interconnection to the output switching stage. With this configuration, all n subnetworks handle identical traffic patterns. If the subflows of two flows conflict in one subnetwork, then their counterpart subflows in the $n-1$ other subnetworks also conflict. The conflicts occur in the same routing stage in all n subnetworks. This is illustrated by example in Figure 6.10.

Suppose an alternative distribution stage configuration could be found that guarantees that within certain routing stages, dual conflicts never occur. This alternative would seem to be desirable, as it is the configuration that would most reduce the cell losses in the presence of conflict in those stages. Some pairs of flows would improve their CLR by a factor of n, since the subflows would conflict in only one subnetwork rather than in n. An example of such a configuration is shown in Figure 6.11. If two subflows of two different flows conflict in routing stage 0 of one subnetwork, their counterparts in the other subnetwork encounter two distinct SEs in routing stage 0 and therefore do not conflict. For larger switches, it is possible to extend this idea to prevent dual conflicts in multiple (but

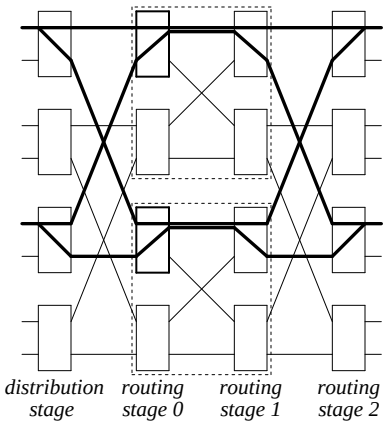


Figure 6.10: Butterfly-connected distribution stage. 8×8 switch, 2×2 SEs, 3 routing stages, one distribution stage. The distribution stage is connected to the routing stages using a butterfly interconnection. Example: two subflows conflict in routing stage 0 in both subnetworks.

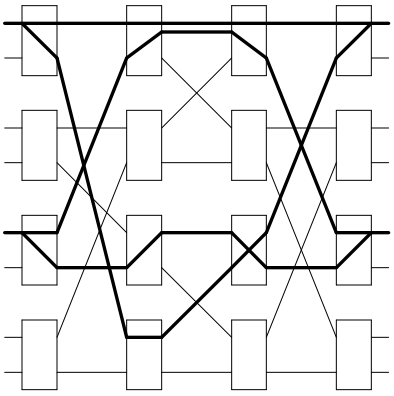


Figure 6.11: Alternative distribution stage configuration. 8×8 switch, 2×2 SEs, 3 routing stages, one distribution stage. The distribution stage is connected to the routing stage such that two flows cannot conflict in the first stage of both subnetworks of the routing part of the switch. Example: two subflows conflict in the top subnetwork but do not conflict in the bottom subnetwork.

not all) routing stages.

To understand the basic trade-offs between the butterfly configuration (Figure 6.10) and the alternative configuration (Figure 6.11), we examine their cell

loss for simple traffic patterns consisting of a pair of flows. There is a trade-off between two factors: 1) under some traffic patterns that result in *two* subflow conflicts with the butterfly configuration, there will be only one conflict (in the same stage) with the alternative configuration, and 2) under some traffic patterns that result in *no* subflow conflicts with the butterfly configuration, there will be one conflict with the alternative configuration. It can be shown that the number of patterns included under item 1 above is the same as the number of patterns under item 2. For the first set of patterns, the butterfly configuration results in twice the CLR of the alternative. For the second set of patterns, there are no conflicts, and thus a CLR of 0, with the butterfly configuration. However, with the alternative configuration, the CLR is the same as the CLR for this configuration under the first set of patterns. Based on these considerations, it can be shown that the expected value of the CLR for the two configurations is the same.

Despite the discussion above, the two configurations are not equally desirable under all traffic conditions and CLR requirements. We illustrate this with a particular example setting of switch and traffic parameters. We obtain the probability distributions of CLR with the two configurations. The distributions indicate which configuration provides the highest probability of meeting specific CLR requirements.

To obtain the probability distribution of CLR, we generated all ‘‘feasible’’ traffic patterns consisting of a mix of two real-time flows, each with load 40%, and a uniform background traffic with load 30%. Feasible traffic patterns are those that apply less than 100% load on each source and destination. We assume that all feasible patterns are equally likely. The traffic patterns are applied to the switches

of Figures 6.10 and 6.11. The extrapolation procedure of Section 6.2.2 is used to obtain estimates of CLR for the real-time flows. Figure 6.12 shows the CLR cumulative probability distribution. Restricting the real-time traffic to only two flows at a time does not change the fundamental trade-offs involved when more flows are allowed, but its simplicity enables a clear interpretation of the shape of the probability distribution. One consequence of using only two flows is that the simulated switch must be fairly small (8×8) in order to observe non-negligible probabilities of cell loss. Our studies of larger switches and more flows yield results that are consistent with those presented here.

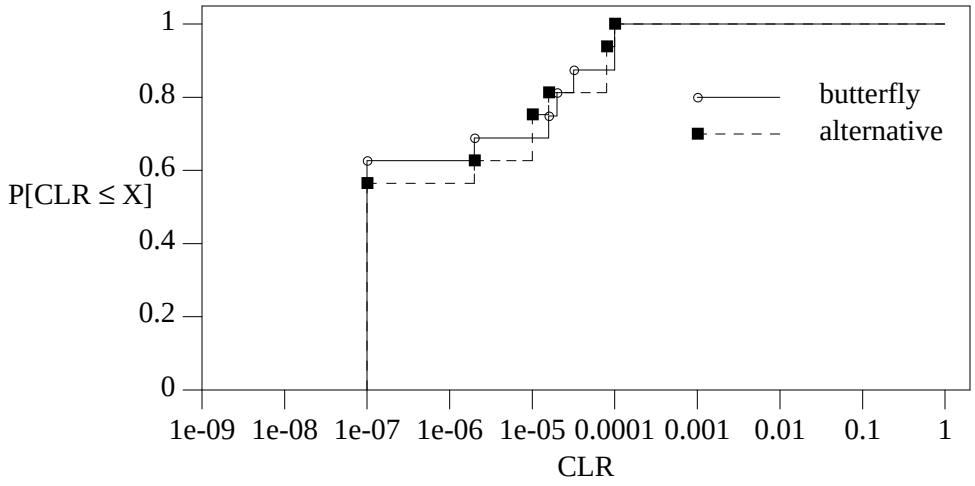


Figure 6.12: CLR cumulative probability distribution for the butterfly and alternative distribution stage configurations. 8×8 switch, 2×2 SEs, 30% background uniform traffic plus two real-time flows of rate 40% each, output buffer capacity = 8. The sources and destinations of the real-time flows are randomly varied to generate the distribution.

From Figure 6.12, we observe several discontinuities. Each discontinuity corresponds to a set of flow pairs that exhibit a unique conflict pattern and have the same CLR. For example, when the two flows do not conflict at any SE output, a

loss ratio of 10^{-7} results. The size of each jump indicates the size of the corresponding set of flow pairs. For example, 62% of flow pairs in the butterfly configuration have no conflicts, compared to only 56% of flow pairs in the alternative configuration.

With the particular parameters chosen for this example, the figure indicates which configuration has the higher probability of satisfying particular CLR requirements. For example, if the maximum acceptable CLR is in the range of 10^{-7} to just less than 10^{-5}, the butterfly configuration has higher probability of satisfying the requirement. However, in a narrow range starting from 10^{-5}, the alternative configuration is preferable. In general, the best configuration is highly dependent on the switch and traffic parameters, and on the CLR requirements.

6.3.2. Routing Versus Distribution

A key motivation of our work is to evaluate the potential benefits of adaptive routing for multi-mode traffic in multi-path multistage switches. In this context, *adaptive routing* means that for each flow one of the possible paths is chosen such that the overall performance of the switch is optimized. We assume idealized conditions that allow: 1) optimal choices to be made between distributing or routing each individual flow, and 2) optimal choice of route for the routed flows. Specifically, we assume global knowledge of the characteristics and destinations of all the flows that need to be handled by the switch. While such global knowledge is typically not available, our results provide an upper bound on the possible benefits of adaptive routing.

With a butterfly interconnection of the distribution stage to the rest of the

switch, we evaluate an ideal optimal policy, where each flow is either routed or distributed in order to minimize the CLR. The optimal choice is made by using the extrapolation technique to predict CLR for all distribution and routing choices exhaustively, and then selecting the best outcome. These results are compared with the policy of always distributing all the flows (Section 6.3.1).

Whether routing flows results in a net reduction in the CLR depends on a trade-off between two opposing effects. Routing can be used to place real-time flows on paths that do not conflict, thus reducing cell losses resulting from contention between these heavily loaded flows. However, routing concentrates all the load of a flow on a single path instead of distributing it uniformly across the n possible paths. Thus, using routed flows instead of distribution results in n times the load on each SE link that is used by a flow. Therefore, using routed flows increases cell loss caused by increased contention between each heavy flow and the uniform background traffic.

Figure 6.13 shows the potential benefits of the idealized optimal routing policy (the curve labeled ‘‘routing’’) compared to the policy of oblivious distribution (the curve labeled ‘‘distribution’’). The same method and traffic patterns as in Section 6.3.1 are used to generate the distribution of CLR among the traffic patterns for the two policies. The results show that use of the optimal routing policy can reduce the worst case CLR by nearly an order of magnitude (from $\mathrm{CLR} = 10^{-4}$ with oblivious distribution to $\mathrm{CLR} = 1.6\times10^{-5}$ with optimal routing). For patterns with low CLR, there are no conflicts even with the distribution policy, so the optimal policy is to distribute all the flows (routing flows could lead to higher contention with the background traffic). A cell loss ratio

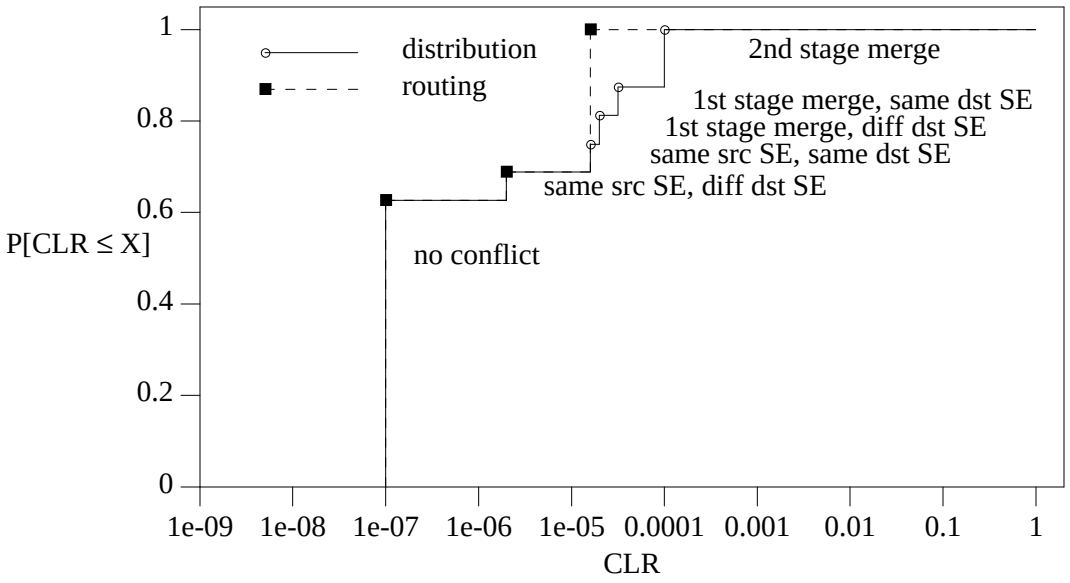


Figure 6.13: CLR distribution, for routing and for full multiple path distribution. 8×8 switch, 2×2 SEs, background traffic load 30% plus 2 real-time flows with load 40% and randomly varying source and destination, output buffer capacity = 8.

above 10^{-5} occurs under the distribution policy for patterns that result in conflicts between the real-time flows. If the background traffic load is reduced from 30% to 10%, as shown in Figure 6.14, the benefits of the optimal policy are increased. Specifically, the results show that the worst-case CLR is reduced by more than two orders of magnitude compared with the distribution policy. The trade-off between routing and distribution shifts in favor of routing since contention between a real-time flow and the background traffic is less likely to cause cell loss with lighter levels of background load.

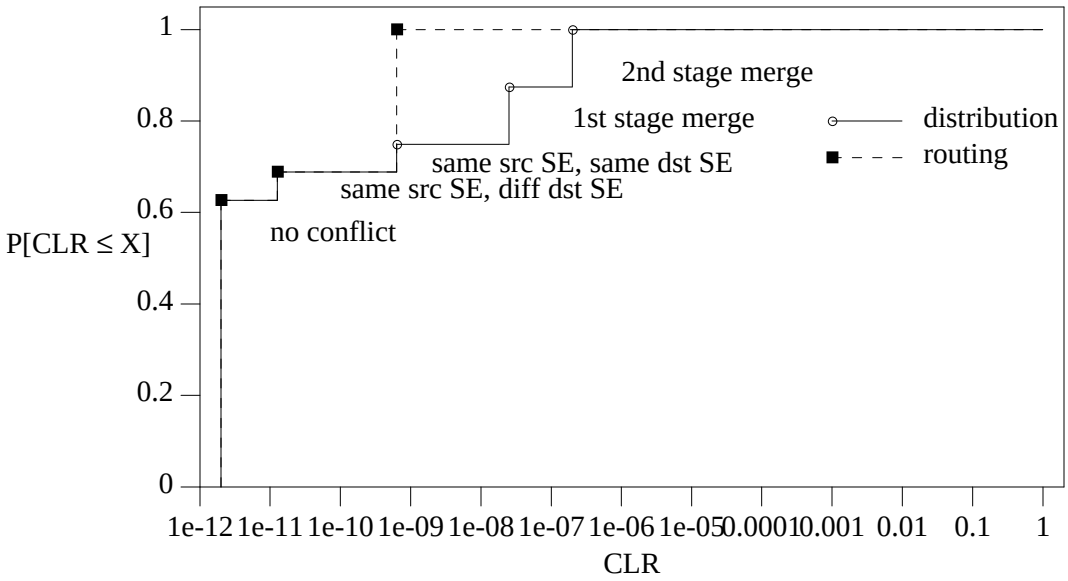


Figure 6.14: CLR distribution, for routing and for full multiple path distribution. 8×8 switch, 2×2 SEs, background traffic load 10% plus 2 real-time flows with load 40% and randomly varying source and destination, output buffer capacity = 8.

6.4. Summary

In this chapter, we investigated routing techniques for reducing the cell loss ratio (CLR) in a large-scale ATM or IP switch by adding a single distribution stage to a multistage switching fabric handling a mix of real-time and best-effort traffic. For a switching fabric composed of $n \times n$ switching elements, the distribution stage provides n alternate paths between each switch input and output. The alternate paths can be used to route each high-bandwidth real-time flow on a non-conflicting path, or each flow can be distributed uniformly across the n alternate paths, reducing the maximum load on any path. Compared to a switch with unique input-output paths, a switch with alternate paths can be used to dramatically reduce the cell loss ratio. For example, consider a traffic pattern consisting of a mix of

best-effort traffic at load 30% combined with two real-time flows, each with load 40%. If the paths of the real-time flows conflict in the switching fabric, an SE output link in the switching fabric will be saturated, causing an output queue in the SE to overflow and resulting in very high cell loss ratios (tens of percent) for the real-time flows. Routing the flows on non-conflicting alternate paths or distributing each flow uniformly across alternate paths results in a dramatic reduction in the CLR to acceptable levels.

For the multi-path switch using oblivious distribution of traffic flows, we have investigated the potential benefits of alternative ways to connect the distribution stage to the first routing stage of the switching fabric. With the butterfly interconnection, if flows that are distributed across all n alternate paths conflict on one path, then they conflict on all n paths. We evaluate an alternative configuration which also provides n alternate paths but avoids several cases in which flows would conflict on all n paths. Our analysis shows that although the alternative configuration reduces cases of multiple conflicts, it introduces cases of single conflicts that are not present with the butterfly configuration. As a result, the butterfly and alternative interconnections have the same average level of cell loss if all possible pairs of flows are equally likely traffic patterns. However, our evaluation results also show that for the specific types of traffic patterns that we considered, the alternative interconnection can provide a slightly higher probability than the butterfly interconnection of satisfying maximum bounds on the CLR, for a narrow range of CLR values.

To investigate the potential for using adaptive routing in the multi-path switch to reduce the CLR, we evaluated the performance of an idealized policy that can

make optimal choices between distribution and routing of the real-time flows. The results show that in comparison to oblivious distribution, this optimal routing policy can reduce the worst-case CLR by one or two orders of magnitude in the operating range of practical interest. These results indicate that routing of flows through multi-path multistage switches can be a useful design option in an environment where the pattern of real-time flows is known or can be inferred.

Chapter Seven
Summary and Conclusions

Continuing improvements in semiconductor technology are increasing the speed and number of transistors on a chip. As a result, microprocessors that form the processing building blocks of future parallel systems will be capable of generating and consuming increasingly heavy traffic loads. Interconnection networks supporting parallel systems must also increase in performance in order to avoid becoming the performance bottleneck. To meet this challenge, interconnection networks can also take advantage of the increasing number and speed of transistors on a chip to implement complex functionality in networking hardware that improves communication performance. In this dissertation, we proposed a mechanism, called *Dynamic Virtual Circuits* (DVCs), that accomplishes this goal.

The DVC mechanism combines the best features of adaptive routing and connection-based routing for multicomputer or cluster interconnection networks. With DVCs, the use of network bandwidth is optimized by minimizing the addressing and control information sent with each packet and the latency of forwarding packets through each intermediate switch. Unlike static virtual circuits, the DVC mechanism enables the network to adapt to changing traffic conditions by allowing any intermediate switch on a circuit's path to reroute the circuit. Unlike prior approaches to combining the benefits of adaptive routing and connection-based routing, with the DVC mechanism switches can reroute circuits quickly by using fast local operations without suffering long delays for coordination with

remote nodes. The DVC mechanism provides high performance routing in arbitrary topologies, thus allowing a single switch architecture to apply to multiple systems.

We presented an overview of the hardware necessary to support DVCs in the context of a communication coprocessor for scalable parallel systems. Dedicated hardware (switching and SRAM-based packet buffers) is used to handle most packets, while a programmable routing processor with DRAM is used to perform the more complex and less frequent circuit manipulation operations.

We described the basic techniques used by switches to break DVCs and adaptively re-establish them on arbitrary paths. In order to maintain the semantics of traditional virtual circuits while supporting fully-adaptive rerouting of circuits, the few packets that can arrive out of sequence at the destination are stamped with sequencing and addressing information. Since only a small fraction of the packets must carry this information, DVCs preserve the low per-hop processing and bandwidth overheads of traditional static virtual circuits.

We presented two approaches for eliminating deadlock in a DVC system: deadlock detection and resolution, and deadlock avoidance. Both approaches were designed to handle complex dependency cycles that include both packet buffer resources and dependencies arising from the circuit manipulation operations which manage Routing Virtual Channel (RVC) resources.

With deadlock detection and resolution, routing is unrestricted, as is the use of most buffer resources. In the absence of deadlock, such flexibility leads to efficient utilization of system resources, and thus to high performance. If deadlock occurs, a mechanism is invoked that identifies the deadlock cycle and temporarily

introduces additional buffer resources into the cycle to enable forward progress in a deadlock resolution procedure called *rotation*. Our scheme is based on a deadlock detection and resolution scheme originally developed for packet switching networks where switches use central queues to store packets. We present extensions that enable the scheme to be used in multicomputer and cluster networks, which typically use input-buffered switches. We present additional extensions that enable the scheme to be used in DVC networks. In particular, two alternative mechanisms were proposed for solving the problem of rotating unmapped data packets that are blocked waiting for RVC resources to be assigned to a new circuit. One approach reserves a single RVC for use by unmapped packets during rotation, whereas the second approach introduces a new packet type that operates as in packet switching. The approaches present a trade-off between the low hardware complexity of the first approach and the higher performance of the second approach.

Our proposed deadlock avoidance scheme for DVC networks allows virtual circuits to be established on any path without the possibility of deadlock. Compared to the deadlock detection and resolution scheme, the deadlock avoidance scheme places additional, minimal restrictions on buffer usage to avoid deadlocks. In particular, deadlocks are prevented by introducing a virtual network, called the *diversion network*, which provides an escape path for any packets that encounter buffer dependency cycles in the primary virtual network. For CDP and CEP control packets which are used to establish and teardown virtual circuits, diversion from virtual circuit paths is not allowed, and therefore an additional mechanism is needed to prevent deadlocks involving these control packets. Our

solution decouples control packets from data packets by providing an additional virtual network for exclusive use by control packets. The virtual network for control packets is provided with sufficient buffer capacity to guarantee that control packets never block waiting for a buffer slot to free at the next switch. We developed a complete DVC-based communication scheme that uses deadlock avoidance and presented correctness arguments to prove that the network is deadlock-free and that all packets are delivered to the correct virtual circuit destinations. We also analyzed the hardware requirements for implementing this mechanism and showed that the requirements are modest. Although in comparison to static virtual circuits, the DVC mechanism requires more packet buffer storage at each switch, most of the additional storage is for the control packet virtual network, which is accessed only when circuits are established, torn down, or rerouted. Therefore, this storage may reside in the Routing Processor's private DRAM without significantly degrading overall performance.

The experimental evaluation of the two approaches to deadlock elimination revealed that deadlock avoidance is preferable to the deadlock resolution approach that we investigated. Although deadlocks are rare, they will occur occasionally in systems with unrestricted routing policies. Since deadlock recovery is typically a relatively slow operation, it is likely that under heavy load the system will persistently introduce new packets into existing deadlock cycles faster than the resolution mechanism can remove them. In contrast, with deadlock avoidance, data packets can be diverted into a virtual network that requires pure packet switching with a restricted routing function to avoid deadlock. In our experiments, diversion is rare and therefore does not negate the bandwidth efficiency of

connection-based routing.

We examined the application of adaptive routing to a networking environment different from multicomputers or cluster interconnection networks. The focus of this investigation was to determine the potential of adaptive routing to exploit multiple paths to reduce cell or packet loss in the switching fabrics that support large ATM or IP switches. The study examined a low-cost class of switching fabrics in which only a single extra stage is added to a minimal banyan topology in order to provide alternate paths. Under a load that consists of light background traffic and a small number of heavy real-time flows, the alternate paths can be used to significantly reduce the peak load on switch resources, thus reducing worst case cell loss. Several different approaches to utilizing the multiple paths of the switch were considered. We considered alternative ways to interconnect the extra stage to the rest of the switching fabric to provide different sets of alternate paths. We evaluated the impact of these different interconnections on cell loss. In addition, we compared traffic management policies for using the alternate paths. In particular, we compared oblivious distribution of the real-time flows to optimal routing of these flows. For the traffic patterns that were evaluated, the results showed that routing of the real-time flows can significantly reduce worst-case cell loss compared to oblivious distribution.

There are several possible directions for future research that extends the results in this dissertation. An interesting and important question is how DVCs behave with shifting traffic patterns. In particular, there are many alternatives for choosing when and how to reroute existing circuits. In addition, there may be opportunities to develop policies for optimizing the performance of the DVC

mechanism. For example, when a switch becomes close to running out of free RVCs on an output link, it may be worthwhile to proactively tear down some existing circuits in anticipation of new circuit establishment requests. Depending on workload properties, such a policy might be able to reduce the latency for establishing new circuits by reducing the probability that a new circuit establishment will have to wait for a victim circuit to be torn down.

Other future investigations could focus on extending DVCs to provide useful advanced functionalities. For example, the DVC mechanism could be extended to provide improved support for fault tolerance. When a link or switch along a virtual circuit's path fails, the circuit can be rerouted to avoid the failure. Since packets can be lost when components fail, one challenge is to provide mechanisms in the switching fabric to restore lost packets efficiently, without involving source or destination nodes. Other techniques are needed in the switching fabric for reducing the overhead of rerouting several circuits when a failed component is shared by the circuits. For example, to avoid introducing large bursts of CEPs and CDPs in the network when a component fails, switches may regulate the rate at which they reroute circuits around failures. As another example of advanced functionality, DVCs could be extended to support multicast capabilities in which a circuit is established as a tree in the network that connects a source node to multiple destination nodes, minimizing the duplicated transmission of packets that must be sent from the source to all the destinations. As a final example, the DVC mechanism could be extended to provide support for quality of service guarantees for individual circuits. For example, mechanisms could be developed to provide some circuits with bandwidth guarantees. An important challenge is to provide

performance guarantees despite the possibility of operations that could reduce the performance for a circuit, such as circuit teardown by an intermediate switch or diversion of some packets of the circuit onto different paths to avoid deadlock. One possible approach could be to avoid selecting high-priority circuits for teardown and for packet diversions, thus increasing the ability of the network to satisfy the performance requirements for these circuits.

Bibliography

[Agar99] A. Agarwal, R. Bianchini, D. Chaiken, F. T. Chong, K. L. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung, ‘‘The MIT Alewife Machine,’’ *Proceedings of the IEEE* **87**(3)(March 1999).

[Alle94] J. D. Allen, P. T. Gaughan, D. E. Schimmel, and S. Yalamanchili, ‘‘Ariadne-an adaptive router for fault-tolerant multicomputers,’’ *21st Annual International Symposium on Computer Architecture*, pp. 278-88 (April 1994).

[Ande94] J. Anderson, B. T. Doshi, S. Dravida, and P. Harshavardhana, ‘‘Fast Restoration of ATM Networks,’’ *IEEE Journal on Selected Areas in Communications* **12**(1) pp. 128-138 (January 1994).

[Ande95] T. E. Anderson, D. E. Culler, and D. A. Patterson, ‘‘A case for NOW (Networks of Workstations),’’ *IEEE Micro* **15**(1) pp. 54-64 (February 1995).

[Anja95] K. V. Anjan and T. M. Pinkston, ‘‘An efficient, fully adaptive deadlock recovery scheme: DISHA,’’ *Proceedings 22nd Annual International Symposium on Computer Architecture*, pp. 201-10 (22-24 June 1995).

[Atha88] W. C. Athas and C. L. Seitz, ‘‘Multicomputers: Message-Passing Concurrent Computers,’’ *Computer* **21**(8) pp. 9-24 (August 1988).

[Awer89] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg, ‘‘Compact

distributed data structures for adaptive routing,'' RJ 6701, IBM, Almaden Research Center (February 1989).

[Bann91] T. R. Banniza, G. J. Eilenberger, B. Pauwels, and Y. Therasse, ''Design and Technology Aspects of VLSI's for ATM Switches,'' *IEEE Journal on Selected Areas in Communications* **9**(8) pp. 1255-1264 (October 1991).

[Batc68] K. E. Batcher, ''Sorting Networks and Their Applications,'' *Proceedings AFIPS Spring Joint Conference* **32** pp. 307-314 (1968).

[Bene64] V. Benes, ''Permutation Groups, Complexes, and Rearrangeable Multistage Connecting Networks,'' *Bell System Technical Journal* **43** pp. 1619-1640 (July 1964).

[Bert87] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall (1987).

[Bode95] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, ''Myrinet -- a Gigabit-per-Second Local Area Network,'' *IEEE Micro* **15**(1) pp. 29-36 (February 1995).

[Boet90] B. Boettle and M. A. Henrion, ''Alcatel ATM Switch Fabric and Its Properties,'' *Electrical Communication* **64**(2/3) pp. 156-165 (1990).

[Bold97] K. Bolding, M. Fulgham, and L. Snyder, ''The case for chaotic adaptive routing,'' *IEEE Transactions on Computers* **46**(12) pp. 1281-1292 (December 1997).

[Bork90] S. Borkar, R. Cohn, G. Cox, T. Gross, H. T. Kung, M. Lam, M.

Levine, B. Moore, W. Moore, C. Peterson, J. Susman, J. Sutton, J. Urbanski, and J. Webb, ‘‘Supporting Systolic and Memory Communication in iWarp,’’ *17th Annual International Symposium on Computer Architecture*, pp. 70-81 (May 28-31, 1990).

[Chen90] M.-S. Chen, K. G. Shin, and D. D. Kandlur, ‘‘Addressing, Routing, and Broadcasting in Hexagonal Mesh Multiprocessors,’’ *IEEE Transactions on Computers* **39**(1) pp. 10-18 (January 1990).

[Chie92] A. A. Chien and J. H. Kim, ‘‘Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors,’’ *19th Annual International Symposium on Computer Architecture*, pp. 268-277 (May 1992).

[Chie95] A. A. Chien and J. H. Kim, ‘‘Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors,’’ *Journal of the Association for Computing Machinery* **42**(1) pp. 91-123 (January 1995).

[Chow87] E. Chow, H. Madan, and J. Peterson, ‘‘A Real-Time Adaptive Message Routing Network for the Hypercube Computer,’’ *Proceedings of the Real-Time Systems Symposium*, pp. 88-96 (December 1987).

[Chow88] E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed, ‘‘Hyperswitch Network for the Hypercube Computer,’’ *15th Annual International Symposium on Computer Architecture*, pp. 90-99 (May 1988).

[Cido87] I. Cidon, J. M. Jaffe, and M. Sidi, ‘‘Local Distributed Deadlock

Detection by Cycle Detection and Clustering,'' *IEEE Transactions on Software Engineering* **SE-13**(1) pp. 3-14 (January 1987).

[Cohe94] R. Cohen, ''Smooth Intentional Rerouting and its Applications in ATM Networks,'' *IEEE INFOCOM'94*, pp. 1490-1497 (June 1994).

[Corb68] F. J. Corbato, ''A Paging Experiment with the MULTICS System,'' Project MAC Memo MAC-M-384, MIT, Cambridge, MA (July 1968).

[Dall86] W. J. Dally and C. L. Seitz, ''The Torus Routing Chip,'' *Distributed Computing* **1**(4) pp. 187-196 (October 1986).

[Dall87] W. J. Dally and C. L. Seitz, ''Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,'' *IEEE Transactions on Computers* **C-36**(5) pp. 547-553 (May 1987).

[Dall93] W. J. Dally and H. Aoki, ''Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels,'' *IEEE Transactions on Parallel and Distributed Systems* **4**(4) pp. 466-475 (April 1993).

[Dao97] B. V. Dao, S. Yalamanchili, and J. Duato, ''Architectural support for reducing communication overhead in multiprocessor interconnection networks,'' *Third International Symposium on High-Performance Computer Architecture*, pp. 343-52 (1-5 Feb. 1997).

[De 96] M. De Prycker, *Asynchronous Transfer Mode: Solution for*

Broadband ISDN (3rd edition), Prentice Hall, New York (1996).

[Denz92] W. E. Denzel, A. P. J. Engbersen, I. A. Iliadis, and G. A. Karlsson, ‘‘A Highly Modular Packet Switch for Gb/s Rates,’’ *International Switching Symposium 1992* **2** pp. 236-240 (October 1992).

[Denz95] W. E. Denzel, A. P. J. Engbersen, and I. A. Iliadis, ‘‘A Flexible Shared-Buffer Switch for ATM at Gb/s Rates,’’ *Computer Networks and ISDN Systems* **27**(4) pp. 611-624 (January 1995).

[Desm91] E. Desmet, B. Steyaert, H. Bruneel, and G. H. Petit, ‘‘Tail Distributions of Queue Length and Delay in Discrete-Time Multiserver Queueing Models, Applicable in ATM Networks,’’ *Queueing, Performance and Control in ATM (Proceedings of the Thirteenth International Teletraffic Congress),* pp. 1-6 (June 1991).

[Duat95] J. Duato, ‘‘A Necessary And Sufficient Condition For Deadlock-Free Adaptive Routing In Wormhole Networks.,’’ *IEEE Transactions on Parallel and Distributed Systems* **6**(10) pp. 1055-1067 (October 1995).

[Duat96] J. Duato, ‘‘A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks,’’ *IEEE Transactions on Parallel and Distributed Systems* **7**(8) pp. 841-54. (August 1996).

[Duat98] J. Duato, ‘‘Deadlock avoidance and adaptive routing in interconnection networks,’’ *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing*, pp. 359-364 (21-23 Jan. 1998).

[Duni05] T. H. Dunigan, J. S. Vetter, J. B. White, and P. H. Worley, ‘‘Performance Evaluation of the Cray X1 Distributed Shared-Memory Architecture,’’ *IEEE Micro* **25**(1) pp. 30-40 (Jan-Feb 2005).

[Flin00] J. Flinch, M. P. Malumbres, P. Lopez, and J. Duato, ‘‘Performance Evaluation of a New Routing Strategy for Irregular Networks with Source Routing,’’ *14th Int’l Conf on Supercomputing*, pp. 34-43 (2000).

[Floy93] S. Floyd and V. Jacobson, ‘‘Random Early Detection Gateways for Congestion Avoidance,’’ *IEEE/ACM Transactions on Networking* **1**(4) pp. 397-413 (August 1993).

[Four97] J.-M. Fourneau, L. Mokdad, and N. Pekergin, ‘‘Bounding the Loss Rates in a Multistage ATM Switch,’’ *Computer Performance Evaluation, Modelling Techniques and Tools: 9th International Conference Proceedings*, pp. 193-205 (June 1997).

[Fuji94] Fujitsu, ‘‘MB86680B ATM Switch Element (Self Routing Element),’’ *Product Literature*, (May 1994).

[Gall97] M. Galles, ‘‘Spider: A High-Speed Network Interconnect,’’ *IEEE Micro* **17**(1) pp. 34-39 (January/February 1997).

[Gers99] A. Gersht and A. Shulman, ‘‘Architecture for Restorable Call Allocation and Fast VP Restoration in Mesh ATM Networks,’’ *IEEE Transactions on Communications* **47**(3) pp. 397-403 (March 1999).

[Giac91] J. N. Giacopelli, J. J. Hickey, W. S. Marcus, W. D. Sincoskie, and M. S. Littlewood, ‘‘Sunshine: A High Performance Self-Routing Broadband Packet Switch Architecture,’’ *IEEE Journal on Selected Areas in Communications* **9**(8) pp. 1289-1298 (October 1991).

[Glas92] C. J. Glass and L. M. Ni, ‘‘The Turn Model for Adaptive Routing,’’ *19th Annual International Symposium on Computer Architecture*, pp. 278-287 (May 1992).

[Glas94] C. J. Glass and L. M. Ni, ‘‘The Turn Model for Adaptive Routing,’’ *Journal of the Association for Computing Machinery* **41**(5) pp. 874-902 (September 1994).

[Henr93] M. A. Henrion, G. L. Eilenberger, G. H. Petit, and P. H. Parmentier, ‘‘A Multipath Self-Routing Switch,’’ *IEEE Communications Magazine* **31**(4) pp. 46-52 (April 1993).

[Hilb89] P. A. J. Hilbers and J. J. Lukkien, ‘‘Deadlock-free message routing in multicomputer networks,’’ *Distributed Computing*, (3) pp. 178-186 (1989).

[Hsu90] J.-M. Hsu and P. Banerjee, ‘‘Hardware Support for Message Routing in a Distributed Memory Multicomputer,’’ *1990 International Conference on Parallel Processing*, (August 1990).

[Hsu92] J.-M. Hsu and P. Banerjee, ‘‘Performance measurement and trace driven simulation of parallel CAD and numeric applications on a hypercube multicomputer,’’ *IEEE Transactions on Parallel and Distributed Systems* **3**(4) pp. 451-464 (July 1992).

[Huan84] A. Huang and S. Knauer, ‘‘STARLITE: A Wideband Digital Switch,’’ *GLOBECOM ’94*, pp. 121-125 (November 1984).

[Jaco88] V. Jacobson, ‘‘Congestion Avoidance and Control,’’ *ACM SIGCOMM*, pp. 314-329 (Aug 1988).

[Jaff89] J. M. Jaffe and M. Sidi, ‘‘Distributed Deadlock Resolution in Store-and-Forward Networks,’’ *Algorithmica* **4**(3) pp. 417-436 (1989).

[Jami97] S. Jamin, P. B. Danzig, S. J. Shenker, and L. Zhang, ‘‘A Measurement-Based Admission Control Algorithm for Integrated Services Packet Networks (extended version),’’ *IEEE/ACM Transactions on Networking* **5**(1) pp. 56-70 (February 1997).

[Kawa99] R. Kawamura and H. Ohta, ‘‘Architectures for ATM Network Survivability and Their Field Deployment,’’ *IEEE Communications Magazine* **37**(8) pp. 88-94 (August 1999).

[Kerm79] P. Kermani and L. Kleinrock, ‘‘Virtual Cut Through: A New Computer Communication Switching Technique,’’ *Computer Networks* **3**(4) pp. 267-286 (September 1979).

[Kim97] J. H. Kim, Z. Liu, and A. A. Chien, ‘‘Compressionless Routing: A Framework For Adaptive and Fault-Tolerant Routing,’’ *IEEE Transactions on Parallel and Distributed Systems* **8**(3) pp. 229-244 (March 1997).

[Kons90a] S. Konstantinidou, ‘‘Adaptive, Minimal Routing in Hypercubes,’’ *6th MIT Conference on Advanced Research in VLSI*, pp. 139-153

(1990).

[Kons90b] S. Konstantinidou and L. Snyder, ‘‘The Chaos Router: A Practical Application of Randomization in Network Routing,’’ *2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 21-30 (July 1990).

[Kons91] S. Konstantinidou and L. Snyder, ‘‘Chaos router: architecture and performance,’’ *18th Annual International Symposium on Computer Architecture*, pp. 212-221 (May 1991).

[Kung94] H. T. Kung, T. Blackwell, and A. Chapman, ‘‘Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation and Statistical Multiplexing,’’ *SIGCOMM ’94* **24**(4) pp. 101-114 (August 1994).

[Lang82] C. R. Lang, ‘‘The extension of object-oriented languages to a homogeneous concurrent architecture,’’ 5014:TR:82, California Institute of Technology, Pasadena, CA (1982).

[Laud97] J. Laudon and D. Lenoski, ‘‘The SGI Origin: a ccNUMA Highly Scalable Server.,’’ *24th Annual International Symposium on Computer Architecture*, pp. 241-251 (May 1997).

[Leig92] F. T. Leighton and B. M. Maggs, ‘‘Fast Algorithms for Routing Around Faults in Multibutterflies and Randomly-Wired Splitter Networks,’’ *IEEE Transactions on Computers* **41**(5) pp. 578-587 (May 1992).

[Lind91] D. H. Linder and J. C. Harden, ‘‘An Adaptive and Fault Tolerant

Wormhole Routing Strategy for *k*-ary *n*-cubes,'' *IEEE Transactions on Computers* **40**(1) pp. 2-12 (January 1991).

[Low97] Y. L. Low and R. C. Frye, ''Signal Integrity and Power Distribution System Analyses for a 4x4 ATM Switch MCM,'' *IEEE International Conference on Multichip Modules*, pp. 278-283 (April 1997).

[McQu80] J. M. McQuillan, I. Richer, and E. C. Rosen, ''The New Routing Algorithm for the ARPANET,'' *IEEE Transactions on Communications* **COM-28**(5) pp. 711-719 (May 1980).

[Metc76] R. M. Metcalfe and D. R. Boggs, ''Ethernet: Distributed Packet Switching for Local Computer Networks,'' *Communications of the ACM* **19**(7) pp. 395-404 (July 1976).

[Mura97] K. Murakami and H. S. Kim, ''Comparative Study on Restoration Schemes of Survivable ATM Networks,'' *IEEE INFOCOM'97*, pp. 345-352 (April 1997).

[Ngai87] J. Y. Ngai and C. L. Seitz, ''A Framework for Adaptive Routing,'' 5246:TR:87, California Institute of Technology, Computer Science Department (July 16, 1987).

[Ngai89a] J. Y. Ngai and C. L. Seitz, ''A FrWork for Adaptive Routing in Multicomputer Networks,'' *Symposium on Parallel Algorithms and Architectures*, pp. 2-9 (June 1989).

[Ngai89b] J. Y. Ngai, ''A Framework for Adaptive Routing in Multicomputer Networks,'' Computer Science Technical Report 89-09, California

Institute of Technology, Pasadena, CA (May 1989).

[Noak93] M. D. Noakes, D. A. Wallach, and W. J. Dally, ‘‘The J-Machine Multicomputer: An Architectural Evaluation,’’ *20th Annual International Symposium on Computer Architecture*, pp. 224-235 (May 16-19, 1993).

[Part94] C. Partridge, *Gigabit networking*, Addison-Wesley, Reading, MA (1994).

[Pert92] M. J. Pertel, ‘‘A Critique of Adaptive Routing,’’ Computer Science Technical Report 92-06, California Institute of Technology, Pasadena, CA (June 1992).

[Pete88] J. Peterson, E. Chow, and H. Madan, ‘‘A High-Speed Message-Driven Communication Architecture,’’ *International Conference on Supercomputing*, pp. 355-366 (July 1988).

[Petr02] F. Petrini, W.-C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, ‘‘The Quadrics Network: High-Performance Clustering Technology,’’ *IEEE Micro* **22**(1) pp. 46-57 (February 2002).

[Pifa91] G. D. Pifarré, L. Gravano, S. A. Felperin, and J. L. C. Sanz, ‘‘Fully-Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes, and Other Networks,’’ *3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, (June 1991).

[Roma95] A. Romanow and S. Floyd, ‘‘Dynamics of TCP Traffic Over ATM Networks,’’ *IEEE Journal on Selected Areas in Communications* **13**(4) pp. 633-641 (May 1995).

[Scot96a] S. L. Scott, ‘‘Synchronization and Communication in the T3E Multiprocessor,’’ *7th International Conference on Architectural Support for Programming Languages and Operating Systems,* pp. 26-36 (October 1996).

[Scot96b] S. L. Scott and G. M. Thorson, ‘‘The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus,’’ *HOT Interconnects IV,* (August 1996).

[Shin87] K. G. Shin and M.-S. Chen, ‘‘Performance Analysis of Distributed Routing Strategies Free of Ping-Pong-Type Looping,’’ *IEEE Transactions on Computers* **C-36**(2) pp. 129-137 (February 1987).

[Stun94a] C. B. Stunkel, M. M. Denneau, B. J. Nathanson, D. G. Shea, P. H. Hochschild, M. Tsao, B. Abali, D. J. Joseph, and P. R. Varker, ‘‘Architecture and implementation of Vulcan,’’ *Proceedings Eighth International Parallel Processing Symposium*, pp. 268-274 (26-29 April 1994).

[Stun94b] C. B. Stunkel, D. G. Shea, D. G. Grice, P. H. Hochschild, and M. Tsao, ‘‘The SP1 High-Performance Switch,’’ *Proceedings of the Scalable High-Performance Computing Conference*, pp. 150-157 (May 1994).

[Stun95] C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. A. Bender, D. G. Grice, P. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, F. Stucke, M. Tsao, and P. R. Varker, ‘‘The SP2 High-Performance Switch,’’ *IBM Systems Journal* **34**(2) pp. 185-204 (1995).

[Sull77] H. Sullivan and T. R. Brashkow, ‘‘A large scale homogeneous

machine,'' *Proceedings of the 4th Annual Symposium on Computer Architecture*, pp. 105-124 (1977).

[Suzu92] H. Suzuki and F. A. Tobagi, ''Fast Bandwidth Reservation Scheme with Multi-Link and Multi-Path Routing in ATM Networks,'' *IEEE INFOCOM '92* **Vol. 3** pp. 2233-2240 (May 1992).

[Taji77] W. D. Tajibnapis, ''A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network,'' *Communications of the ACM* **20**(7) pp. 477-485 (July 1977).

[Tami88a] Y. Tamir and G. L. Frazier, ''High-Performance Multi-Queue Buffers for VLSI Communication Switches,'' *15th Annual International Symposium on Computer Architecture*, pp. 343-354 (May 1988).

[Tami88b] Y. Tamir and J. C. Cho, ''Design and Implementation of High-Speed Asynchronous Communication Ports for VLSI Multicomputer Nodes,'' *International Symposium on Circuits and Systems*, pp. 805-809 (June 1988).

[Tami91] Y. Tamir and Y. F. Turner, ''High-Performance Adaptive Routing in Multicomputers Using Dynamic Virtual Circuits,'' *6th Distributed Memory Computing Conference*, pp. 404-411 (April 1991).

[Tami92] Y. Tamir and G. L. Frazier, ''Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches,'' *IEEE Transactions on Computers* **41**(6) pp. 725-737 (June 1992).

[Turn86] J. S. Turner, ‘‘Design of an Integrated Services Packet Network,’’ *IEEE Journal on Selected Areas in Communications* **SAC-4**(8) pp. 1373-1380 (November 1986).

[Turn98] Y. F. Turner and Y. Tamir, ‘‘Connection-Based Adaptive Routing Using Dynamic Virtual Circuits,’’ *International Conference on Parallel and Distributed Computing and Systems*, pp. 379-384 (October 1998).

[Turn05] Y. F. Turner and Y. Tamir, *Deadlock-Free Connection-Based Adaptive Routing with Dynamic Virtual Circuits*. submitted for publication May 2005.

[Tyme81] L. Tymes, ‘‘Routing and flow control in TYMNET,’’ *IEEE Transactions on Communication* **COM-29** pp. 392-398 (1981).

[Venk96] A. K. Venkatramani, T. M. Pinkston, and J. Duato, ‘‘Generalized theory for deadlock-free adaptive wormhole routing and its application to Disha Concurrent,’’ *The 10th International Parallel Processing Symposium*, pp. 815-21 (15-19 April 1996).