ABSTRACT

Tailoring Router Architectures to Performance Requirements in Cut-Through Networks

by
Jennifer Rexford

Chair: Kang G. Shin

Message-passing parallel machines have emerged as a cost-effective platform for exploiting concurrency in a variety of applications. These multicomputer systems employ a wide range of policies for routing, switching, arbitration, queueing, and flow control, implemented in the router hardware that connects an individual processing node to the interconnection fabric and manages traffic flowing through the node en route to other destinations. To address the requirements of emerging applications, we develop new techniques for designing and evaluating new router architectures that tailor network policies to application characteristics. These results facilitate the development of effective support for communication in real-time systems and local area networks, as well as more traditional multicomputer domains like high-speed scientific computing.

Most modern routers employ *cut-through* switching schemes, such as virtual cut-through and wormhole switching, that permit an arriving packet to proceed directly to an idle outgoing link. We develop analytical models for evaluating cut-through routing algorithms with different degrees of adaptivity. The analytical results permit an efficient evaluation of large networks, while detailed comparisons with simulation results characterize the subtle effects of the simplifying assumptions in the analysis; in particular, cut-through networks introduce unique dependencies between adjacent nodes. Additional simulation experiments show that the network topologies, routing algorithms, and traffic patterns in modern multicomputers exacerbate these effects. Based on these results, we present a routing algorithm that capitalizes on inter-node dependencies to improve network performance.

The limitations of analytical models motivate the development of flexible simulation tools that can model a wide range of network policies. We present a configurable router model and simulation environment that enables experimentation with a variety of network policies under diverse traffic patterns and performance metrics. The object-oriented simulation environment supports the high-level specification of routing-switching algorithms, separate from the low-level timing details in the router model; by associating these algorithms with the communication workload, instead of the underlying router, the simulator can evaluate emerging networks that support *multiple* coexisting routing-switching schemes, with different communication characteristics and performance requirements.

Drawing on these novel features, we investigate router architectures that tailor their switching and arbitration policies to the conflicting performance requirements of two traffic classes, *time-constrained* and *best-effort* packets, common in real-time applications. Detailed simulation experiments motivate the choice of packet switching for time-constrained

traffic and wormhole switching for best-effort traffic. Separate logical resources, coupled with fine-grain link arbitration, enables time-constrained and best-effort communication to share network resources without sacrificing the performance requirements of either traffic class.

These results motivate a new router architecture that bounds end-to-end delay and throughput for time-constrained traffic, while ensuring low average latency for best-effort packets. The router implements deadline-based scheduling and rate-based flow control, under packet switching and table-driven multicast routing, to guarantee performance for time-constrained traffic. For an efficient hardware realization, the architecture integrates the traffic-shaping and link-scheduling functionality into a single unit, which is shared amongst the multiple outgoing links. The router explicitly handles clock rollover and provides early service to best-effort wormhole traffic when no time-constrained packets are in danger of missing their deadlines. A careful combination of routing, switching, queueing, arbitration, and flow-control policies permits a single-chip implementation that can support the performance requirements of parallel real-time applications.

Tailoring Router Architectures to Performance Requirements in Cut-Through Networks

by

Jennifer Rexford

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1999

Doctoral Committee:
Professor Kang G. Shin, Chair
Associate Professor Richard B. Brown
Associate Professor Farnam Jahanian
Professor Toby Teorey

To my parents.

ACKNOWLEDGEMENTS

> There is really no such creature as a single individual; he has no more a life of his own than a cast-off cell marooned from the surface of your skin.
> – *Lewis Thomas*

The support of enthusiastic and dedicated mentors, colleagues, and friends has immeasurably enriched my graduate school experience. In particular, I would like to thank

- Professor Kang Shin, my advisor, for exposing me to a broad range of exciting issues in communication networks, parallel computing, and real-time systems. His emphasis on combining "theory and practice" has shaped the direction of my thesis, as well as my future research endeavors.

- Professors Toby Teorey, Farnam Jahanian, and Richard Brown for serving on my thesis committee. Thanks especially to Farnam Jahanian for his many valuable insights into research, teaching, and advising, and to John Meyer for serving on the committee for my thesis proposal and for sharing his views on performance evaluation and communication networks.

- Albert Greenberg for his mentorship throughout my research projects at AT&T Bell Laboratories and his guidance in improving my technical writing.

- Ashish, Jim, Anees, Wu-chang, Scott, Stuart, and Dave for creating an office environment full of comradeship, humor, and caffeine. Thanks especially to Jim Dolter for his philosophical musings and invaluable guidance and to Ashish Mehra for his many years of friendship and candor, as well as numerous enjoyable technical discussions.

- My other close friends in Ann Arbor, including my roommates who endured many bad puns during my first three years in Ann Arbor. Thanks to Sujatha for her friendship and kooky sense of humor and to Carlos for the many evenings of delicious food and good conversation. Special thanks also to Tris and Farnam for welcoming me into their home, to B.J. Monaghan for being both a friend and a mom-away-from-home, and to Zakia for her engaging phone calls to the office.

- Casper for his patience, support, and humor over many enjoyable visits and countless late-night phone conversations about topics ranging from research and education to literature and life. Thanks also to Yvonne for her uncanny understanding and infectious laughter.

- John and Susan Rexford, my parents, for their love and support, especially during the final stretch.

- The Office of Naval Research, AT&T Bell Laboratories, the Rackham School of Graduate Studies, and Intel Corporation for the fellowship and grant funds that enabled me to pursue my thesis work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Message-passing parallel machines have emerged as a cost-effective platform for exploiting concurrency in a variety of applications. In these *multicomputer* systems, fast message exchange enables efficient, fine-grained cooperation between processing elements. No longer the purview of tightly-coupled parallel machines, multicomputer components are also finding new uses in local area networks and real-time systems. While multicomputer systems have traditionally targetted a relatively narrow range of applications in scientific computing, emerging networks must support diverse applications such as file and video servers, databases, scientific visualization, and process control. These application domains impose a broader range of communication characteristics and performance requirements on the underlying interconnection network. To address these performance challenges, this thesis presents techniques for designing and evaluating new multicomputer router architectures that tailor network policies to application characteristics.

1.1 Cut-Through Routers

Maximizing system performance requires matching application communication patterns with a suitable network design. Multicomputer networks employ a wide range of topologies, routing algorithms, switching schemes, and flow-control policies. Application characteristics directly impact these design decisions by determining the distribution of traffic in the network. End-to-end performance metrics, such as throughput and latency, are extremely sensitive to the network policies implemented in the router hardware that connects an individual processing node to the interconnection fabric and manages traffic flowing through the node en route to other destinations; for example, Figure 1.1 shows a router in a 4×4 square mesh. To communicate with another node, a processor injects a packet into its router; then, the packet travels through one or more intermediate routers before reaching the reception port of the router at the destination node.

To address the requirements of emerging applications, router designs should tailor their policies to application traffic patterns and performance metrics. In particular, the network's switching scheme can greatly influence communication performance by determining what link and buffer resources a packet consumes at the nodes in its route. Traditional *packet switching* requires an arriving packet to buffer completely before transmission to a subsequent node can begin. In contrast, *cut-through* switching schemes, such as virtual cut-through [65] and wormhole [28] switching, attempt to directly forward the incoming packet to an idle output link. If the packet encounters a busy outgoing link, virtual cut-through

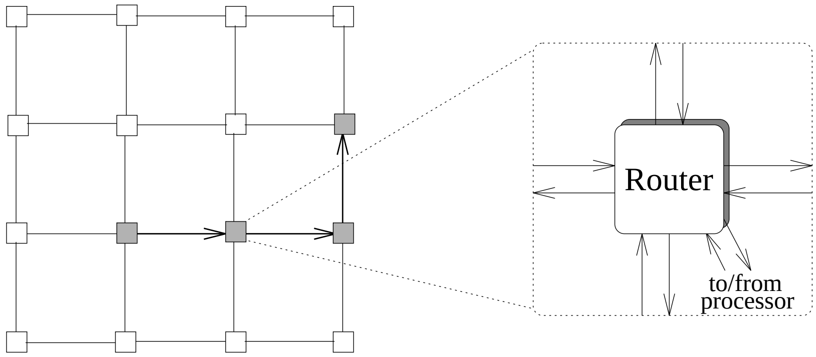


Figure 1.1: Router in a mesh network with a 4×4 grid of processing nodes

switching buffers the packet. On the other hand, a blocked wormhole packet stalls in the network, holding the earlier links in its route until the outgoing link becomes free.

Although first-generation multicomputers employed packet switching, most contemporary research and commercial routers utilize cut-through switching for lower latency and reduced buffer space requirements [7, 99, 127]. For example, the Intel iPSC/1, nCube/3200, and Ametek/14 implemented store-and-forward packet switching, with each processing node interrupting application tasks to buffer and forward incoming packets [127]. Due to advances in VLSI technology, these commercial vendors were able to incorporate dedicated router hardware for cut-through switching in subsequent multicomputer systems, such as the iPSC/2 [89], nCube/6400 [45], and Ametek/2010 [109], during the late 1980s. Keeping with this trend, companies like Intel, IBM, Thinking Machines, and Cray Research developed new multicomputer systems based on wormhole-switched networks [25, 59, 76, 85, 95, 114, 115] during the first half of the 1990s. Similarly, most research multicomputer networks employ virtual cut-through or wormhole switching, or hybrid schemes, as discussed in Chapter 2.

Although cut-through switching schemes have the potential to improve communication latency and throughput, most existing local and wide area networks employ packet switching to reduce flow-control requirements and simplify operation over a range of different link speeds. Still, wormhole and virtual cut-through switching are viable options in more homogeneous, tightly-coupled domains, such as workstation clusters and high-speed switch designs. As a result, several new systems incorporate multicomputer routers to achieve low-latency, high-bandwidth communication in local area networks. The Atomic [24] and Shrimp [13] research projects demonstrate the power of this paradigm, which has been incorporated in commercial products such as Sun Microsystem's S-Connect [88] and Myricom's Myrinet [14]. In addition, researchers are investigating the use of cut-through routers as building blocks for constructing larger ATM (asynchronous transfer mode) switches for use in large-scale networks [82, 123, 124].

Compared to traditional parallel machines, high-speed switches and local area networks impose a wide range of communication characteristics and performance requirements on multicomputer routers. In addition, multicomputer components are increasingly used for multimedia and real-time applications, such as scientific visualization, process control, and video-on-demand servers. These applications require *predictable* communication performance, in addition to low latency and high throughput. Initially, real-time multicomputer applications have employed existing interconnection networks, such as the Intel Paragon [12, 52] and the nCube/3 [20], with no explicit support for time-constrained communication. However, on many existing multicomputer systems, worst-case packet latency can theoretically be as large as several *days* [69]. As a result, real-time and multimedia applications must under-utilize the network to achieve predictable performance. To address this problem, several recent projects consider new router architectures that can *guarantee* end-to-end communication performance in multicomputer networks [10, 69, 77, 83, 91, 108, 119, 122].

1.2 Structure of the Thesis

This thesis presents effective techniques for the design and evaluation of router architectures that accommodate the performance requirements of emerging multicomputer applications. Chapter 2 surveys related work on multicomputer networks and application workloads, with a classification of the routing, switching, queueing, and arbitration policies in existing router designs. Focusing specifically on the impact of *routing* on the performance of cut-through networks, Chapter 3 compares several routing algorithms based on both analytical models and simulation experiments. The analytical models permit an efficient evaluation of large networks, while comparisons with simulation results reveal the effects of the simplifying assumptions in the analysis. Based on this comparison, we provide a precise characterization of the unique dependencies between adjacent nodes in realistic cut-through networks.

These results facilitate more accurate performance models, as well as novel routing algorithms that capitalize on inter-node dependencies to improve communication performance. To evaluate a wider range of network policies, Chapter 4 presents a configurable router model and simulation environment. This flexible and extensible simulation framework enables experimentation with a variety of routing, switching, queueing, and arbitration schemes under diverse traffic patterns and performance metrics. Unlike existing tools for evaluating multicomputer networks, this simulator can model emerging router architectures that support *multiple* coexisting routing-switching schemes, tailored to different communication characteristics and performance requirements. Chapter 5 capitalizes on these novel features to investigate router architectures that tailor their switching schemes and arbitration policies to the performance requirements of parallel real-time systems.

Real-time applications typically handle a mixture of time-constrained and best-effort communication, where time-constrained packets require bounds on latency and throughput, while best-effort traffic settles for good average performance [6, 54, 112, 113]. Based on simulation experiments, Chapter 5 proposes effective techniques for mixing the time-constrained and best-effort packets without sacrificing the performance goals of either traffic class. Chapter 6 employs these policies to develop a new router architecture that bounds end-to-end delay and throughput for time-constrained traffic, while ensuring low average latency for best-effort packets. The chapter presents a single-chip implementation of the router that minimizes complexity by sharing buffer space and arbitration logic amongst the multiple outgoing links. Chapter 7 concludes the dissertation with a recapitulation of the research contributions and possible avenues for future work.

CHAPTER 2

MULTICOMPUTER ROUTER ARCHITECTURE

Multicomputer performance hinges on the subtle interaction between application traffic patterns and the router policies in the underlying network. This chapter reviews the communication workloads and architectural parameters in modern multicomputer networks. Section 2.1 describes the diverse range of packet sizes, interarrival times, and destination distributions in parallel applications, while Section 2.2 provides an overview of architectural issues. Based on this discussion, Section 2.3 classifies and compares existing router architectures to highlight current trends in multicomputer network design.

2.1 Multicomputer Communication Workloads

Multicomputer applications consist of a collection of tasks that communicate by exchanging messages. At the source node, the sending task submits the message to the communication subsystem for transmission to receiving tasks on one or more destination nodes. For efficiency reasons, the source node may decompose a message into a collection of smaller packets that proceed through the interconnection network and are reassembled into a message at the destination node. As a result, multicomputer routers typically coordinate communication at the packet level, where each packet consists of a header field followed by one or more bytes of data. The packet header often includes information such as a task identifier, message number, and packet sequence number to aid the destination node in delivering the reconstructed message to the receiving task. In addition, the header typically includes additional fields to assist the routers in shepherding the packet through the interconnection network.

Parallel applications generate a wide range of communication workloads depending on the application's granularity and mapping across multiple nodes, as well as the policies employed by the interconnection network. Scientific computations, parallel databases, and real-time applications generate distinct distributions for message/packet lengths, interarrival times, and target destination nodes [27, 57, 77, 107]. Multi-user systems exacerbate these effects since different applications may run *simultaneously*; these applications may execute on different parts of the network or even time-share the same processing elements. These diverse traffic patterns significantly impact the suitability of certain router architectural features, such as switching and routing schemes, as well as the accuracy of network performance models.

Analytical studies of multicomputer networks have typically modeled packet arrivals as a *Poisson process*, with exponentially-distributed interarrival times. This assumption was made, in part, due to the analytical tractability of such models and the lack of more realistic data. However, detailed measurements of multicomputer applications have led to more sophisticated message generation models. In particular, recent studies show that applications typically generate *bursty* network traffic [27, 57], due to multi-packet messages and fine-grain interaction between cooperating nodes. Similarly, many parallel applications invoke *multicast* operations that send a message to a collection of destination nodes, injecting multiple copies of each packet into the interconnection network. These traffic models have significant impact on network design and evaluation.

Like interarrival distributions, message and packet sizes depend on several factors. Some router architectures or communication protocols impose strict upper and lower bounds on packet length, or permit just a few different sizes or formats. Although fixed-length packets or exponentially-distributed lengths simplify analytic performance models, recent studies show that real multicomputer applications often generate a mixture of large data transfers and small request/acknowledgement packets [27, 57]. Without careful support in the router architecture, these *bimodal* length distributions can seriously degrade average latency by forcing short packets to queue behind long packets for access to link and buffer resources [67]. In addition, the presence of large packets can introduce greater *variability* in end-to-end communication delay, particularly for short packets [68].

Message/packet destination distributions vary a great deal depending on the network topology and the application's mapping onto different processing elements. Although many analytical and simulation studies evaluate a *uniform random* distribution of destination nodes, this model does not accurately represent the traffic patterns that arise in many multicomputer applications. To reduce link load and communication latency, many parallel applications place cooperating tasks near each other in the underlying network, introducing *spatial* locality. In addition, many applications exhibit *temporal* locality, where a node sends several messages to the same destination over a small time interval. Also, parallel algorithms introduce specific *non-uniform* traffic patterns. For example, many scientific applications generate communication workloads that correspond to the matrix-transpose (dimension-reversal), bit-complement, and bit-reversal permutations [23, 32, 66, 98]. Other application communication constructs, such as synchronization or multicast operations, may induce *hot-spots* of heavily-utilized nodes and links [17, 36, 98].

2.2 Multicomputer Interconnection Networks

2.2.1 Network Topology

By defining the connections between processing nodes, the network topology determines the number of communication links at each node and how far a packet must travel to reach its destination [47, 100]. As a result, the choice of a topology impacts both the complexity of network wiring and the achievable communication bandwidth in the system [2, 33]. Multicomputer networks vary from bus or ring topologies to fully-connected configurations, as shown in Figure 2.1. Connecting processors on a single bus provides an inexpensive solution that provides sufficient link bandwidth to support a small number of nodes; in contrast, a fully-connected network has a dedicated link between each pair of nodes, at the expense of increased implementation complexity. As a result, most systems attempt to strike a careful balance between implementation cost and achievable performance.

Most existing multicomputers have *direct* networks, which consist of point-to-point links, while other *indirect* networks connect processors through multiple stages of switches. Many

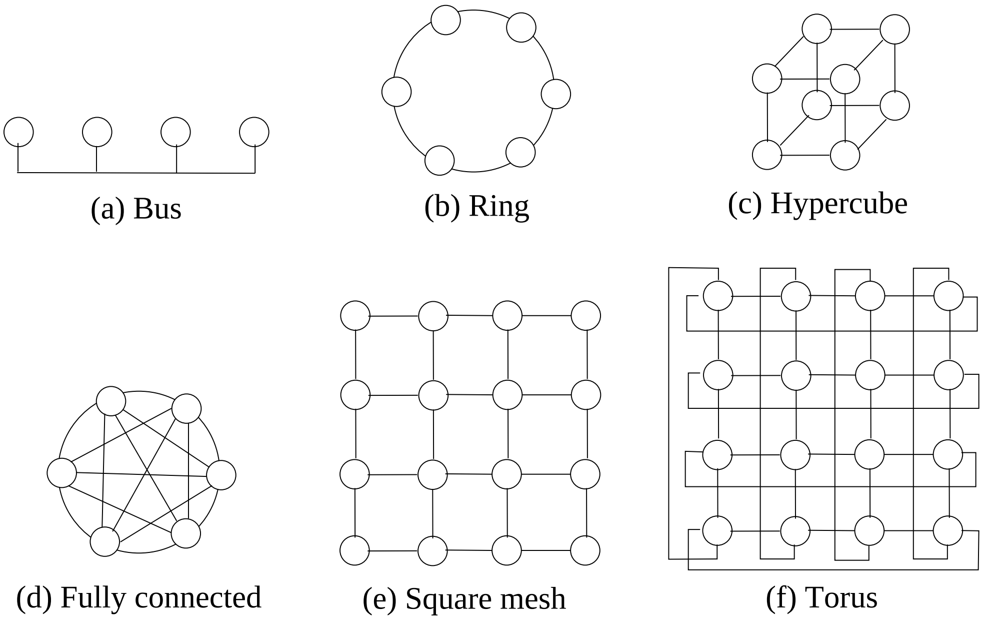


Figure 2.1: Multicomputer topologies

systems employ the k-ary n-cube family of topologies, with k nodes along each of n dimensions [33]; Figures 2.1(b) and 2.1(f) show a 6-ary 1-cube and a 4-ary 2-cube, respectively. To restrict wiring complexity, some systems omit the wrap-around links in each dimension, turning a homogeneous toroidal topology into a mesh network, such as the 4×4 square mesh in Figure 2.1(e). Unlike earlier hypercube (2-ary n-cube) multicomputers, most contemporary parallel machines have low-dimensional topologies, such as two and three dimensional meshes and tori, to allow wider inter-node links and reduced wiring costs.

2.2.2 Routing Algorithm

These topologies simplify the selection of a packet's path through the network by allowing the routing header to include the *relative address* of the destination node(s). For example, in a two-dimensional mesh or torus, the header can consist of x and y offsets that represent the distance the packet must travel in each direction; the offsets reach zero when the packet arrives at its destination node. When a new header arrives on an incoming link, the router inspects these fields to determine which outgoing link(s) should handle the incoming packet. *Oblivious* routing generates a single, deterministic outgoing link for an incoming packet. Many multicomputer networks employ *dimension-ordered* (*e-cube*) routing, an oblivious scheme which routes a packet completely in one dimension before proceeding to the next dimension, as shown by the shaded nodes in Figure 1.1.

In contrast, *adaptive* schemes can incorporate prevailing network conditions into the routing decision. By considering multiple outgoing links, adaptive algorithms can balance network load and decrease packet latency by avoiding busy nodes and links. Under adaptive routing, packets from a single message may follow different paths through the network; this can cause out-of-order packet arrival, which complicates message processing at the destination node [64]. Opportunities for adaptive routing vary depending on the underlying topology and communication pattern. Although most oblivious routing algorithms generate only minimum-hop routes between the source and destination nodes, some adaptive schemes consider *non-minimal* routes in the hope of circumventing congested or faulty links. In the last few years, researchers have proposed a wide variety of minimal and non-minimal adaptive routing algorithms, with different degrees of adaptivity and implementation complexity [4, 17, 46, 53, 87, 98].

When a routing algorithm must select from multiple output links at a node, the actual route chosen depends on a *selection function* that determines the *order* in which the algorithm considers these candidate links. For example, if a packet is traveling from node $(4,3)$ to node $(6,10)$ in a square mesh, a dimension-ordered selection function would favor the positive x-direction over the positive y-direction, if both links are idle. Alternatively, the router could rank outgoing links according to how much further the packet must travel in each direction; in the example, this selection function would favor the y-direction, since the packet still has 7 hops to travel. This link ordering improves the packet's chance of considering multiple outgoing links at future nodes in its route [8, 39]. Communication performance can be extremely sensitive to how the selection function interacts with the application workload to create traffic patterns in the interconnection network [50, 104].

2.2.3 Switching Scheme

The switching scheme impacts performance by determining the link and buffer resources a packet consumes at a given node in its route. Traditionally, multi-hop networks employ either a *circuit-switched* or *store-and-forward* model of communication. The classic example of a circuit-switched network in the Plain Old Telephone Service, which creates a dedicated connection between the source of a call and the desired destination. Once the network establishes a circuit, the call has guaranteed access to the link bandwidth, which limits delay and obviates the need for buffering; however, circuit switching *statically* allocates link bandwidth, which can be wasteful for transporting bursty traffic. To avoid this problem, most multicomputer networks coordinate link access at the message or packet level by using store-and-forward switching schemes.

First-generation multicomputer networks employed *packet switching*, which requires each incoming packet to buffer completely before transmission to a subsequent node can begin, as discussed in Section 1.1. Even in the absence of network congestion, packet switching incurs delay proportional to the *product* of packet size and the length of the route. In contrast, cut-through switching schemes, such as *virtual cut-through* [65] and *wormhole* [28] switching, try to forward an in-transit packet directly to an idle output link, based on the routing header and prevailing network conditions. In a lightly-loaded network, communication latency under cut-through switching is proportional to the *sum* of packet size and the length of the route, since the packet only incurs a small delay for processing the routing header at each node. As a result, most contemporary routers utilize cut-through switching for lower latency and reduced buffer space requirements [127].

If the outgoing link is busy, virtual cut-through switching *buffers* the packet, effectively degrading to packet switching. In contrast, a blocked wormhole packet *stalls* pending access to the outgoing link. Instead of storing entire packets, a wormhole router can simply include small *flit* (flow control unit) buffers to hold a few bytes of the incoming packet; when this buffer is full, inter-node flow control halts further transmission from the previous node. Wormhole switching can achieve low latency, particularly in a lightly-loaded network; however, a stalled wormhole packet can block other traffic from accessing the outgoing links. This limits network throughput and complicates the effort to avoid packet deadlock. Wormhole networks can improve throughput by including additional flit buffers or by incorporating logical lanes on each link to enable other traffic to bypass a blocked packet, as discussed below in Section 2.2.4.

Several routers employ *pipelined circuit switching*, which blurs the distinction between between wormhole and circuit switching. Unlike traditional circuit switching, pipelined circuit switching dynamically reserves network resources on a *per-packet* basis; however, unlike traditional wormhole switching, the router does not transmit the data portion of the packet until the header successfully reaches the destination node. This variant on wormhole switching can simplify deadlock-avoidance, since the small header can typically fit in a single flit buffer and can backtrack when it encounters congestion, if necessary; after route establishment, the data portion of the packet does not encounter any link contention. However, reserving the packet's path in advance incurs additional overhead, particularly in a lightly-loaded network. The router can amortize this overhead by temporarily maintaining the channel reservations, since subsequent packets may reuse the route in the near future [56].

2.2.4 Virtual Channels

For additional routing and switching flexibility, a multicomputer can extend its underlying physical topology by dividing each link into multiple *virtual channels* [31]. Although the router allocates these logical links on a *per-packet* basis, each link interleaves its virtual channels at the *flit* level. As a result, the link can service any virtual channel that has a flit to send, as long as the downstream router has sufficient buffer space. By supporting multiple virtual channels on each link, a router can allow incoming traffic to bypass stalled packets on different virtual channels. This can substantially improve the achievable network throughput of wormhole switching. In addition, virtual channels play an important role in avoiding packet deadlocks, particularly in wormhole networks [29, 87].

Deadlock arises when a chain of packets stall in the network, with each packet blocking the forward progress of another packet. A router can preclude such circular dependencies by restricting the order in which packets can access virtual channels. For example, the square mesh topology has no cycles *within* a dimension, due to the absence of wrap-links; coupled with dimension-ordered routing, packets also cannot develop cycles *between* directions, since each packet routes in the x-direction before proceeding in the y-direction. However, in a torus topology a chain of blocked packets can halt all forward progress in a particular row or column of the network. To avoid potential packet deadlocks, the router can divide each link into two virtual channels and break the circular dependencies in the underlying topology [29]. Additional virtual channels permit the construction of adaptive routing algorithms with deadlock-avoidance guarantees [44]; typically, the more flexible adaptive routing algorithms require more virtual channels to prevent deadlock [17, 87, 98].

Multicomputer networks can also use virtual channels to logically segregate traffic with different characteristics or performance requirements. For example, a router could employ separate virtual channels to separate short and long packets [67, 68, 73] or control and data packets [30, 35]. By assigning separate logical resources to each set of packets, the router effectively limits the interaction between the traffic classes. This can improve performance by limiting the number of long data packets that can receive service ahead of a short control packet. Although virtual channels improve router flexibility, they also affect network speed and implementation complexity, since each virtual channel requires arbitration logic as well as a small flit buffer [4].

Router	Topology	Routing	V-chans	Arbitration
Mesh Routing Chip [110]	2-D mesh	e-cube	1	—
Network Design Frame [30]	2-D mesh	e-cube	2	priority
Message-Driven Processor [35]	3-D mesh	e-cube	2	priority
Torus Routing Chip [28]	2-D torus	e-cube	2	fair
Reliable Router [34]	2-D mesh	adaptive	5	fair
IMS C104 Switch [117]	flexible	interval routing, universal routing	1	—

Table 2.1: Wormhole routers

2.2.5 Buffer Architecture and Link Arbitration

Careful selection of a routing algorithm reduces packet contention, improving a packet's chance of cutting through intermediate nodes. However, if two incoming packets route to the same outgoing link, one packet must be blocked and incur delay. The simplest buffering scheme places a first-in first-out (FIFO) queue at each input link to store packets unable to access their chosen output link. Unfortunately, a blocked packet at the head of an input FIFO detains any other packets in that buffer, even if the routing algorithm would assign these packets to other (possibly idle) output links. This *head-of-line* blocking significantly reduces achievable network throughput [55, 92, 116].

Alternatives to input queueing avoid this artificial contention at the expense of increased hardware complexity. *Partitioned* input-queueing replaces the single input FIFO with multiple queues to separate traffic destined for different outgoing links [92, 116]. In contrast, *output queueing* places packet buffers at the outgoing links, requiring the buffers to simultaneously accept packets from multiple input links. The router may provide separate buffers for each output queue or allow the outgoing links to share a central packet memory. While shared packet buffers enable higher memory utilization, separate queues can effectively insulate a link from other, heavily-loaded links.

If an output link services multiple virtual channels or packet queues, the arbitration scheme coordinates access amongst competing traffic. The simplest approach statically assigns link bandwidth to each queue or virtual channel, but demand-driven arbitration schemes better utilize the available network bandwidth. Priority-based schemes may improve performance by favoring longer queues or packets with stricter end-to-end delay requirements [31, 77, 120]. However, most multicomputer routers implement demand-driven, round-robin arbitration to divide bandwidth fairly amongst the competing traffic. Other arbitration policies, such as fair queueing algorithms, dispense bandwidth in proportion to a set of weights [5, 125, 126]. These schemes ensure a minimum bandwidth to each channel, independent of the traffic on other channels.

2.3 Cut-Through Router Designs

2.3.1 Wormhole Routers

Many contemporary research and commercial routers employ wormhole switching or pipelined circuit switching, as shown in Table 2.1 and Table 2.2. These tables highlight the network topology, routing algorithm, and virtual channel support in these router designs.

Router	Topology	Routing	V-chans	Arbitration
iPSC/2 Direct-Connect [89]	hypercube	e-cube	1	—
NCube 6400 [45]	hypercube	e-cube	1	—
Virtual Channel Router [56]	hypercube	e-cube	2	fair
Ariadne [3]	2-D torus	adaptive	3	fair

Table 2.2: Pipelined circuit-switched routers

The last column describes how the router arbitrates amongst multiple virtual channels contending for the same outgoing link; the Mesh Routing Chip, IMS C104, iPSC/2 Direct-Connect, and the NCube 6400 do not require channel arbitration, since each link has only one virtual channel. Most of the wormhole routers use dimension-ordered (e-cube) routing for deadlock-free communication with low implementation complexity. With dimension-ordered routing and an unwrapped mesh topology, a router requires only one virtual channel per link to avoid deadlock, as in the Caltech Mesh Routing chip; the Network Design Frame and the Message-Driven Processor include a second virtual channel to separate user and system messages (at different priority levels). In contrast, the torus topologies require two virtual channels per link, coupled with e-cube routing, to prevent communication deadlocks.

Some recent wormhole routers employ adaptive routing to circumvent network congestion. The Reliable Router includes five virtual channels on each link to support adaptive routing in a 2-D mesh, even allowing non-minimal routes when a packet must avoid a faulty link. The IMS C104 switch uses *interval routing*, which associates each output link with a programmable range of node identifiers; as a packet arrives, the routing algorithm determines which output links cover the packet's destination node. To further reduce congestion, the C104 supports *universal routing*, which sends a packet to a random node before routing to its ultimate destination. While this randomness balances network traffic load, it forces packets to travel through more intermediate nodes before reaching their destinations.

2.3.2 Virtual Cut-Through Routers

While wormhole switching stalls blocked packets, virtual cut-through routers include packet buffers to remove these packets from the network. As shown in Table 2.3, virtual cut-through routers support a diverse range of topologies, routing algorithms, and queue architectures. The Mayfly Post-Office router, designed for hexagonal mesh topologies, implements adaptive routing and queues blocked packets in a central, shared buffer. When a packet has shortest-path links in multiple dimensions, the Post-Office favors the direction with the most remaining hops; this increases the packet's likelihood of having multiple routing options at later nodes in its route. When the packet cannot route to a shortest-path link, the Post-Office considers links that leave the packet no farther away from its destination. When these links are unavailable, non-minimal routing allows the packet to travel away from its destination.

The Chaos router also allows non-minimal routing to prevent buffer deadlock in the network. If an arriving packet cannot proceed to a shortest-path output link, the Chaos router stores the packet in a small central packet buffer. When blocked packets exhaust this buffer the router transmits one of the blocked packets in a non-minimal direction to free space for arriving traffic. Deflecting packets from their ultimate destination can cause *livelock*, where a packet continually travels in the network without reaching its destination,

Router	Topology	Routing	Queueing
Mayfly Post-Office [39]	wrapped hexagonal	adaptive	shared output
Chaos Router [15]	2-D torus	non-minimal adaptive	shared pool
DEC AN1 [94]	flexible	table look-up, adaptive	input
ComCoBB [116]	flexible	table look-up	partitioned input
Artic Router [18]	multistage	static	input pools

Table 2.3: Virtual cut-through routers

but Chaos routing is *probabilistically* livelock-free [74].

Other virtual cut-through routers allow flexible construction of routing algorithms and network topologies through routing tables. The DEC AN1 switch has an adjustable routing table. When faulty links or nodes change the underlying network topology, this software automatically reconfigures each switch's routing table. While the DEC AN1 has flexible routing support, FIFO input buffers limit network performance, since a blocked head-of-line packet prohibits other traffic from utilizing idle output links. In contrast, the ComCoBB switch avoids head-of-line blocking by separating packets destined for different output links. While each input link has a single packet buffer, this memory includes a separate logical FIFO for each output link [116].

Instead of constructing separate logical queues, the Artic routing chip allows *every* buffered packet to contend for access to the output links. Since each Artic input link can buffer at most three packets, the implementation connects each packet buffer to a crossbar port. Although this provides an efficient queue architecture for Artic, the approach is not scalable to routers with larger buffer sizes due to the implementation cost of large crossbar switches. Artic prevents buffer overflow through a handshake protocol between adjacent routers. Although the Artic chip implements static routing, the system can exploit multiple paths in the network by assigning different packet routes at the source node.

2.3.3 Hybrid Routers

Recognizing the benefits of wormhole and virtual cut-through switching, some recent router designs support both switching schemes, as shown in Table 2.4. These architectures implement wormhole switching, but also provide sufficient buffer space to remove some blocked packets from the network. This combination relieves network congestion while preserving the low buffer requirements of wormhole switching. The Cray T3D router implements wormhole switching, with two pairs of virtual channels ensuring separate deadlock-avoidance for request and response messages. Each virtual channel has enough buffer space to store a small packet, while longer packets spread across multiple nodes. Since multicomputer applications typically generate a mixture of short and long packets [27, 67], router support for these two traffic classes can significantly improve performance.

Short packets incur large end-to-end latency when forced to wait behind long packets for access to the network links [49, 67]. The Segment router addresses this problem by assigning the two traffic classes to separate virtual channels with different switching schemes; long packets use wormhole switching to limit buffer-space requirements, while short packets use virtual cut-through switching to reduce channel contention. The router queues a blocked short packet in a central output buffer pending access to its output link, freeing the incoming channel for other short packets. The IBM Vulcan also reduces network congestion through

Router	Topology	Routing	V-chans	Queueing
Cray T3D Router [90]	3-D torus	e-cube	4	input
Segment Router [73]	—	—	2	shared output
IBM Vulcan [114]	multistage	static	1	shared output
PRC [38]	flexible	programmable	3	software-controlled

Table 2.4: Hybrid cut-through routers

a shared output buffer. This central queue dynamically buffers blocked wormhole packets in 8-flit chunks.

Since wormhole and virtual cut-through switching performance varies with the application workload, the PRC (Programmable Routing Controller) router can implement *micro-programmable* routing-switching schemes in small, custom processors close to the physical links. These processors execute low-level routing-switching algorithms, tailored to the application traffic patterns. This enables the PRC to implement wormhole, virtual cut-through, and packet switching, as well as hybrid schemes, each under variety of routing algorithms. In addition, the router can support multiple routing-switching schemes *simultaneously*, as discussed further in Section 5.2.2. This flexibility can significantly improve application performance, particularly in multi-user systems.

CHAPTER 3

ANALYTICAL MODELS OF CUT-THROUGH ROUTING

Cut-through switching schemes, coupled with effective routing algorithms, significantly reduce end-to-end packet delay. As networks grow larger, scalable performance requires packets to cut through as many nodes as possible; fortunately, larger networks also provide more opportunities for adaptive routing algorithms. Effective analytical models can predict the behavior of large networks and help weigh the cost-performance trade-offs of various oblivious and adaptive routing algorithms. However, analytical models require certain simplifying assumptions about the underlying interconnection network and application traffic patterns, for the sake of tractability. Simulation studies can characterize the effects of these assumptions to determine when analytical models overestimate or underestimate actual router performance.

Performance evaluation of cut-through schemes began with the work in [65], which presented a mean-value analysis of end-to-end latency for virtual cut-through switching under oblivious routing, derived from a queueing theory model for packet-switched networks. Other work extends this M/M/1 analysis to consider delivery-time distributions [41] and fixed-length packets [1]. Extensions of virtual cut-through switching address specific error recovery mechanisms [58] and partial cut-throughs [1, 124]. In addition, using virtual cut-through switching within multistage ATM switches introduces new communication characteristics and performance requirements [82, 123, 124], including bursty traffic with delay constraints. However, these analytical models do not address the impact of *routing algorithms* on cut-through switching performance.

Using both analytical models and simulation experiments, this chapter compares several shortest-path routing strategies in homogeneous virtual cut-through networks, such as tori and other k-ary n-cube topologies; in particular, the performance evaluation considers influence of the *selection function* used to rank candidate outgoing links. Adaptive routing schemes that dynamically select from multiple outgoing links best capitalize on cut-through switching, particularly when the selection function strives to increase a packet's routing opportunities at subsequent nodes in the the route. The simulation model illustrates that virtual cut-through switching introduces unique inter-node dependencies that affect the accuracy of the analytical model; novel routing algorithms can capitalize on these effects to improve network performance.

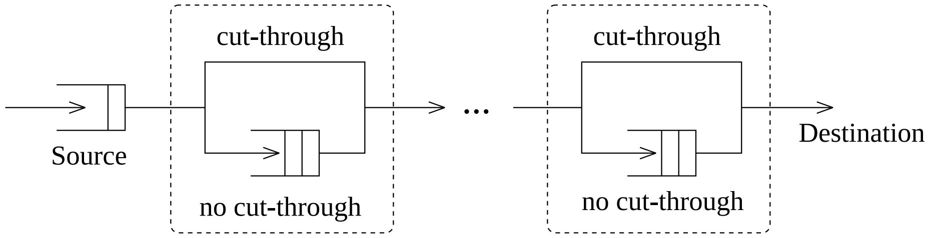


Figure 3.1: Conceptual model of a cut-through network

3.1 Router Model

A basic queueing-theory model of cut-through performance can decompose the interconnection network into a collection of independent links and queues. An arriving packet can cut through a node if its outgoing link is idle; otherwise, the packet buffers in a service queue, as shown in Figure 3.1. The various routing algorithms differ in how they affect the cut-through probability p_c, which depends on the number of candidate output links a packet can consider at each node in its route.

3.1.1 Queueing Theory

As in packet-switched networks, an M/M/1 model can accurately represent the service queue at each link, if queue size is not restricted and packet lengths and interarrival times are exponentially distributed [71]; in this framework, each source node generates packets as a Poisson process with rate λ, as shown in Table 3.1. With sufficient mixing of traffic from different sources, Kleinrock's independence assumption enables the analysis to decouple the packet length and interarrival distributions at each node [70]. In effect, a packet receives a new length at each node in its route, permitting the analysis to model the network links as independent exponential servers operating at rate $1/\overline{\ell}$, where $\overline{\ell}$ is the average packet length.

Under this assumption, a mixture of sources with Poisson arrivals produces a Poissonian output stream for the link [21]; this link, in turn, forms the traffic source for an input link to an adjacent node. With sufficient randomization of packet routes, the Poissonian traffic permits a product-form solution that separately analyzes each link in a packet's route, by Jackson's theorem [60]. Since the network links are independent, a packet traveling h hops has $h-1$ independent opportunities to cut through intermediate nodes, resulting in a binomial distribution for the number of cut-throughs [65]. Based on this binomial distribution, analytical models can determine other higher-level metrics, such as average latency or delivery-time distributions, for cut-through networks.

For example, a packet traveling h hops has average delay

$$\frac{h\overline{\ell}}{1-\rho} - p_c(h-1)\overline{\ell},$$

where $\rho < 1$ is link utilization [65]. The first term is the average delay in a packet-switched network, where the packet encounters h independent M/M/1 queues, each with mean service rate $1/\overline{\ell}$; in an M/M/1 queue, customers spend average time $\ell/(1-\rho)$ in the system [71]. The second term captures the performance benefit of employing cut-through switching. When an arriving packet routes directly to an idle link, the router avoids buffering the packet, so the average packet reduces its latency by $\overline{\ell}$ whenever it cuts through one of the $h-1$ intermediate nodes in its route. To include header processing delay at each router,

Parameter	Setting
Switching scheme	Virtual cut-through switching
Buffer architecture	Infinite packet FIFO for each output link
Routing algorithm	Random, oblivious routing
Network topology	Homogeneous network
Packet arrival	Poissonian packet generation (rate λ)
Packet destination	Uniform traffic load
Packet length	Exponentially distributed (mean $\overline{\ell}$)

Table 3.1: Idealized network and workload parameters for queueing model

the $p_c(h-1)\overline{\ell}$ term reduces to $p_c(h-1)(\overline{\ell}-\tau)$, for a header of length τ, but τ is typically small in comparison to $\overline{\ell}$.

3.1.2 Routing Algorithms

Routing algorithms impact communication performance by affecting the cut-through probability p_c. Oblivious routing directs an incoming packet to a single outgoing link, selected randomly or according to a static ordering; hence, under the assumptions in the analytical model, an arriving packet encounters an empty service queue with probability $p_c = 1-\rho$. However, adaptive routing algorithms are more difficult to analyze, since the cut-through probability depends on the *number* of links a packet can consider at each hop in its route. For example, cut-through probability increases to $1-\rho^2$ if *two* links lie along minimal paths to the destination node, since the arriving packet can establish a cut-through unless *both* outgoing links are busy; if neither link is available, we assume that the packet enters a single service queue to await transmission.

In a two-dimensional torus (k-ary 2-cube), the average cut-through probability depends on the likelihood P_2 that a packet has two routing options at an intermediate node. That is,

$$p_c = (1-\rho^2)P_2 + (1-\rho)(1-P_2) = (1-\rho)(1+\rho P_2),$$

with link utilization $\rho = \lambda\,\overline{h}\,\overline{\ell}/4$, where packets travel an average of $\overline{h}$ hops in the network; the $\overline{h}\,\overline{\ell}$ term represents the average bandwidth consumed by a packet, while the denominator corresponds to the four links emanating from each node. The probability P_2 depends on the routing algorithm, as well as the relative position of the source and destination nodes; although an empirical approach can tabulate P_2 for a specific topology and routing algorithm [84], analytical expressions allow efficient prediction of communication performance, particularly in large networks.

3.1.3 Traffic Pattern

To study uniform traffic loads, we assume that a source node is equally likely to communicate with any destinations that are h hops away; this *hop-uniform* traffic pattern generates rings of destination nodes within a fixed distance of the source node, which can represent spheres of communication locality, as shown in Figure 3.2. The node-uniform traffic pattern, which selects each node with equal probability, is a special case of the hop-uniform distribution. The hop-uniform traffic, coupled with the homogeneous network topology, permits the analysis to focus on a single source node $(0,0)$ and a "quadrant" of destination nodes

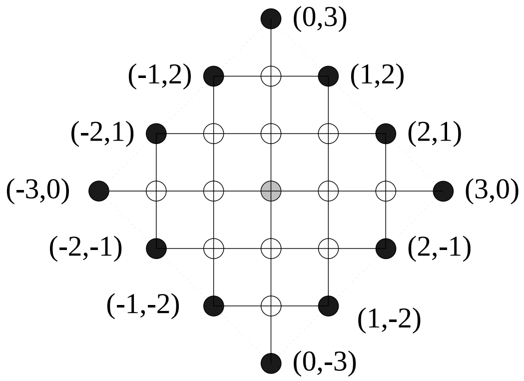


Figure 3.2: Ring of destinations $h = 3$ hops from the source node $(0, 0)$

$(0, h), (1, h-1), \ldots, (h-1, 1)$ in the torus network. Each of these destinations corresponds to a unique collection of possible shortest-path routes; the likelihood of selecting each path depends on the routing algorithm and the traffic load ρ.

To determine P_2, consider a packet that travels to destination node (x, y), with $x, y \geq 0$ and $x + y = h$. Any nodes (i, j) with $0 \leq i \leq x$ and $0 \leq j \leq y$ may lie a shortest-path route. Nodes (i, j) with $i < x$ and $j < y$ have *two* outgoing links on minimal paths to the destination node, as shown in Figure 3.3. These *internal* nodes have greater routing flexibility than the remaining *border* nodes [8]. The next section analyzes various adaptive routing algorithms by deriving

$$S^n_{(x,y)} = P[\text{packet visits } n \text{ internal nodes}], \quad n = 0, 1, \ldots h-1$$

in traveling x hops in the x-direction and y-hops in the y-direction, where $h = x + y$. After defining recurrences for $S^n_{(x,y)}$, the analysis computes P_2 by averaging across a quadrant of destinations that are h hops away from the source node. Determining P_2 as a function of h facilitates comparisons between the different routing algorithms based on cut-through probability and average packet latency.

3.2 Analysis for Cut-Through Probability

The analysis evaluates three adaptive, minimal routing algorithms in a torus network to study the influence of selection functions on communication performance, as shown in Table 3.2. In contrast to the random and dimension-ordered selection functions, the diagonal adaptive algorithm actively strives to improve a packet's chance of having multiple routing options at intermediate nodes. This can significantly improve the average cut-through probability p_c, particularly in large networks.

3.2.1 Random and Dimension-Ordered Adaptive Routing

By considering the likelihood of traveling in the x and y directions, the analysis can determine $S^n_{(x,y)}$ for each of the routing algorithms. When $x = 0$ or $y = 0$, a packet is at a border node and, hence, can only consider one direction, resulting in

$$S^n_{(x,y)} = \begin{cases} 1 & n = 0 \\ 0 & n > 0. \end{cases}$$

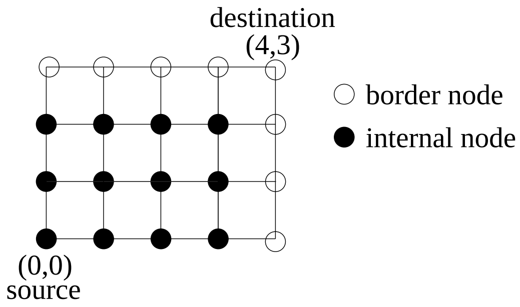


Figure 3.3: Internal and border nodes along minimal paths to node $(4,3)$

If both x and y are non-zero, define α as the probability that a packet travels in the x-direction; if a packet cannot establish a cut-through on either link, we assume that the packet joins the service queue for its first-choice direction. Under the random selection function, $\alpha = 1/2$, since neither direction has preference over the other. However, the dimension-ordered selection function routes a packet in the x-direction, unless the x link is busy and the y link is idle, resulting in $\alpha = 1 - \rho(1-\rho)$.

When both x and y are non-zero, the source node is an internal node; any other internal nodes also appear in the routes for $(x-1, y)$ and $(x, y-1)$. Thus, for $x, y > 0$,

$$S^n_{(x,y)} = \begin{cases} 0 & \text{if } n = 0 \\ \alpha S^{n-1}_{(x-1,y)} + (1-\alpha) S^{n-1}_{(x,y-1)} & \text{otherwise.} \end{cases}$$

To average $S^n_{(x,y)}$ across the quadrant of destination nodes, let S^n_h be the sum of $S_{(x,y),n}$ for $x = 0, 1, \ldots, h-1$, where $h = x + y$. For $n > 0$,

$$\begin{aligned} S^n_h = \sum_{\substack{x=1 \\ y=h-x}}^{h-1} S^n_{(x,y)} &= \alpha \sum_{x=1}^{h-1} S^{n-1}_{(x-1,y)} + (1-\alpha) \sum_{x=1}^{h-1} S^{n-1}_{(x,y-1)} \\ &= \alpha \sum_{\substack{x=0 \\ y=(h-1)-x}}^{(h-1)-1} S^{n-1}_{(x,y)} + (1-\alpha) \sum_{\substack{x=1 \\ y=(h-1)-x}}^{h-1} S^{n-1}_{(x,y)}. \end{aligned}$$

If $n = 1$, both summations evaluate to 1 because $S^0_{(0,y)} = S^0_{(x,0)} = 1$, while all other terms are zero, resulting in $S^1_h = 1$. When $n > 1$, $S^{n-1}_{(0,y)} = S^{n-1}_{(x,0)} = 0$, so both sums range over $x = 1, \ldots, (h-1)-1$. Thus, for $n > 1$,

$$S^n_h = \alpha S^{n-1}_{h-1} + (1-\alpha) S^{n-1}_{h-1} = S^{n-1}_{h-1}.$$

Hence, $S^n_h = S^{n-1}_{h-1} = \ldots = S^0_{h-n}$. Since $S^0_h = 1$ for all $h > 0$, this implies $S^n_h = 1$, for $0 \le n < h$. Thus, for h-hop packets, routes with $0, 1, \ldots, h-1$ internal nodes are encountered with equal probability $1/h$.

However, note that $S^n_{(x,y)}$ includes the source whenever it is an internal node, even though a packet cannot actually cut through the source node. Any route with one or more internal nodes has the source as one of these nodes. Removing the source node yields

$$P[N = n \text{ internal } \textit{intermediate} \text{ nodes}] = \begin{cases} \frac{2}{h} & \text{if } n = 0 \\ \frac{1}{h} & \text{if } n = 1, 2, \ldots, h-2 \\ 0 & \text{if } n = h-1\,. \end{cases}$$

Selection Function	Ranking of x and y links
Random	Select x and y links with equal probability
Dimension-ordered	Prefer x link over y link
Diagonal	Prefer direction with most hops remaining

Table 3.2: Selection functions for adaptive, minimal routing

Note that each packet has at least one intermediate node on the *border*, since the packet has only one routing option for its final hop. Averaging over the intermediate nodes,

$$P_2 = \frac{1}{h-1} \sum_{n=0}^{h-1} n \cdot P[N=n] = \frac{1}{2} - \frac{1}{h} \qquad \text{for } h \geq 2.$$

Hence, the cut-through probability simplifies to

$$p_c = (1-\rho)\left(1 + \rho\left(\frac{1}{2} - \frac{1}{h}\right)\right),$$

which approaches $p_c = (1-\rho)(1+\rho/2)$ for large h. Considering a second outgoing link, when possible, increases the cut-through probability by nearly a factor of $\rho/2$ over the oblivious routing algorithms. It is interesting to note that P_2, and hence p_c, does not depend on α. This suggests that choosing the first-choice link randomly (instead of using a static strategy, such as dimension-order routing) does not necessarily improve the likelihood of encountering internal nodes; on average, both approaches lead the packet to a border node in the same number of hops.

3.2.2 Diagonal Adaptive Routing

As seen in the previous subsection, capitalizing on multiple shortest paths improves the likelihood of cut-throughs. Oblivious routing ignored this opportunity, while the random and dimension-ordered adaptive schemes capitalize on it. To further improve communication performance, diagonal routing actively *creates* such opportunities by favoring the x-direction when $x > y$ and the y-direction when $y > x$; when $x = y$, we assume that the router breaks ties in favor of the x-direction[1]. Let α equal the likelihood of traveling in the preferred direction. The packet travels in the alternate direction only when the preferred link is busy and the alternative link is idle, so $1 - \alpha = \rho(1-\rho)$; since $\rho \in [0,1]$, $\alpha \geq 3/4$, ensuring that packets travel in the preferred directions at least three-fourths of the time.

As in the previous section, if $x = 0$ or $y = 0$, $S^0_{(x,y)} = 1$ and $S^n_{(x,y)} = 0$ for all $n > 0$. For $x, y > 0$,

$$S^n_{(x,y)} = \begin{cases} 0 & \text{if } n = 0 \\ \alpha S^{n-1}_{(x,y-1)} + (1-\alpha) S^{n-1}_{(x-1,y)} & y > x, n > 0 \\ \alpha S^{n-1}_{(x-1,y)} + (1-\alpha) S^{n-1}_{(x,y-1)} & x \geq y, n > 0 \,. \end{cases}$$

Except for destination $(0, h)$, all destination nodes in the quadrant have at least one internal node; hence, only the $(0, h)$ destination contributes to S^0_h, resulting in $S^0_h = S^0_{(0,h)} = 1$. For

[1] It can be shown that this assumption does not affect the outcome of the derivation.

$n > 0$ and even h,

$$S_h^n = \sum_{\substack{x=1 \\ y=h-x}}^{h-1} S_{(x,y)}^n \quad = \quad \sum_{\substack{x=1 \\ y=h-x}}^{\frac{h}{2}-1} \left\{ \alpha S_{(x,y-1)}^{n-1} + (1-\alpha) S_{(x-1,y)}^{n-1} \right\}$$
$$+ \sum_{\substack{x=h/2 \\ y=h-x}}^{h-1} \left\{ \alpha S_{(x-1,y)}^{n-1} + (1-\alpha) S_{(x,y-1)}^{n-1} \right\} .$$

This simplifies to

$$S_h^n = \alpha(S_{h-1}^{n-1} + S_{(\frac{h}{2}-1,\frac{h}{2})}^{n-1}) + (1-\alpha)(S_{h-1}^{n-1} - S_{(\frac{h}{2}-1,\frac{h}{2})}^{n-1}) .$$

A similar expression holds for odd h, resulting in

$$S_h^n = \begin{cases} S_{h-1}^{n-1} + (2\alpha-1) S_{(\frac{h}{2}-1,\frac{h}{2})}^{n-1} & \text{for even } h \\ S_{h-1}^{n-1} + (2\alpha-1) S_{(\frac{h-1}{2},\frac{h-1}{2})}^{n-1} & \text{for odd } h . \end{cases}$$

Note that when $\alpha = 1/2$ these two recurrences both reduce to the analysis for random and dimension-ordered adaptive routing. Since diagonal routing has $\alpha \geq 3/4$, the algorithm increases S_h^n for larger n, since routes with a larger number of internal nodes become more likely. These recurrences can be used to generate P_2, as in the previous section.

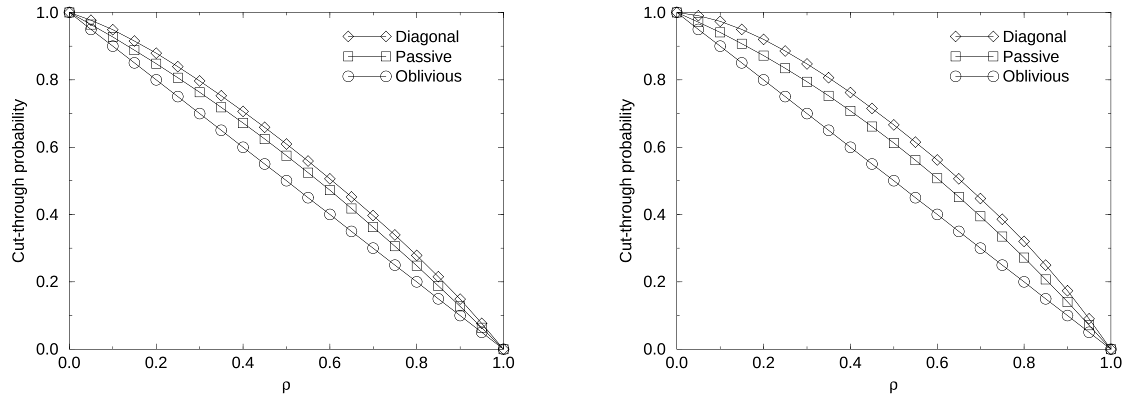
3.2.3 Performance Comparison

The analytical expressions for p_c enable performance comparisons between the oblivious and adaptive routing algorithms. Figure 3.4 shows the performance of the three algorithms with $\overline{\ell} = 64$ under changing load ρ, with $h = 5$ and $h = 20$ respectively; the "passive" curve corresponds to the random and dimension-ordered adaptive routing algorithms. At low loads, packets almost always cut through intermediate nodes, so p_c is nearly unity; in this situation, transmission delay is the main component of packet latency, so all three algorithms would exhibit similar end-to-end performance. As load increases, the adaptive schemes outperform the oblivious algorithm by increasing p_c.

In Figure 3.4(a), diagonal routing operating at $\rho = 0.5$ achieves the same cut-through probability that oblivious routing does at just $\rho = 0.4$. The benefit of diagonal routing is most evident when the packets travel a large number of hops, as in Figure 3.4(b). To further illustrate this effect, Figure 3.5 compares the three routing algorithms as h increases, for fixed values of ρ. Since cutting through intermediate nodes significantly shortens packet delay, even small increases in p_c can translate into significant reductions in end-to-end latency. Adaptive routing, and especially diagonal routing, can significantly improve performance, particularly for large networks; these effects are particularly dramatic under non-uniform traffic patterns, as shown in the next section.

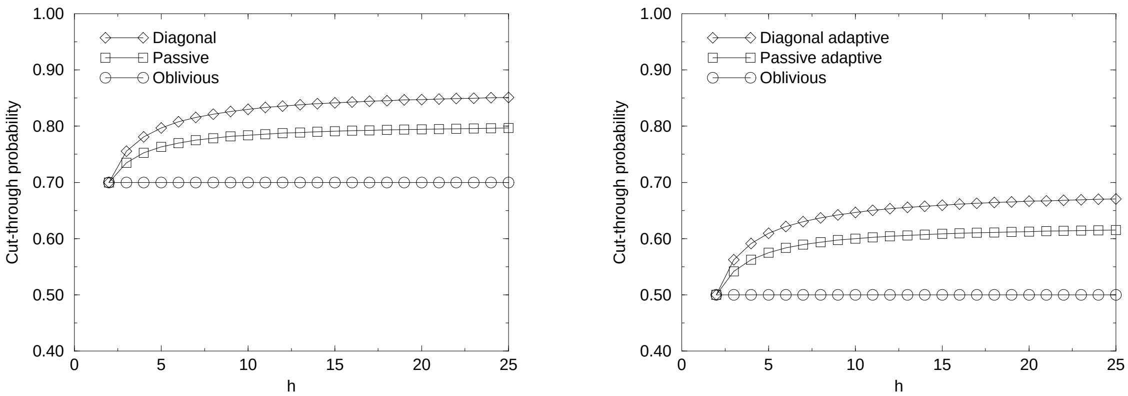
3.3 Simulation Results

The analytical expressions for p_c and average latency estimate the routing performance, but some low-level design decisions and inter-node dependencies are difficult to capture mathematically. Experiments with a discrete-event simulation model provide deeper insight



(a) $h = 5$ (b) $h = 20$

Figure 3.4: Cut-through probability p_c as a function of load ρ



(a) $\rho = 0.3$ (b) $\rho = 0.5$

Figure 3.5: Cut-through probability p_c as a function of hop-count h

into the interaction between routing and cut-through switching. To verify the analytical model, the initial experiments evaluate the router under idealized traffic assumptions; subsequent experiments consider how well the routing algorithms perform under the non-uniform traffic patterns that can arise in realistic multicomputer applications.

3.3.1 Uniform Traffic

To enable comparisons with the analytical model, the simulation experiments evaluate a crossbar switch connecting the input ports to unbounded output queues, using the simulator presented in Chapter 4. Each packet has a one-byte routing header to identify the destination node in a 16×16 torus (16-ary 2-cube) network; hence, the analytical expressions slightly underestimate average at low loads, since the analysis does not include header processing delay at each node. The simulation model describes the flow of each packet through the network, capturing contention for access to the outgoing links and the port to the local processor. In these preliminary experiments, each node independently generates packets with exponentially distributed inter-arrival times (with $\overline{\ell} = 64$ bytes) and uniform random selection of destination nodes, following the workload assumptions in Table 3.1.

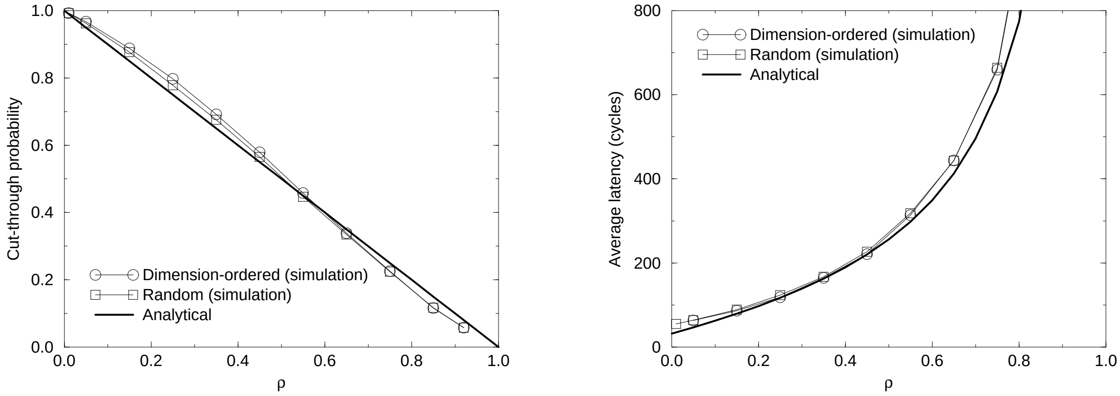
Figure 3.6 compares the analytical models to the simulation results for packets traveling five hops in a 16×16 torus network. The analytical models closely matches the simulation results. In Figure 3.6(a), oblivious routing performs slightly better when a router selects outgoing links in dimension order, choosing the x-direction over the y-direction. Under dimension-ordered routing, a packet entering a node in one direction generally exits the node traveling in the same direction. This reduces the likelihood that packets from different incoming links contend for the same output port. As a result, dimension-ordered routing has a higher cut-through probability and lower average latency, although the analytical model predicts that the dimension-ordered and random selection functions should have the same performance.

As shown in Figure 3.7, the benefit of dimension-order routing becomes more significant for larger values of h, since packets travel through more nodes in each direction, changing dimensions only once [2, 106]. In contrast, Figure 3.6(b) shows little difference between the dimension-ordered and random selection functions under *adaptive* routing, since the extra routing flexibility is sufficient to resolve these link conflicts. The diagonal algorithm exhibits the best performance by actively increasing the number of routing options, particularly at later nodes in a packet's route; however, diagonal routing does not significantly outperform the other two adaptive algorithms under uniform traffic loads, as seen in Figure 3.7.

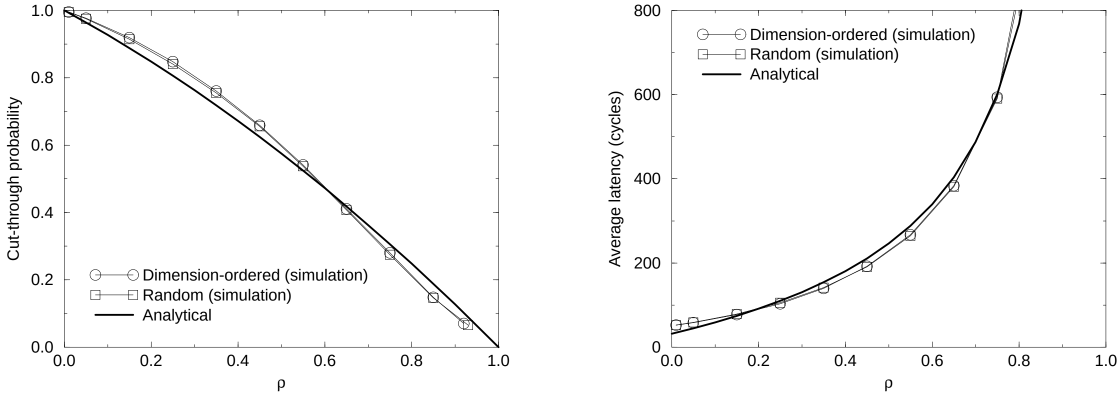
3.3.2 Non-Uniform Traffic

While the analytical model considers a uniform network load, common multicomputer constructs, such as synchronization or multicast operations, induce "hot-spots" of heavily-utilized nodes and links. Figure 3.8 compares the performance of the three routing algorithms in a simulated 16×16 torus network under non-uniform load. A single "hot" node receives 5% of the traffic, while the remaining packets select a destination at random; the "normal" uniform traffic loads the network to 20% capacity, with the non-uniform traffic generating a congested region near the hot-spot node. The plots in Figure 3.8 show average end-to-end latency for packets traveling h hops. For low values of h, normal packets do not experience significant delay, since most packets do not encounter the congested region.

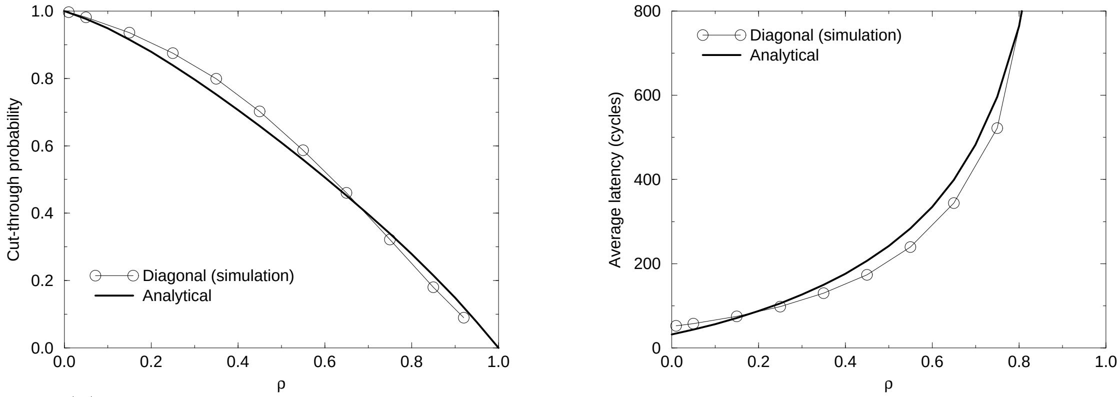
However, for dimension-ordered oblivious routing, average delay rises steeply as packets



(a) Cut-through probability and average latency for oblivious routing

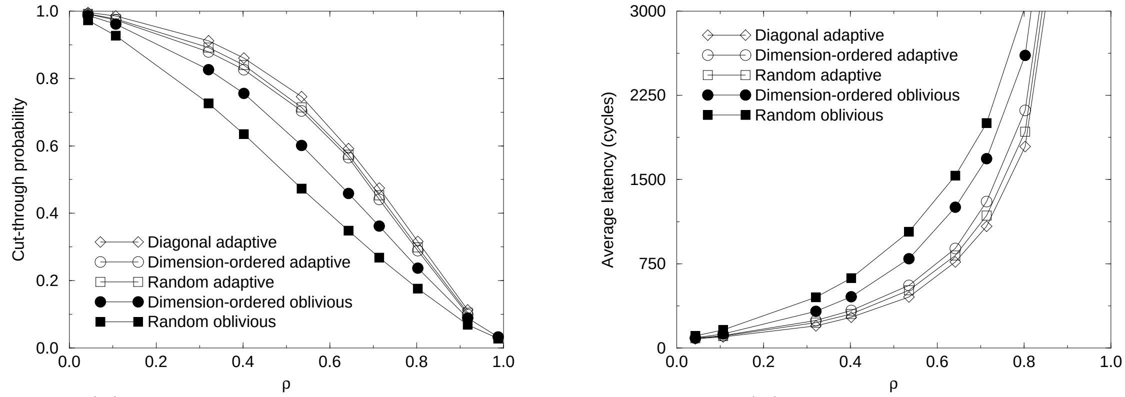


(b) Cut-through probability and average latency for passive adaptive routing



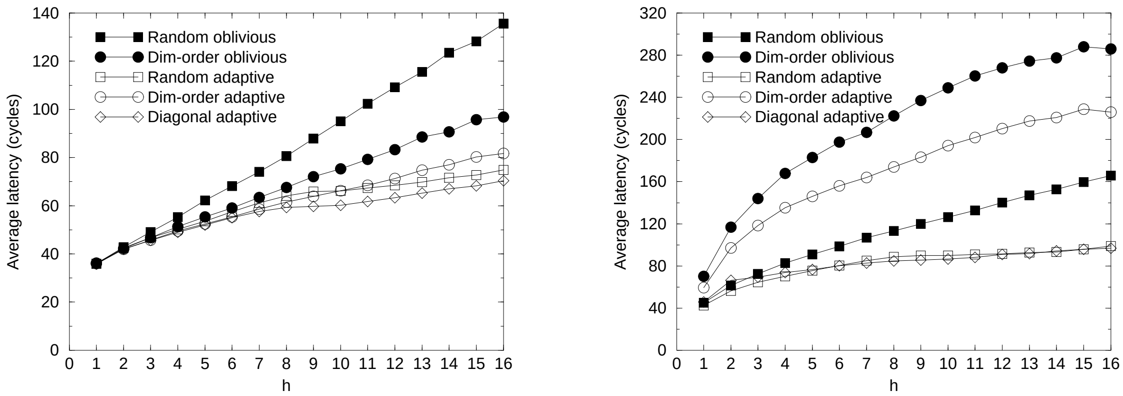
(c) Cut-through probability and average latency for diagonal adaptive routing

Figure 3.6: Analytical and simulation performance for $h = 5$



(a) Cut-through probability (b) Average latency

Figure 3.7: Simulation performance of routing algorithms ($h = 12$)



(a) "Normal" packets (b) "Hot" packets

Figure 3.8: Simulation performance under non-uniform, "hot-spot" traffic

travel further in the network. In contrast, the plots for adaptive routing stay relatively flat, since packets reduce their average latency by circumventing busy links and nodes. As h increases, the adaptive algorithms have more routing flexibility, since packets typically have more internal nodes. This improves the scalability of virtual cut-through switching as network sizes and packet distances increase. Diagonal routing performs especially well in this context, since it preserves routing flexibility even as packets near their destination nodes, clearing congestion near the hot-spot region. Packets destined for "normal" destinations avoid the nodes and links near the hot spot, significantly reducing the delay for hot packets that must enter the congested region.

The random and dimension-ordered selection functions exhibit quite different performance under hot-spot traffic, in contrast to the expectations of the analytical model. For the "normal" traffic, dimension-ordered routing has lower average latency, since packets arriving on different incoming links tend to head to different outgoing links, as in Figure 3.7. However, the random routing algorithms significantly outperform dimension-ordered routing for the "hot" traffic. This occurs because dimension-ordered routing forces most hot-spot packets to enter the hot-spot node on one of the y-direction links; this results in heavy congestion on the two y-directions links, even though the x-direction links have a relatively light load. In constrat, the random selection function better balances the traffic load across the various links near the hot-spot node. This suggests that multicomputer networks could benefit from support for multiple coexisting routing schemes, each tailored to a specific type of traffic.

3.4 Inter-Node Dependencies

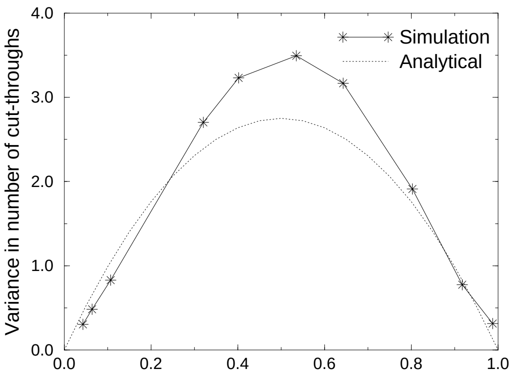
Although the simulation results match the expectations of the analytical model fairly well, the plots in the previous section show consistent performance differences. Unlike the analytical expressions, the simulation experiments capture the effects of dependencies between adjacent nodes in the network. This section isolates the effects of the independence assumption through a detailed study of the analytical and simulation models under oblivious routing algorithms. Additional experiments show that realistic router architectures and application workloads exacerbate the effects of inter-node dependencies.

3.4.1 Cut-Through Correlation

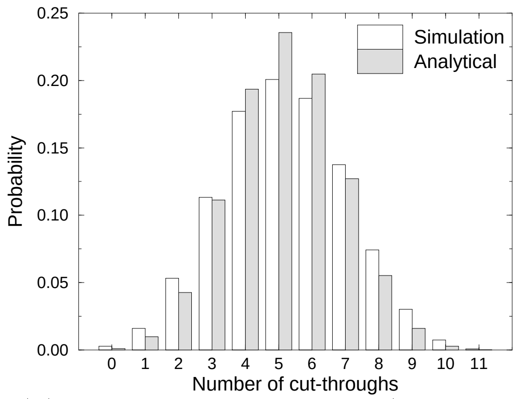
To characterize the impact of the independence assumption, Figure 3.9 considers the performance of random oblivious routing under the architectural and workload parameters in Table 3.1. While Figure 3.7 plotted the *average* values for the cut-through probability and communication latency, Figure 3.9 considers the *distribution* of the number of cut-throughs. As discussed in Section 3.1, the number of cut-throughs should obey a binomial distribution, where a packet has cut-through probability $1-\rho$ at each of the $h-1$ intermediate nodes in its route. As a result,

$$P[c \text{ cut-throughs}] = \binom{h-1}{c} (1-\rho)^c \rho^{h-1-c} \qquad c = 0, 1, \ldots, h-1,$$

with a variance of $\rho(1-\rho)(h-1)$ in the number of cut-throughs. Instead, the simulated network experiences *greater* variability, particularly for moderate values of ρ. As shown in Figure 3.9(b), the simulated network has a *flatter* distribution for the number of cut-throughs, with heavier tails than the analytical model would suggest.



(a) Variance in number of cut-throughs



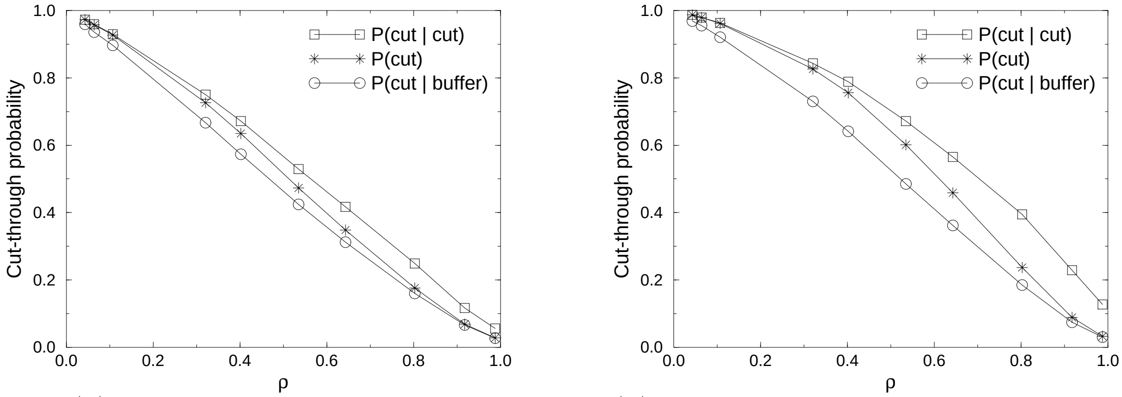
(b) Number of cut-throughs ($\rho = 0.535$)

Figure 3.9: Random oblivious routing in a 16×16 torus ($h = 12$)

As a result, the simulated network has more packets that experience a large number of cut-throughs or a small number of cut-throughs. This, in turn, translates into greater variability in end-to-end packet latency [106], even for an idealized router model. A closer look at the simulation data reveals the cause of this variability. Figure 3.10(a) shows the packet cut-through probability, conditioned on the switching decision at the packet's previous hop. In theory, the likelihood of cutting through an intermediate node should be independent of the packet's experience at previous hops in its route. However, Figure 3.10(a) shows significant variation in the cut-through probability, depending on whether or not a cut-through occurred on the packet's previous hop; the middle curve is effectively the weighted average of the two conditioned plots.

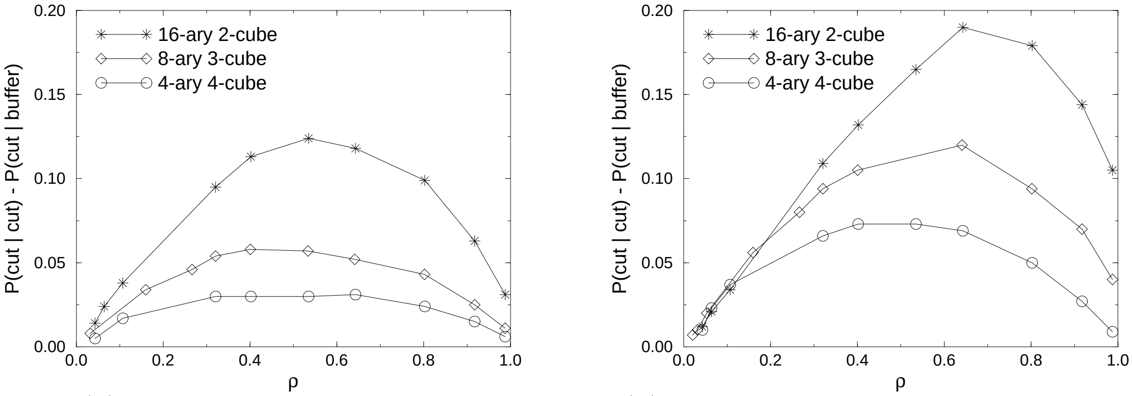
A prior cut-through encourages an additional cut-through, while buffering a packet is more likely to lead to future packet bufferings. As a result, some packets cut through several nodes in a row, while other packets consistently buffer at intermediate nodes. At low loads, this phenomenon increases the likelihood of cut-throughs, since the frequent cut-throughs perpetuate themselves, while the reverse occurs at high loads. Hence, for small ρ, the simulated packets consistently experience slightly more cut-throughs than the analytical model predicts, while the opposite is true for large ρ, as seen in Figure 3.6(a). This type of dependency seems unusual, since random oblivious routing considers exactly one outgoing link at each node, irrespective of a packet's past history.

Correlation in the cut-through probability occurs because in an actual system, as in the simulator, nodes are not truly independent. In a real network, if a packet buffers at one node, then it eventually exits the router behind at least one other packet. This increases the likelihood that the packet's next outgoing link is busy, too. Likewise, if a packet cuts through a node, then the link it uses has been idle for some length of time, increasing the likelihood that the packet encounters a light load at the subsequent node. Although such inter-node correlations also affect packet-switched networks, the performance of cut-through networks is much more sensitive to the probability of encountering idle outgoing links. As shown in Figure 3.10(b), the dependencies between nodes are even more pronounced under *dimension-ordered* routing, due to the tighter coupling between incoming and outgoing links in the same dimension of the network.



(a) Random oblivious routing (b) Dimension-ordered oblivious routing

Figure 3.10: Conditional cut-through probability in a 16×16 torus ($h = 12$)



(a) Random oblivious routing (b) Dimension-ordered oblivious routing

Figure 3.11: Cut-through correlation for k-ary n-cube topologies ($h = 6$)

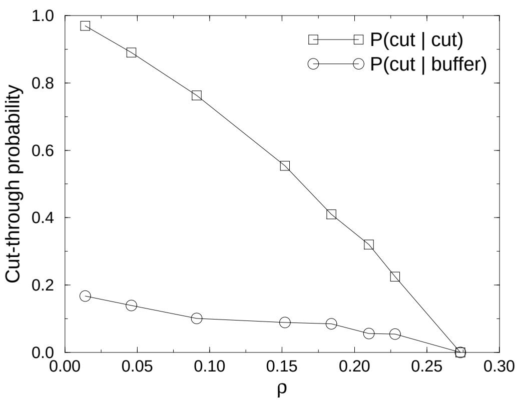


Figure 3.12: Dimension-ordered routing under bit-reversal traffic

3.4.2 Traffic Mixing

In addition, modern cut-through networks typically employ low-dimensional topologies that limit the mixing of incoming traffic streams; a larger number of incoming and outgoing links would ensure more traffic mixing, reducing the dependencies between adjacent nodes [70]. Figure 3.11 illustrates this effect by comparing cut-through correlation in three k-ary n-cube topologies with different dimensions n. The graph plots the difference

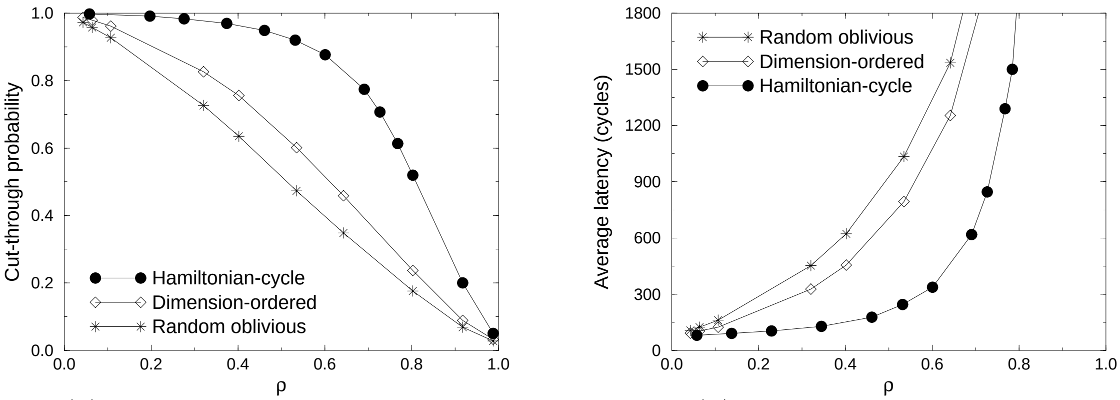
$$P[\text{cut-through} \mid \text{cut-through on previous hop}] - P[\text{cut-through} \mid \text{buffering on previous hop}]$$

for packets traveling 6 hops in the simulated networks. In the absence of inter-node dependencies, this metric should equal zero, independent of network load. While the metric consistently exceeds zero for all three topologies, the 16-ary 2-cube network exhibits a much stronger dependence on past packet history, particularly under dimension-ordered routing.

Similarly, many common communication patterns, such as scientific permutations, limit the interaction of traffic from different incoming links. Figure 3.12 shows the cut-through performance of dimension-ordered routing under a *bit-reversal* permutation, where a packet travels to the destination node whose binary address is the reverse of the source identifier; for example, in a 16×16 torus, node $(3, 10)$ communicates with node $(12, 5)$. This generates a non-uniform traffic load[2] as seen by the small peak link utilization in Figure 3.12. Under this traffic pattern, nodes in a common row (same y coordinate) of the torus communicate with destinations in the same column (same x coordinate), and vice versa. Consequently, packets that contend for the same outgoing link tend to share several links in common during the remainder of their routes; as a result, a blocked packet must repeatedly buffer behind other traffic, significantly reducing the cut-through probability, even at low network loads.

The performance of dimension-ordered routing, under uniform and non-uniform communication workloads, suggests that routers can improve network performance by restricting the mixing of traffic from different incoming links. To illustrate this effect, Figure 3.13 shows the performance of a routing algorithm that associates each input link with *exactly one* output link, effectively requiring each packet to travel in a single "dimension" for its entire route. This *Hamiltonian-cycle* routing decomposes the network topology into link-disjoint cycles that include every node in the system; for example, a 4×4 torus (4-ary

[2] Since the various parts of the network experience different traffic loads, the experiment focuses on a single pair of adjacent links; other link pairs show similar cut-through correlation effects.



(a) Cut-through probability p_c (b) Average packet latency

Figure 3.13: Performance of Hamiltonian cycle routing ($h = 12$)

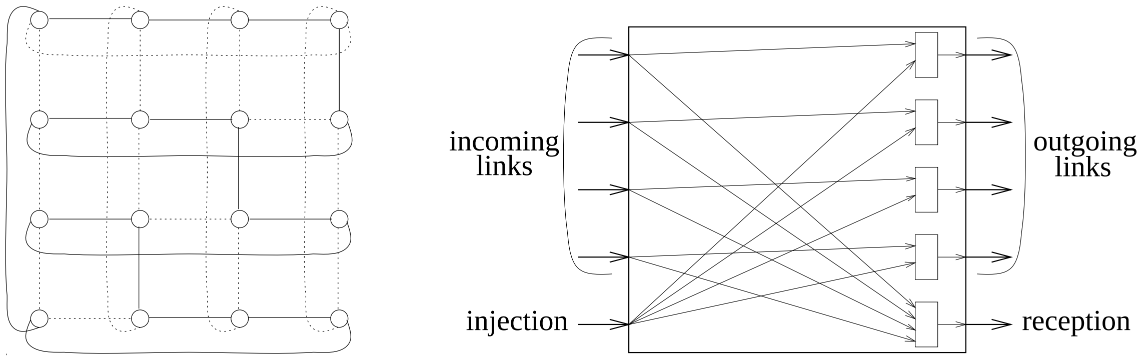
2-cube) has four disjoint, unidirectional Hamiltonian cycles, as shown in Figure 3.14. For the results in Figure 3.13, each packet travels on the cycle with the shortest path to the destination.

By routing traffic on disjoint cycles, two packets that arrive on different incoming links never compete for access to the same outgoing link; likewise, two packets arriving on the same incoming link never route to different outgoing links. Consequently, this routing algorithm has a significantly higher cut-through probability, and much lower latency, than random and dimension-ordered routing, for the same value of ρ. In addition, Hamiltonian-cycle routing facilitates an efficient router implementation by restricting the complexity of the switch that connects the incoming and outgoing links. The reduced connectivity between links also simplifies the buffer architecture for storing blocked packets, permitting the router to place packet queues at the incoming links without introducing contention between traffic headed to different outgoing links.

However, Hamiltonian-cycle routing can significantly increase the distance a packet must travel to reach its destination, unless communicating tasks are mapped to nodes that are near each other in one of the cycles; in Figure 3.13, the Hamiltonian-cycle algorithm has a much smaller λ value than the other two routing schemes, for the same value of ρ. To reduce bandwidth requirements, a network could implement a hybrid routing scheme that employs dimension-ordered routing for source-destination pairs that would have long routes under Hamiltonian-cycle routing. Alternately, to reduce the length of packet routes, the application could place communicating tasks on nodes that are near each other on one of the underlying Hamiltonian cycles. Hybrid routing algorithms, coupled with effective task allocation schemes, could significantly improve cut-through performance and reduce router complexity by restricting traffic mixing.

3.5 Conclusions and Future Work

This chapter compares oblivious and adaptive routing algorithms with different selection functions, based on both analytical and simulation models. The analysis determines packet cut-through probability in torus networks, based on the likelihood that a packet has two routing choices at intermediate nodes in its route. Due to the natural dependencies between neighboring nodes, analytical performance models can consistently underestimate or



(a) Four unidirectional cycles

(b) Two-dimensional router

Figure 3.14: Hamiltonian-cycle routing in a 4×4 torus network

overestimate cut-through probability and, in turn, other metrics like the mean and variance of end-to-end packet latency. Modern multicomputer networks employ router architectures and application workloads that exacerbate these correlation effects. This chapter introduces several research contributions:

- *Analytical models for k-ary 2-cube topologies*: The analysis in Section 3.2 facilitates the study of routing performance in large torus (k-ary 2-cube) networks common in modern multicomputers. To determine average cut-through probability, the analysis exploits the concept of *internal* and *border* nodes to formulate a recursion for computing the likelihood that a packet has two routing options at intermediate nodes. To model higher-dimensional topologies, the analysis could extend this notion to represent nodes that have $i = 0, 1, \ldots, n-1$ routing choices. Although the analysis for diagonal routing may be intractable, it may be possible to derive the cut-through probability for random and dimension-ordered adaptive routing in k-ary n-cube networks.

- *Characterization of inter-node dependencies:* The results in Section 3.4 demonstrate that cut-through switching introduces unique dependencies between adjacent nodes. By comparing analytical and simulation results, the section characterizes these correlation effects under different routing algorithms, network topologies, and communication workloads. The simulation experiments show that the network policies and traffic patterns in modern multicomputer exacerbate these dependencies by limiting traffic mixing. These results can motivate the development of novel routing algorithms and flow-control policies that exploit the natural dependencies between neighboring nodes, as discussed in Section 3.4.2.

- *Task mapping and processor allocation:* This chapter shows that communication performance depends on the likelihood that packets have multiple routing options at intermediate nodes, as well as the effects of inter-node dependencies. These observations, and the quantitative results, can guide the development of new task mapping algorithms. For example, a packet has greater routing flexibility when its has to travel one or more hops in each dimension of the underlying topology, instead of traveling in a single direction; to improve performance, processor allocation schemes can place communicating tasks on nodes with this issue in mind. Similarly, the task mapping scheme can exploit inter-node dependencies by placing tasks to minimize traffic mixing under a given routing algorithm.

By comparing analytical and simulation results, this chapter isolates the effects of inter-node dependencies and characterizes how the independence assumption interacts with realistic network topologies, routing algorithms, switching schemes, and traffic patterns. This chapter focuses on an idealized router model that facilitates direct comparisons between analytical and simulation results. Although this is important for verifying the analytical model, and characterizing the effects of the simplifying assumptions, modern routers have other architectural features that affect performance. More detailed studies of realistic networks requires greater flexibility in specifying and evaluating router models. The next chapter presents a flexible and extensible simulation environment for evaluating practical router architectures under a wide range of communication workloads and network policies.

CHAPTER 4

FLEXIBLE SIMULATION MODELS FOR EVALUATING ROUTER ARCHITECTURES

Although performance modeling provides a cost-effective way to explore router design issues, analytical models often require simplifying assumptions that degrade the accuracy of the evaluation. To overcome these limitations, several researchers have developed multicomputer simulators in the past few years. In contrast to toolkits for evaluating local and wide area networks [75], these simulators attempt to capture the unique characteristics of multicomputer applications and interconnection networks. Execution-driven simulators [11,40,93] typically model the instruction-level operation of applications on particular parallel machines. Other simulation tools emphasize multicomputer network architectures, allowing users to evaluate routing algorithms and switching schemes under various synthetic traffic patterns [16,62,81]. However, these simulators do not capture the subtle relationship between router architecture and application communication requirements.

To address the traffic patterns and performance requirements of modern multicomputer networks, this chapter presents a flexible router model and simulation environment that can evaluate systems that tailor network policies to application communication workloads. The pp-mess-sim (point-to-point message simulator) environment provides an extensible, object-oriented framework for evaluating multicomputer routers [42,102,104]. Implemented in C++, pp-mess-sim separates its major components into different classes, representing the network topology, application workloads, routing-switching algorithms, and the router architecture. The simulator includes a router model that decouples routing, switching, queueing, and arbitration policies to facilitate multifactor experiments that can independently vary each design parameter. This "virtual" router (v-router) model consists of data structures that represent architectural components, as well as simulation events that capture low-level network policies and timing details.

To allow the user to augment the simulator, each pp-mess-sim module consists of a general base class and an extensible collection of derived classes. Clean, well-defined boundaries between the components allows users to extend one pp-mess-sim module without altering the internal representation of other classes. The simulator's structure facilitates experimentation with flexible router architectures that can support multiple classes of traffic *simultaneously*. At run time, the simulator can instantiate unique communication workloads, performance metrics, and routing-switching schemes for each traffic class. The routing algorithm class defines a powerful language which can be used to write a large number of routing-switching algorithms, independent of the timing characteristics of the underlying router model. By associating these algorithms with collections of packets, instead of the

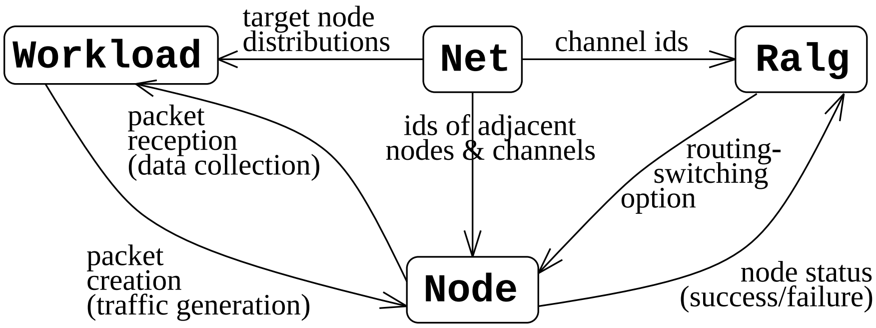


Figure 4.1: Structure of pp-mess-sim

router model, the simulator is able to support multiple routing algorithms and switching schemes *simultaneously*, with different traffic patterns and performance metrics.

4.1 Simulator Structure

As shown in Figure 4.1, pp-mess-sim's structure reflects the important architectural issues outlined in Chapter 2. The main components are a set of C++ classes supporting: network topologies (**Net**), communication patterns and data collection routines (**Workload**), routing and switching policies (**Ralg**), and particular router models (**Node**), as shown in Figure 4.1. The arrows in the figure highlight the interaction between the pp-mess-sim components. Although network design parameters interact in subtle ways, pp-mess-sim defines clean and powerful interfaces between the main simulation components, without restricting the flexibility of the tool. Thus, the simulator can easily incorporate new topologies, routing algorithms, router models, traffic patterns, and data collection routines.

The simulator also defines a specification language for composing complex experiments, with a variety of traffic patterns and network policies. In order to evaluate diverse network architectures, under complex traffic patterns, pp-mess-sim interprets a high-level language that can represent a wide range of simulation experiments, as shown in Figure 4.2. Input specification is supported by a lexical analyzer generator and a parser generator, which generate code that is linked with the rest of the simulator during compilation. The input grammar includes blocks for selecting the experiment parameters for each of the other pp-mess-sim modules. For example, Figure 4.2 specifies an 8-ary 2-cube (8×8 torus) with three virtual channels on each link, carrying a mixture of two traffic classes with different communication characteristics, network policies, and performance metrics.

The simulator's **Workload** module encapsulates the details of the traffic generation and data collection by handling all functions related to packet creation and reception. To construct a wide variety of traffic patterns, pp-mess-sim generates packets from a collection of independent "tasks," which are mapped onto individual nodes in the network to represent application behavior. For example, lines 8–15 of Figure 4.2 instantiate uniform background traffic (line 20) and a many-to-one "hot-spot" pattern (line 33), with node 0 receiving extra packets from each of the other nodes in the network. For the background traffic class, each node generates packets from a Poissonian arrival process and a bimodal length distribution; 70% of packets are short (16 bytes), while the remaining 30% are long (512 bytes). The hot-spot traffic is periodic, with each node generating a 32-byte packet every 300 time units.

Flexible composition of tasks can generate complex network workloads, with multiple

```
1 - topology begin
2 -   select kary-ncube;
3 -   size 8;
4 -   dimension 2;
5 -   channels 3;
6 - end
7 -
8 - node default begin
9 -   tasks 2;
10 -   select task hot_spot 1;
11 - end
12 -
13 - node 0 begin
14 -    tasks 1;
15 - end
16 -
17 - task default begin
18 -   arrival NegativeExpntl(400.00);
19 -   length  Discrete(0.7,16,0.3,512);
20 -   target  NodeUniform();
21 -   routing_spec begin
22 -     routing wh_adaptive(0,1,2);
23 -     order diagonal;
24 -   end
25 -   history latency;
26 -   packets 2000;
27 -   drop 200;
28 - end
29 -
30 - task hot_spot begin
31 -   arrival Uniform(300,300);
32 -   length  Uniform(32,32);
33 -   target  DestDiscrete(1.0,0);
34 -   routing_spec begin
35 -     routing wh_oblivious(0,1);
36 -     order dimorder;
37 -   end
38 -   history histogram(0,1000,50);
39 -   packets 2000;
40 -   drop 200;
41 - end
42 -
43 - general begin
44 -   random seed 1353625084;
45 -   output  mix.out;
46 -   errors  mix.err;
47 -   results mix.results;
48 -   debug   mix.debug;
49 - end
```

Figure 4.2: Example simulation specification

traffic classes, while flexible data collection allows the user to define different performance metrics for each class. The simulator computes end-to-end latency statistics (line 25), such as the mean, variance, and confidence intervals, for the background traffic, while maintaining a histogram of packet delay for the hot-spot tasks (line 38). Since the performance of routing and switching policies vary significantly depending on application communication characteristics, each task can select from the various routing-switching schemes in the **Ralg** module. The hot-spot packets employ dimension-ordered wormhole routing on two virtual channels (lines 35–36), while the background traffic uses adaptive routing, the diagonal selection function, and an extra virtual channel (lines 22-23) to circumvent congested regions of the network.

Together with the **Workload** and **Ralg** classes, the pp-mess-sim **Net** module insulate the **Node**'s event flow from the details of application characteristics and the network configuration. For example, the **Net** class encapsulates the specific network topology, by providing an interface for other modules to identify and translate node, link, and virtual channel identifiers; currently, pp-mess-sim supports k-ary n-cube topologies, square meshes, and wrapped hexagonal meshes. When a pending **Node** event sends data or a flow-control acknowledgement across an outgoing link, **Net** functions spawn the corresponding reception event in the adjacent router. Each router, then, receives new packets from the **Workload** and in-transit packets from adjacent nodes, with no dependence on the network topology, communication patterns, or internal router policies at other nodes.

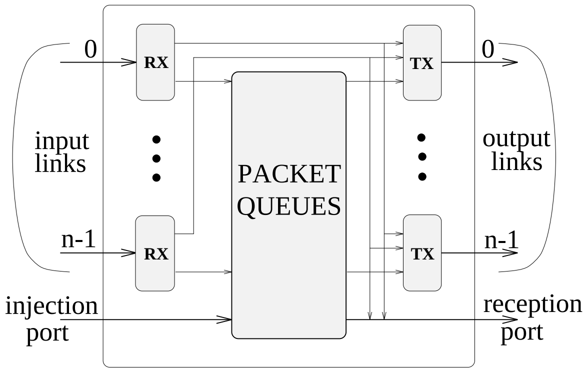


Figure 4.3: V-router model

4.2 Virtual Router Model

By defining strict interfaces between individual parts of the code, pp-mess-sim insulates the **Node** module from the **Net**, **Ralg**, and **Workload** modules. Within this framework, the v-router (virtual router) **Node** model introduces several useful abstractions for representing flow control and resource arbitration.

4.2.1 Router Model

Figure 4.3 shows the high-level architecture of the v-router; Figure 4.4 shows the corresponding class definition in pp-mess-sim. A packet enters the router from the local injection port or one of the n incoming links and departs through the reception port or one of the n outgoing links, where n depends on the network topology. Each physical link multiplexes traffic for c virtual channels at the granularity of a flit cycle, while the injection and reception ports handle packets on behalf of the nc outgoing and incoming virtual channels, respectively. Although every router design implements its internal policies in different ways, each device proceeds through common operations to service an incoming packet. The v-router model decouples these phases to allow simulation experiments to independently vary the internal routing, switching, queueing, and arbitration policies.

Upon receiving the header flits of an incoming packet, the receiver (RX) decides whether to buffer, stall, or forward the packet, based on the routing and switching policies and prevailing network conditions. By treating outbound virtual channels as individually reservable resources, the model can invoke a variety of routing and switching schemes through flexible control over reservation policies. The routing algorithm generates candidate outgoing virtual channels, while the switching scheme determines whether or not an incoming packet waits to acquire a selected outgoing virtual channel or buffers instead. If the switching decision requires the packet to buffer at the current node, the router submits the arriving packet into the queue.

The data structures in the queue model determine the admission and scheduling of the buffered packets. When the router tries to enqueue a packet, the queue module either accepts or rejects this request, depending on the current buffer space and the size of the packet. Separate from the enqueueing mechanism, the queue model ranks competing packets, determining which buffered packets can contend for access to the physical links. The v-router module currently implements first-in first-out queueing at each output link; extensions can consider other policies, based on packet length, priority, or age, as well as alternative ar-

```
class v_router : public Node {
  // Sizes and identifiers
  NodeId id;          // Identifier of node
  DevId  max_dev_id;  // Number of virtual channels

  // Virtual channels
  VCRX*  vc_rx;       // Incoming virtual channels (c * n)
  VCIN*  vc_in;       // Injection virtual channels (c * n)
  VCTX*  vc_tx;       // Outgoing virtual channels (c * n)

  // Internal interconnections
  RXBus  rxbus;       // Reception bus (c * n slots)
  TXBus  txbus;       // Transmission bus (c * n slots)
  CTBus  ctbus;       // Cut-through bus (2 * c * n slots, for RX and TX)
}
```

Figure 4.4: Internal components in the v-router Node model

chitectures, such as input queueing. Separate arbitration policies enable each outgoing link to select packets from the heads of competing queues. Once a packet reserves an outgoing virtual channel, it contends with other virtual channels for access to the physical link (TX) through an arbitration policy, independent of the queue architecture. The model includes several arbitration policies, including round-robin and priority-driven schemes.

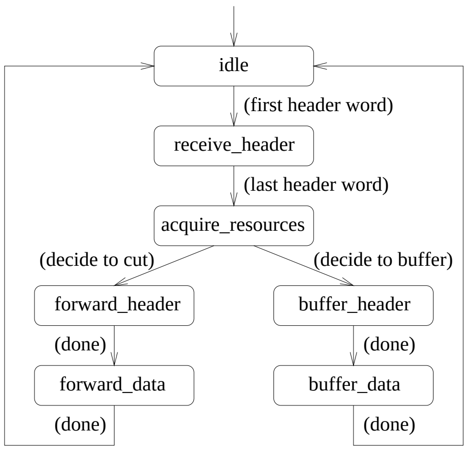
4.2.2 Router Components

Similar to behavioral hardware description languages, the v-router represents each router component as a state machine, where simulation events trigger each state transition. For example, Figure 4.5(a) shows a state machine for an incoming virtual channel. The channel remains idle until the incoming link signals the arrival of the header flits for a new packet. After receiving the full packet header, the channel must make a routing-switching decision, with the help of the **Ralg** module; this may require the channel to reserve buffer space or an outgoing virtual channel, as discussed in Section 4.3. Once these resources are available, the channel can forward the accumulated header bytes, followed by the remainder of the packet, before returning to the idle state.

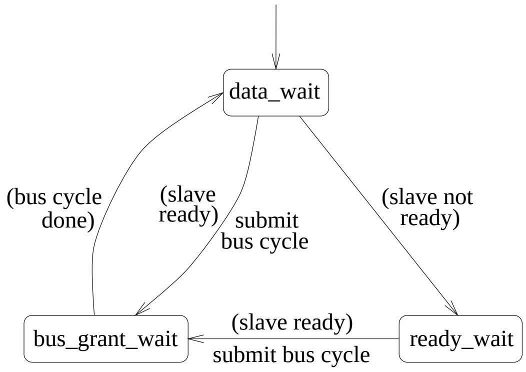
Flow Control

Visiting each state involves the passage of simulation time, represented by one or more simulation events. At a lower level, some operations require the virtual channel to interact with other router components, such as an incoming or outgoing link. A separate state machine can encapsulate the low-level flow control at each interface, as shown in Figure 4.5(b). For example, an incoming virtual channel (`vi_rx`) waits for arriving data before forwarding the new word to an outgoing virtual channel (or the reception port). However, this transfer cannot occur until the slave device (`vi_tx`) has sufficient buffer space; then, the `vi_rx` can request a slot on the bus to forward the word to the `vi_tx`. Similarly, the `vi_tx` model consists of a small state machine for transmitting words and awaiting flow-control acknowledgements from the adjacent router.

The simulator models the flow control between router components using a *wake-up queue* interface, hiding the internal details of each module. For example, Figure 4.5(b)



(a) Receiver state machine



(b) Bus interface control

Figure 4.5: Node state machines

shows how an incoming virtual channel remains in the `ready_wait` state until its slave device becomes available. To encapsulate the details of the slave device, the v-router allows the incoming channel to register a pending simulation event in the outgoing channel's wake-up queue. Once the outgoing channel becomes available, the simulator drains the wake-up queue and inserts the entries into the main event queue; then, the simulation event can execute and notify the incoming channel that the slave device has become available, allowing the channel to transition to the `bus_grant_wait` state. The simulator uses a similar wake-up queue mechanism to notify waiting packets when an outgoing channel becomes eligible for reservation.

Resource Arbitration

Similarly, the v-router **Node** model incorporates useful abstractions for representing arbitration for shared internal resources, such as buses. When multiple modules can compete for access to a resource, each module's state transitions and delays depend on the arbitration policy. To insulate each module from these details, the v-router associates an arbiter with each shared resource. To access to the shared resource, a module registers its pending simulation event with the *arbiter* model, instead of the simulator's main event queue; the arbiter class includes an array `requests` of pending events, as shown in Figure 4.6. For example, in Figure 4.5(b), an incoming virtual channel requests access to the bus by submitting its bus-cycle event to the arbiter; the channel remains in the `bus_grant_wait` state until the bus arbiter triggers execution of the bus-cycle event.

Separate from the other components, each arbiter schedules arbitration events at regular intervals, depending on the speed of the unit; if no virtual channels are awaiting access to the shared resource, the next `submit` operation spawns the next arbitration event. The event handler `process_events` implements the arbitration policy, determining which registered events should be transferred to the main event queue for subsequent execution. This flexible framework allows users to extend the v-router to include new arbitration policies without affecting the other simulation modules. In particular, the v-router can instantiate several different arbiter models, including various demand-slotted buses, physical crossbars, and

```
class Arbiter {
  // Counter variables
  short chans_requested;   // Total number of channels sharing the resource
  short pending_requests;  // Number of channels awaiting access to resource
  short cycle_time;        // Time unit for resource allocation and scheduling

  // Registered events
  EventPtr* requests;     // Array of queued events (size chans_requested)
  void submit(ArbitrationId,EventPtr);  // Enqueue event and schedule arbitration
  void process_events();  // Dequeue and invoke event(s)
}
```

Figure 4.6: Arbiter class in pp-mess-sim Node models

virtual-channel crossbars. Crossbar models can transfer *multiple* events to the simulator's event queue in a single arbitration cycle, in contrast to bus models that active a single event in each arbitration cycle. By changing the order the arbiter scans the array of pending events, the v-router can also evaluate priority-based arbitration schemes.

4.3 Routing and Switching Algorithms

Tuning a network design requires evaluating routing and switching schemes under a variety of arbitration, queueing, and flow-control policies. The simulator facilitates such experimentation by decoupling the router models (**Node**) from the routing-switching algorithms (**Ralg**), as shown in Figure 4.1. This functional separation allows the user to easily prototype new routing-switching algorithms without changing the **Node** models. additional support in the **Net** module insulates the algorithms from the details of the network topology, as discussed in Section 4.4. By associating a routing-switching algorithm with each **Workload** task, pp-mess-sim can allow multiple policies to coexist in the simulated network.

4.3.1 Routing-Switching Instructions

Although multicomputer routers implement routing and switching in various ways, every router proceeds through common operations to service an incoming packet, as shown in Figure 4.5(a). When a packet arrives from a host injection port or an incoming link the router parses the header flits to make a routing-switching decision. The **Ralg** module decouples this decision-making process from the simulation event flow in the **Node** model. Invoked after packet header collection, the **Ralg** module interacts with the **Node** using a series of *instructions* (`RouterInstr`) until they agree upon a suitable routing-switching decision. This allows the high-level routing-switching algorithm to make its decisions based on feedback from the **Node**, without low-level knowledge of the router architecture.

Similarly, while the router model must accept commands from the routing algorithm, the **Node** does not need to know how this algorithm selects the sequence of operations. The **Ralg** instruction set embodies basic primitives for constructing routing-switching algorithms. Each instruction consists of a candidate switching decision and an ordered list of outgoing virtual channels, as shown in Figure 4.7. The list of virtual channels encapsulates the routing options generated by the algorithm, while the candidate switching decision helps the router decide whether to buffer, stall, drop, or forward the packet. The **Node**

```
class Ralg {
  // Construct packet's initial routing instruction at source or intermediate node
  void i_inject_init(PacketPtr, NodePtr, DevId);

  // Construct new routing instruction in response to feedback from Node
  void i_complete(PacketPtr, NodePtr, DevId);

  // Commit final routing-switching decision
  void i_commit(PacketPtr, NodePtr, DevId);
}

class RouterInstr {
  RouterOp  op;              // Candidate switching decision
  DevOp*    devop;           // Ordered list of virtual channels and their status
  short     num_devops;      // Number of devices (virtual channels) to attempt
}
```

Figure 4.7: Ralg routines for interacting with the Node model

examines each instruction and determines whether or not the output channel(s) can satisfy the request. The algorithm and the router model continue this request-response handshake until they agree on a common routing-switching decision.

4.3.2 V-Router Handler

Node simulation events distinguish the architectural and timing details in different router models. Between successive interactions with **Ralg**, the **Node** may try to reserve channel or buffer resources to successfully complete the operation; this process may involve multiple simulation events and, perhaps, advancement in simulation time. For example, the v-router includes a routing-algorithm handler that coordinates interaction with **Ralg**, as shown by the pseudocode in Figure 4.8. An arriving packet invokes this routine in three scenarios. First of all, when the incoming link receives the packet's last header flit, the v-router executes the handler and invokes the **Ralg** `i_inject_init()` routine. Then, the handler executes again when the **Ralg** returns a new routing-switching instruction upon invocation of the `i_complete` routine.

The third scenario arises when an incoming packet stalls waiting for an outgoing virtual channel to become available. As discussed in Section 4.2.2, the packet blocks by registering an event in the virtual channel's wake-up queue. Later in simulation time, the channel completes the transmission of its current packet, causing the v-router to drain the wake-up queue and invoke the handler for the blocked packet. The generality of the **Ralg** instruction set allows pp-mess-sim to include other **Node** models with different timing properties. For example, while the v-router model allows a packet to "instantly" reserve an idle virtual channel, other router architectures may incur delay or contention in acquiring a channel. These timing details are completely encapsulated in the **Node**, allowing other models to include additional simulation events to capture internal delays without affecting the routing-switching algorithms in the **Ralg** class [102].

4.3.3 Routing-Switching Algorithms

Using the **Ralg** instruction set, pp-mess-sim can easily incorporate additional routing algorithms and switching schemes, without altering the **Node** models. For example, Fig-

```
if (packet header has just arrived)
  call i_inject_init for the packet;

handler:
  // Process instruction
  switch (op)
    Cut:
      // Sequence through the candidate virtual channels
      for (i=0; i<num_devops; i++)
        if (channel devop[i] is not reserved)
          reserve channel devop[i] and register success;
          break;

      if (all channels are reserved)
        register failure;
      call i_complete for the packet;
      goto handler;

    Wait:
      // Sequence through the candidate virtual channels
      for (i=0; i<num_devops; i++)
        if (channel devop[i] is not reserved)
          reserve channel devop[i] and register success;
          cancel any registered wake-up events for this packet;
          call i_complete for the packet;
          goto handler;

      // All channels are busy, so packet must block waiting for wake-up event
      if (all channels are reserved)
        register failure;
        submit a wake-up event to each candidate channel;
        break;

    Buffer:
      // Sequence through the candidate virtual channels
      for (i=0; i<num_devops; i++)
        register success;
      call i_complete for the packet;
      goto handler;

    Commit:
      // Ralg and Node have agreed on a routing-switching decision
      cancel any registered wake-up events for this packet;
      submit simulation event to start forwarding the packet to buffer or link;
```

Figure 4.8: V-router interaction with Ralg

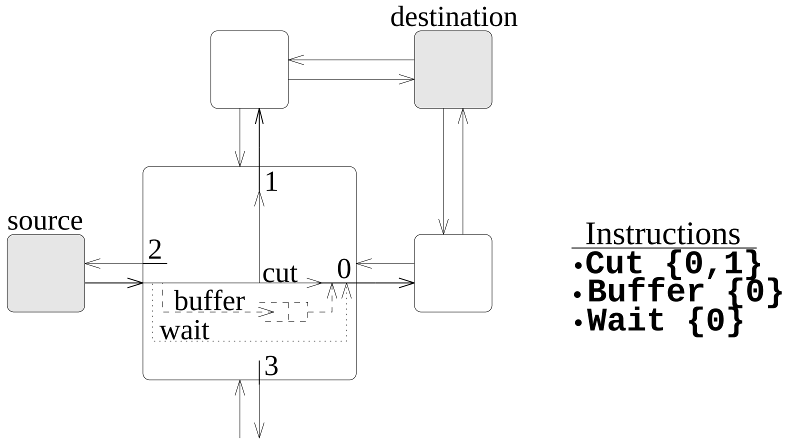


Figure 4.9: Sequence of routing-switching instructions

ure 4.9 shows a shortest-path routing algorithm that tries to buffer a packet when its outgoing links are busy; if the buffers are full, the incoming packet waits for a link to become available (similar to wormhole switching). To implement this algorithm, **Ralg** first asks the **Node** to establish a cut-through along outgoing channel 0 or 1. Upon receiving the *cut* instruction, the **Node** module first tries to reserve outgoing channel 0, resorting to channel 1 if the first link is busy. To acquire a channel, the **Node** module may invoke one or more simulation events to model internal router delays. If neither attempt is successful, the **Ralg** responds with another instruction, asking the router to *buffer* the packet for later transmission on channel 0.

The router's queue architecture determines if the node can accommodate the new packet; if the router cannot store the incoming packet, the **Node** rejects the *buffer* instruction. Ultimately, the **Ralg** requests that the packet *wait* until channel 0 becomes available. Eventually, a simulation event frees the channel 0, allowing the **Node** to reserve the outgoing channel and successfully complete the *wait* instruction; then, the packet begins transmission to the next node in its route. The **Ralg** instruction set enables pp-mess-sim to model a wide range of routing-switching algorithms, as shown in Table 4.1. The *buffer* instruction implements packet switching algorithms, while virtual cut-through schemes employ a combination of *cut* and *buffer*; wormhole switching schemes utilize the *wait* instruction, where the underlying **Node** model determines the flow-control and arbitration policies.

In addition to traditional switching schemes, sequences of **Ralg** instructions can generate hybrid algorithms that incorporate aspects of both virtual cut-through *and* wormhole switching, such as the example in Figure 4.9. These hybrid switching schemes dynamically balance the use of channel and memory resources for "storing" blocked packets. For example, the h-hop hybrid algorithm in Table 4.1 allows a blocked packet to stall (using the *wait* construct) only if the packet spans fewer than h links; otherwise, the blocked packet buffers at the intermediate node, releasing any channel resources [111]. This algorithm limits channel contention, while still restricting the use of packet buffers.

The **Ralg** instructions can also implement various routing algorithms by generating different lists of candidate virtual channels. As shown in Table 4.1, pp-mess-sim includes several oblivious and adaptive routing algorithms for the different switching schemes. The simulator uses Duato's theory [44] to construct deadlock-free adaptive routing algorithms under wormhole switching. Each algorithm requires a minimum number of virtual channels

Switching	Routing
Packet switching	Minimal oblivious Minimal adaptive
Virtual cut-through	Minimal oblivious Minimal adaptive Non-minimal adaptive
Wormhole	Minimal oblivious Minimal adaptive Non-minimal adaptive
Hybrid (h-hop)	Minimal oblivious

Table 4.1: Examples of routing-switching schemes in pp-mess-sim

for deadlock-free routing; the algorithm uses any additional channels to improve network throughput. The specification file determines how many and which virtual channels are assigned to the routing algorithm, as shown in lines 22 and 35 of Figure 4.2. The routing instructions provide flexibility and extensibility, allowing pp-mess-sim users to add new routing-switching algorithms and experiment with a mixture of policies in the simulated network.

4.4 Workload and Topology Support

In multicomputer networks, communication performance hinges on the subtle interaction of routing-switching algorithms with the application traffic pattern and the network topology. To facilitate a wide range of experiments, pp-mess-sim supports the flexible composition of complex communication workloads, with distinct router policies and performance metrics, on different network topologies.

4.4.1 Communication Workload

The **Workload** module generates packets from a collection of independent tasks, which are mapped onto individual nodes in the network to represent application behavior, as shown in the example in Figure 4.2. Since the performance of routing and switching policies vary significantly depending on application communication characteristics, each task can select from the routing-switching schemes in the **Ralg** module. In order to simulate realistic workloads, pp-mess-sim provides a rich set of packet length, interarrival time, and destination node distributions[1], as shown in Table 4.2. The simulator can generate complex, non-uniform workloads by selectively mapping tasks onto particular nodes in the network. Flexible task specification and mapping, combined with diverse traffic models,

[1] Each task on each node requires access to random number streams to generate packet lengths, interarrival times, and destination nodes. The simulator extends the additive congruential generator (ACG) [72] in the GNU libg++ libraries to provide a multi-threaded generator with a separate random number streams for each stochastic process in each task. Starting with a single input seed (e.g., line 44 in Figure 4.2), pp-mess-sim divides the resulting random number stream into consecutive chunks, assigning a separate chunk to each stochastic process. This significantly reduces correlation between the processes by generating multiple non-overlapping random number streams [61]. If a process exhausts its chunk, the next unused chunk is allocated from the original stream.

Distribution	Definition
Negative exponential (λ)	Exponential distribution with mean λ
Uniform (a, b)	Select integers between a and b with equal probability
Discrete ($\{p_i, \ell_i\}$)	Select ℓ_i with probability p_i
Normal (μ, σ)	Normal distribution with mean μ and standard deviation σ
Two-stage normal $(p, \mu_1, \sigma_1, \mu_2, \sigma_2)$	Select from normal distribution (μ_1, σ_1) with probability p; otherwise, select from normal distribution (μ_2, σ_2)

(a) Packet length and interarrival distributions

Distribution	Description
DimensionReversal	Source $(w, x, \ldots, z)$ selects destination $(z, \ldots, x, w)$
BitComplement	Destination node id is the bit-complement of the source id
BitReversal	Destination node id is the bit-reversal of the source id
HopUniform ($\{p_i\}$)	Select a destination i hops away with probability p_i
NodeUniform	Uniform random selection of destination node
Discrete ($\{n_i, p_i\}$)	Select "hot spot" destination node n_i with probability p_i

(b) Destination node distributions

Table 4.2: Traffic patterns in pp-mess-sim

enable pp-mess-sim to impose a wide range of communication patterns on the underlying network.

The **Workload** module has a simple interface to the event flow in the **Node**, facilitating extensions that incorporate new packet generation models. The simulator encapsulates packet length, interarrival, and destination node distributions through generic functions, as shown in Figure 4.10. **Workload** schedules one packet creation event for each task on each node, with the event handler employing the `next_packet_time()` function to submit the next creation event. Since pp-mess-sim isolates creation times in the task model, the user can incorporate new packet generation schemes, including multi-state models, without affecting the rest of the simulator. Using these generic functions, pp-mess-sim can easily be extended to run in a trace-driven mode by simply writing functions which read packet arrival times, packet lengths, and packet destinations from a file containing application traces rather than generating the values from distributions.

The simulator provides effective data collection by associating performance metrics with the task construct, as shown in Figure 4.10. This allows the user to specify a different metric for each task, as shown in Figure 4.2, to evaluate traffic with diverse performance requirements. Since the behavior of the simulated network changes over time, performance metrics are extremely sensitive to the interval of data collection. Accurate measures of steady-state performance require both a sufficient warm-up period and a reasonable averaging interval. To prime the network, each task on each node must deliver a certain minimum number of packets to their destinations before any data collection commences. The user may configure a different number of "warm-up" packets for each type of task through the "drop" field in the task specification (as in lines 27 and 40 of Figure 4.2).

After all tasks have completed their required "warm-up" packets, each task accumulates performance data until the required number of its packets have completed service (as specified in lines 26 and 39 of Figure 4.2); the task continues to generate packets until every

```
class Task {
  TaskId         id;                    // Task identifier
  NodeId         node;                  // Node identifier

  Random*        arrival;               // Stochastic process for interarrival times
  MTACG*         arrivalacg;            // Random number stream for arrival times
  delta_time     next_packet_time();    // Return next packet arrival time

  Random*        length;                // Stochastic process for packet lengths
  MTACG*         lengthacg;             // Random number stream for packet lengths
  PacketLength   next_packet_length();  // Return next packet length

  NodeIdRand*    target;                // Stochastic process for destination node
  MTACG*         targetacg;             // Random number stream for destination node
  NodeId         next_packet_target();  // Return next packet destination

  unsigned int   generated;             // Number of task's packets generated
  unsigned int   delivered;             // Number of task's packets delivered
  unsigned int   collected;             // Number of task's packets collected
  RoutingAlgPtr  r_prog;                // Pointer to routing algorithm
  HistCollect*   history;               // History list for data collection
}
```

Figure 4.10: Workload Task model

task in the network has completed data collection. During the data collection phase, each task accumulates performance statistics as its packets reach their destinations. The simulator provides an extensible mechanism for collecting packet statistics for each task. As a packet travels through the simulated network, the router model maintains a history list that records significant events during the packet's journey. For example, if a packet cuts through an intermediate node, the location, time, and event (e.g., **cut**) are appended to the history list. When the packet arrives at its destination node, the data collection routine processes the list to extract the desired performance metrics.

With help from the **Node** modules, the data collection routines can accumulate a wide variety of performance statistics, as shown in Table 4.3. The timestamps on the history records indicate the end-to-end latency of the packet, as well as the components of this delay. Logging the event type allows the collection routines to evaluate the routing and switching decisions that occurred for each packet. Existing history collection routines capture end-to-end delay statistics, packet cut-through probabilities, and latency histograms. For example, in Figure 4.2 the `hot_spot` tasks capture a histogram of latency data to estimate the probability distribution of packet delay (line 38), while the `default` tasks collect basic latency metrics (line 25), including the mean, max, variance, and confidence intervals.

Since performance may vary with communication distance, these routines also maintain separate statistics based on the number of hops a packet travels. Tasks may also select a null collection routine; this avoids accumulating unnecessary performance data for any "background" traffic in the system. The history collection mechanism also allows for simple extensions for additional performance metrics to study specific research issues. For example, Chapter 3 used statistics on packet cut-through history to investigate the effects of internode dependencies on the performance of virtual cut-through switching. Adding customized entries to the history list can create a fairly detailed list, allowing the collection routines to reconstruct the behavior of the packet and the network.

Metric	Description
Latency	Mean, max, variance, and confidence intervals for packet latency
Histogram(a, b, c)	Histogram of packet latency with c bins over range $[a, b]$
Cut-through statistics	Histogram of packet cut-through history
Null	No data collection

Table 4.3: History-list data collection routines in pp-mess-sim

4.4.2 Network Topology

While some routing algorithms depend on a particular topology, most schemes require only high-level information about the various output links at each node. To facilitate simulation experiments that vary network topology, the pp-mess-sim **Net** class, as shown in Figure 4.11, includes functions which encapsulate the labeling scheme used to number each node, link, and virtual channel in the network; derived classes implement the numbering schemes for the k-ary n-cube, square mesh, and hexagonal mesh topologies. To decouple the routing-switching algorithms from the network topology, the **Net** class includes functions that generate a list of possible directions for a packet to travel. For example, given the current node and the packet's destination, **Net** identifies which output links lie on a minimal path; in Figure 4.9, **Net** returns the link set $\{0, 1\}$. Alternatively, **Net** can determine which outgoing links would deflect a packet *away* from a shortest-path route.

Since routing performance often depends on the *order* the router considers the output links, **Net** includes functions for ranking the candidate outgoing links. These *selection functions*, coupled with the **Ralg** routing-switching instructions, enable pp-mess-sim and the v-router to model a wide range of communication policies on different network topologies. For example, in line 36 of Figure 4.2, the `hot_spot` traffic is assigned dimension-ordered routing, whereas the `default` traffic employs the diagonal selection function, as described in Table 3.2. In addition to these two options, and the random selection function, **Net** can rank outgoing links according to network congestion, giving preference to links with fewer busy virtual channels [32]; this balances traffic load amongst the outgoing links, reducing contention and packet delay.

In addition to supporting routing-switching algorithms, **Net** also insulates the **Node** and **Workload** modules from the details of the network topology. As shown in Figure 4.11, **Net** includes a set of neighbor functions to identify adjacent nodes. When a pending **Node** event sends data or a flow-control acknowledgement across an outgoing link, these neighbor functions identify the node, link, and virtual channel that should receive this information, as shown in Figure 4.1; this allows a transmission event in one node to spawn the corresponding reception event in the adjacent router. Coupled with the support in **Ralg**, these mapping functions decouple the **Node** models from the labeling scheme in the **Net** class. This facilitates simulation experiments that evaluate a router design under different network topologies.

In addition to the neighbor and direction functions, the **Net** module includes routines that assist **Workload** in generating traffic patterns. Mapping parallel applications across multiple nodes results in unique communication workloads that depend on the network topology. In order to capture the communication behavior of scientific applications, pp-mess-sim can select packet destination nodes from several common permutations, such as matrix-transpose (dimension-reversal), bit-complement, and bit-reversal, as shown in Table 4.2(b). Since these distributions depend on the underlying numbering scheme for each

```
class Net {
  unsigned int  total_nodes;                // Total number of nodes
  unsigned int  chan_per_link;              // Number of virtual channels per link
  unsigned int  diameter;                   // Diameter of the topology
  Dimension     edge_dimension;             // Dimension of the topology
  Direction     max_direction               // Number of links per node
  OffsetVec*    get_offset(NodeId,NodeId); // Compute relative address of a node
  NodeId**      hops;                       // Table for HopUniform distribution

  // Selection functions
  Direction*    dimension_order(), random_order(), min_congestion(), diagonal(),
                   nonmin_dim_order(), nonmin_random(), nonmin_min_congestion();

  // Neighbor and direction functions
  NodeId        neighbor_via_dir(NodeId, Direction), neighbor_via_dev(NodeId, DevId);
  DevId         neighbor_dev_via_vnet_and_dir(NodeId, VNet, Direction);
  DevId         neighbor_dev_via_dev(NodeId,DevId);
  Direction     dir_via_dev(DevId), reverse_dir_via_dir(Direction);

  // Destination node distributions
  NodeId        bit_reverse(NodeId), bit_complement(NodeId), node_uniform(),
                   dimen_reverse(NodeId), dimen_rotate(NodeId, unsigned int);
}
```

Figure 4.11: Internal components in the Net class

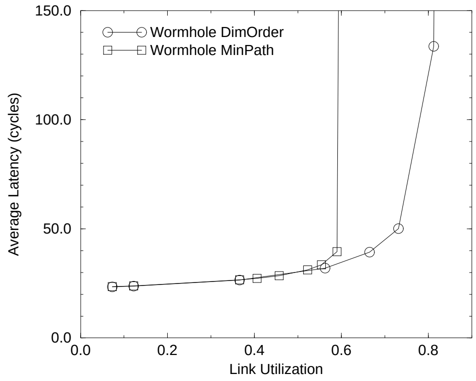
topology, the **Net** class includes functions to compute a packet's destination, based on the source node. For example, **Workload** can invoke **Net**'s `dimension_reverse()` function to return the `id` of the node whose dimension coordinates are the same as the given node, but in the reverse order. The **Net** module also includes a `hops` table, used by the **Workload** model to construct a the `hop_uniform` distribution.

4.5 Routing Experiments

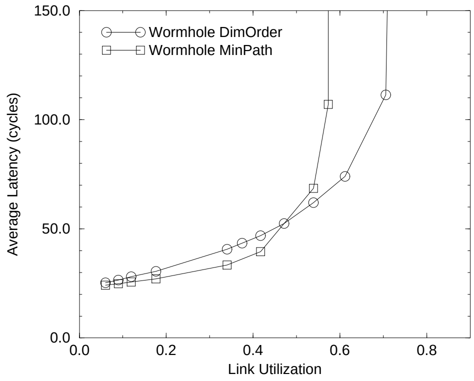
The flexibility of the v-router model and the pp-mess-sim environment facilitate a wide range of simulation experiments evaluating multicomputer router designs. This paper presents an experiment that demonstrates the utility of flexible routing by comparing oblivious and adaptive routing under two traffic patterns. A second experiment capitalizes on the simulator's flexibility to evaluate a network that supports two routing algorithms simultaneously. Chapter 3 and Chapter 5 include additional experiments evaluating routing and switching, respectively.

4.5.1 Routing Experiment

Traffic patterns significantly impact the performance of routing algorithms, as shown in Figure 4.12. This experiment evaluates an 8×8 square mesh of v-router **Nodes** carrying 16-flit packets. The plots compare the performance of oblivious and adaptive routing under wormhole switching and the dimension-ordered selection function; experiments with virtual cut-through switching show the same qualitative trends. The adaptive algorithm is a fully-adaptive minimal-path routing scheme that requires two virtual channels per link to prevent network deadlocks [44]; in these experiments, both routing algorithms employ a



(a) Poisson arrival process



(b) Bursty arrival process

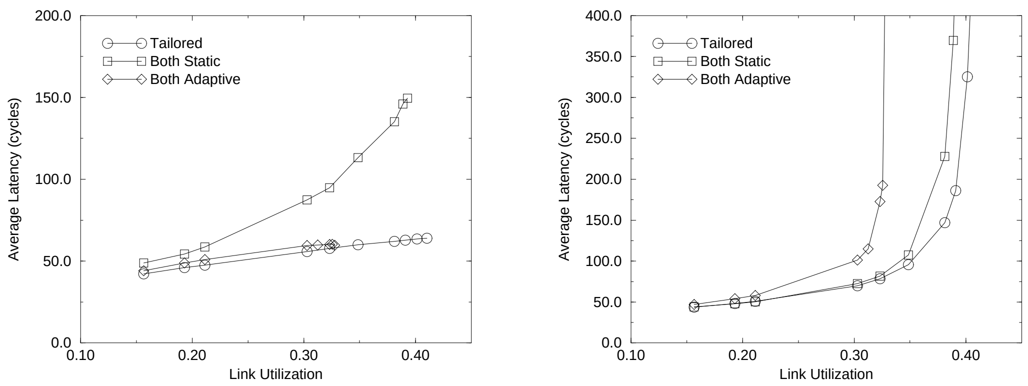
Figure 4.12: Comparing routing algorithms under wormhole switching and bit-complement traffic

pair of virtual channels to enable fair performance comparisons. The oblivious dimension-order routing algorithm uses the extra virtual channel to reduce contention between packets traveling on the same link [31].

Contrary to intuition, oblivious routing consistently outperforms adaptive routing in Figure 4.12(a). In an 8×8 square mesh, the bit-complement permutation requires source node (c, d) to communicate with node $(7 - c, 7 - d)$. As a result, all packets must eventually cross both the middle row and the middle column of the mesh, irrespective of the routing algorithm. Dimension-order oblivious routing tends to avoid the center of the network, where the middle row and column meet, by exhausting the x-direction before routing a packet in the y-direction. In contrast, adaptive algorithms try to avoid the heavily-congested middle column (or row) by routing packets to more lightly-loaded rows (or columns); this ultimately pushes traffic closer to the congested center of the network. A local decision at one node causes a packet to travel a lightly-loaded link into a more congested region.

In addition, the extra routing flexibility provided by adaptive algorithms allows source nodes to inject more packets, further increasing contention at the middle of the network. Hence, in some situations, restricted routing flexibility can effectively limit the overuse of network resources. However, this effect varies with the network load and the underlying traffic pattern, as shown in Figure 4.12(b). This experiment considers bursty traffic, in contrast to the traditional Poissonian packet arrival process in Figure 4.12(a). The source nodes generate bursty traffic using a two-stage normal distribution of packet interarrivals [57]. Packet interarrivals stem from two independent normal distributions, with different means, as shown in Table 4.2(a); sources randomly select 80% of interarrivals from the distribution with the small mean.

In Figure 4.12(b), the applied traffic load (x-axis) changes by varying the large mean, keeping the small mean fixed at 10 cycles. This generates relatively small packet interarrival times within a burst to capture the transmission of a multi-packet message or a handful of related messages. Figures 4.12(a) and (b) exhibit similar trends at high loads, but bursty traffic limits the effectiveness of static routing at low network loads since packets in



(a) Bit-reversal traffic (b) Bit-complement traffic

Figure 4.13: Average latency under traffic mixing

a burst are queued awaiting transmission. The adaptive algorithm helps dissipate bursts by capitalizing on multiple paths between each source and destination, thus reducing the queueing delay at the sending node.

4.5.2 Tailoring Experiment

The simulator's **Ralg** instructions and **Net** selection functions enable multi-factor experiments that study the interaction of routing algorithms and selection functions [50, 105]. For routers which support multiple routing schemes, these results can serve as a guide for selecting an appropriate routing algorithm based on the application workload. In a multi-user environment, where multiple applications execute at the same time, supporting multiple routing schemes *simultaneously* can significantly improve performance. For example, Figure 4.13 plots results from an experiment that mixes bit-reversal and bit-complement traffic in an 8×8 square mesh with wormhole switching. To protect the traffic classes from each other, the network devotes two virtual channels to each traffic class on each link; this ensures that heavy load in one class does not deny service to packets in the other class.

The graphs show average packet latency for both traffic patterns, under increasing bit-complement load; the bit-reversal pattern remains fixed at a link load of 0.12. As shown in Figure 4.13(a), the bit-reversal traffic has poor performance when both tasks are forced to use the static routing algorithm. Bit-reversal performance improves significantly when both tasks employ diagonal minimal-path routing, but this configuration degrades the bit-complement performance, as shown in Figure 4.13(b). The bit-complement traffic has low average latency under static dimension-ordered routing, independent of the algorithm assigned to the bit-reversal traffic. The network performs best when it tailors the routing policies to the application traffic patterns.

The cooperation between the pp-mess-sim components facilitates experiments that permit multiple routing-switching schemes to coexist in the underlying network. The **Ralg** support enabled packets to invoke different routing and switching schemes while the **Workload** module generated the communication patterns and collected the performance statistics. Run-time construction of the internal **Node** policies enabled the v-router to vary the number of virtual channels and the routing algorithms, while the **Net** transparently handled the details of the square-mesh topology. This synergy results in a flexible and extensible environment for specifying and evaluating modern multicomputer router architectures.

4.6 Conclusions and Future Work

Evaluating multicomputer router designs requires a flexible simulation framework. The object-oriented pp-mess-sim environment provides a toolkit for studying different network topologies, routing-switching policies, and router models, under a variety of communication workloads. Although every router design implements its internal policies in different ways, each device proceeds through common operations to service an incoming packet. The v-router model decouples these phases to allow pp-mess-sim experiments to independently vary the internal routing, switching, queueing, and arbitration policies. This chapter presents several research contributions:

- *Simulation of flow-control and arbitration policies:* The v-router **Node** module introduces useful abstractions for representing flow control and resource arbitration policies in router architectures. The v-router models the flow control between router components using a *wake-up queue* interface, hiding the internal details of each module. This is particularly useful for modeling wormhole switching, which requires a blocked packet to wait for an outgoing virtual channel to become available. Similarly, the v-router supports multiple link arbitration policies by representing shared resource with an arbiter model that can register pending simulation events; a separate handler invokes an arbitration policy to determine which event(s) to activate.

- *Routing-switching instructions:* To decouple network policies from the router model, the **Ralg** module represents routing-switching algorithms as a sequence of instructions, independent of the timing details of the **Node** model. This policy-mechanism split facilitates the development of new routing-switching algorithms, as well as fair comparisons between different router models, as discussed in Section 4.3.

- *Task model and workload composition:* The simulator introduces a novel task construct that associates routing-switching policies and performance metrics with the underlying traffic patterns, instead of the router model. To represent the communication characteristics of multicomputer applications, tasks can invoke a variety of distributions for selecting packet lengths, interarrival times, and destination nodes, as described in Section 4.4.1. Flexible task specification and mapping, combined with diverse traffic models, enable pp-mess-sim to impose a wide range of communication patterns on the simulated network.

- *History-list data collection:* To support a wide variety of performance metrics, the simulator introduces an extensible mechanism for accumulating packet statistics for each task, as described in Section 4.4.1. With basic support from the **Node** model, the **Workload** module maintains a *history list* that records significant events during a packet's journey. When a packet arrives at its destination node, the data collection routine processes the list to extract the desired performance metric(s) for the task.

- *Performance evaluation of router architectures and network policies:* The v-router model and the pp-mess-sim environment enable a broad range of experiments that evaluate multicomputer router designs. As a result, several studies have used pp-mess-sim to evaluate **Node** models under various routing-switching schemes, network topologies, and application workloads. In particular, recent research work exploits the simulator's ability to model networks that support multiple routing-switching schemes simultaneously [48–50, 103, 104, 107]. In addition to experiments with the v-router **Node** model [50, 102, 104, 107], pp-mess-sim has been instrumental in evaluating the Programmable Routing Controller [42, 48, 49, 103, 111], described further in Chapter 5.

The simulator's flexibility and extensibility stem from a careful definition of the **Net**, **Workload**, **Ralg**, and **Node** modules, as shown in Figure 4.1. These components encapsulate important design parameters in multicomputer networks, while the interfaces represent the subtle interaction between network topologies, communication workloads and performance metrics, routing-switching algorithms, and particular router models. The four main components, and their well-defined interfaces, result in an extensible environment that enables independent enhancements to the simulator. Capitalizing on this flexibility, the next chapter investigates router architectures that tailor routing-switching policies and arbitration schemes to address the requirements of emerging real-time applications.

CHAPTER 5

SWITCHING POLICIES IN REAL-TIME MULTICOMPUTERS

Although multicomputer router design has traditionally emphasized providing low-latency communication, modern parallel applications require additional services from the interconnection network [24, 27]. Multimedia and real-time applications, such as scientific visualization and process control, necessitate control over throughput, worst-case latency, and delay variance (jitter) [51, 122]. While *time-constrained* traffic necessitates deterministic or probabilistic bounds on throughput or end-to-end delay, *best-effort* service often suffices for the remaining traffic. For example, control or audio/video messages may mandate explicit performance guarantees, while data transfer may tolerate delay variability in exchange for improved average latency.

Handling this mixture of disparate traffic classes affects the suitability of architectural features in multicomputer routers. While the router alone cannot satisfy application performance requirements, design decisions should not preclude the system from providing necessary guarantees. Servicing time-constrained traffic requires control over network access time and bandwidth allocation, so the router should bound the influence best-effort packets have on these parameters. The software, or even the hardware, can then utilize these bounds to satisfy quality-of-service requirements through packet scheduling and resource allocation for communicating tasks. Additionally, the design should not unduly penalize the performance of best-effort packets.

Research in networking considers techniques for the effective mixing of multiple traffic classes in a communication fabric [5, 9]. However, the design trade-offs for parallel machines differ significantly from those in a heterogeneous, distributed environment. The shorter, wider communication links in most parallel machines result in much lower packet transmission delays, compared to distributed systems. These low-latency channels broaden the spectrum of flow-control schemes that can be implemented efficiently. Although multicomputer routers can employ low-level flow control and cut-through switching schemes to reduce *average* delay, these techniques often impinge on control over packet scheduling and bandwidth allocation. Other multicomputer router features, such as FIFO queueing and adaptive routing, further complicate the effort to provide predictable or guaranteed service.

Using the v-router model and pp-mess-sim environment from Chapter 4, this chapter investigates how switching schemes affect the network's ability to accommodate the performance requirements of time-constrained and best-effort traffic. With a careful selection of routing-switching policies, coupled with fine-grain link arbitration, multicomputer routers can provide low average latency for best-effort packets without compromising the pre-

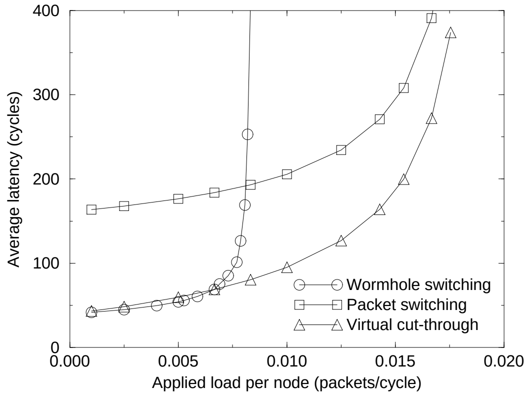


Figure 5.1: Average packet latency

dictability of time-constrained communication [103, 107]. To realize this scheme, the chapter considers the design and evaluation of a Programmable Routing Controller [38, 42, 103] that can implement packet switching for time-constrained traffic and wormhole switching for best-effort packets, with separate virtual channels for each traffic class. Additional experiments with the v-router model consider the utility of priority-based link arbitration to further insulate time-constrained traffic from best-effort packets.

5.1 Evaluation of Switching Schemes

In defining how packets flow through the network, the various switching schemes use different resources at nodes along a packet's route. This section evaluates the ability of wormhole, virtual cut-through, and packet switching to meet different performance requirements in multicomputer routers. Each switching scheme is best-suited for certain traffic classes with particular characteristics and performance requirements. To effectively support multiple traffic classes, the router should bound both network access time and the service rate for time-constrained packets. These bounds provide necessary abstractions for the scheduling and mapping of communicating tasks. Best-effort packets, on the other hand, may forego these restrictions in exchange for lower latency and reduced buffer requirements.

5.1.1 Average Latency

The usage of memory and link resources determines both average packet latency and the influence an in-transit packet can have on other network traffic. Figure 5.1 shows the average end-to-end packet latency for the three switching schemes as a function of the packet injection rate. Using pp-mess-sim's v-router model, the experiment evaluates an 8×8 torus (8-ary 2-cube) network with dimension-ordered routing, where each node generates 16-flit packets with exponentially distributed interarrival times and uniform random selection of destination nodes. Virtual cut-through and packet switching utilize one virtual channel for each physical link and store buffered packets in output queues in the router. Wormhole packets employ deadlock-free routing on a pair of virtual channels [29] with demand-driven, round-robin arbitration amongst the virtual channels; each virtual channel can hold a single flit pending access to the output link.

Even with this small amount of memory resources, wormhole switching performs well at low loads, slightly outperforming virtual cut-through switching. At high loads, virtual cut-through and packet switching performance gradually merge, since high network utilization decreases the likelihood that an in-transit packet encounters an idle output link. By removing blocked packets from the network, virtual cut-through and packet switching consume network bandwidth proportional to the offered load. In contrast, a blocked wormhole packet stalls in the network, effectively dilating its length until its outgoing channel becomes available. As a result, wormhole networks typically utilize only a fraction of the available network bandwidth [31, 86], as seen by the early saturation of the wormhole plot in Figure 5.1. At higher loads, this effect enables packet switching to outperform wormhole switching, even though packet switching introduces buffering delay at each hop in a packet's route.

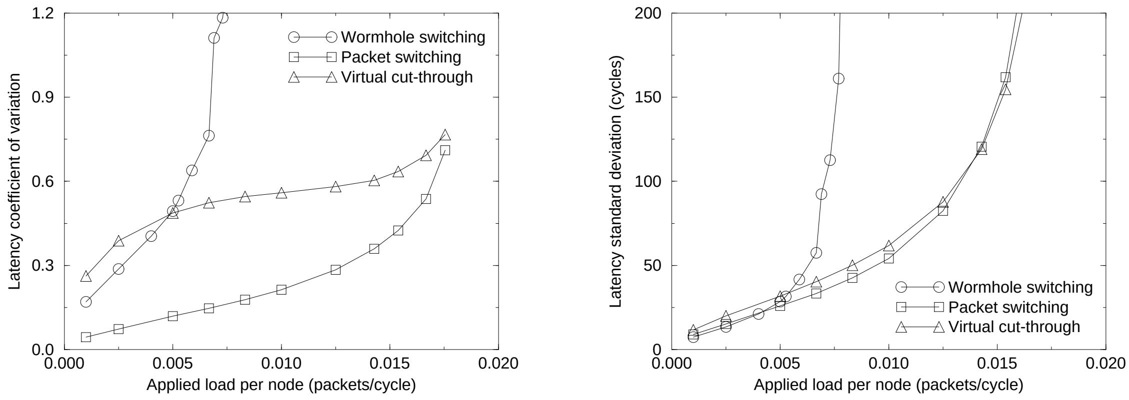
Although including additional virtual channels can improve the throughput of a wormhole network [31], channel contention still creates dependencies amongst packets spanning multiple nodes. The sensitivity of wormhole networks to slight changes in load, including short communication bursts [32], complicates the use of wormhole switching for time-constrained traffic. Still, wormhole switching is particularly well-suited to best-effort packets, due to its low latency and minimal buffer space requirements. While flow-control costs limit the utility of wormhole switching in distributed systems, parallel machines can dynamically transfer or stall wormhole flits without complicating buffer allocation for other traffic. Section 6.1 describes how, with effective flow-control and arbitration schemes, best-effort packets can employ wormhole switching without compromising the performance of the guaranteed traffic.

5.1.2 Predictability

While the router should provide low average latency for best-effort packets, guaranteed communication requires predictable network delay and throughput. Figure 5.2(a) shows the coefficient of variation for packet latency for the three switching schemes, where the coefficient of variation measures the ratio of the standard deviation to the mean [61]. Since latency characteristics vary depending on the distance between source-destination pairs, the graph shows results only for packets traveling a fixed distance in the network. While each source generates traffic with uniform random selection of destination nodes, data collection for Figure 5.2 includes only packets traveling exactly five hops. Hence, the plots illustrate the *variability* in end-to-end latency, indicating the potential for jitter in a stream of packets with a common source and destination.

Across all loads, packet switching incurs the least variability since packets deterministically buffer at intermediate nodes. Coupled with static routing, a packet-switched transfer utilizes deterministic memory and channel resources at fixed nodes and links along the route. This greatly simplifies the allocation and scheduling of resources throughout the interconnection network. In contrast, virtual cut-through switching imparts variable load on memory resources at intermediate nodes by basing the buffering decision on the status of the output links. At high loads, virtual cut-through and packet switching merge, as in Figure 5.1, due to the decreasing likelihood of packet cut-throughs.

Wormhole switching, though conceptually similar to virtual cut-through, has quite different characteristics. Since a blocked wormhole packet never buffers, it imparts no memory demands on intermediate nodes, but instead consumes unpredictable amounts of channel bandwidth. In Figure 5.2(a), wormhole latency variation increases dramatically with rising



(a) Coefficient of variation (b) Standard deviation

Figure 5.2: Variability of packet latency (5-hop packets)

load, even under a moderate injection rate below the saturation throughput. Below the saturation load, wormhole switching results in a low average latency, as seen in Figure 5.1, but a portion of the traffic incurs much larger delay due to pockets of channel contention and the small amount of buffer resources. In addition to a large coefficient of variation, wormhole traffic suffers a large standard deviation of packet latency, as shown in Figure 5.2(b).

Depending on the number of active virtual channels at each link, flits within a single wormhole packet may encounter different service rates. Demand-driven arbitration for access to the physical links, while important for low average latency, complicates the effort to export a predictable flit or packet service rate to a static or run-time scheduling algorithm. Although adding virtual channels can reduce contention [31], and hence average latency, additional virtual channels also increase the potential variability in the number of flits awaiting access to each physical link, further complicating the flit service rate. Hence, although wormhole switching can provide low average latency at low cost, bounding worst-case delay for time-constrained packets requires additional mechanisms.

5.1.3 Packet Scheduling

The router must have control over packet scheduling and bandwidth allocation to ensure that time-constrained packets meet their latency and bandwidth requirements. Virtual cut-through and packet switching generate *physical* queues in each node, facilitating priority-based scheduling amongst competing packets. In contrast, stalled wormhole packets form *logical* queues spanning multiple nodes. These decentralized queues complicate packet scheduling. Still, a wormhole router can influence resource allocation through its virtual channel reservation and arbitration policies. Priority assignment of virtual channels to incoming packets improves predictability; adaptive arbitration policies can further reduce variability by basing flit bandwidth allocation on packet deadlines or priority [31, 77, 78, 120].

While assigning priorities to virtual channels provides some control over packet scheduling, this ties priority resolution to the number of virtual channels. If packets at different priority levels share virtual channels, the application must account for blocking time when a lower priority packet holds resources needed by higher priority traffic. Although adding more virtual channels can improve priority resolution, this also incurs increased latency overhead and implementation complexity for the router [4]. In addition, the router must enforce the multiple priority levels at its injection and reception ports to avoid unpredictable stalling at the network entry and exit points.

Providing separate buffers for each priority level is effective for coarse-grain priority assignment, but this approach incurs significant cost for fine-grain resolution. With *packet* queues at each node, the router can effectively utilize fine-grain priorities, such as deadlines, to assign access to output links [5, 63]. By buffering packets at each node, packet switching enables the router to schedule traffic to provide latency or bandwidth guarantees [63], as shown by the router architecture in Chapter 6. For example, suppose a time-constrained packet enters an intermediate node well in advance of its deadline. The router may wish to detain this packet, even if its outgoing link is available, to avoid unexpectedly overloading the subsequent node. The next section considers mechanisms for supporting packet switching for time-constrained traffic, while permitting best-effort packets to capitalize on low-cost, low-latency cut-through switching schemes.

5.2 Router Architectures for Traffic Mixing

Best-effort and time-constrained traffic have conflicting performance goals that complicate interconnection network design. The effective mixing of time-constrained and best-effort traffic hinges on controlling the interaction between these two classes. In particular, best-effort packets cannot consume arbitrary amounts of link or buffer resources while time-constrained packets await service. Fine-grain arbitration between the traffic classes allow time-constrained and best-effort packets to share network bandwidth without sacrificing the performance of either class.

5.2.1 Tailoring Switching Schemes

As seen in Section 5.1, wormhole and packet switching exercise complementary resources in the interconnection network, with wormhole switching reserving virtual channels and packet switching consuming buffers in the router. Hence, the combination of wormhole switching for best-effort traffic and packet switching for time-constrained communication enables an effective partitioning of router resources. However, since the traffic classes share network bandwidth, the router must regulate access to the physical links to control the interaction between the two classes. Otherwise, a blocked wormhole packet can delay the advancement of time-constrained traffic, even when the time-constrained traffic does not share any links with the stalled packet.

The router can regulate the interaction between traffic classes by assigning best-effort and time-constrained packets to separate logical networks. In this approach, the router divides each physical link into multiple virtual channels, where some virtual channels carry best-effort packets and the rest accept only time-constrained traffic. Virtual channels provide an effective mechanism for reducing the interaction between packets while still allowing traffic to share network bandwidth [30, 35, 43, 73, 95]. Exporting the virtual channel abstraction to the injection and reception ports further prevents intrusion between packets at the network entry and exit points [43, 67, 95]. This ensures that time-constrained and best-effort traffic have separate logical resources through the entire path from the source to the destination node.

Virtual networks, coupled with appropriate policies for each traffic class, enable the router to limit the resources consumed by best-effort communication to ensure sufficient

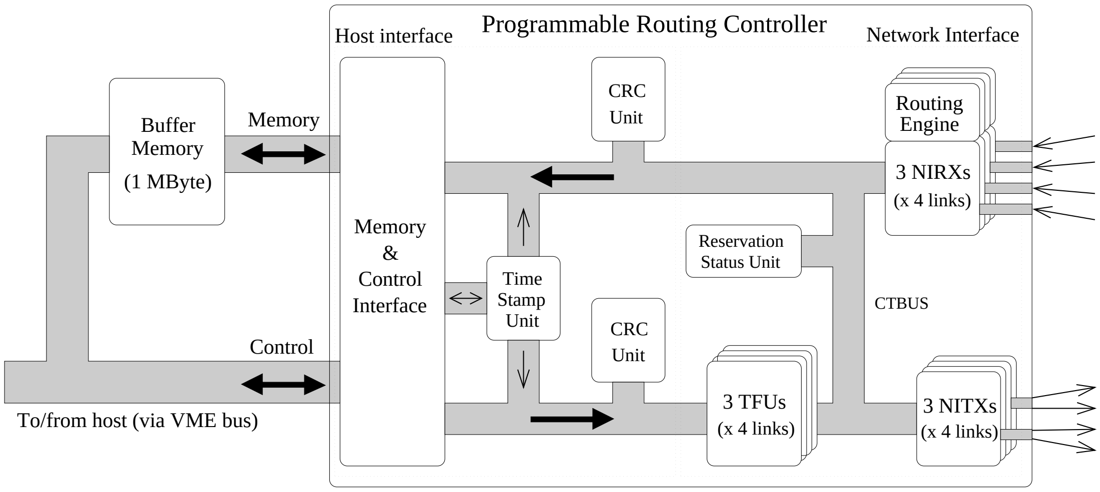


Figure 5.3: Programmable Routing Controller

buffer space and link bandwidth for time-constrained packets. By tailoring the routing, switching, and flow-control policies for each virtual network, multicomputer routers can support traffic classes with conflicting performance requirements. Packets on separate virtual networks interact only to compete for access to the physical links and ports. Under fine-grain multiplexing of virtual channels, this bounds network access time for time-constrained packets, independent of the amount or length of best-effort packets. The communication software, or hardware, can then build on these underlying abstractions to provide various services, such as connection-oriented communication with latency or bandwidth guarantees. Fine-grain flow control on the wormhole virtual network enables best-effort flits to capitalize on slack link bandwidth left unclaimed by time-constrained packets.

5.2.2 Programmable Routing Controller

With proper hardware support, real-time systems can capitalize on multicomputer switching schemes and flow-control policies to accommodate the performance requirements of time-constrained and best-effort traffic. For example, the Programmable Routing Controller (PRC) [37, 38, 42, 43, 103], shown in Figure 5.3, is designed to implement programmable routing-switching schemes for best-effort traffic, while facilitating host control over scheduling and resource allocation for time-constrained communication. Designed to reside on the host processor's private memory bus, the PRC has direct access to a packet buffer and provides the host with a memory-mapped control interface. The PRC architecture has been implemented as an application-specific integrated circuit using the Verilog hardware description language and the Epoch silicon compiler [37, 38].

The router coordinates bidirectional communication with up to four neighboring nodes, with three virtual channels on each unidirectional link, with transparent support for timestamping and CRC (cyclic redundancy code) error detection. The PRC exploits concurrency amongst the virtual channels and provides fair, fine-grain arbitration at the memory and network interfaces. The twelve NITXs (network-interface transmitters) provide low-level control of packet transmission, while the twelve NIRXs (network-interface receivers) coordinate packet reception. The host transmits a packet by feeding page tags to one of the twelve TFUs (transmitter fetch units), where each page tag includes a memory address and the number of words to transmit. Similarly, the host processor supplies each NIRX with pointers to free pages in the buffer memory for use by arriving packets. The network interface components communicate over the CTBUS (cut-through bus), a demand-slotted, time-divisioned multiplexed bus that provides fair service to the incoming and outgoing virtual channels.

For flexibility in selecting network policies, each incoming link has a dedicated programmable routing engine for implementing routing-switching policies for in-transit traffic. The PRC treats the outbound virtual channels (NITXs) as individually reservable resources, allowing the device to support a variety of routing and switching schemes through flexible control over channel allocation policies. Upon receiving the header bytes of an incoming packet, the routing engine decides whether to buffer, stall, forward, or drop the packet. The microprogrammable routing engine bases its routing-switching decision on the incoming virtual channel, the arriving header, and prevailing network conditions. By downloading different microcode routines for each NIRX, the routing engines can tailor the low-level communication policies of each virtual channel to address the requirements of best-effort and time-constrained traffic.

5.3 Performance Evaluation of Traffic Mixing

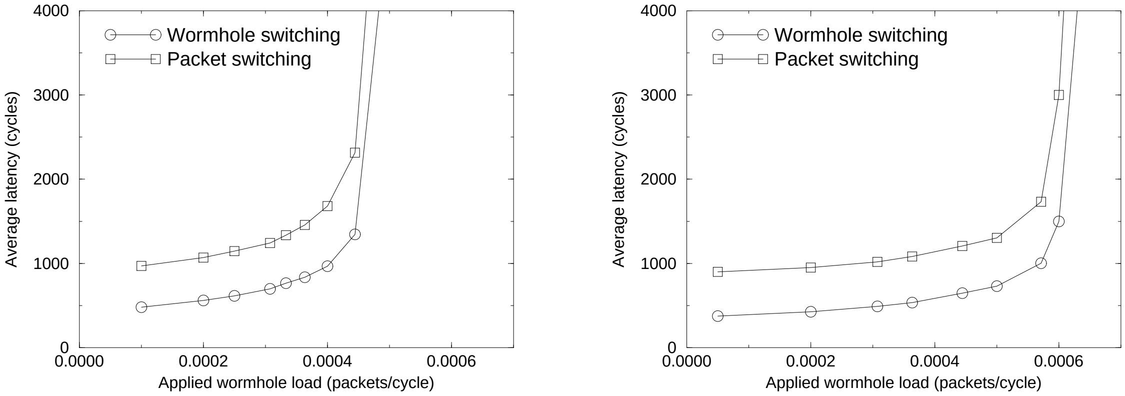
The pp-mess-sim environment includes a cycle-level model of the PRC that captures the details of flow control, resource arbitration, and microcode execution [102, 104]. Experiments with the PRC model show that segregating best-effort wormhole traffic from time-constrained packet-switched traffic can accommodate the performance requirements of both classes. Additional experiments with the v-router model consider the effects of using priority arbitration to improve the predictability of the time-constrained communication.

5.3.1 Traffic Mixing on the PRC

Figure 5.4 shows the interaction of time-constrained and best-effort packets in an 8×8 square mesh of PRCs. Both traffic classes generate 16-flit packets with a node-uniform distribution of destination nodes and dimension-ordered routing. The best-effort packets have exponentially distributed interarrival times, while time-constrained packets are generated in a periodic fashion. Figure 5.4 plots the average latency of the best-effort and time-constrained traffic when both classes share a single virtual channel on each physical link. The graphs plot the average latency for both traffic classes as a function of best-effort traffic load for a fixed injection rate for time-constrained packets; Figure 5.4(a) and Figure 5.4(b) consider time-constrained traffic with periods of 1000 and 1500 flit cycles, respectively. The increase in best-effort load has deleterious effects on the time-constrained traffic, since stalled best-effort packets block the forward progress of time-constrained packets.

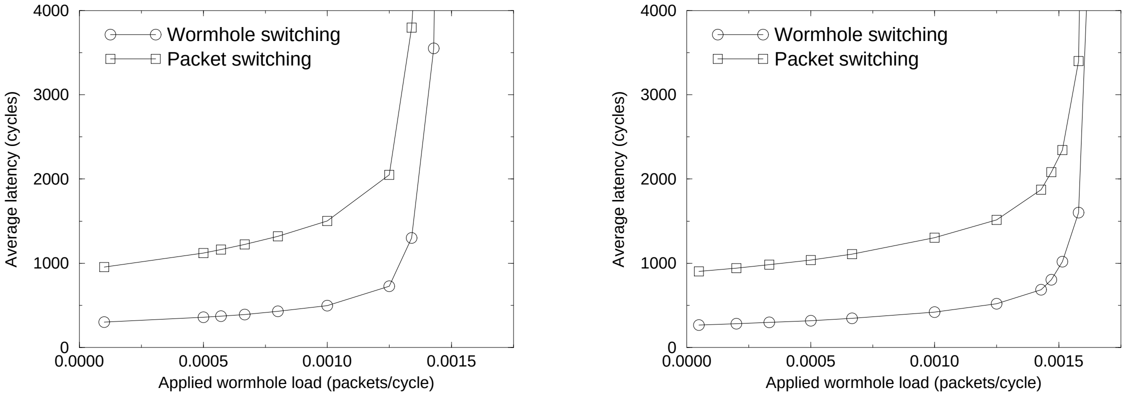
The intrusion of the best-effort traffic is particularly noticeable in Figure 5.7, which plots the standard deviation of packet latency for the time-constrained traffic as a function of the best-effort wormhole load. The router can improve performance by increasing the number of virtual channels on each link, as shown in Figure 5.5. With a pair of virtual channels, both the best-effort and time-constrained traffic can achieve higher throughput and lower delay variance, since the additional virtual channel provides extra flexibility for bypassing blocked best-effort packets. As a result, both traffic classes achieve a much higher peak throughput in Figure 5.5 than in Figure 5.4. Still, under heavy best-effort load, the time-constrained traffic has high average latency and delay variance.

In contrast, time-constrained packets have much better performance under heavy best-



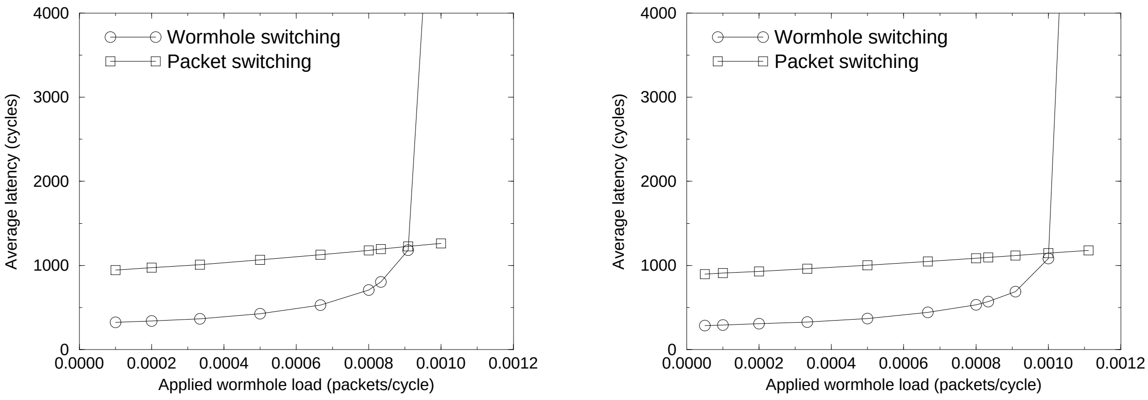
(a) Packet-switching period of 1000 cycles (b) Packet-switching period of 1500 cycles

Figure 5.4: Average latency of time-constrained and best-effort traffic sharing a single virtual channel on each link



(a) Packet-switching period of 1000 cycles (b) Packet-switching period of 1500 cycles

Figure 5.5: Average latency of time-constrained and best-effort traffic sharing two virtual channels on each link



(a) Packet-switching period of 1000 cycles (b) Packet-switching period of 1500 cycles

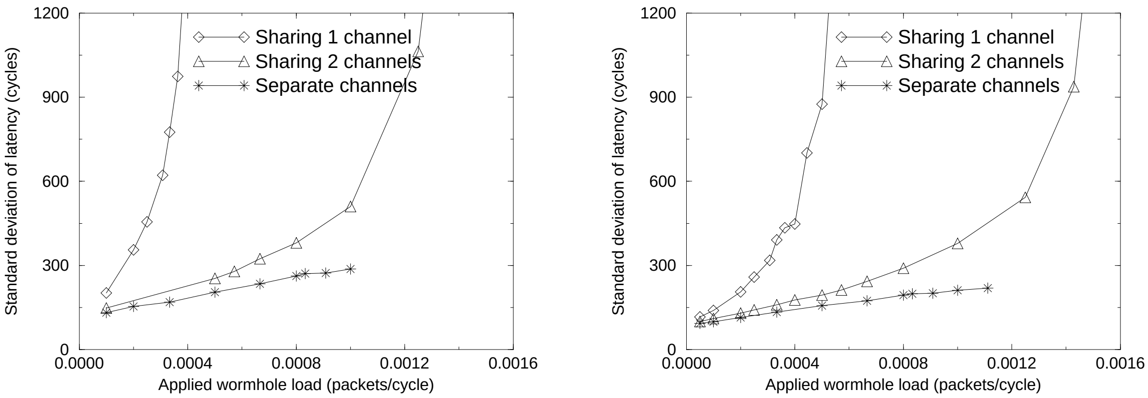
Figure 5.6: Average latency of time-constrained and best-effort traffic on separate virtual channels on each link

effort load when the network partitions the traffic classes onto separate virtual channels, as shown in Figure 5.6. In this configuration, channel contention on the best-effort virtual network does not impede the forward progress of time-constrained packets, since blocked wormhole packets temporarily stall in their own virtual network instead of depleting physical link or buffer resources. This permits the time-constrained packets to have low average latency and delay variance, even when the best-effort virtual channels are saturated. However, segregating access to the virtual channels can hurt best-effort performance, as shown by the lower peak throughput for the wormhole traffic in Figure 5.6, relative to Figure 5.5. Still, separating the traffic classes significantly improves the predictability of the time-constrained communication, while permitting best-effort traffic to capitalize on any excess link bandwidth.

5.3.2 Tighter Bounds for Time-Constrained Packets

Although the separate virtual networks limit the interaction between the traffic classes, the arbitration for access to the physical link still permits an active best-effort virtual channel to increase delay for time-constrained packets. In Figure 5.6 and Figure 5.7, this is seen by the increase in the mean and standard deviation of latency for packet-switched traffic in the presence of a heavier load of wormhole traffic; for example, in the bottom curve in Figure 5.7(b), the standard deviation varies from 92.7 cycles (under low best-effort load) to a high of 218.6 cycles (when the best-effort virtual network is saturated). More significantly, round-robin arbitration amongst the virtual channels varies the service rate for the time-constrained packets; in the worst case, time-constrained traffic receives only half of the link bandwidth.

The router can further minimize the intrusion on time-constrained traffic by imposing *priority arbitration* between the virtual networks, where time-constrained packets always receive service ahead of best-effort packets. For a time-constrained packet, this effectively provides flit-level *preemption* of best-effort traffic across its entire path through the network. In contrast to the results in Figures 5.4– 5.7, assigning priority to time-constrained traffic



(a) Packet-switching period of 1000 cycles (b) Packet-switching period of 1500 cycles

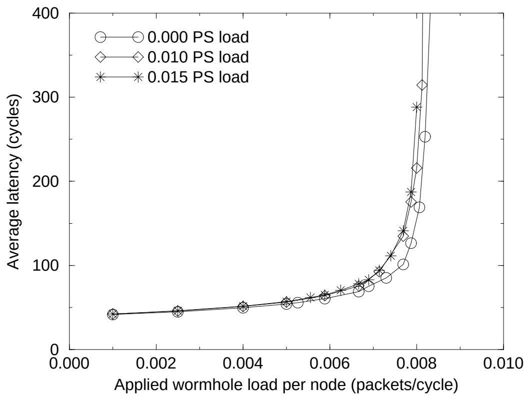
Figure 5.7: Standard deviation of packet latency for time-constrained traffic

removes any sensitivity to the best-effort load. As a result, a time-constrained packet travels at the same rate through each link in its journey, independent of the number of active best-effort virtual channels. Building on this abstraction, a higher-level scheduling algorithm can allocate resources based only on the worst-case requirements of the time-constrained traffic, while still allowing best-effort traffic to dynamically consume any unused link bandwidth.

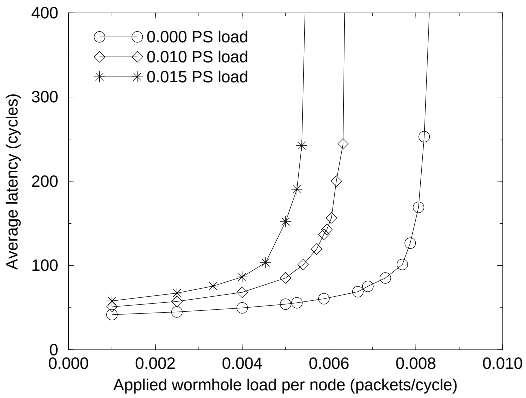
However, priority arbitration can exact a heavy toll on the best-effort packets, particularly at higher loads, as illustrated by Figure 5.8 which evaluates an 8×8 torus of v-router nodes carrying 16-flit packets with separate virtual channels for the two traffic classes. This graph shows the average latency of best-effort wormhole packets in the presence of three different packet-switching (PS) injection rates under both round-robin and priority arbitration for the physical links. In contrast to Figure 5.8(a), Figure 5.8(b) shows significant degradation in the performance of best-effort packets, since the strict priority-based scheme restricts the forward progress of wormhole traffic. Even in the absence of livelock, lengthy blocking of wormhole flits increases contention and delay in the best-effort virtual network. Chapter 6 addresses this problem by permitting best-effort traffic to claim link bandwidth *ahead* of some time-constrained traffic, as long as each time-constrained packet is still guaranteed to receive service by its deadline.

5.4 Conclusions and Future Work

Emerging parallel real-time and multimedia applications impose diverse communication requirements on multicomputer interconnection networks. The conflicting performance goals of best-effort and time-constrained traffic affect the suitability of routing, switching, and flow-control schemes. Traditionally, real-time systems have employed packet switching, coupled with packet scheduling algorithms, to achieve predictable communication performance; however, in tightly-coupled parallel machines, this approach unduly penalizes the best-effort traffic. As shown in this chapter, low-level control over routing and switching, coupled with fine-grain arbitration, enables multicomputer routers to effectively mix time-constrained and best-effort communication. This chapter introduces several research contributions:



(a) Round-robin arbitration



(b) Priority arbitration

Figure 5.8: Average wormhole latency under different packet switching loads

- *Characterization of switching schemes:* The simulation experiments and discussion in Section 5.1 characterize wormhole, virtual cut-through, and packet switching, in terms of their ability to support the performance requirements of best-effort and time-constrained communication. Packet switching, combined with static routing, consumes predictable bandwidth and buffer resources, making the scheme well-suited to time-constrained traffic. In contrast, wormhole packets can stall in the network, consuming an unpredictable amount of link bandwidth while blocking the advancement of other traffic. Still, the small average latency and minimal buffer requirements make wormhole switching ideal for best-effort communication.

- *Traffic mixing with virtual networks:* To address the conflicting performance requirements of best-effort and time-constrained communication, Section 5.2 proposes a scheme that permits best-effort traffic to employ wormhole switching, without compromising the predictability of time-constrained communication. By separating best-effort and time-constrained traffic onto separate virtual channels, a router can insulate time-constrained traffic from the contention between best-effort packets. Fine-grain, demand-driven arbitration for each link and injection/reception port, ensures that both traffic classes can capitalize on the available bandwidth resources.

- *Evaluation of router arbitration policies:* The Programmable Routing Controller, with its flexible support for multiple routing-switching schemes, provides an effective platform for evaluating the proposed scheme. The experiments in Section 5.3 demonstrate the benefits of assigning best-effort wormhole traffic and time-constrained packet-switched traffic to separate virtual channels. With priority arbitration, the router can completely insulate time-constrained packets from the best-effort communication, at the expense of increasing average latency for the best-effort traffic. Chapter 6 addresses this limitation by allowing best-effort traffic to receive service ahead of some time-constrained packets, when possible.

The effective mixing of best-effort and time-constrained traffic requires a combination of low-level hardware support and higher-level protocols. This chapter has investigated effective switching schemes that enable the development of such higher-level protocols. Effective arbitration and flow-control policies enable the router to export bounded values for network access delay, packet service time, and throughput for time-constrained traffic, even in the presence of best-effort flits. Hardware or software protocols can then build on these abstractions to allocate communication resources and schedule time-constrained packets. The next chapter presents a router architecture that integrates low-level routing-switching policies with packet scheduling to provide end-to-end performance guarantees for time-constrained traffic.

CHAPTER 6

REAL-TIME ROUTER ARCHITECTURE

Time-constrained and best-effort traffic have conflicting performance goals that complicate network design. To improve predictability for time-constrained packets, router architectures can isolate the two traffic classes and prioritize access to network resources, as discussed in Chapter 5, However, this can significantly degrade the average performance of best-effort packets and does not necessarily provide explicit end-to-end delay guarantees for time-constrained packets. Ultimately, bounding worst-case latency requires prior reservation of link and buffer resources, based on the application's anticipated traffic load [5, 125, 126]. Under this traffic contract, the network can provide end-to-end performance guarantees through effective link-scheduling and buffer-allocation policies. This chapter presents a real-time router design that handles a wide range of throughput and delay requirements by implementing the *real-time channel* [63] abstraction for packet scheduling.

A real-time channel is a unidirectional virtual connection between two nodes, with a source traffic specification and end-to-end delay bound; separate parameters for delay and bandwidth permit the model to accommodate a wider range and larger number of connections than other disciplines [125], at the expense of increased implementation complexity. At run-time, the network guarantees end-to-end performance through a combination of bandwidth regulation and deadline-based packet scheduling at each link, as shown in Figure 6.1. When a connection temporarily exceeds its traffic contract, the router delays the early time-constrained packets to avoid buffer overflow at the next node in the route. In addition, bandwidth regulation of the time-constrained traffic permits best-effort packets to access the link ahead of any *early* time-constrained traffic. This can significantly improve average best-effort performance without compromising the worst-case latency of the time-constrained packets.

Implementing bandwidth regulation and deadline-based scheduling in software would impose a significant burden on the processing resources at each node and would prove too slow to serve multiple high-speed links. This software would have to sort packets by deadline for each outgoing link, in addition to scheduling and executing application tasks. With high-speed links and tight timing constraints, real-time parallel machines require hardware support for communication scheduling. An efficient, low-cost solution requires a design that integrates this run-time scheduling with packet transmission. Hence, we present a *chip-level* router design that handles run-time packet scheduling, while relegating non-real-time operations (such as admission control and route selection) to the protocol software. In contrast to existing designs, the real-time router tailors network routing, switching, arbitration, and flow-control policies to the conflicting requirements of best-effort and time-

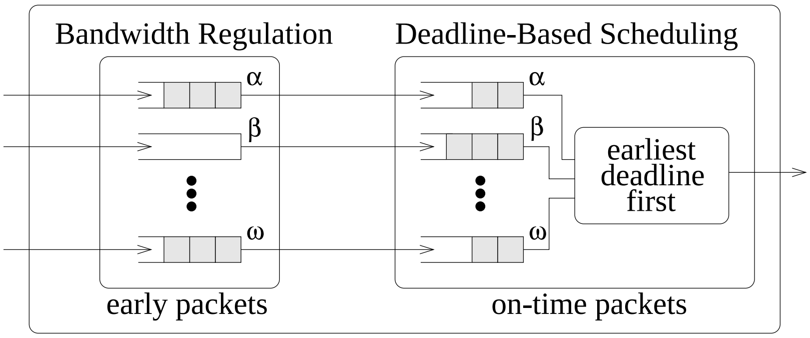


Figure 6.1: Bandwidth regulation and packet scheduling for connections $\alpha, \beta, \ldots \omega$

constrained traffic, as discussed in Section 6.1.

Section 6.2 describes the real-time channel model for communication in point-to-point networks, while Section 6.3 discusses the router's support for run-time scheduling of time-constrained packets. To reduce hardware complexity, the architecture shares packet buffers and sorting logic between the router's multiple output links. The router overlaps communication scheduling with packet transmission to maximize utilization of the network links. The design limits the complexity of the link scheduler by bounding the range of packet deadlines and handling the effects of clock rollover. Section 6.4 describes the router implementation, using the Verilog hardware description language and the Epoch silicon compiler. Verilog simulations demonstrate that the design satisfies the performance goals of both traffic classes in a single-chip solution. Section 6.5 concludes the chapter with a discussion of future research directions.

6.1 Mixing Best-Effort and Time-Constrained Traffic

Best-effort and time-constrained traffic have conflicting performance goals that complicate network design, as discussed in Chapter 5. As a result, the real-time router architecture has separate control and data path for the two traffic classes, as shown in Figure 6.2; solid lines denote the flow of packet data, while dashed lines indicate control information. To insulate the local processor from packet scheduling, the design has separate injection ports for time-constrained and best-effort traffic, while the router coordinates access to a shared reception port and the four outgoing links. Careful selection of router policies, coupled with fine-grain link arbitration, enables time-constrained and best-effort packets to share network bandwidth without sacrificing the performance of either class.

6.1.1 Switching

To ensure that time-constrained packets meet their delay requirements, the router must have control over bandwidth and memory allocation. In most real-time systems, time-constrained communication consists of 10–20 byte exchanges of command or status information [97]. Consequently, our design restricts time-constrained traffic to small, fixed-size packets, as shown in Table 6.1; this bounds network access latency and buffering delay while simplifying memory allocation in the router. To ensure predictable consumption of link and buffer resources, time-constrained traffic employs store-and-forward packet switching. By

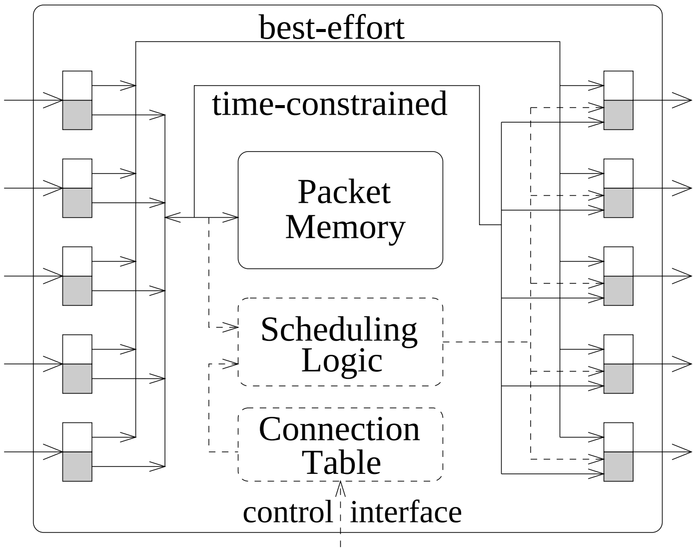


Figure 6.2: Real-time router architecture

buffering packets at each node, packet switching allows each router to independently schedule packet transmissions to satisfy per-hop delay requirements, as discussed in Chapter 5.

However, this approach unduly penalizes the performance of best-effort traffic. Most modern parallel machines employ cut-through switching schemes for lower latency and reduced buffer space requirements. In the real-time router, best-effort traffic employs wormhole switching switching for low latency and reduced buffer space requirements. Instead of storing entire best-effort packets at intermediate nodes, the router simply includes small flit buffers to hold a few bytes of a packet from each input link; inter-node flow control stalls further transmission of the packet until this buffer space is available. This permits best-effort traffic to use variable-size packets, to reduce or even avoid packetization overheads, without increasing buffer complexity in the router.

6.1.2 Arbitration

By cutting through intermediate nodes, best-effort packets can avoid unnecessary buffering delay. However, these wormhole packets can stall in the network for an unpredictable amount of time, delaying the advancement of other packets heading for different destinations. The effective mixing of time-constrained and best-effort traffic hinges on controlling the interaction between these two classes, as discussed in Chapter 5. In particular, best-effort packets should not consume arbitrary amounts of bandwidth resources while time-constrained packets await service. To control the interaction between the two traffic classes, the real-time router divides each link into two virtual channels; a single bit on each link differentiates between time-constrained and best-effort packets, as shown in Figure 6.3. Each link also includes an acknowledgement bit for flow control on the best-effort virtual channel.

In contrast to the PRC architecture in Section 5.2.2, the real-time router incorporates arbitration policies that address the performance requirements of best-effort and time-constrained traffic. Each wormhole virtual channel performs round-robin arbitration on the input links to select an incoming best-effort packet for service. By sequencing through the incoming links, round-robin arbitration ensures that arriving packets receives service in

	Time-Constrained	**Best-Effort**
Switching	Packet switching	Wormhole switching
Packet size	20 bytes	Variable length
Link arbitration	Deadline-driven	Round-robin on input links
Routing	Table-driven multicast	Dimension-ordered unicast
Buffers	Shared output queues	Flit buffers at input links
Flow control	Rate-based	Flit acknowledgements

Table 6.1: Architectural parameters in real-time router design

a fair and timely manner; in addition, round-robin schedulers are relatively simple to implement. For time-constrained traffic, the packet-switched virtual channel schedules *on-time* packets based on their deadlines, as discussed in Section 6.2. Priority arbitration amongst the virtual channels tightly regulates the intrusion of best-effort traffic on time-constrained packets. This effectively provides *flit-level* preemption of best-effort traffic whenever an on-time time-constrained packet awaits service, while permitting wormhole flits to consume any excess link bandwidth. The link transmits best-effort flits ahead of any *early* time-constrained packets.

6.1.3 Routing

As part of establishing a real-time channel, the network reserves link bandwidth and buffer space along a fixed path between the source and destination nodes; the chosen route depends on the resources available at various nodes and links in the network. Consequently, the real-time router maintains a routing table, indexed on the connection identifier of the arriving time-constrained packet, as shown in Figure 6.4(a); Section 6.3 describes how the controlling processor can edit this table as part of a connection establishment protocol. Since a node may wish to send information to a *collection* of destination nodes (i.e., multicast), the router can forward an incoming time-constrained packet to multiple outgoing links; this facilitates efficient, timely communication between a set of cooperating nodes.

In contrast, best-effort traffic does not require resource reservation along packet routes. Instead, the real-time router implements dimension-ordered routing, a shortest-path scheme that completely routes a packet in the x-direction before proceeding in the y-direction to the destination. Dimension-ordered routing avoids packet deadlock in a square mesh [29] and also facilitates an efficient implementation based on x and y offsets in the packet header, as shown in Figure 6.4(b); the offsets reach zero when the packet has arrived at its destination node. The router could improve best-effort performance by implementing *adaptive* wormhole routing, with additional virtual channels to avoid deadlock, at the expense of increased implementation complexity [4, 87]. In particular, non-minimal adaptive routing would enable best-effort packets to circumvent links with a heavy load of time-constrained traffic.

6.1.4 Buffer Architecture

The real-time router includes a packet memory for storing time-constrained traffic awaiting access to the outgoing links; in contrast, blocked best-effort packets stall in the network. The router queues time-constrained packets at the *output* ports to avoid the throughput limitations of input queueing [118]; this permits each output link to select a packet for trans-

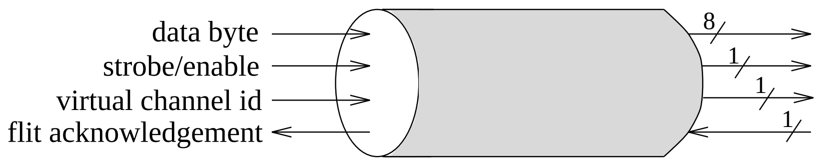


Figure 6.3: Link encoding in real-time router

mission amongst *all* time-constrained traffic buffered in the router. The reception port and four output links *share* a single packet memory to maximize usage of the available buffer space. To accommodate the aggregate bandwidth of the five input and five output ports, the router stores packets in 10-byte chunks, with demand-driven round-robin arbitration amongst the ports, as shown in Figure 6.5. As shown in Figure 6.2, each port includes nominal buffer space to avoid stalling the flow of data while waiting for bus access to the packet memory. Similarly, each port includes two small flit buffers to permit continuous transmission of wormhole packets in the absence of link contention.

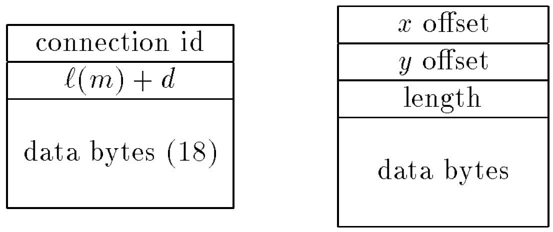
Similar to many shared-memory switches in high-speed networks [118], the router maintains a pool of unused memory locations to assign to arriving time-constrained packets. Initially, this *idle-address FIFO* includes every location in the memory. An incoming packet retrieves an address from this FIFO; upon packet departure, the router returns the location to the idle-address pool. To avoid buffer overflow or packet loss, a real-time channel reserves sufficient buffer slots at each node in its route, as described in Section 6.2. Although the output ports share a single packet memory, the connection establishment procedure can *logically* partition the memory by limiting the number of packet buffers dedicated to connections on each outgoing link; otherwise, one link could reserve the bulk of the memory slots, limiting the chance of establishing real-time channels on the other outgoing links. By implementing a physically shared memory, the router permits the protocol software to balance the trade-offs between buffer partitioning and complete sharing to enhance future channel admissability.

6.2 Real-Time Channels

Real-time communication requires reservation of bandwidth and buffer resources, coupled with packet scheduling at the network links. The *real-time channel* model [63] provides a useful abstraction for bounding end-to-end network delay for time-constrained packets, under certain application traffic characteristics, without compromising the performance of best-effort packets.

Traffic parameters: A real-time channel is a unidirectional virtual connection that traverses one or more network links. Since time-constrained communication is typically periodic, or nearly periodic, in real-time systems, each connection is characterized by its minimum temporal spacing between messages ($I_{\min}$) and maximum message size ($S_{\max}$ bytes). To permit some variation from purely periodic traffic, a connection can generate a burst of up to $B_{\max}$ messages in excess of the periodic restriction $I_{\min}$. Together, these three parameters form a *linear bounded arrival process* [26] that governs a connection's traffic generation at the source node.

End-to-end delay bound: In addition to these traffic parameters, a connection has a bound D on end-to-end message delay, based on the minimum message spacing $I_{\min}$. At



(a) Time-constrained (b) Best-effort

Figure 6.4: Packet formats in real-time router

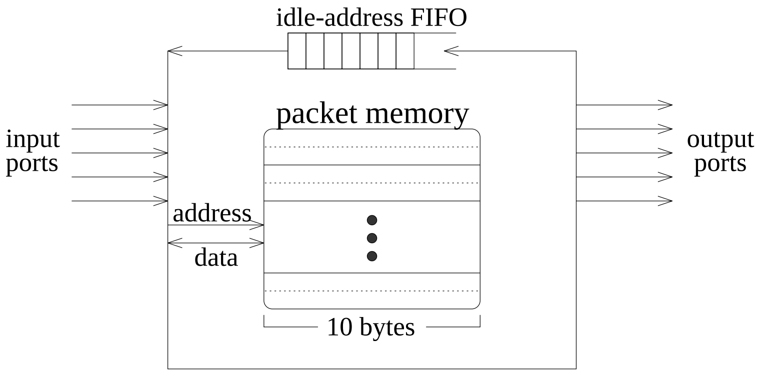


Figure 6.5: Buffer architecture for time-constrained traffic

	Traffic	**Data Structure**
Queue 1	On-time time-constrained packets	Priority queue (by deadline $\ell(m)+d$)
Queue 2	Best-effort packets	First-in-first-out queue
Queue 3	Early time-constrained packets	Priority queue (by logical arrival time $\ell(m)$)

Table 6.2: Link scheduling queues in real-time channels model

Scheduling Algorithm
if (queue_1 is non-empty)
transmit packet from head of queue_1 (minimum $\ell + d$);
else if (best-effort flits await service)
transmit best-effort flits;
else if ((queue_3 is non-empty) and (head has $\ell - t \leq h$))
transmit packet from head of queue_3 (minimum ℓ);
else
do not transmit any packet;

Figure 6.6: Link-scheduling algorithm in the the real-time router

the source node, a message m_i generated at time t_i has a *logical arrival time*

$$\ell_0(m_i) = \begin{cases} t_i & \text{if } i = 0 \\ \max\{\ell_0(m_{i-1}) + I_{\min},\ t_i\} & \text{if } i > 0. \end{cases}$$

By basing performance guarantees on these *logical* arrival times, the real-time channels model limits the influence an ill-behaving or malicious connection can have on other traffic in the network. The run-time link scheduler guarantees that message m_i reaches its destination node by its deadline $\ell_0(m_i) + D$.

Per-hop delay bounds: The network does not admit a new connection unless it can reserve sufficient buffer and bandwidth resources without violating the requirements of existing connections [63, 129]. A connection establishment procedure decomposes the connection's *end-to-end* delay bound D into *local* delay bounds d_j for each hop in its route such that $d_j \leq I_{\min}$ and $\sum_j d_j \leq D$. Based on the local delay bounds, a message m_i has a logical arrival time

$$\ell_j(m_i) = \ell_{j-1}(m_i) + d_{j-1} \quad \text{for } j > 0$$

at node j in its route, where $j = 0$ corresponds to the source node. Link scheduling ensures that message m_i arrives at node j no later than time $\ell_{j-1}(m_i) + d_{j-1}$, the local deadline at node $j-1$; however, message m_i could reach node j *earlier*, due to variations in delay at previous hops in the route.

Run-time link scheduling: Each link *schedules* time-constrained traffic based on logical arrival times and deadlines in order to bound message delay without exceeding the reserved buffer space at intermediate nodes. The scheduler, which employs a multi-class variation of the earliest due-date algorithm [80], gives highest priority to time-constrained messages that have reached their logical arrival time (i.e., $\ell_j(m_i) \leq t$), transmitting the message with the smallest deadline $\ell_j(m_i) + d_j$, as shown in Table 6.2. If Queue 1 is empty, the link services best-effort traffic from Queue 2, ahead of any early time-constrained messages, thus improving the average performance of best-effort traffic without violating the delay requirements of time-constrained communication; in the real-time router, Queue 2 is a *logical* queue of wormhole packets that may span multiple nodes. Queue 3 holds early time-constrained traffic, effectively absorbing variations in delay at the previous node; upon reaching its logical arrival time, a message moves from Queue 3 to Queue 1.

Buffer requirements: By postponing the transmission of early time-constrained traffic, the link scheduler avoids overloading the buffer space at the downstream node [63, 125]. If the first two scheduling queues are empty, the link can transmit early time-constrained traffic

Write Command	Fields
Connection parameters	outgoing connection id
	local delay bound d
	bit-mask of output ports
	incoming connection id
Horizon parameter	bit-mask of output ports
	horizon value h

Table 6.3: Control interface commands

from Queue 3, as long as these messages are within a small distance $h \geq 0$ of their logical arrival time; Figure 6.6 summarizes the router's link-scheduling algorithm. Incorporating this *horizon* parameter improves average latency and bandwidth utilization, at the expense of increased buffer requirements at the downstream node. A connection's local delay bound, coupled with the incoming link's horizon parameter, limits the required buffer space at the next node in the route. Node j can receive a message as early as $\ell_j(m_i)-(h_{j-1}+d_{j-1})$, if the incoming link has horizon h_{j-1}; the node can hold a message until its deadline $\ell_j(m_i)+d_j$. If messages arrive as early as possible, and depart as late as possible, then node j could have to store as many as

$$\left\lceil \frac{(h_{j-1}+d_{j-1})+d_j}{I_{\min}} \right\rceil$$

messages from this connection at the same time. Although each connection could conceivably have its own horizon value, employing a single h parameter allows the link to transmit early traffic directly from the head of Queue 3, without any per-connection data structures.

6.3 Real-Time Support

Supporting time-constrained communication in a single chip requires careful consideration of the interface to the controlling protocol software. The real-time router permits flexible software control of connection establishment, while implementing efficient run-time packet scheduling on the outgoing ports.

6.3.1 Control Interface

Establishing a real-time channel requires the application to specify the traffic parameters and performance requirements for the new connection. Admitting a new connection, and selecting a multi-hop route with suitable local delay parameters, is a computationally-intensive procedure [5, 63, 129]. Fortunately, channel establishment typically does not impose tight timing constraints; in most cases, the network can create the required channels before data transfer commences. To permit a single-chip solution, the real-time router relegates these non-real-time operations to the protocol software. Software control also permits greater flexibility in route selection and buffer allocation policies.

As part of establishing a new real-time channel, each node in the connection's route writes control information into a table in the router. Indexed off the connection identifier, the table stores the channel's local delay bound d and a bit mask for routing incoming packets to the appropriate output port(s); to simplify the design, a multicast connection uses the same value of d for any outgoing ports at the node. To minimize the number of pins on the chip, the controlling processor updates the connection table as a sequence of

On-time:	0	0	$\ell(m) + d - t$
Early:	0	1	$\ell(m) - t$
Ineligible:	1	—	——

Figure 6.7: Sorting key for time-constrained packets

four write operations, as shown in Table 6.3. When a time-constrained packet arrives, the router reads the deadline and routing information and assigns a new connection identifier for use at the next node in the packet's route. The router also assigns the packet's local deadline, based on the delay parameter d and the logical arrival time $\ell(m)$, as shown in Figure 6.4(a).

The packet deadline at one node serves as the logical arrival time at the downstream node in the route. Carrying these logical arrival times in the packet header implicitly assumes that the network routers have a common notion of time, within some bounded clock skew. Although this is not appropriate in a wide-area network context, the tight coupling in parallel machines minimizes the effects of clock skew. Alternatively, the router could store additional information in the connection table to compute $\ell_j(m_i)$ from a packet's actual arrival time and the logical arrival time of the connection's previous packet [128]; however, this approach would require the router to periodically refresh this connection state to correctly handle the effects of clock rollover.

In addition to the connection table, the router maintains a separate horizon parameter h for each outgoing port. As discussed in Section 6.2, these horizon values permit the router to transmit a time-constrained packet in advance of its logical arrival time, when no on-time packets or best-effort flits await service. The local processor can write the horizon registers through the control interface, as shown in Table 6.3. Larger horizon values permit earlier transmission of time-constrained packets, but require connections to reserve more buffer space at the downstream node. If necessary, the protocol software could reduce a port's horizon parameter as more connections are established, to free downstream buffer space for reservation by the new connections.

6.3.2 Scheduling Logic

The real-time router schedules time-constrained traffic for transmission based on logical arrival times and deadlines, as well as the link horizon parameters. To maximize link utilization and channel admissability, the router overlaps run-time communication scheduling with packet transmission on each of the five output ports. As a result, packet size determines the acceptable worst-case scheduling delay, limiting both the maximum number of time-constrained packets and the size of the sorting keys [96]; to facilitate a single-chip solution, our design efficiently handles a moderate number of packets. Since packet sorting can introduce considerable hardware complexity [19, 22, 79, 96, 105, 121], particularly when connections have a wide range of delay and bandwidth parameters, the real-time router *shares* the scheduling logic amongst the early and on-time packets headed for any of the five outgoing ports.

Table 6.2 suggests that each outgoing port requires separate priority queues for early and on-time packets. However, implementing two priority queues for each link would incur significant hardware cost and would require logic to transfer packets from the early queue to

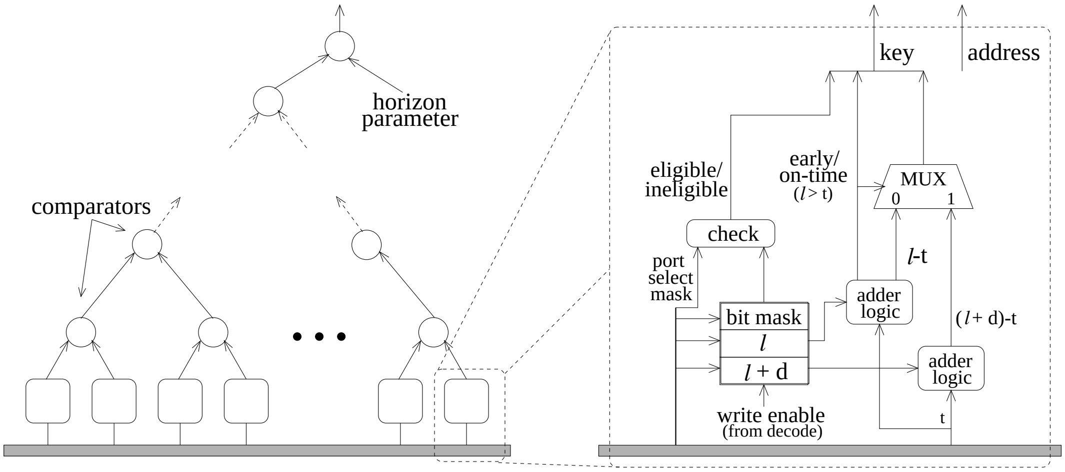


Figure 6.8: Comparator tree for run-time scheduling

the on-time queue; also, multiple packets can reach their logical arrival times *simultaneously*, further complicating movement between the two priority queues, as shown in Figure 6.1. Hence, the real-time router does not attempt to store time-constrained packets in sorted order; instead, the router employs a tree of comparators to select the packet with the smallest key. The base of the tree computes a key for each packet, based on the packet state and the current time t; a bit in the packet key differentiates between early and on-time traffic, as shown in Figure 6.7.

For on-time traffic, the lower bits of the key represent packet *laxity*, the time remaining till the local deadline expires, whereas the key for early traffic represents the time left before reaching the packet's logical arrival time. Normalizing the packet keys, relative to current time t, allows the rest of the tree to perform simple, unsigned comparison operations, even in the presence of clock rollover. To avoid replicating the scheduling logic, all five outgoing ports share access to a single comparator tree that arbitrates amongst all time-constrained packets, as shown in Figure 6.8. Pipelining the comparator tree provides the necessary throughput to overlap run-time scheduling with packet transmission on each outgoing port. This also permits the ports to conveniently share the same packet memory. Although this buffer memory stores the actual packet data, the base of the comparator tree maintains a small amount of per-packet state to coordinate run-time scheduling.

As shown in Figure 6.8, each leaf in the tree stores a logical arrival time $\ell(m)$, a deadline $\ell(m)+d$, and a bit mask of outgoing ports, assigned at packet arrival based on the connection state. The bit mask determines if the leaf is eligible to compete for access to a particular outgoing port. When a port transmits a selected packet, it clears the corresponding field in the leaf's bit mask; a bit mask of zero indicates an empty packet leaf slot and a corresponding idle slot in the packet memory. The base of the tree also determines if packets are early ($\ell(m) > t$) or on-time ($\ell(m) \leq t$) and computes the sorting keys based on the current value of t. At the top of the sorting tree, an additional comparator checks to see if the winning packet is early traffic that falls within the link's horizon parameter; if so, the link transmits this packet, unless best-effort flits await service.

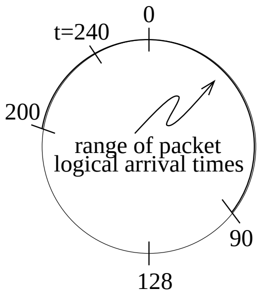


Figure 6.9: Handling clock rollover with an 8-bit clock

6.3.3 Handling Clock Rollover

The number of bits in the sorting keys directly affects the latency and implementation complexity of the comparator tree. However, by limiting the size of the keys, the router also restricts the range of local delay bounds d that can be selected by time-constrained connections. To formalize this trade-off, consider a connection traversing consecutive links $j-1$ and j, with local delay parameters d_{j-1} and d_j, respectively, and a horizon parameter h_{j-1} at link $j-1$. A packet can arrive as much as $h_{j-1}+d_{j-1}$ time units ahead of its logical arrival time $\ell_j(m)$, if link $j-1$ transmits the packet as early as possible. Similarly, link j must transmit the packet by its deadline $\ell_j(m)+d_j$. Hence, at time t and link j, any packets from this connection have logical arrival times $\ell_j(m) \in [t-d_j,\ t+(h_{j-1}+d_{j-1})]$.

This property permits the router to limit the size of the packet sorting keys, as well as the required number of bits in the on-chip clock, where the clock ticks once per packet transmission time. The router can correctly interpret logical arrival times and deadlines, even in the presence of clock rollover, as long as each connection has $h_{j-1}+d_{j-1}$ and d_j values that are less than *half* the range of the on-chip clock register. For example, Figure 6.9 shows a range of $\ell_j(m)$ values for different connections under an 8-bit clock, with a range of 256 time units. A packet with $\ell(m)=80$ would be considered early traffic (since t–$80 \geq 128$), while a packet with $\ell(m)=210$ would be considered on-time traffic (since t–$210 < 128$). This enables the leaves of the sorting tree to compute the normalized keys, relative to current time t, using modulo arithmetic.

6.4 Implementation and Evaluation

To demonstrate the feasibility of the architecture, a prototype of the router chip has been designed using the Verilog hardware description language and the Epoch silicon compiler from Cascade Design Automation. This framework facilitates a detailed evaluation of the implementation and performance properties of the architecture and suggests possible mechanisms for improving the router design.

6.4.1 Chip Design

The Epoch tools compile the structural and behavioral Verilog models to generate a chip layout and an annotated Verilog model for timing simulations. Using a three-metal, 0.5μm CMOS process, the 123-pin chip has dimensions 8.1 mm $\times$ 8.7 mm for an implementation with 256 time-constrained packets and up to 256 connections, as shown in Table 6.4. Manual intervention in the layout process significantly reduced the chip area and increased the

Parameter	Value
Connections	256
Time-constrained packets	256
Clock (sorting key)	8 (9) bits
Comparator tree pipeline	2 stages
Flit input buffer	10 bytes

(a) Architectural parameters

Parameter	Value
Process	0.5μm 3-metal CMOS
Signal pins	123
Transistors	$905,104$
Area	8.1 mm $\times$ 8.7 mm
Power	2.3 watts

(b) Chip complexity

Table 6.4: Router specification

achievable clock speed. The link-scheduling logic accounts for the majority of the chip area, with the packet memory consuming much of the remaining space, as shown in Table 6.5. Operating at 50 MHz, the chip can transmit or receive a byte of data on each of its ten ports every 20 nsec; this closely matches the access time of the 10-byte-wide, single-ported SRAM for storing time-constrained traffic.

Since time-constrained packets are 20-bytes long, the scheduling logic must select a packet for transmission every 400 nsec for each of the five output ports. To achieve the necessary throughput, the comparator tree consists of a two-stage pipeline, where each stage requires approximately 50 nsec; the boundary between the two pipeline stages consists of a set of latches across a row of comparators. Although the tree could incorporate up to five pipeline stages, the two-stage design provides sufficient throughput to satisfy the output ports. This suggests that the link scheduler could effectively support a larger number of packets or additional output ports, for a higher-dimensional mesh topology.

6.4.2 Experiments

Verilog simulations were used to test a single router chip under a variety of traffic patterns. A preliminary experiment tests the baseline performance of best-effort wormhole packets. To study a multi-hop configuration, the router connects its links in the x and y directions. The packet proceeds from the injection port to the positive x link, then travels from the negative x input link to the positive y direction; after reentering the router on the negative y link, the packet proceeds to the reception port. In this test, a b byte wormhole packet incurs an end-to-end latency of $30+b$ cycles, where the link transmits one byte in each cycle. This delay is proportional to packet length, with a small overhead for synchronizing the arriving bytes, processing the packet header, and accumulating five-byte chunks for access to the router's internal bus. In contrast, packet switching would introduce additional delay to buffer the packet at each hop in its route.

An additional experiment illustrates how the router schedules time-constrained packets to satisfy delay and throughput guarantees, while allowing best-effort traffic to capitalize on any excess link bandwidth. Figure 6.10 plots the link bandwidth consumed by best-effort traffic and each of three time-constrained connections with the following parameters, in units of 20-byte slots:

	d	$I_{\min}$
0	8	9
1	5	7
2	3	4

Unit	Area	Transistors
Priority queue	34.02 mm^2	555025
Memory and control	5.97 mm^2	268161
Best-effort support	1.55 mm^2	45352
Connection table	0.65 mm^2	20966
Idle-address pool	0.35 mm^2	15600

Table 6.5: Components of chip area for real-time router

All three connections compete for access to a single network link with horizon parameter $h = 0$, where each connection has a continual backlog of traffic. The time-constrained connections receive service in proportion to their throughput requirements, since a packet is not eligible for service till its logical arrival time. Similarly, the link transmits each packet by its deadline, with best-effort flits consuming any remaining link bandwidth.

6.4.3 Reducing Scheduler Complexity

As discussed in Section 6.4.1, the comparator tree is the main source of complexity in the real-time router architecture. Extensions to the architecture can reduce this complexity by sharing logic amongst *groups* of packets. To handle n packets, the scheduler in Figure 6.8 has a total of $2+\lg n$ stages of logic, including the operations at the base of the tree as well as the comparator for the horizon parameter. In terms of implementation cost, the tree requires n comparators and n leaf nodes, for a total of $2n$ elements of similar complexity; for large n, the number of leaf nodes can have a significant influence on the bus loading at the base of the tree. To reduce the logic complexity and bus loading, the router could share logic between leaves in the same subtree; in effect, this collapses some of the large layers of logic at the bottom of the tree, as shown in Figure 6.11.

In this approach, the router combines several leaf units into a single module with a small memory (e.g., a register file) to store the deadlines and logical arrival times for k packets, where k is a power of two. At the base of the tree, each of the n/k modules can *sequentially* compare its k sorting keys, using a single comparator, to select the packet with the minimum key; this incurs k stages of delay. Then, a smaller comparator tree finds the smallest key amongst n/k packets. As a result, the scheduler incurs

$$(k+1) + \lg\left(\frac{n}{k}\right)$$

stages of delay; note that, for $k = 1$, this reduces to the $2 + \lg n$ stages of logic in the architecture in Figure 6.8. For larger values of k, the scheduler has larger arbitration delay but reduced implementation complexity; the architecture in Figure 6.11 has $2n/k$ comparators, as well as a lighter bus loading of n/k elements at the base of the tree. Figure 6.12 highlights the cost-performance trade-offs of logic sharing, based on Epoch implementations and Verilog simulation experiments.

For example, the implementation described in Table 6.4 has $n = 256$ and $k = 1$, for a total of 512 logic elements and 10 stages of delay, with 256 units on the bus at the base of the tree. The router could reduce scheduler complexity by grouping packets in sets of four (i.e., $k = 4$). This would result in just 128 logic elements, with 64 units at the base of the tree, at the expense of just three extra stages of delay. Since latency in the comparator tree

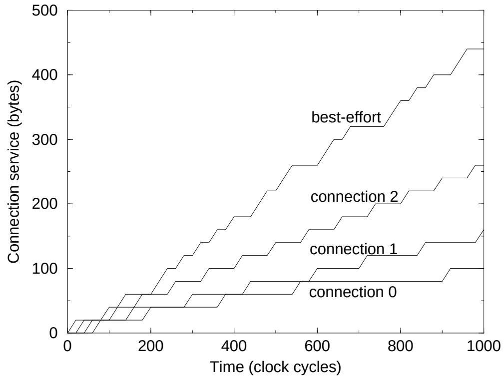


Figure 6.10: Time-constrained and best-effort service

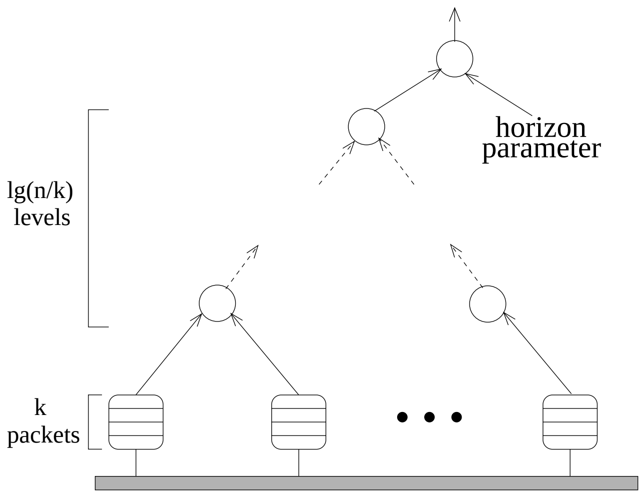


Figure 6.11: Comparator tree with logic sharing amongst subtrees

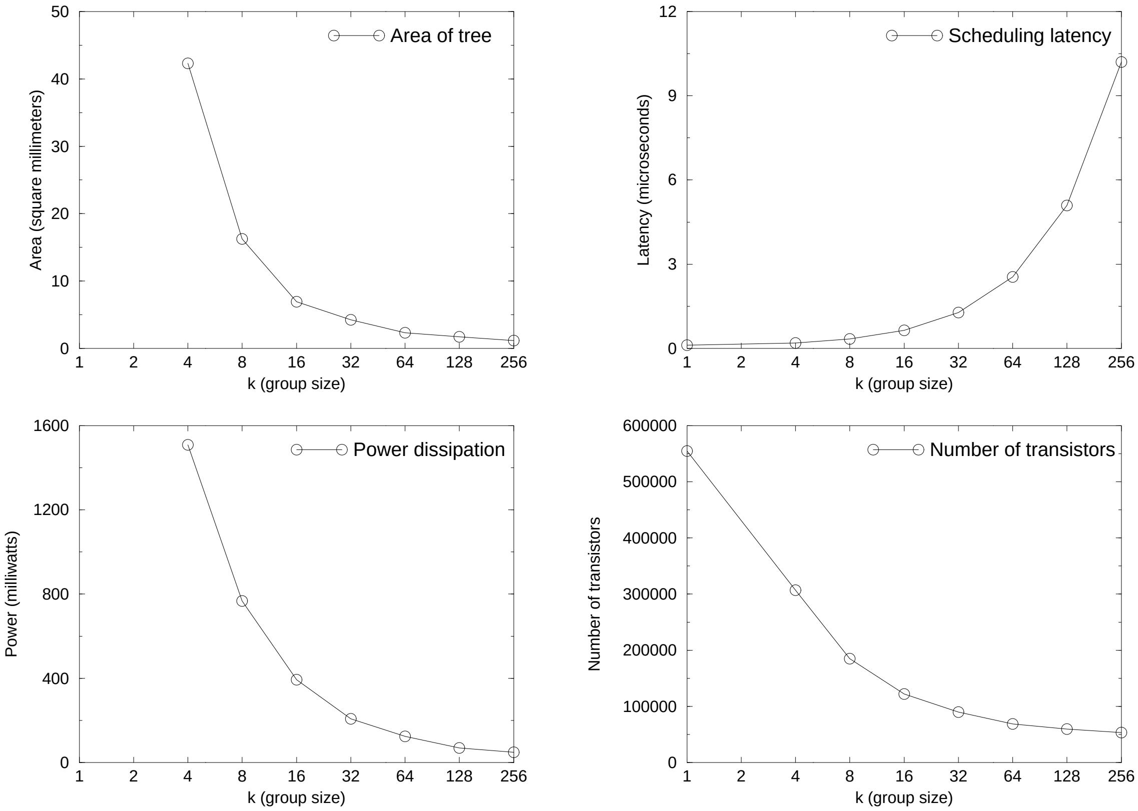


Figure 6.12: Comparison of comparator tree architectures with group size k

is not the bottleneck in the implementation, the router could reduce scheduler complexity without affecting the operating speed. Future work can investigate other mechanisms for reducing the cost of the scheduling logic, including schemes that relax the accuracy of packet sorting to permit a cheaper and faster design [79, 101]. Such approximate schemes may prove necessary for implementing real-time packet scheduling algorithms in emerging high-speed networks.

6.5 Conclusion

Parallel real-time applications impose diverse communication requirements on the underlying interconnection network. The real-time router design supports these emerging applications by bounding packet delay for time-constrained traffic, while ensuring good average performance for best-effort traffic. Low-level control over routing and switching, coupled with fine-grain arbitration at the network links, enables the router to effectively mix these two diverse traffic classes. Sharing the scheduling logic amongst the multiple output ports significantly reduces implementation complexity, while careful handling of clock rollover enables the router to support connections with diverse delay and throughput parameters with small packet sorting keys. This chapter introduces several research contributions:

- *Router architecture that supports best-effort and time-constrained traffic:* Based on the results from Chapter 5, this chapter presents the architecture and implementation of a router that supports the conflicting performance requirements of best-effort and time-constrained traffic, as described in Section 6.1. The router employs bandwidth regulation and deadline-based scheduling, with packet switching and table-driven multicast routing, to bound end-to-end delay and avoid buffer overflow for time-constrained traffic. In contrast, best-effort traffic uses wormhole switching, dimension-ordered routing, and variable-length packets for low average latency and reduced buffer space requirements.

- *Sharing of scheduling logic amongst links and early/on-time packets:* To permit a single-chip solution, the real-time router architecture shares the bandwidth regulation and link scheduling logic amongst the multiple output ports, as described in Section 6.3. In contrast to the model in Figure 6.1, the router *integrates* the bandwidth regulation and packet scheduling units to both avoid the replication of complex sorting logic and the expensive movement of packets between two separate priority queues. Sharing sorting logic and packet buffers amongst the five output ports permits a single-chip solution that handles up to 256 time-constrained packets simultaneously, as seen in Section 6.4.

- *Handling of clock rollover in link scheduling:* To handle the effects of clock rollover, Section 6.3.3 derives the relationship between the range of the clock and the permissible delay and horizon parameters in the real-time channel model; this relates directly to implementation complexity and scheduling delay by determining the number of bits in the packet sorting keys in Figure 6.7. Given an upper bound on the possible delay and horizon parameters, the analysis can derive the minimum number of bits for the router's real-time clock and the packet sorting keys. Similarly, for a given clock size, the analysis can determine the required restrictions on the values of d and h.

- *Formalization of space/time trade-offs in comparator trees:* Figure 6.8 and Figure 6.11 propose a comparator tree architecture for run-time packet scheduling in the real-time channel model. Section 6.3.2 and Section 6.4.3 propose several techniques for balancing the trade-offs between the implementation complexity, throughput, and latency of the scheduler. Section 6.3.2 suggests pipelining access to the shared data structure, to overlap packet transmission with run-time scheduling on each outgoing port. To reduce implementation complexity, Section 6.4.3 presents a mechanism that trades time for space by reusing logic at the base of the comparator tree. For a given packet size and number of outgoing ports, the analysis in Section 6.4.3 can determine the maximum possible value of k (degree of logic sharing).

The real-time router tailors low-level routing, switching, arbitration, and flow-control policies to the conflicting demands of time-constrained and best-effort traffic. The router's delay and throughput guarantees for time-constrained packets, combined with low average latency for best-effort packets, can efficiently support a wide range of communication performance requirements. As a result, the single-chip solution can serve as an effective building block for constructing multicomputer networks, or large high-speed switches, that support the quality-of-service requirements of emerging real-time and multimedia applications.

CHAPTER 7

CONCLUSIONS

The performance of multicomputer routers hinges on the subtle interplay between application workloads and network policies, such as switching, routing, queueing, flow control, and resource arbitration. This thesis investigates the subtle interplay between cut-through switching and other architectural parameters to improve the design and evaluation of multicomputer routers.

7.1 Research Contributions

Chapter 2 provides a classification of multicomputer router architectures, highlighting the diversity of existing designs. Focusing on the influence of routing, Chapter 3 develops analytical models for several cut-through routing algorithms with different degrees of adaptivity, based on a recurrence that computes the number of *internal* and *border* nodes in an h-hop route. The analytical models can help weigh the cost-performance trade-offs of implementing adaptive routing algorithms, particularly in large multicomputer networks. The detailed comparisons with simulation results reveal the unique dependencies between adjacent nodes in cut-through networks. Modern multicomputer systems employ network topologies, routing algorithms, and application workloads that exacerbate these correlation effects. Based on these results, the chapter introduces a new routing algorithm that capitalizes on inter-node dependencies to improve network performance.

To facilitate broader comparisons between different network architectures, Chapter 4 presents a general router model (v-router) and a simulation testbed (pp-mess-sim) for evaluating multicomputer networks. The objected-oriented pp-mess-sim and v-router framework decomposes multicomputer network and router design into separate components with well-defined interfaces. The development of this simulation environment introduces several new techniques for evaluating router architectures that support multiple *coexisting* routing-switching schemes, tailored to different traffic patterns and performance metrics. In particular, the simulator includes an abstract language for representing routing-switching algorithms, independent of the low-level timing details in the underlying router model. Additionally, the framework incorporates effective mechanisms for simulating flow-control and arbitration policies, a flexible history-list data collection scheme, and an extensible task model for composing complex communication workloads to evaluate a mixture of traffic classes.

Drawing on these novel features, Chapter 5 investigates architectural features that allow time-constrained and best-effort traffic to share access to network resources in parallel real-time systems. Simulation experiments demonstrate that packet switching, coupled with static routing, can support predictable communication for time-constrained packets, while wormhole switching can provide low average latency for best-effort traffic. Multicomputer routers can support both traffic classes by partitioning best-effort and time-constrained packets onto separate virtual channels. Fine-grain, demand-driven arbitration for each link and injection/reception port can ensure that each traffic class can capitalize on the available bandwidth resources, as shown by simulation experiments with the v-router mode, as well as an implementation of the Programmable Routing Controller.

These results motivate a new router architecture that bounds worst-case latency for time-constrained traffic, while ensuring good average latency for best-effort packets, as discussed in Chapter 6. The router implements deadline-based scheduling, with packet switching and table-driven multicast routing, to bound end-to-end delay for time-constrained traffic, while allowing best-effort traffic to capitalize on the low-latency routing and switching schemes common in modern parallel machines. To limit the cost of servicing time-constrained traffic, the router shares packet buffers and link-scheduling logic between the multiple output ports and implicitly handles the effects of clock rollover in computing packet deadlines. Tailoring the low-level routing, switching, arbitration and flow-control policies to the conflicting demands of each traffic class results in a single-chip implementation that can serve as a building block for constructing real-time multicomputer systems.

7.2 Avenues for Future Work

This thesis presents analytical, simulation, and architectural techniques for the design and evaluation of cut-through routers that tailor network policies to application performance requirements. These results motivate future research on more accurate analytical and simulation models that capture the interaction of application workloads and router architectures. Further simulation experiments could evaluate cut-through correlation effects under a wider range of router architectures and application workloads. The precise characterization of inter-node dependencies can guide the development of more accurate performance models, as well as novel network architectures and task allocation schemes that exploit the natural dependencies between neighboring nodes.

To study more diverse architectures, further work on pp-mess-sim and the v-router model can address the interplay between switching schemes and buffer architectures. With extensions to the **Ralg** instruction set, the simulator could interact more closely with the queueing models in the v-router **Node** model. This would facilitate experimentation with a wider range of schemes for avoiding packet deadlock in modern cut-through networks. Specific extensions to the v-router model could broaden the collection of arbitration and buffer models to construct a more comprehensive set of candidate router designs. For example, this would enable the simulator to evaluate a mesh network based on the real-time router architecture.

These simulation experiments could also compare the real-time router design with other techniques for supporting time-constrained and best-effort communication in multicomputer networks. Future research on the real-time router can also include further investigation of scalable techniques for arbitrating between packets. High-speed multicomputer and local-area networks, coupled with the emergence of real-time and multimedia applications, motivate the need for effective hardware support for regulating bandwidth and scheduling traffic for a large number of connections with diverse delay and throughput requirements. Ultimately, these modern applications demand effective router architectures that tailor network policies to performance requirements.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] S. Abraham and K. Padmanabhan, "Performance of multicomputer networks under pin-out constraints," *Journal of Distributed Computing*, pp. 237–248, 1991.

[2] A. Agarwal, "Limits on interconnection network performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398–412, October 1991.

[3] J. D. Allen, P. T. Gaughan, D. E. Schimmel, and S. Yalamanchili, "Ariadne: An adaptive router for fault-tolerant multicomputers," in *Proceedings of the International Symposium on Computer Architecture*, pp. 278–288, April 1994.

[4] K. Aoyama and A. Chien, "Cost of adaptivity and virtual lanes in a wormhole router," *Journal of VLSI Design*, vol. 2, no. 4, pp. 315–333, 1995.

[5] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne, "Real-time communication in packet-switched networks," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 122–139, January 1994.

[6] K. Arvind, K. Ramamritham, and J. A. Stankovic, "A local area network architecture for communication in distributed real-time systems," *Journal of Real-Time Systems*, vol. 3, no. 2, pp. 115–147, May 1991.

[7] W. Athas and C. Seitz, "Multicomputers: Message-passing concurrent computers," *IEEE Computer Magazine*, pp. 9–24, August 1988.

[8] H. G. Badr and S. Podar, "An optimal shortest-path routing policy for network computers with regular mesh-connected topologies," *IEEE Transactions on Computers*, vol. C-38, no. 10, pp. 1362–1370, October 1989.

[9] J. J. Bae and T. Suda, "Survey of traffic control schemes and protocols in ATM networks," *Proceedings of the IEEE*, vol. 79, no. 2, pp. 170–189, February 1991.

[10] S. Balakrishnan and F. Ozguner, "Providing message delivery guarantees in pipelined flit-buffered multiprocessor networks," in *Proceedings of the Real-Time Technology and Applications Symposium*, pp. 120–129, June 1996.

[11] R. C. Bedichek, "Talisman: Fast and accurate multicomputer simulation," in *Proceedings of ACM SIGMETRICS/Performance*, pp. 14–24, May 1995.

[12] F. Blitzer, "Militarized touchtone program," in *Proceedings of the IEEE National Aerospace and Electronics Conference*, pp. 137–143, 1993.

[13] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. W. Felten, and J. Sandberg, "Virtual memory mapped network interface for the SHRIMP multicomputer," in *Proceedings of the International Symposium on Computer Architecture*, pp. 142–153, April 1994.

[14] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W.-K. Su, "Myrinet: A gigabit-per-second local area network," *IEEE Micro*, pp. 29–36, February 1995.

[15] K. Bolding, S.-C. Cheung, S.-E. Choi, C. Ebeling, S. Hassoun, T. A. Ngo, and R. Wille, "The Chaos router chip: Design and implementation of an adaptive router," in *Proceedings of the IFIP International Conference on VLSI*, pp. 311–320, September 1993.

[16] K. Bolding, S.-E. Choi, and M. Fulgham. *The Chaos Router Simulator*. Presentation at the *Parallel Computer Routing and Communication Workshop*, May 1994.

[17] R. Boppana and S. Chalasani, "A comparison of adaptive wormhole routing algorithms," in *Proceedings of the International Symposium on Computer Architecture*, pp. 351–360, 1993.

[18] G. A. Boughton, "Artic routing chip," in *Proceedings of the Parallel Computer Routing and Communication Workshop*, pp. 310–317, June 1994.

[19] P. E. Boyer, F. M. Guillemin, M. J. Servel, and J.-P. Coudreuse, "Spacing cells protects and enhances utilization of ATM network links," *IEEE Network Magazine*, pp. 38–49, September 1992.

[20] R. Buck, "The Oracle media server for nCUBE massively parallel system," in *Proceedings of the International Parallel Processing Symposium*, pp. 670–673, April 1994.

[21] P. J. Burke, "The output of a queueing system," *Operations Research*, vol. 4, pp. 699–704, 1956.

[22] H. J. Chao, "A novel architecture for queue management in the ATM network," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 7, pp. 1110–1118, September 1991.

[23] S. Chittor and R. Enbody, "Performance evaluation of mesh-connected wormhole-routed networks for interprocessor communication in multicomputers," in *Supercomputing*, pp. 647–656, November 1990.

[24] D. Cohen, G. G. Finn, R. Felderman, and A. DeSchon, "The use of message-based multicomputer components to construct gigabit networks," *Computer Communication Review*, vol. 23, no. 3, pp. 32–44, July 1993.

[25] *Cray T3D System Architecture Overview*, Cray Research, Inc., 1993.

[26] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, January 1991.

[27] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "Architectural requirements of parallel scientific applications with explicit communication," in *Proceedings of the International Symposium on Computer Architecture*, pp. 2–13, May 1993.

[28] W. J. Dally and C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

[29] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, May 1987.

[30] W. J. Dally and P. Song, "Design of a self-timed VLSI multicomputer communication controller," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 230–234, 1987.

[31] W. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, March 1992.

[32] W. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466–475, April 1993.

[33] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775–785, June 1990.

[34] W. J. Dally, L. R. Dennison, D. Harris, K. Kan, and T. Zanthopoulos, "The reliable router: A reliable and high-performance communication substrate for parallel computers," in *Proceedings of the Parallel Computer Routing and Communication Workshop*, pp. 241–255, June 1994.

[35] W. J. Dally, J. A. S. Fiske, J. S. Keen, R. A. Lethin, M. D. Noakes, P. R. Nuth, R. E. Davison, and G. A. Fyler, "The Message-Driven Processor: A multicomputer processing node with efficient mechanisms," *IEEE Micro*, pp. 23–39, April 1992.

[36] S. Dandamudi and D. Eager, "Hot-spot contention in binary hypercube networks," *IEEE Transactions on Computers*, vol. 41, no. 2, pp. 239–244, February 1992.

[37] S. Daniel, *Flexible Router Architectures for Point-to-Point Networks*, PhD thesis, University of Michigan, May 1996.

[38] S. Daniel, J. Rexford, J. Dolter, and K. Shin, "A programmable routing controller for flexible communications in point-to-point networks," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 320–325, October 1995.

[39] A. L. Davis, "Mayfly: A general-purpose, scalable, parallel processing architecture," *Lisp and Symbolic Computation*, vol. 5, no. 1/2, pp. 7–47, May 1992.

[40] P. M. Dickens, P. Heidelberger, and D. M. Nicol, "Parallelized network simulators for message-passing parallel programs," in *Proceedings of the International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 72–76, 1995.

[41] J. W. Dolter, P. Ramanathan, and K. G. Shin, "Performance analysis of virtual cut-through switching in HARTS: A hexagonal mesh multicomputer," *IEEE Transactions on Computers*, vol. 40, no. 6, pp. 669–680, June 1991.

[42] J. Dolter, *A Programmable Routing Controller Supporting Multi-Mode Routing and Switching in Distributed Real-Time Systems*, PhD thesis, University of Michigan, September 1993.

[43] J. Dolter, S. Daniel, A. Mehra, J. Rexford, W. Feng, and K. Shin, "SPIDER: Flexible and efficient communication support for point-to-point distributed systems," in *Proceedings of the International Conference on Distributed Computing Systems*, pp. 574–580, June 1994.

[44] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1320–1331, December 1993.

[45] T. H. Dunigan, "Performance of the Intel iPSC/860 and Ncube 6400 hypercubes," *Parallel Computing*, vol. 17, no. 10/11, pp. 1285–1302, December 1991.

[46] S. Felperin, L. Gravano, G. Pirarre, and J. Sanz, "Routing techniques for massively parallel communication," *Proceedings of the IEEE*, vol. 79, no. 4, pp. 488–503, April 1991.

[47] T.-Y. Feng, "A survey of interconnection networks," *IEEE Computer Magazine*, vol. 14, no. 12, pp. 12–27, December 1981.

[48] W. Feng, J. Rexford, S. Daniel, A. Mehra, and K. Shin, "Tailoring routing and switching schemes to application workloads in multicomputer networks," Computer Science and Engineering Technical Report CSE-TR-239-95, University of Michigan, May 1995.

[49] W. Feng, J. Rexford, A. Mehra, S. Daniel, J. Dolter, and K. Shin, "Architectural support for managing communication in point-to-point distributed systems," Technical Report CSE-TR-197-94, University of Michigan, March 1994.

[50] W. Feng and K. Shin, "Impact of selection functions on routing algorithm performance in multicomputer networks," Computer Science and Engineering Technical Report CSE-TR-287-96, University of Michigan, March 1996.

[51] D. Ferrari, "Client requirements for real-time communication services," *IEEE Communication Magazine*, pp. 65–72, November 1990.

[52] R. Games, A. Kanevsky, P. Krupp, and L. Monk, "Real-time communications scheduling for massively parallel processors," in *Proceedings of the Real-Time Technology and Applications Symposium*, pp. 76–85, May 1995.

[53] P. Gaughan and S. Yalamanchili, "Adaptive routing protocols for hypercube interconnection networks," *IEEE Computer Magazine*, pp. 12–23, May 1993.

[54] A. Gupta and D. Ferrari, "Resource partitioning for real-time communication," *IEEE/ACM Transactions on Networking*, vol. 3, no. 5, pp. 501–508, October 1995.

[55] M. G. Hluchyj and M. J. Karol, "Queueing in high-performance packet switching," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1587–1597, December 1988.

[56] J.-M. Hsu and P. Banerjee, "Hardware support for message routing in a distributed memory multicomputer," in *Proceedings of the International Conference on Parallel Processing*, pp. I–508–I–515, 1990.

[57] 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*, vol. 3, no. 4, pp. 451–464, July 1992.

[58] M. Ilyas and H. T. Mouftah, "Towards performance improvement of cut-through switching in computer networks," *Performance Evaluation*, vol. 6, pp. 125–133, July 1986.

[59] *Paragon XP/S Product Overview*, Intel Corporation, 1991.

[60] J. R. Jackson, "Networks of waiting lines," *Operations Research*, vol. 5, no. 4, pp. 518–521, August 1957.

[61] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc., 1991.

[62] J. R. Jump and S. Lakshmanamurthy, "NETSIM: A general-purpose interconnection network simulator," in *Proceedings of the International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 121–125, January 1993.

[63] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multihop networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1044–1056, October 1994.

[64] V. Karamcheti and A. A. Chien, "Software overhead in messaging layers: Where does the time go?," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 51–60, October 1994.

[65] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks*, vol. 3, no. 4, pp. 267–286, September 1979.

[66] J. H. Kim and A. A. Chien, "An evaluation of planar-adaptive routing (PAR)," in *Proceedings of the International Symposium on Parallel and Distributed Processing*, 1992.

[67] J. H. Kim and A. A. Chien, "Evaluation of wormhole routed networks under hybrid traffic loads," in *Proceedings of the Hawaii International Conference on System Sciences*, pp. 276–285, January 1993.

[68] J. H. Kim and A. A. Chien, "The impact of packetization in wormhole-routed networks," in *Proceedings of Parallel Architectures and Languages, Europe*, 1993.

[69] J. H. Kim and A. A. Chien, "Rotated combined queueing (RCQ)," in *Proceedings of the International Symposium on Computer Architecture*, pp. 226–236, May 1996.

[70] L. Kleinrock, *Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill, New York, 1964.

[71] L. Kleinrock, *Queueing systems*, volume I: Theory, John Wiley & Sons, 1975.

[72] D. E. Knuth, *The Art of Computer Programming, Vol 2: Seminumerical Algorithms*, Addison Wesley, 1969.

[73] S. Konstantinidou, "Segment router: A novel router design for parallel computers," in *Proceedings of the Symposium on Parallel Algorithms and Architectures*, June 1994.

[74] S. Konstantinidou and L. Synder, "The Chaos router," *IEEE Transactions on Computers*, vol. 43, no. 12, pp. 1386–1397, December 1994.

[75] A. M. Law and M. G. McComas, "Simulation software for communications networks: The state of the art," *IEEE Communication Magazine*, pp. 44–50, March 1994.

[76] C. Leiserson, Z. Abuhamdeh, D. Douglas, C. Feynman, M. Ganmukhi, J. Hill, W. D. Hillis, B. Kuszmaul, M. St. Pierre, D. Wells, M. Wong, S.-W. Yang, and R. Zak, "The network architecture of the connection machine CM-5," in *Proceedings of the Symposium on Parallel Algorithms and Architectures*, pp. 272–285, June 1992.

[77] J.-P. Li and M. W. Mutka, "Priority based real-time communication for large scale wormhole networks," in *Proceedings of the International Parallel Processing Symposium*, pp. 433–438, April 1994.

[78] J.-P. Li and M. W. Mutka, "Real-time virtual channel flow control," in *Proceedings of the Phoenix Conference on Computers and Communication*, pp. 97–103, April 1994.

[79] J. Liebeherr and D. Wrege, "Versatile packet multiplexer for quality-of-service networks," in *Proceedings of the IEEE International Symposium on High Performance Distributed Computing*, pp. 148–155, August 1995.

[80] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[81] P. K. McKinley and C. Trefftz, "MultiSim: A simulation tool for the study of large-scale multiprocessors," in *Proceedings of the International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 57–62, January 1993.

[82] T. D. Morris and H. G. Perros, "Approximate analysis of a discrete-time tandem network of cut-through queues with blocking and bursty traffic," *Performance Evaluation*, vol. 17, no. 3, pp. 207–223, 1993.

[83] M. W. Mutka, "Using rate monotonic scheduling technology for real-time communications in a wormhole network," in *Proceedings of the IEEE Workshop on Parallel and Distributed Real-Time Systems*, pp. 194–199, April 1994.

[84] W. A. Najjar, A. Lagman, S. Sur, and P. K. Srimani, "Modeling adaptive routing in k-ary n-cube networks," in *Proceedings of the International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 120–125, 1994.

[85] *nCUBE-3: The Scalable Server Platform*, nCube Corporation, March 1995.

[86] J. Ngai and C. Seitz, "A framework for adaptive routing in multicomputer networks," in *Proceedings of the Symposium on Parallel Algorithms and Architectures*, pp. 1–9, June 1989.

[87] L. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer Magazine*, pp. 62–76, February 1993.

[88] A. Nowatzyk, M. Browne, E. Kelly, and M. Parkin, "S-connect: From networks of workstations to supercomputer performance," in *Proceedings of the International Symposium on Computer Architecture*, pp. 71–82, June 1995.

[89] S. F. Nugent, "The iPSC/2 direct-connect communications technology," in *Proceedings of Hypercube Concurrent Computers and Applications*, pp. 51–60, January 1988.

[90] W. Oed, *The Cray Research Massively Parallel Processor System: Cray T3D*, November 1993.

[91] Y. Ofek and M. Yung, "The integrated MetaNet architecture: A switch-based multimedia LAN for parallel computing and real-time traffic," in *Proceedings of IEEE INFOCOM*, pp. 802–811, 1994.

[92] Y. Oie, T. Suda, M. Murata, D. Kolson, and H. Miyahara, "Survey of switching techniques in high-speed networks and their performance," in *Proceedings of IEEE INFOCOM*, pp. 1242–1251, June 1990.

[93] E. Olk, "PARSE: Simulation of message passing communication networks," in *Proceedings of the Annual Simulation Symposium*, pp. 115–1245, April 1994.

[94] S. S. Owicki and A. R. Karlin, "Factors in the performance of the AN1 computer network," in *Proceedings of ACM SIGMETRICS*, pp. 167–180, June 1992.

[95] C. Peterson, J. Sutton, and P. Wiley, "iWarp: A 100-MOPS LIW microprocessor for multicomputers," *IEEE Micro*, pp. 26–29,81–87, June 1991.

[96] D. Picker and R. D. Fellman, "Scaling and performance of a priority packet queue for real-time applications," in *Proceedings of the Real-Time Systems Symposium*, pp. 56–62, December 1994.

[97] R. S. Raji, "Smart networks for control," *IEEE Spectrum*, vol. 31, no. 6, pp. 49–55, June 1994.

[98] S. Ramany and D. Eager, "The interaction between virtual channel flow control and adaptive routing in wormhole networks," in *Proceedings of the International Conference on Supercomputing*, pp. 136–145, July 1994.

[99] D. Reed and R. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, MIT Press, 1987.

[100] D. A. Reed and D. C. Grunwald, "The performance of multicomputer interconnection networks," *IEEE Computer Magazine*, vol. 20, no. 6, pp. 63–73, June 1987.

[101] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong, "Scalable architectures for integrated traffic shaping and link scheduling in high-speed ATM switches," in submission to *IEEE Journal on Selected Areas in Communications*, May 1996.

[102] J. Rexford, J. Dolter, W. Feng, and K. G. Shin, "PP-MESS-SIM: A simulator for evaluating multicomputer interconnection networks," in *Proceedings of the Annual Simulation Symposium*, pp. 84–93, April 1995.

[103] J. Rexford, J. Dolter, and K. G. Shin, "Hardware support for controlled interaction of guaranteed and best-effort communication," in *Proceedings of the IEEE Workshop on Parallel and Distributed Real-Time Systems*, pp. 188–193, April 1994.

[104] J. Rexford, W. Feng, J. Dolter, and K. G. Shin, "PP-MESS-SIM: A flexible and extensible simulator for evaluating multicomputer networks," to appear in *IEEE Transactions on Parallel and Distributed Systems*.

[105] J. Rexford, A. Greenberg, and F. Bonomi, "Hardware-efficient fair queueing architectures for high-speed networks," in *Proceedings of IEEE INFOCOM*, March 1996.

[106] J. Rexford and K. G. Shin, "Shortest-path routing in homogeneous point-to-point networks with virtual cut-through switching," Computer Science and Engineering Technical Report CSE-TR-146-92, University of Michigan, November 1992.

[107] J. Rexford and K. G. Shin, "Support for multiple classes of traffic in multicomputer routers," in *Proceedings of the Parallel Computer Routing and Communication Workshop*, pp. 116–130, May 1994.

[108] A. Saha, "Simulator for real-time parallel processing architectures," in *Proceedings of the Annual Simulation Symposium*, pp. 74–83, April 1995.

[109] C. L. Seitz, W. C. Athas, C. M. Flaig, A. J. Martin, J. Seizovic, C. S. Steele, and W.-K. Su, "The architecture and programming of the Ametek series 2010 multicomputer," in *Proceedings of Hypercube Concurrent Computers and Applications*, pp. 33–36, January 1988.

[110] C. L. Seitz and W. Su, "A family of routing and communication chips based on the Mosaic," in *Proceedings of the Symposium on Integrated Systems*, 1993.

[111] K. G. Shin and S. Daniel, "Analysis and implementation of hybrid switching," in *Proceedings of the International Symposium on Computer Architecture*, pp. 211–219, June 1995.

[112] K. G. Shin and Q. Zheng, "Mixed time-constrained and non-time-constrained communications in local area networks," *IEEE Transactions on Communications*, pp. 1668–1676, November 1993.

[113] J. A. Stankovic, "Distributed real-time computing: The next generation," Technical Report COINS 92-01, University of Massachusetts, Amherst, January 1992.

[114] C. Stunkel et al., "SP2 high-performance switch," *IBM Systems Journal*, vol. 34, no. 2, pp. 185–204, 1995.

[115] D. Talia, "Message-routing systems for transputer-based multicomputers," *IEEE Micro*, pp. 62–72, June 1993.

[116] Y. Tamir and G. Frazier, "Dynamically-allocated multi-queue buffers for VLSI communication switches," *IEEE Transactions on Computers*, vol. 41, no. 6, pp. 725–737, June 1992.

[117] P. Thompson, "Concurrent interconnect for parallel systems," *The Computer Journal*, vol. 36, no. 8, pp. 778–784, 1993.

[118] F. A. Tobagi, "Fast packet switch architectures for broadband integrated services digital networks," *Proceedings of the IEEE*, vol. 78, no. 1, pp. 133–167, January 1990.

[119] K. Toda, K. Nishida, E. Takahashi, N. Michell, and Y. Yamaguchi, "Design and implementation of a priority forwarding router chip for real-time interconnection networks," *International Journal of Mini and Microcomputers*, vol. 17, no. 1, pp. 42–51, 1995.

[120] B. Tsai and K. G. Shin, "Sequencing of concurrent communication traffic in a mesh multicomputer with virtual channels," in *Proceedings of the International Conference on Parallel Processing*, pp. I–126–I–133, August 1994.

[121] E. Wallmeier and T. Worster, "The Spacing Policer, an algorithm for efficient peak bit rate control in ATM networks," in *Proceedings of the International Switching Symposium*, pp. 22–26, October 1992.

[122] L. R. Welch and K. Toda, "Architectural support for real-time systems: Issues and trade-offs," in *Proceedings of the International Workshop on Real-Time Computing Systems and Applications*, pp. 145–152, December 1994.

[123] I. Widjaja, A. Leon-Garcia, and H. T. Mouftah, "The effect of cut-through switching on the performance of buffered Banyan networks," *Computer Networks and ISDN Systems*, vol. 26, pp. 139–159, 1993.

[124] Y. S. Youn and C. K. Un, "Performance analysis of an integrated voice/data cut-through switching network," *Computer Networks and ISDN Systems*, vol. 21, no. 1, pp. 41–51, 1991.

[125] H. Zhang, "Providing end-to-end performance guarantees using non-work-conserving disciplines," *Computer Communications*, vol. 18, no. 10, pp. 769–781, October 1995.

[126] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proceedings of the IEEE*, vol. 83, no. 10, pp. 1374–1396, October 1995.

[127] X. Zhang, "System effects of interprocessor communication latency in multicomputers," *IEEE Micro*, pp. 12–15, 52–55, April 1991.

[128] Q. Zheng, K. G. Shin, and C. Chen, "Real-time communication in ATM," in *Proceedings of the Annual Conference on Local Computer Networks*, pp. 156–164, October 1994.

[129] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks," *IEEE Transactions on Communications*, vol. 42, no. 2–4, pp. 1096–1105, February/March/April 1994.