PP-MESS-SIM: A Flexible and Extensible Simulator for Evaluating Multicomputer Networks*

Jennifer Rexford, Wu-chang Feng, James Dolter, and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
E-mail: {jrexford, wuchang, jdolter, kgshin}@eecs.umich.edu
Web page: http://www.eecs.umich.edu/RTCL

Abstract

This paper presents pp-mess-sim, an object-oriented discrete-event simulation environment for evaluating interconnection networks in message-passing systems. The simulator provides a toolbox of various network topologies, communication workloads, routing-switching algorithms, and router models. By carefully defining the boundaries between these modules, pp-mess-sim creates a flexible and extensible environment for evaluating different aspects of network design. The simulator models emerging multicomputer networks that can support multiple routing and switching schemes *simultaneously*; pp-mess-sim achieves this flexibility by associating routing-switching policies, traffic patterns, and performance metrics with *collections of packets*, instead of the underlying router model. Besides providing a general framework for evaluating router architectures, pp-mess-sim includes a cycle-level model of the PRC, a programmable router for point-to-point distributed systems. The PRC model captures low-level implementation details, while another high-level model facilitates experimentation with general router design issues. Sample simulation experiments capitalize on this flexibility to compare network architectures under various application workloads.

Key words: multicomputers, routers, routing, switching, object-oriented simulation

1 Introduction

Message-passing parallel machines have emerged as a cost-effective platform for exploiting concurrency or parallelism in applications. These multicomputer systems consist of processors linked by an interconnection network, where fast message exchange enables efficient cooperation between processing elements [1, 2]. Router hardware connects each processing node to the interconnection fabric and manages traffic flowing through the node *en route* to other nodes. The router architecture greatly affects the ability of the interconnection network to deliver good communication performance to parallel applications. This paper presents pp-mess-sim (point-to-point message simulator), a flexible and extensible simulation environment for evaluating and tuning multicomputer router designs [3, 4].

*The work reported in this paper was supported in part by the National Science Foundation under Grant MIP-9203895. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the NSF. James Dolter is now a senior engineer at Qualcomm Inc.

Maximizing system performance requires matching application communication requirements with a suitable network design. While many simulation toolkits can flexibly model local and wide area networks [5], few simulators sufficiently capture the characteristics of emerging multicomputer router architectures. In contrast to LANs/WANs, parallel systems typically employ regular network topologies that facilitate efficient, flexible routing schemes. Tighter coupling between nodes enables network designers to consider more diverse switching schemes and flow-control policies. In addition, mapping concurrent applications across multiple nodes generates unique communication patterns and requirements in message-passing parallel machines [6–9]. These communication workloads affect the performance of particular routing algorithms and switching schemes [10–13].

As a result, several recent router architectures support multiple routing or switching schemes *simultaneously* to tailor network policies to application performance requirements [13–17]. Evaluating and tuning such router designs requires special support in the network simulator. Several recent simulation tools evaluate various aspects of multicomputer applications and interconnection networks. Execution-driven simulators [18–20] typically capture the instruction-level operation of applications on particular multicomputer architectures. Other simulation tools emphasize multicomputer network architectures, allowing users to vary the router's buffer architecture, switching scheme, and routing algorithm, under different synthetic traffic patterns [21–23]. However, existing multicomputer simulators do not accommodate router architectures that allow multiple routing algorithms or switching schemes to coexist in the underlying network.

This paper presents pp-mess-sim, an object-oriented, discrete-event simulator for experimenting with flexible router policies in multicomputer interconnection networks. Implemented in C++, pp-mess-sim achieves a high degree of flexibility and extensibility by separating its major components into different classes, representing the network topology, application workloads, routing-switching algorithms, and the router architecture. By carefully defining the boundaries between these components, pp-mess-sim enables users to extend one module without altering the internal representation of the other classes. For example, users can easily incorporate new routing algorithms or traffic patterns without knowing the details of the underlying router model. In fact, as long as each model implements the necessary interfaces, these router models can vary from high-level architectures to detailed, low-level specifications of actual devices, allowing incremental investigation of implementation approaches and design enhancements.

The simulator includes several novel features that facilitate experimentation with flexible router architectures that can support multiple classes of traffic, with different communication workloads, performance metrics, and network policies. 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 router model, the simulator is able to support multiple routing algorithms and switching schemes *simultaneously*. The workload class also supports the flexible composition of diverse traffic patterns, as well as a history-list mechanism for accumulating performance metrics with user-selectable data collection functions. Finally, pp-mess-sim includes a powerful input specification language for constructing simulation experiments with diverse network parameters and complex communication workloads.

The next section of the paper provides an overview of multicomputer router design issues; the components of pp-mess-sim derive directly from these main architectural parameters, as shown in Section 3. Section 4 describes how this framework enables the evaluation of a variety of router models. Currently, pp-mess-sim includes a cycle-level model of the *programmable routing controller* (PRC) [16], a router for point-to-point distributed systems, as well as a general, higher-level router model. The simulator supports a broad spectrum of routing and switching schemes by decoupling them from the router models, as discussed in Section 5. Section 6 describes how pp-mess-sim insulates these routing-switching algorithms from the details of the network topology. The simulator can also construct diverse traffic patterns and performance metrics, as discussed in Section 7. In Section 8, sample simulation experiments capitalize on this flexibility to compare network architectures under a variety of application workloads. Section 9 concludes the paper with a discussion of future pp-mess-sim enhancements.

2 Motivation

This section gives an overview of the major architectural issues in multicomputer network design to motivate the need for a flexible simulation environment. The selection of these design parameters impacts both the cost and performance of the design. Router performance is further influenced by the characteristics of the applied communication workload.

2.1 Topology

The choice of network topology affects multicomputer performance and implementation complexity. By defining the connections between processing nodes, the topology determines the number of communication links at each node and how far a packet must travel to reach its destination. This impacts both the complexity of network wiring and the achievable communication bandwidth in the system [24, 25]. Depending on the communication workload, the network topology affects how non-uniform traffic patterns, such as hot-spots, form and dissipate over time. Although parallel machines may connect processing nodes in a variety of topologies, many multicomputers have regular, direct networks. Regular topologies simplify packet routing and the placement of application processes in the network; direct networks, consisting of point-to-point links, capitalize on spatial communication locality. Many multicomputers employ the k-ary n-cube family of topologies, with k nodes along each of n dimensions [24]. Currently pp-mess-sim supports k-ary n-cube topologies, square meshes, and wrapped hexagonal meshes.

A multicomputer can construct logical topologies on top of the physical network by providing multiple virtual channels on each physical link. These logical resources may be employed to prevent communication deadlocks [26] and improve network throughput [27]. Although virtual channels improve router flexibility, they also affect network speed and implementation complexity [28]; since these trade-offs greatly influence communication performance, pp-mess-sim can vary the number of virtual channels in the network. In addition, multicomputer networks can use virtual channels to separate traffic with different characteristics or performance requirements. To evaluate such network partitioning schemes, pp-mess-sim can associate each traffic class with a communication pattern, performance metric, and routing-switching policy on a set of virtual channels.

2.2 Switching

Switching schemes have significant influence on router performance and implementation complexity. The switching scheme impacts performance by determining the link and buffer resources a packet consumes at a given node in its route. Traditional *packet switching* requires incoming packets to buffer completely before transmission to a subsequent node can begin. In contrast, cut-through switching schemes, such as *virtual cut-through* [29] and *wormhole* [30], try to forward incoming packets directly to an idle output link. If the outgoing link is busy, virtual cut-through switching buffers the packet, whereas a blocked wormhole packet stalls pending access to the link. While first-generation multicomputers employed packet switching, most contemporary routers utilize cut-through switching for lower latency and reduced buffer space requirements [2].

Wormhole switching achieves low latency without requiring packet buffers, but virtual cut-through and packet switching may achieve larger throughput at high loads. Packet size also impacts switching performance, since multicomputer communication often consists of large data transfers, coupled with small request and acknowledgement packets [7]. A flexible router can accommodate this mixture of traffic by having long packets use wormhole switching to reduce buffer space requirements, while allowing short packets to use virtual cut-through switching to reduce network contention [13]. To study these effects, pp-mess-sim supports virtual cut-through, wormhole, and packet switching, as well as hybrids of these schemes [31], each under a variety of routing algorithms.

2.3 Routing

The routing algorithm determines which nodes a packet traverses to reach its destination. *Oblivious* routing generates a single, deterministic outgoing link for an incoming packet, whereas *adaptive* schemes can incorporate prevailing network conditions into the routing decision. By considering multiple outgoing links, adaptive algorithms can balance network load and increase a packet's chances of cutting through intermediate nodes. Additionally, adaptive schemes may consider *nonminimal* paths in the hope of circumventing network congestion or faulty links. When the 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.

Minimal and non-minimal adaptive routing can reduce end-to-end delay, but out-of-order packet arrival can complicate protocol processing at the receiving node [32]. Opportunities for adaptive routing vary depending on the topology, the distance a packet must travel, and the traffic patterns. A router could balance the trade-off between network latency and depacketization overheads by implementing adaptive routing only for single-packet messages or packets that must visit a large number of intermediate nodes; for these messages, additional routing adaptivity may significantly reduce network latency, outweighing the cost of packet reordering. By allowing multiple routing-switching policies to coexist in the simulated network, pp-mess-sim facilitates experimentation with flexible router designs that tailor their routing-switching policies to application performance trade-offs.

2.4 Queueing, Arbitration, and Flow Control

While routing and switching determine how each packet flows through the network, the router at each node determines how the individual link and buffer resources are accessed. The router's internal policies for queueing, arbitration, and flow control affect network performance and implementation complexity by coordinating resource sharing amongst competing packets. A bottleneck in the router design may limit achievable network throughput, potentially outweighing the effects of the topology or the communication patterns. Thus, a crucial aspect of interconnection network design is determining the size, speed, and structure of internal components. The simulator decouples these router policies from the network topology, routing-switching schemes, communication workloads, and data collection to enable a broad range of experiments on different router models. Within the router models, pp-mess-sim represents internal components as separate simulation modules, connected by generic flow-control and arbitration models.

A particular router design may queue packets at the input links, the output links, and the interface to the local node. Depending on the structure and placement of these buffers, packets may incur significant queueing delay [33]. When several queues vie for a resource, the router invokes an arbitration policy, such as round-robin or a priority-based scheme, to select the winner. The arbitration policy may differentiate between application traffic classes to assign priority to more urgent packets. Closely tied to both queueing and arbitration is flow control, which affects latency and throughput by limiting the

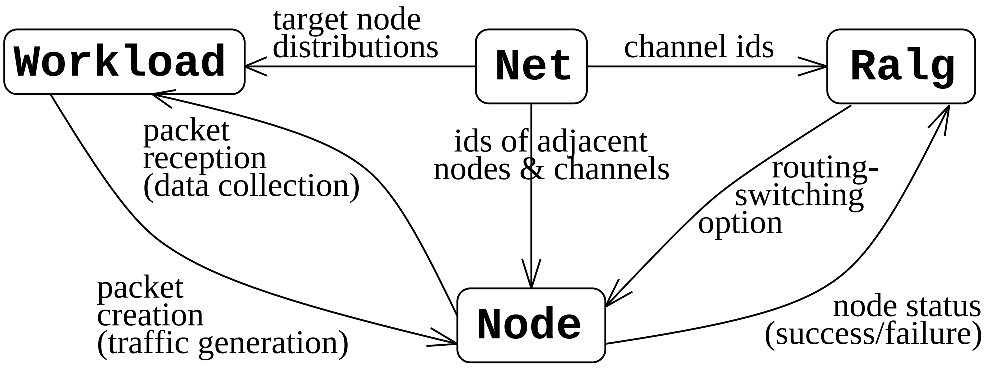


Figure 1: Structure of pp-mess-sim

rate at which packets travel through the network. Flow control can occur anywhere from the byte level at the physical link to the message level in the software and can influence both communication latency and network throughput. For example, wormhole routers typically divide the packets of a message into small flow control units (flits); packets on different virtual channels share the physical bandwidth, based on flit-level link arbitration [27].

3 Simulator Structure

As shown in Figure 1, pp-mess-sim's structure reflects the important architectural issues outlined in Section 2. 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. The simulator also defines a specification language for composing complex experiments, with a variety of traffic patterns and network policies.

3.1 Simulator Components and Interfaces

The main components of the simulator 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 1. The arrows in the figure highlight the interaction between the pp-mess-sim components. By identifying the types of information that each module needs from the other modules, and by carefully defining the interface for accessing this information, components can be independently developed without sacrificing flexibility. Thus, the simulator can easily incorporate new topologies, routing algorithms, router models, traffic patterns, and data collection routines. As long as new modules use the well-defined interfaces, they can interoperate with the pre-existing modules.

By handling all event flow in the simulator, the **Node** module encapsulates all information about the internal operation of the router. For example, the **Ralg** module uses a routing instruction interface to pass **Node** a series of instructions for it to execute; the **Node** executes these instructions and passes back the status of each one back through the same interface. In the same manner, **Workload** composes traffic patterns by mapping diverse application "tasks" onto the simulated network and handles the details of packet generation and data collection. The **Net** class insulates other modules from the details of the specific topology, by providing an interface for other modules to identify and translate node addresses, link identifiers, and virtual channels. Each router receives new packets from the

```
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 time_constr 1;
11 - end
12 -
13 - task default begin
14 -   arrival NegativeExpntl(400.00);
15 -   length  Discrete(0.7,16,0.3,512);
16 -   target  NodeUniform();
17 -   routing_spec begin
18 -     routing wh_oblivious(1,2);
19 -     order dimorder;
20 -   end
21 -   history latency;
22 -   packets 2000;
23 -   drop 200;
24 - end
25 - task time_constr begin
26 -   arrival Uniform(100,100);
27 -   length  Uniform(10,10);
28 -   target HopUniform(0.2,0.7,0.1);
29 -   target_stack params(0.5,0.2,0.1);
30 -   routing_spec begin
31 -     routing ps_oblivious(0);
32 -     order random;
33 -   end
34 -   history histogram(0,1000,50);
35 -   packets 2000;
36 -   drop 200;
37 - end
38 -
39 - general begin
40 -   random seed 1353625084;
41 -   parameter RX::ack_xmit_time 1;
42 -   parameter CTBus::arbiter priority_crossbar;
43 -   output    e4_mix_400.00.out;
44 -   errors    e4_mix_400.00.err;
45 -   results   e4_mix_400.00.results;
46 -   debug     e4_mix_400.00.debug;
47 - end
```

Figure 2: Example simulation specification

Workload and in-transit packets from adjacent nodes, with no dependence on the network topology, communication patterns, or internal router policies at other nodes.

3.2 Input Specification

A separate **Spec** module encapsulates all functions related to experiment specification. In order to evaluate diverse network architectures, under complex traffic patterns, **Spec** interprets a high-level language that can represent a wide range of simulation experiments, as shown in Figure 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 2 specifies an 8-ary 2-cube (8×8 torus) network that carries a mixture of time-constrained and best-effort traffic, with different traffic patterns, performance metrics, and network policies [12, 34].

The input specification file includes "task" blocks to define the communication workload, routing-switching algorithm, and performance metric for each traffic class. As shown in lines 18 and 31 of Figure 2, the best-effort traffic employs two virtual channels (channels 1 and 2) for wormhole routing while the time-constrained packets use packet switching and oblivious routing on a single virtual channel (channel 0). The network assigns these two traffic classes to separate virtual channels to minimize interference and to assign priority to the time-constrained packets. For example, line 42 selects a priority arbiter to govern access to the outgoing links; this arbiter is written to favor virtual channel 0 over the two wormhole virtual channels in order to better serve the time-constrained traffic.

The simulator defines a flexible, string-mapping mechanism for assigning internal router policies. In parsing the input file, the simulator creates an entry in an associative string map, with the key

"CTBus::arbiter" and value "priority_crossbar." As nodes are created, pp-mess-sim queries the string map to retrieve the parameter values; if no string is present the parameter is initialized with a default value. To provide more control over router features, the string parameters can identify specific nodes or devices (virtual channels) in the network. For example, "node(10)::dev(8)::TX::xmit_time 100" would assign a large, 100 cycle, transmission delay for outgoing virtual channel 8 at node 10. Parameters without node and device numbers apply to all nodes and devices, as in lines 41 and 42 in Figure 2. This flexibility enables pp-mess-sim to model heterogeneous and even faulty networks, with a range of link speeds and router features.

4 Router Models

By defining strict interfaces between individual parts of the code, pp-mess-sim insulates the **Node** module from the **Net**, **Ralg**, and **Workload** modules. This extensible framework allows the user to develop and evaluate a variety of router models without changing any of the other pp-mess-sim modules. In addition, pp-mess-sim introduces several useful abstractions for representing flow control and resource arbitration within the router models.

4.1 Node Modules

The simulator includes a cycle-level model of the PRC [3, 16], a *programmable routing controller* for point-to-point distributed systems. As shown in Figure 3, the PRC coordinates bidirectional communication with up to four neighboring nodes, with three virtual channels on each physical link; the corresponding pp-mess-sim **Node** model includes simulation modules for each of the router's main components, as shown in Figure 4. The host transmits a packet by feeding one or more page tags to a *transmitter fetch unit* (TFU), where each tag includes a memory address and the number of words to transmit. Similarly, the host processor supplies each *network interface receiver* (NIRX) with pointers to free pages in the buffer memory, for use by arriving packets. To represent software and system overheads, the pp-mess-sim model captures the details of the page-level interaction with the host processor; this includes the control and data transfer delays associated with moving pages to/from the router's off-chip buffer memory.

The PRC treats the outgoing virtual channels, the *network interface transmitters* (NITXs), as individually reservable resources, allowing the device to support a variety of routing and switching schemes through flexible control over channel allocation policies. The demand-slotted *cut-through bus* (CTBUS) and the *reservation status unit* coordinate the low-level interaction between the incoming and outgoing virtual channels on a word-by-word basis. Each incoming link has a microprogrammable *routing engine* that can interpret the wide variety of header formats used in routing-switching algorithms; an arriving header could indicate the packet's destination node(s), or a relative address(es) for the destination(s), as well as the packet's traffic type or priority. The microcode can invoke different routing-switching algorithms based on the packet header fields, allowing the PRC to support multiple network policies simultaneously.

By capturing the low-level timing details for flow control, resource arbitration, and microcode execution, the PRC **Node** model enables precise simulation experiments for tuning interface speeds and flow-control policies. However, efficiently exploring alternative architectures requires more versatile models. Hence, pp-mess-sim includes a configurable, high-level model for examining techniques and performance trends in router architecture. This "virtual" router (v-router) supports various queueing, arbitration, and flow-control policies. Since the v-router model captures less detail than the PRC model, v-router experiments can efficiently consider a broader range of simulation parameters before testing

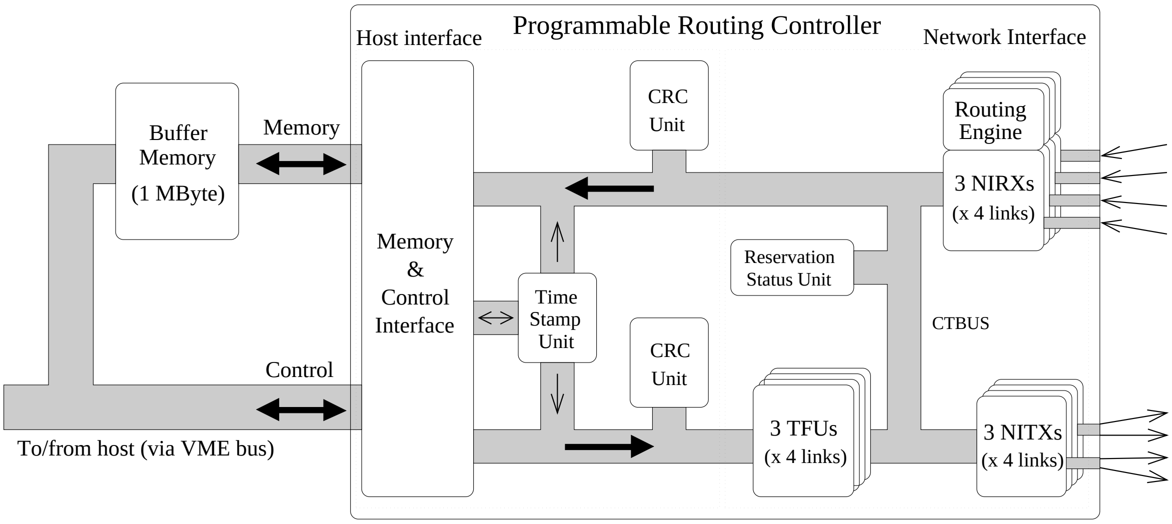


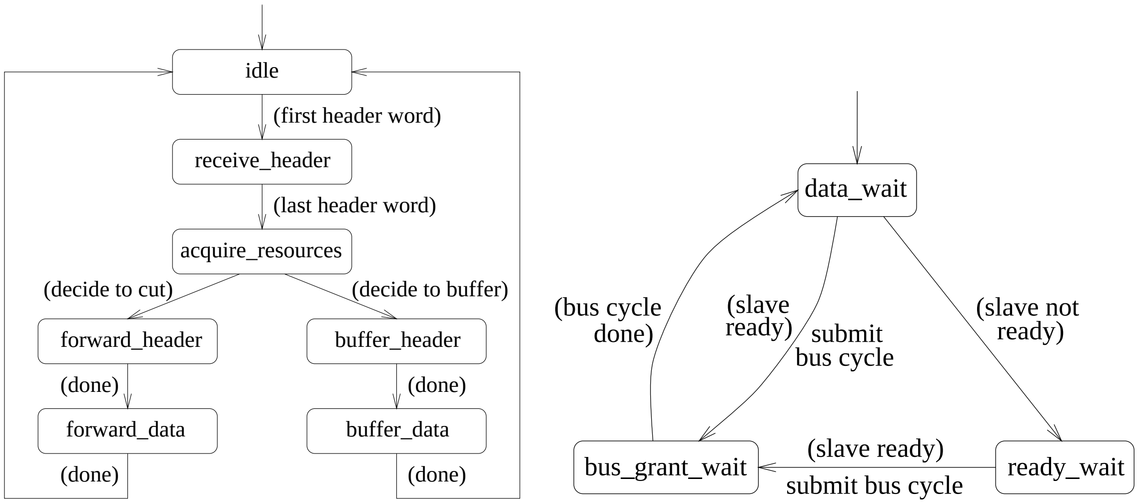
Figure 3: PRC architecture

```
class PRC : public Node {
  NodeId     id;             // Identifier of node
  DevId      max_dev_id;     // Number of virtual channels
  NodeStat   node_stat;      // Data collection for Node

  HOST       host;           // Local host processor
  TFU*       tfu;            // TFU transmission ports: tfu[max_dev_id]
  NIRX*      nirx;           // NIRX reception channel: nirx[max_dev_id]
  NITX*      nitx;           // NITX transmission channel: nitx[max_dev_id]
  Link*      link;           // Physical links: link[max_direction]

  CTBus      ctbus;          // Cut-through bus to network interface
  TXBus      txbus;          // Bus from memory interface to TFUs
  RXHost     rxhost;         // Data interface to host
  PCBus      pcbus;          // Control interface
}
```

Figure 4: Internal components in the PRC **Node** model



(a) Receiver state machine

(b) Bus interface control

Figure 5: **Node** state machines

specific options on a more detailed model. For virtual cut-through and packet switching experiments, the v-router simulations often execute an order of magnitude faster than comparable PRC simulations by modeling only the head and tail flits in each packet; wormhole switching experiments typically execute 50% faster, since the v-router does not model low-level timing details, such as the delay to reserve access to an idle virtual channel.

4.2 Router Components

Similar to behavioral hardware description languages, pp-mess-sim represents each router component as a state machine, where simulation events trigger each state transition. For example, Figure 5(a) shows a state machine for an incoming virtual channel. The channel remains idle until the incoming link signals the arrival of a header word 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 5. 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

Each of these steps 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 5(b). For example, in the PRC model, an incoming virtual channel (NIRX) waits for arriving data before forwarding the new word across the CTBUS to an outgoing virtual channel (or the memory interface). However, this transfer cannot occur until the slave device (NITX) has sufficient buffer space; then, the NIRX can request a slot on the CTBUS to forward the word to the NITX. Similarly, the NITX 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 5(b) 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, pp-mess-sim allows the incoming channel to register a pending simulation event in the outgoing channel's wake-up queue. Once the outgoing channel becomes available, pp-mess-sim drains the wake-up queue and inserts the entries into the simulator's 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 pp-mess-sim **Node** models introduce useful abstractions for representing arbitration for internal resources, such as buses. When multiple modules can compete for access to a resource, each module's state transitions depend on the behavior of other components, as well as the arbitration policy. To insulate each module from these details, pp-mess-sim represents each shared resource with an arbiter model that can register pending simulation events. For example, in Figure 5(b), an incoming virtual channel requests access to the bus by submitting its bus-cycle event to the arbiter, instead of the simulator's main event queue. 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. The event handler implements the arbitration policy, determining which registered events should be transferred to the main event queue for subsequent execution. This flexible framework allows pp-mess-sim users to add 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 virtual-channel crossbars. Unlike bus models, crossbar models can transfer *multiple* events to the simulator's event queue in a single arbitration cycle. By changing the order the arbiter scans pending events, the v-router can evaluate priority-based arbitration schemes, as shown in line 42 of Figure 2.

Router Statistics

Evaluating a router design requires detailed information about how internal features perform in operation; this information can then help pinpoint weaknesses and bottlenecks in the design. Although the **Workload** module computes performance statistics for arriving packets, this approach is unnatural for capturing fine-grain information about resource usage in the router model. Consequently, each pp-mess-sim **Node** model accumulates statistics for its outgoing channels, internal buses, and packet buffers. By maintaining separate performance data at each node, the user can investigate the impact of non-uniform communication patterns on resource usage across all nodes in the network. Each **Node** module updates these statistics whenever a simulation event models access to an internal resource; for example, if an event models the use of an internal bus, the **Node** updates its count of active bus cycles. Using these statistics, pp-mess-sim can gather detailed information about network performance.

5 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

```
class Ralg {
  // Construct packet's initial routing instruction at source or intermediate node
  void i_source_inject(PacketPtr, NodePtr);
  void i_net_inject(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
  short     current_devop;   // Place holder for sequencing through virtual channels
}
```

Figure 6: **Ralg** routines for interacting with **Node** routing-switching state machine

the router models (**Node**) from the routing-switching algorithms (**Ralg**), as shown in Figure 1. This functional separation allows the user to easily prototype new routing-switching algorithms without changing the **Node** models. By associating a routing-switching algorithm with each **Workload** task, pp-mess-sim can allow multiple policies to coexist in the simulated network.

5.1 Routing and Switching

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 5(a). When a packet arrives from a host injection port or an incoming link the router parses the header bytes 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* 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.

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 6. 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** examines each instruction and determines whether or not the output channel(s) can satisfy the request. If necessary, the **Node** tries to reserve channel or buffer resources to successfully complete the operation; this process may involve multiple simulation events and, perhaps, advancement in simulation time. The algorithm and the router model continue this request-response handshake until they agree on a common routing-switching decision. Additional support in the **Net** module insulates the algorithms from the details of the network topology, as discussed in Section 6.

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. **Node** simulation events

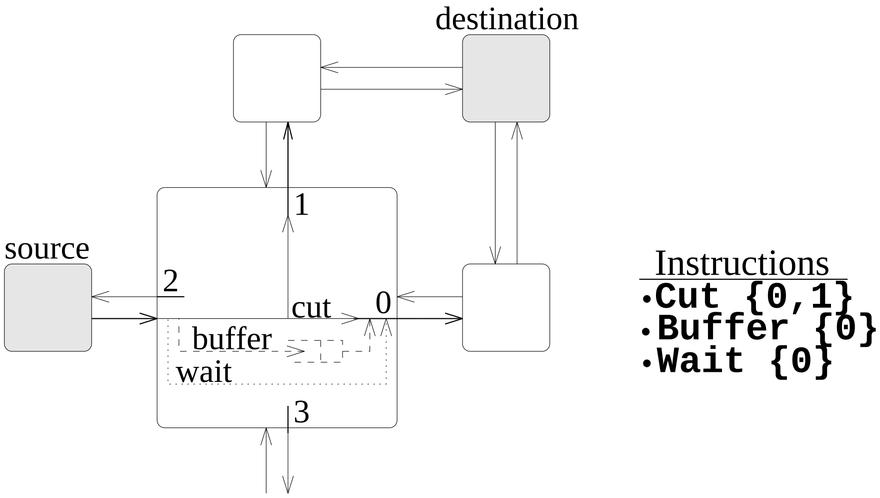


Figure 7: Sequence of routing-switching instructions

distinguish the architectural and timing details in different router models. For example, the PRC and v-router models differ in how they acquire link and buffer resources for packets. The PRC has a central reservation unit for assigning outgoing virtual channels to incoming packets. Hence, the PRC **Node** module proceeds through multiple simulation events to gain access this unit and reserve virtual channels; in contrast, the v-router module assumes that reserving idle channels does not incur any delay. Using the **Ralg** instruction set, pp-mess-sim can easily incorporate additional routing algorithms and switching schemes, without altering the **Node** models.

5.2 Routing-Switching Algorithms

For example, Figure 7 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 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 7. These hybrid switching schemes dynamically balance the use of channel and

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

Table 1: Examples of routing-switching schemes in pp-mess-sim

memory resources for "storing" blocked packets. For example, the h-hop hybrid algorithm in Table 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 [31]. This algorithm limits channel contention, while still restricting the use of packet buffers.

The **Ralg** instructions can also implement a variety of routing algorithms by generating different lists of candidate virtual channels. As shown in Table 1, pp-mess-sim includes a variety of oblivious and adaptive routing algorithms for the different switching schemes. The simulator uses Duato's theory [35] to construct deadlock-free adaptive routing algorithms under wormhole switching. Each algorithm requires a minimum number of virtual channels 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 18 and 31 of Figure 2. By specifying the same virtual channels, algorithms can also share virtual channels between them. 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.

6 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 8, 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. The **Net** functions also assist the **Workload** class in constructing traffic patterns.

6.1 Selection Functions

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 7, **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 a variety of selection functions for ranking the candi-

```
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 8: Internal components in the **Net** class

date outgoing links. These selection functions, coupled with the **Ralg** routing-switching instructions, enable pp-mess-sim to model a wide range of communication policies on a variety of network topologies.

In line 19 of Figure 2, the default best-effort traffic is assigned *dimension-ordered* routing, which requires a packet to complete all hops in one direction before proceeding to the next dimension; in contrast, line 32 specifies that time-constrained packets consider their minimal-path output links in a random order. In addition to these two selection functions, **Net** can rank outgoing links according to how much further the packet must travel in each direction; this link ordering improves the packet's chance of considering multiple outgoing links at future nodes in its route [36, 37]. For example, if a packet is traveling from node $(4, 3)$ to node $(6, 10)$ in a square mesh, the diagonal selection function would recommend traveling in the y-direction, instead of the x-direction, if both links are idle. Another selection function ranks output links according to network congestion, giving preference to links with fewer busy virtual channels [38]; this balances traffic load amongst the outgoing links, reducing contention and packet delay.

6.2 Mapping Functions

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 8, **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 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 a variety of simulation experiments that evaluate a router design under

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

Table 2: Destination node distributions in pp-mess-sim

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 2. Since these distributions depend on the underlying numbering scheme for each 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 underlying topology also affects the proximity of communicating nodes. To minimize packet latency and network throughput requirements, multicomputer applications often place communicating tasks near each other in the network. To capture these spheres of communication locality, pp-mess-sim includes a hop-uniform distribution, as shown in line 28 of Figure 2. In this example, 20% of packets have destinations just one hop away, while 70% travel two hops, and the remaining packets traverse three hops. In order to support this distribution, the **Net** class instantiates a hop table as shown in Figure 8; this multidimensional array, constructed at run-time, keeps track of which nodes are within a certain distance of other nodes. The **Workload** module consults this table to compute a packet's destination, based on the source node, without any dependency on the specific network topology.

7 Communication Workloads

Network traffic patterns and performance requirements vary significantly across different applications. Hence, pp-mess-sim provides flexible support for generating communication patterns and collecting performance statistics. The **Workload** module insulates the rest of the simulator from the details of the traffic generation and data collection by handling all functions related to packet creation and reception. Flexible composition of tasks can generate complex network workloads, with multiple traffic classes, while history-list data collection allows the user to define different performance metrics for each class.

7.1 Task Mapping

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

```
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 9: **Workload** Task model

behavior. For example, lines 8–11 of Figure 2 instantiate a time-constrained task and a "default" best-effort task on each node. 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 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, 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 9. **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. By isolating creation times in the task model, pp-mess-sim allows the user to incorporate new packet generation schemes, including multi-state models, without affecting the rest of the simulator. By using these generic functions, pp-mess-sim can easily be extended to run in 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.

7.2 Traffic Models

Analytical studies of multicomputer networks have typically modeled packet length, interarrival time, and destination node distributions as simple stochastic processes. For example, packet arrivals have commonly been modeled as a Poisson process, with exponentially-distributed interarrival times. Detailed measurements of multicomputer applications, however, have led to more sophisticated traffic generation models [6, 7]. In order to simulate realistic workloads, pp-mess-sim provides a rich set of

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)

Table 3: Packet length and interarrival distributions in pp-mess-sim

packet length, interarrival time, and destination node distributions[1], as shown in Tables 2 and 3. Since many network protocols enforce limits on packet size, the length distributions may be trimmed to enforce upper and lower bounds on packet length.

In Figure 2 (line 15), the best-effort packets use the discrete distribution to generate packet lengths. In this case, 70% of the best-effort packets are short, while the remaining are long; such *bimodal* distributions are common in multicomputer applications [7]. Packet interarrival times also stem from a variety of distributions, as shown in Table 3. For example, in Figure 2, time-constrained tasks create periodic (line 26), fixed-length (line 27) packets, while default best-effort tasks generate packets according to a Poisson process (line 14). Parallel applications also generate bursty network traffic, due to multi-packet messages and fine-grain handshaking between cooperating nodes. The **Workload** module can generate bursty traffic by using a two-stage normal distribution, where the smaller mean represents the spacing between packets within a burst and the larger mean represents the spacing between bursts; the burst length depends on the probability p of selecting a packet's interarrival time from the first normal distribution.

As shown in Section 6, pp-mess-sim implements several destination node distributions ranging from the simple node-uniform pattern to permutations derived from scientific computations. While these distributions capture specific traffic patterns which can exist in the network, many multicomputer applications exhibit temporal communication locality, where a node sends several messages to the same destination over a small time interval. The simulator captures this effect through a *most-recently used* stack model for selecting destination nodes [6]. For example, line 29 of Figure 2 declares a three-element stack for each time-constrained task. Half of the time, a new packet selects its destination from the top of the stack; the second and third destinations in the stack are chosen 20% and 10% of the time, respectively. The task randomly selects a packet's destination node from the hop-uniform distribution (in line 28 of Figure 2) the remaining 20% of the time. The destination stack, coupled with hop-uniform communication, generates a unique combination of spatial and temporal locality in the simulated network.

[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) [39] 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 40 in Figure 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 [40]. If a process exhausts its chunk, the next unused chunk is allocated from the original stream.

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

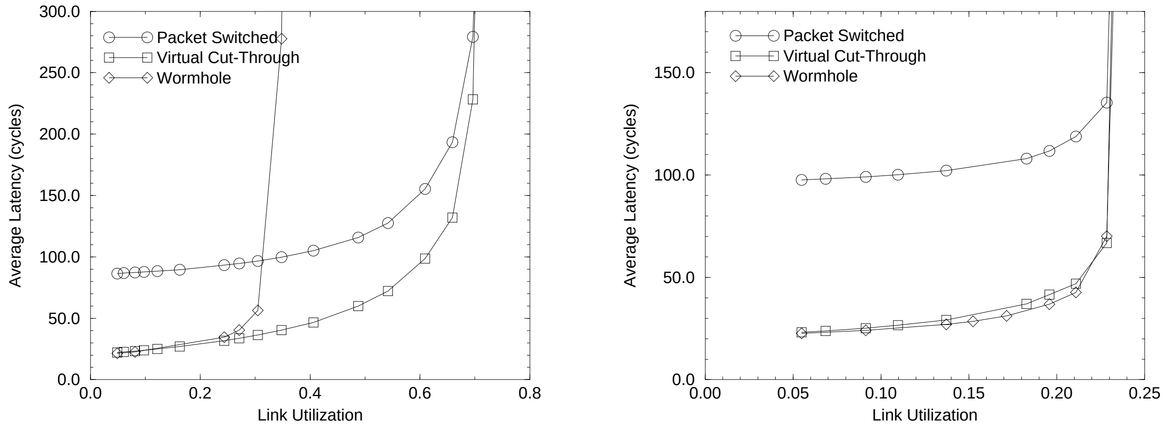
7.3 Packet History List

Similarly, pp-mess-sim provides effective data collection by associating performance metrics with the task construct, as shown in Figure 9. These performance statistics also allow the user to study various communication patterns with different 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 23 and 36 of Figure 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 22 and 35 of Figure 2); the task continues to generate packets until every 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. 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 2 the time-constrained tasks capture a histogram of latency data to estimate the probability distribution of packet delay (line 34), while the best-effort tasks collect basic latency metrics (line 21), 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, using statistics on packet cut-through history, we have investigated the effects of inter-node dependencies on the performance of cut-through networks by comparing simulation results with analytical models [41]. 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.



(a) Node uniform traffic (b) Matrix transpose traffic

Figure 10: Comparing switching schemes under dimension-ordered routing

8 Simulation Experiments

The simulator's flexibility enables a broad range of experiments for evaluating multicomputer router designs. As a result, several studies have used pp-mess-sim to evaluate the PRC and the v-router models under various routing-switching schemes, network topologies, and application workloads [3, 4, 12, 31, 34, 42–44]. This section presents the results of sample pp-mess-sim experiments that consider the interaction of routing-switching algorithms and communication patterns. Additional experiments show that multicomputer networks can support a diverse mixture of application workloads by implementing multiple routing-switching schemes on different virtual channels.

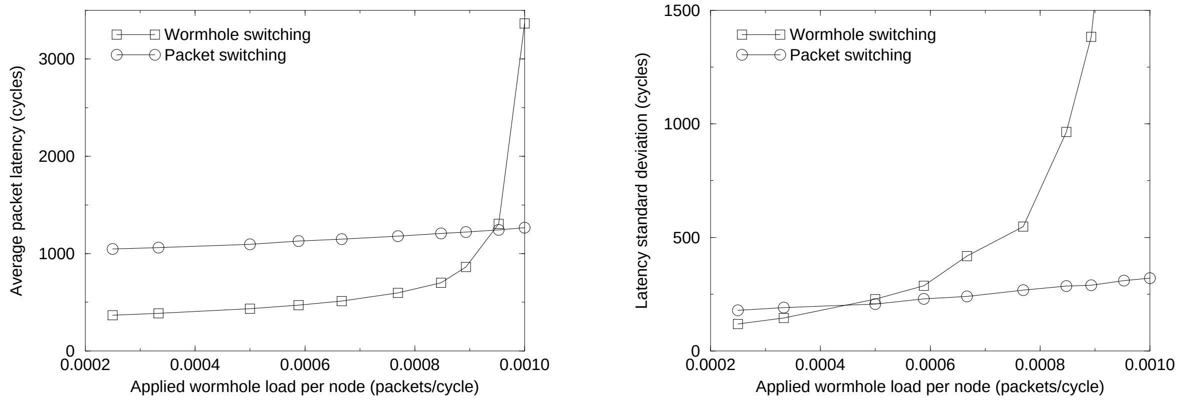
8.1 Switching Schemes

In defining how packets flow between nodes, the various switching schemes stress different resources along a packet's route. This section evaluates the use of packet buffers and virtual channels to relieve congestion in the interconnection network.

Node-Uniform and Matrix-Transpose Traffic

Figure 10 shows the performance of various switching schemes using static dimension-ordered routing on an 8×8 square mesh; each node independently generates 16-word packets with exponentially-distributed interarrival times. The graphs plot the mean packet latency as a function of the link utilization for node-uniform and matrix-transpose traffic patterns. As expected, virtual cut-through switching consistently outperforms packet switching, since virtual cut-through packets often avoid buffering delay at intermediate nodes; at high loads, virtual cut-through and packet switching performance gradually merge, as high network utilization decreases the likelihood that an in-transit packet encounters an idle output link. At low loads, wormhole switching performs extremely well for both traffic patterns, even though blocked packets stall in the network.

However, the relative performance of virtual cut-through and wormhole switching varies significantly between Figure 10(a) and 10(b). Under node-uniform traffic, the two switching schemes exhibit comparable performance at low loads; however, network contention limits wormhole throughput at



(a) Average latency (all packets) (b) Latency standard deviation (5-hop packets)

Figure 11: Mixing wormhole and packet switching under dimension-ordered routing

higher loads. 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 until its outgoing channel becomes available; this stalled packet may then block other traffic destined for different output links. 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.

Despite channel contention effects, wormhole switching excels for the matrix-transpose permutation, as shown in Figure 10(b). This occurs because matrix-transpose traffic, coupled with dimension-ordered routing, limits contention between packets heading to different parts of the network. In a square mesh, the matrix-transpose permutation requires node (c, d) to communicate with node (d, c). With dimension-ordered routing, each packet starting on row d proceeds in the x-direction to node (d, d), before traveling in the y-direction to reach the destination node. As a result, source nodes in row d inject packets that use the same row and column links. Although a blocked wormhole packet may still restrict other traffic from entering a node, this traffic must ultimately traverse the same links as the stalled packet; buffering the blocked packet cannot alleviate this contention. Neither wormhole nor virtual cut-through switching performs best in all situations, suggesting that router designs can incorporate the salient features of the both schemes [31]; pp-mess-sim's **Ralg** support facilitates experimentation with such hybrid algorithms.

Mixing Wormhole and Packet Switching

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. By removing blocked packets from the network, packet switching can improve network throughput and reduce the variability in packet delay. While packet switching forms physical queues at each outgoing link, blocked wormhole packets form logical queues that span multiple nodes. These decentralized queues complicate resource allocation and scheduling, while avoiding the cost of packet buffers in the router. As described in Figure 2, a flexible router can implement packet switching for predictable communication, while permitting best-effort traffic to employ wormhole switching [12, 34].

Figure 11 shows results from a similar traffic mixing experiment, using the PRC **Node** model in a

6×6 torus network. Each node injects 64-byte packets with uniform random selection of destination nodes and dimension-ordered routing. The experiment varies the injection rate for best-effort traffic, while generating a time-constrained packet every 1500 cycles at each node. With three virtual channels on each physical link, the experiment assigns two virtual channels to deadlock-free wormhole routing for best-effort packets, while dedicating the remaining virtual channel to time-constrained, packet-switched traffic. As shown in Figure 11, best-effort packets incur increased latency and delay variance as the traffic load increases. This contention does not harm the time-constrained traffic, since blocked best-effort packets temporarily stall in their own virtual network instead of consuming link and buffer resources at intermediate nodes.

The average latency and predictability of time-constrained packets are largely unaffected by the best-effort load, due to fine-grain arbitration amongst the virtual channels. The router can further reduce the interference of best-effort traffic by imposing priority arbitration amongst the virtual channels [34], as shown in Figure 2. For a time-constrained packet, this effectively provides flit-level preemption of best-effort traffic across its entire path through the network. These results motivate a router architecture that tailors its low-level routing, switching, arbitration, queueing, and flow-control policies to the conflicting requirements of time-constrained and best-effort communication [17]. The simulator's flexibility and extensibility will permit development of a new **Node** model to compare this router design to other architectures under a wide range of traffic patterns and performance metrics.

8.2 Routing Algorithms

The simulator's **Ralg** instructions and **Net** selection functions enable multi-factor experiments that study the interaction of routing algorithms with the network topology and application traffic patterns. This subsection compares the dimension-order, diagonal, and random selection functions under oblivious and adaptive minimal routing. In addition, the experiment includes a class of deflection algorithms that use a dimension-ordered selection function to select from minimal-path links before considering any nonminimal possibilities; the number of nonminimal hops is restricted by a hop-threshold. The simulations consider the peak achievable network throughput under virtual cut-through switching and a Poissonian packet arrival process at each node; additional experiments with bursty arrivals demonstrate the ability of adaptive routing algorithms to dissipate traffic bursts [4,44]. Finally, the experiments evaluate 16-word packets in a 16×16 square mesh, but simulations with 64-word packets and torus networks show similar trends [44].

Bit-Complement Traffic

Figure 12 shows the peak throughputs of the routing algorithms under bit-complement traffic. This pattern fundamentally congests the center of the network, in both the torus and square mesh topologies. The bit-complement permutation requires source node (c, d) to communicate with node $(15-c, 15-d)$; as a result, all packets must eventually cross both the middle row and the middle column of the mesh, irrespective of the routing algorithm. As the figure shows, the selection function plays a role in determining the achievable network throughput. The diagonal minimal algorithm performs the worst since it tends to route packets towards the center of the network; in contrast, the dimension-ordered selection function performs well by keeping packets closer to the periphery of the mesh.

Adding adaptivity to the the dimension-ordered algorithm actually *harms* performance since the adaptive algorithm mistakenly tries 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. This effect is worst in a square mesh and on larger networks, where the

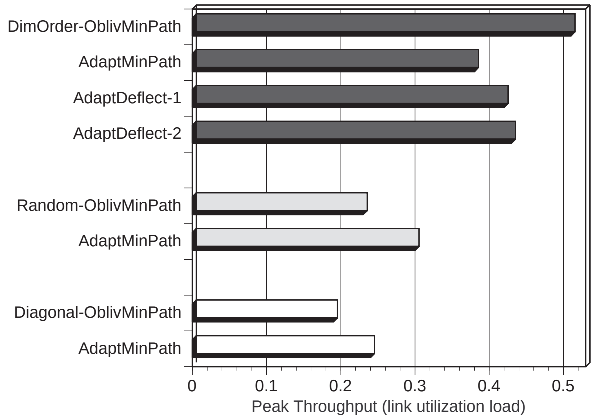


Figure 12: Peak throughput in a 16×16 square mesh with bit-complement traffic

regions of congestion are magnified. Adaptivity does help improve performance of algorithms which use the random and diagonal selection functions, however. Since these selection functions tend to direct traffic into the center of the network, adaptivity helps by allowing the algorithm to avoid creating large regions of congestion. Deflection algorithms also perform well under this traffic pattern since they are able to circumvent the regions of congestion and correct mistakes made earlier in the route.

Bit-Reversal Traffic

Figure 13 shows the peak throughputs of the routing algorithms under a bit-reversal traffic pattern. In contrast to the bit-complement experiment, the static dimension-ordered routing algorithm performs poorly under bit-reversal traffic since packets cannot circumvent regions of congestion to utilize many of the lightly-loaded links in the network. Adding adaptivity to this algorithm, however, still does not improve performance significantly as shown by the low peak throughputs for the adaptive dimension-ordered algorithms. This indicates that just as the diagonal selection function is pathologically bad for bit-complement traffic, the dimension-ordered selection function is pathologically bad for bit-reversal traffic. While changing selection functions improves performance for the static routing algorithms, it is not enough to obtain peak performance. As shown in Figure 13, adding adaptivity to the static random and diagonal routing algorithms improves performance considerably.

It is interesting to note that the adaptive diagonal algorithm saturates at a higher peak throughput than the adaptive random algorithm, while the reverse is true for their oblivious counterparts. For the adaptive algorithms, the diagonal selection function attempts to increase a packet's chance of having multiple routing options at later hops in the route. As a result, the diagonal routing algorithm increases the likelihood of avoiding network congestion and establishing packet cut-throughs, improving performance over the adaptive random scheme. Since the diagonal selection function performed the best out of the adaptive minimal routing algorithms, additional experiments using a diagonal deflection algorithm were also performed; as Figure 13 shows, the extra adaptivity makes little difference. In contrast, deflections improve performance for the dimension-ordered routing algorithm by allowing packets to route around areas of congestion caused by the poorly performing selection function.

Tailoring Experiments

The multi-factor experiments demonstrate the value of tailoring routing policies to application traffic patterns. For routers which support multiple routing schemes, these results serve as a guide for selecting

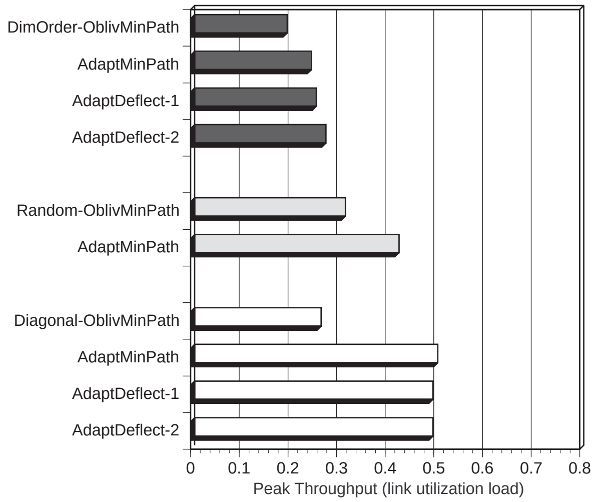


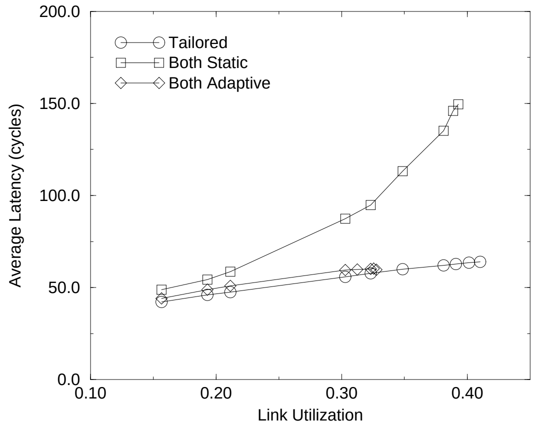
Figure 13: Peak throughput in a 16×16 square mesh with bit-reversal traffic

an appropriate routing algorithm based on the application workload. For networks which only support a single routing scheme, these results can influence how the operating system maps tasks onto different nodes in the network, in effect, tailoring the communication workload to the routing algorithm. In a multi-user environment, where multiple applications execute at the same time, supporting multiple routing schemes *simultaneously* can significantly improve performance. This subsection considers a set of experiments that mix bit-reversal and bit-complement traffic in an 8×8 square mesh. The experiments evaluate a wormhole network with four virtual channels on each link; the two tasks route messages on separate pairs of virtual channels to partially insulate the applications from each other.

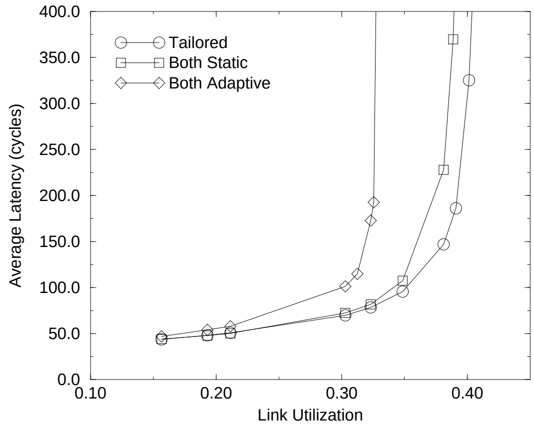
The graphs show average packet latency for both traffic patterns, under increasing bit-complement load; the bit-reversal pattern remains fixed at a link load of 0.12. According to Figure 12 and Figure 13, bit-complement has peak performance under static dimension-ordered routing, while the bit-reversal pattern performs best under the diagonal minimal-path algorithm. As shown in Figure 14(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 14(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 simulator's **Ralg** and **Workload** support facilitate such experimentation by allowing multiple routing-switching schemes and traffic patterns to coexist in the underlying network.

9 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. Well-defined interfaces between the simulator components create an extensible environment that enables independent enhancements to the code. As part of ongoing work, additional features are continually added to the simulator. In particular, we are extending **Workload** to generate more realistic communication patterns through



(a) Bit-reversal traffic



(b) Bit-complement traffic

Figure 14: Average latency under traffic mixing

the use of complex arrival processes, application traces, and accurate communication models. These options will complement the existing probability distributions for packet length, inter-arrival times, and target destination nodes.

We are also capitalizing on the **Ralg** instruction set, and the **Net** selection functions, to construct new routing-switching algorithms that address application characteristics and performance requirements. Using pp-mess-sim's flexible mechanisms for specifying and evaluating complex network configurations, we are experimenting with how to effectively combine multiple routing-switching policies in the underlying network. To represent more complex routing and switching algorithms, we are extending the **Ralg** instruction set to support multicast routing algorithms and to interact more closely with the queueing models in the **Node** modules.

To study general router design issues, we are extending the v-router **Node** module to allow more control over internal router organization. With a diverse library of arbiters and buffer architectures, the v-router could construct a wider range of candidate router designs. Ultimately, multicomputer performance depends on the interaction of these internal router policies with the network topology, routing-switching policies, and application workloads. Drawing on the **Net**, **Ralg**, and **Workload** support, pp-mess-sim users could then compare candidate router architectures under the same network policies and application demands[2].

References

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

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

[2] For more information about the availability of pp-mess-sim, please see our web page at http://www.eecs.umich.edu/RTCL/

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

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

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

[6] J.-M. Hsu and P. Banerjee, "Performance measurement and trace driven simulation of parallel CAD and numeric applications on a hypercube multicomputer," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, pp. 451–464, July 1992.

[7] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "Architectural requirements of parallel scientific applications with explicit communication," in *Proc. Int'l Symposium on Computer Architecture*, pp. 2–13, May 1993.

[8] M. G. Norman and P. Thanisch, "Models of machines and computation for mapping in multicomputers," *ACM Computing Surveys*, vol. 25, pp. 263–302, September 1993.

[9] V. M. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M. A. Mohamed, B. Nitzberg, J. A. Telle, and X. Zhong, "OREGAMI: Tools for mapping parallel computations to parallel architectures," *International Journal of Parallel Programming*, vol. 20, pp. 237–270, June 1991.

[10] F. Hady and D. Smitley, "Adaptive vs. non-adaptive routing: An application driven case study," Tech. Rep. SRC-TR-93-099, Supercomputing Research Center, Bowie, Maryland, March 1993.

[11] J. H. Kim and A. A. Chien, "Evaluation of wormhole routed networks under hybrid traffic loads," in *Proc. Hawaii Int'l Conf. on System Sciences*, pp. 276–285, January 1993.

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

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

[14] D. Smitley, F. Hady, and D. Burns, "Hnet: A high-performance network evaluation testbed," Tech. Rep. SRC-TR-91-049, Supercomputing Research Center, Institute for Defense Analyses, December 1991.

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

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

[17] J. Rexford, J. Hall, and K. G. Shin, "A router architecture for real-time point-to-point networks," in *Proc. International Symposium on Computer Architecture*, pp. 237–246, May 1996.

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

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

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

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

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

[23] K. Bolding, S.-E. Choi, and M. Fulgham, "The Chaos router simulator." Presentation at *Parallel Computer Routing and Communication Workshop*, May 1994.

[24] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Trans. Computers*, vol. 39, pp. 775–785, June 1990.

[25] A. Agarwal, "Limits on interconnection network performance," *IEEE Trans. Parallel and Distributed Systems*, vol. 2, pp. 398–412, October 1991.

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

[27] W. Dally, "Virtual-channel flow control," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, pp. 194–205, March 1992.

[28] A. A. Chien, "A cost and speed model for k-ary n-cube wormhole routers," in *Proc. Hot Interconnects*, August 1993.

[29] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, September 1979.

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

[31] K. G. Shin and S. Daniel, "Analysis and implementation of hybrid switching," in *Proc. International Symposium on Computer Architecture*, pp. 211–219, June 1995. Extended version to appear in *IEEE Trans. on Computers*.

[32] V. Karamcheti and A. A. Chien, "Software overhead in messaging layers: Where does the time go?," in *Proc. Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 51–60, October 1994.

[33] M. G. Hluchyj and M. J. Karol, "Queueing in high-performance packet switching," *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 1587–1597, December 1988.

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

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

[36] H. G. Badr and S. Podar, "An optimal shortest-path routing policy for network computers with regular mesh-connected topologies," *IEEE Trans. Computers*, vol. C-38, pp. 1362–1370, October 1989.

[37] A. L. Davis, "Mayfly: A general-purpose, scalable, parallel processing architecture," *Lisp and Symbolic Computation*, vol. 5, pp. 7–47, May 1992.

[38] W. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 466–475, April 1993.

[39] D. E. Knuth, *The Art of Computer Programming, Vol 2: Seminumerical Algorithms.* Addison Wesley, 1st ed., 1969.

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

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

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

[43] 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.

[44] W. Feng and K. G. 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.