PP-MESS-SIM: A Simulator for Evaluating Multicomputer Interconnection Networks *

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

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
{jrexford, jdolter, wuchang, kgshin}@eecs.umich.edu

Abstract

This paper presents pp-mess-sim, an object-oriented discrete-event simulation environment for evaluating multicomputer networks. The simulator provides a toolbox of various network topologies, communication workloads, routing-switching algorithms, and router models. These router models can vary from high-level architectures to low-level specification of actual devices. By decoupling individual parts of the code, pp-mess-sim enables independent code development and creates a flexible and extensible environment for evaluating different aspects of network design. Sample simulation experiments capitalize on this flexibility to compare routing-switching schemes under various application workloads.

1 Introduction

Message-passing multicomputers have emerged as a cost-effective platform for exploiting parallelism in applications. Multicomputers consist of processors joined by an interconnection network, where fast message exchange enables efficient cooperation between processing elements [1, 2]. Router hardware connects an individual processing node to the interconnection fabric and manages traffic flowing through the node en route to other destinations. The router architecture greatly affects the ability of the interconnection network to deliver good communication performance to parallel applications.

*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.

Achieving good overall system performance requires matching application communication requirements with a suitable network design. Networks can employ a wide range of topologies, routing algorithms, switching schemes, and flow control policies. Application characteristics directly impact these design decisions by determining the quantity and frequency of inter-node communication. While modeling provides a cost-effective way to explore design issues, analytical models often impose simplifying assumptions that degrade the accuracy of the evaluation. On the other hand, monitoring an actual system can capture the effects of low-level design choices, but this restricts experimentation with different router policies since it can be prohibitively expensive and time-consuming to change these features.

Instead, an extensible simulation environment can provide an extensible framework for evaluating multicomputer networks. While many simulation toolkits can flexibly model local and wide area networks [3], few simulators sufficiently capture the characteristics of multicomputer networks. In contrast to LANs/WANs, multicomputers typically employ regular network topologies that facilitate efficient, flexible routing schemes. Tighter coupling between nodes enables multicomputer 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 multicomputer networks [4–6].

This paper presents pp-mess-sim (point-to-point message simulator), a flexible simulation environment for evaluating multicomputer routers [7]. Implemented in C++, pp-mess-sim is an object-oriented discrete-event simulator that provides a toolbox of primitives for various network topologies, communication workloads, routing algorithms, and router models. Router models may vary from high-level architectures to low-level specification of actual devices, allowing incremental investigation of implementation approaches and design enhancements. By enforcing strict boundaries between these components, pp-mess-sim facilitates multi-factor experiments that independently explore network design issues.

The next section describes how application traffic patterns impact the suitability of router design options; the components of pp-mess-sim derive directly from these main architectural parameters, as shown in Section 3. The simulator supports a broad spectrum of routing and switching schemes by decoupling them from network topologies and router models, as described in Section 4. Section 5 discusses how pp-mess-sim constructs diverse communication workloads and performance metrics. This framework enables the flexible evaluation of a variety of multicomputer router models, as described in Section 6. Sample simulation experiments capitalize on this flexibility to compare routing-switching combinations under a variety of application workloads. Section 7 concludes the paper with a discussion of future pp-mess-sim enhancements.

2 Motivation

This section overviews the major architectural issues in multicomputer network design to motivate the need for a flexible simulation environment. The selection of these 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 [8, 9]. Many multicomputers employ the k-ary n-cube family of topologies, with k nodes along each of n dimensions [8]. Currently pp-mess-sim supports k-ary n-cube topologies, square meshes, and wrapped hexagonal meshes.

Logical topologies can be built 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 [10] and improve network throughput [11]. Additionally, they can be used to separate traffic with different characteristics or performance requirements. Although virtual channels improve router flexibility, they also affect network speed and implementation complexity [12]. Since these trade-offs greatly influence communication performance, pp-mess-sim can vary the number of virtual channels in the network.

2.2 Routing and Switching

Switching and routing 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. 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* [13] and *wormhole* [14], 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].

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 increase the likelihood of cut-through at intermediate nodes and can balance the load on the network. Additionally, adaptive schemes may also 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 may depend on the *order* in which the algorithm considers these candidate links.

Each routing and switching policy is best suited for traffic with particular characteristics and performance requirements. Wormhole switching achieves low latency without requiring packet buffers, but virtual cut-through and packet switching may achieve larger throughput at high loads. Similarly, adaptive routing can reduce end-to-end delay, but out-of-order packet arrival can complicate protocol processing at the receiving node. Opportunities for adaptive routing vary depending on the topology, the distance a packet must travel, and network congestion. To study these effects, pp-mess-sim supports virtual cut-through, wormhole, and packet switching, as well as hybrid schemes, each under a variety of routing algorithms.

2.3 Router Architecture

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 implements queueing, arbitration, and flow-control policies to manage resource contention. These policies determine the complexity of the router, as well as overall network performance. Thus, a crucial aspect of interconnection network design is determining the size, speed, and structure of internal components.

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 [15]. 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. Closely tied to both queueing and arbitration is flow control, which affects latency and throughput by limiting the 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.

These internal router policies affect network performance by coordinating resource sharing amongst competing packets. When traffic patterns heavily load certain input or output links, these policies significantly impact network delay and achievable throughput. Isolating potential bottlenecks in the network design requires detailed performance metrics that capture the utilization of internal router resources. The simulator decouples 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.

3 Simulator Structure

The simulator's structure reflects the important design issues outlined in Section 2. The main components of pp-mess-sim are a set of C++ classes supporting: input specification (**Spec**), network topologies (**Net**), communication patterns (**Workload**), routing and switching policies (**Ralg**), and particular router models (**Node**). These components export clean and powerful interfaces to enable independent code development without sacrificing flexibility. The simulator can easily incorporate new topologies, routing algorithms, node models, traffic patterns, and data collection routines.

3.1 Input Specification

Simulation experiments are specified through a high-level language, as shown in the example in Figure 1. This language allows users to specify the range of experiments necessary to explore the large design space of multicomputer networks. The simulator parses an input file written in this language at run-time to initialize the experiment parameters. Input specification is supported by a lexical analyzer generator (flex) and a parser generator (GNU bison). These generators output C code, which is linked with the rest of the pp-mess-sim code during compilation.

The **Spec** grammar includes blocks for selecting the experiment parameters for each of the other pp-mess-sim modules and can be easily be extended to incorporate new simulation parameters. The sample experiment in Figure 1 involves an 8-ary 2-cube (8×8 torus) network that carries a mixture of time-constrained and best-effort traffic, with different traffic characteristics and performance requirements [16]. As shown in lines 18 and 30 of Figure 1, the time-constrained packets use packet switching and oblivious routing on a single virtual channel (channel 0), while the best-effort traffic employs two virtual channels (channels 1 and 2) for wormhole routing.

3.2 Network Creation

Given the topology parameters in lines 1-6 of Figure 1, the simulator first creates the 8-ary 2-cube **Net** topology. The **Net** class includes various functions for identifying and translating node addresses, link identifiers, and virtual channels; this insulates the **Node** and **Ralg** modules from the details of the specific network topology. As part of **Net** creation, pp-mess-sim generates each of the 64 router **Nodes**. Each node then instantiates its internal components, such as queues or arbiters, with their own simulation events and associated event handlers. The simulator uses an associative string map, initialized by the specification file, to assign internal router parameters. For example, line 41 selects a priority arbiter to govern

```
 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 LengthDiscrete(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(100.0,100.0);
27 -   length Uniform(10.0,10.0);
28 -   target HopUniform(0.2,0.7,0.1);
29 -   routing_spec begin
30 -     routing ps_oblivious(0);
31 -     order random;
32 -   end
33 -   history histogram(0,1000,50);
34 -   packets 2000;
35 -   drop 200;
36 - end
37 -
38 - general begin
39 -   random seed 1353625084;
40 -   parameter RX::ack_xmit_time 1;
41 -   parameter TX::arbiter priority;
42 -   output  e4_mix_400.00.out;
43 -   errors  e4_mix_400.00.err;
44 -   results e4_mix_400.00.results;
45 -   debug   e4_mix_400.00.debug;
46 - end
```

Figure 1: Example simulation specification

access to the outgoing links; this arbiter favors virtual channel 0 over the two wormhole virtual channels to better serve the time-constrained traffic. In parsing the input file, **Spec** creates an entry in the string map with the key "TX::arbiter" and value "priority."

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 defined in the **Node** code. 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 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 40 and 41 in Figure 1. This flexibility enables pp-mess-sim to model heterogeneous and even faulty networks, with a range of link speeds and router features.

3.3 Communication Patterns

Once pp-mess-sim constructs the network, the **Workload** module initializes the communication patterns for the experiment. In pp-mess-sim, traffic patterns are generated by a collection of independent "tasks," which are mapped onto individual nodes in the network to represent application behavior. As part of task creation, pp-mess-sim binds each task to a node and schedules its first packet creation event. For example, lines 8-11 of Figure 1 assign two independent tasks to each node in the network; every node instantiates one time-constrained and one "default" best-effort task. Since routing and switching policies significantly impact multicomputer performance, the tasks may adopt different routing-switching schemes, tailored to application communication demands. By changing task characteristics and mappings, the pp-mess-sim user can compose the diverse communication patterns necessary for realistic network evaluation.

4 Routing and Switching Algorithms

Tuning a network design requires evaluating a variety of router architectures and routing-switching schemes. The simulator facilitates such experimentation by decoupling these schemes (**Ralg**) from the router models (**Node**) and the network topologies (**Net**). This functional separation allows the user to easily implement new routing-switching algorithms without changing the node models.

4.1 Routing-Switching Instructions

Routers implement routing and switching in various ways, closely tied with internal timing and arbitration, but every device proceeds through common operations to service an incoming packet. The routing algorithm support in pp-mess-sim identifies these phases and represents them *outside* of the router model. Invoked after packet header collection, the routing algorithm interacts with the **Node** using a series of *routing-switching instructions* until they agree upon a suitable routing-switching decision. This allows the high-level routing algorithm to make its decisions based on feedback from the device, without low-level knowledge of the router architecture. Similarly, while the router model must accept commands from the routing algorithm, the router need not know how this algorithm selects the sequence of operations. This decoupling is instrumental in supporting multiple routing-switching schemes across a collection of router models.

On each interaction with the router, the algorithm generates a routing-switching instruction consisting of an ordered list of outgoing virtual channels and a candidate switching decision for the router's consideration. 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 router examines each instruction and determines whether or not the output channel(s) can satisfy the request; if necessary, the router tries to reserve any internal resources necessary to successfully complete the operation. For example, the algorithm may ask the router to reserve a single outgoing channel from a list of channels on a shortest-path route. If all of these channels are busy, the router may reject this request, requiring the algorithm to suggest an alternate way to service the packet (e.g., buffering the packet at the current node). The algorithm and the router model continue this request-response handshake until they agree on a common routing-switching decision.

The routing-switching instructions transcend event processing in the discrete-event simulation. For example, **Ralg** may instruct the **Node** to stall the incoming packet until one of its candidate output channels becomes available. If all of these channels are busy with other traffic, the **Node** cannot immediately respond to this instruction. When some later simulation event frees one of the channels, the **Node** may then try to reserve this channel and continue its interaction with **Ralg**. This allows the **Node** to invoke channel allocation policies transparent to the routing instructions. Similarly, detailed router models may capture delay in acquiring internal buffer resources; such models may proceed through multiple simulation events before responding to a routing instruction. The handshake between **Ralg** and **Node** hides the low-level timing details of the router model and, thus, allows the construction of generic routing algorithms.

4.2 Selection Functions

The simulator also includes **Net** support to minimize the dependency of the routing-switching algorithms on the underlying 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. The **Net** selection functions categorize and rank these links, based on certain routing primitives; **Ralg** uses these functions to generate a list of possible directions for a packet to travel. For example, given the current node and the packet's destination, the selection functions can identify which output links lie on a minimal path or, alternatively, which links would deflect the packet away from a shortest-path route.

Routing algorithm performance also depends on the *order* the router considers the set of output directions. Hence, the selection functions also rank the set of output links, returning an ordered list of candidate outgoing links. For example, line 19 in Figure 1 assigns a dimension-order ranking to the default best-effort packets. This requires a packet to complete all hops in one direction before proceeding to the next dimension. In contrast, the time-constrained packets consider their output links in a random order. **Net** also includes a selection function that ranks links according to how far the packet must still travel in each direction; this link ordering improves a packet's chance of considering multiple outgoing links at future nodes in its route. Another selection function orders output links according to network congestion, giving preference to links with fewer busy virtual channels; this balances traffic load amongst the outgoing links. These selection functions, coupled with the routing-switching instructions, enable pp-mess-sim to model a wide range of routing-switching algorithms on a variety of network topologies.

Existing schemes include both oblivious and adaptive shortest-path routing for wormhole, virtual cut-through, and packet switching, with several selection functions. The user can also specify various nonminimal routing algorithms for virtual cut-through and wormhole switching. The simulator includes several deadlock-free wormhole routing algorithms, with varying degrees of adaptivity [17]. In addition to traditional routing and switching schemes, sequences of routing-switching instructions can generate hybrid algorithms that incorporate both virtual cut-through and wormhole switching, depending on the underlying router conditions. The generality of the routing-switching instructions and the selection functions significantly reduces the difficulty of adding new algorithms to pp-mess-sim.

5 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. To better evaluate network policies, each **Node** also monitors the utilization of its internal resources during the course of the simulation.

5.1 Traffic Generation

The simulator generates traffic patterns as a collection of tasks with varying characteristics. Because the packet characteristics in an actual network depend on the application or protocol software, pp-mess-sim allows the derivation of packet length and interarrival times from a variety of stochastic processes, including uniform, exponential, geometric, normal, and discrete distributions. 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 1, time-constrained tasks generate periodic, fixed-length packets, while default best-effort tasks create packets according to a Poisson process. Using the discrete distribution of packet lengths, 70% of the best-effort packets are short, while the remaining are long; such bimodal distributions are common in multicomputer applications [5].

Application constructs also impact the selection of a target destination node for each packet. Line 28 of Figure 1 assigns a hop-uniform target distribution to the time-constrained task. In this example, 20% of packets have destinations just one hop away, while 70% travel two hops, and the remaining packets traverse three hops. While the hop-uniform distribution captures spheres of communication locality, the node-uniform distribution in line 16 represents a random permutation, with uniform random selection of destination nodes. To capture the communication behavior of scientific applications, target destinations may stem from common permutations, such as matrix-transpose (dimension-reversal), bit-complement, and bit-reversal. The simulator also includes a destination-discrete distribution, where all packets are destined for a certain subset of nodes, to generate "hot-spots" of heavily-utilized nodes and links; common multicomputer constructs, such as synchronization or multicast operations, may induce such non-uniform traffic.

5.2 Packet Statistics

The simulator associates performance metrics with each task to make data collection more flexible. These packet statistics 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. In pp-mess-sim the tasks proceed through three distinct phases: priming the empty network, collecting performance data, and draining the system of any remaining packets.

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 35 of Figure 1). 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 34 of Figure 1). The task continues to generate packets until every task in the network has completed data collection. Then, all tasks stop creating packets and the simulator executes any remaining events to handle traffic left in the system; this serves as a precaution to identify possible communication deadlocks.

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 router model, the data collection routines can accumulate a wide variety of performance statistics. 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 (e.g., mean, variance, minimum, and maximum), packet cut-through probabilities, and latency histograms. For example, in Figure 1 the best-effort default tasks collect basic latency metrics (line 21), while the time-constrained tasks capture a histogram of latency data to estimate the probability distribution of packet delay (line 33).

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. 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.

5.3 Router Statistics

Pinpointing weaknesses and bottlenecks in a router design requires detailed information about how internal features perform in operation. While the history-list approach is suitable for collecting packet statistics, it is unnatural for capturing fine-grain information about resource usage in the router. Instead, each pp-mess-sim router model accumulates statistics for its outgoing channels, internal buses, and packet buffers throughout the data collection phase of the simulation. By maintaining separate statistics for each node, the user can investigate the impact of non-uniform communication patterns on resource usage across all nodes in the network.

Each node model accumulates this data whenever a simulation event models access to an internal resource. The **Node** maintains a single object that updates these statistics in response to simulation events. For example, if an event models the use of an internal bus, the **Node** updates its count of active bus cycles. Monitoring resource usage through the simulation events also enables the **Node** to compute average queue lengths for any buffers in the router design. When a simulation event augments or depletes a buffer, the statistics object records the time and the size of the buffer; this allows the **Node** to maintain the time-average of the queue length. Using these statistics, along with the history-lists, pp-mess-sim can gather detailed information about network performance.

6 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 additional **Node** modules to be developed without changing any of the other modules. In addition, this support enables individual **Node** models to focus completely on internal policies for queueing, arbitration, and flow control.

6.1 Node Modules

The simulator includes a cycle-level model of SPIDER [18], a network router for point-to-point distributed systems. SPIDER coordinates bidirectional communication with up to four neighboring nodes, with three virtual channels on each physical link. A demand-driven TDM bus connects the incoming and outgoing links, with microprogrammable engines implementing the routing-switching schemes for arriving packets; the **Ralg** routing programs mimic the behavior of these programmable devices. The SPIDER **Node** model facilitates precise performance evaluation by capturing the low-level details of flow control, resource arbitration, and microcode execution. Run-time specification of interface speeds and internal buffer sizes has been instrumental in tuning the router design and implementation.

While the SPIDER **Node** module effectively represents a specific router design, exploring alternative architectures requires more robust 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 SPIDER model, v-router experiments can efficiently consider a broader range of simulation parameters before testing specific options on a more detailed model. The v-router simulations often execute an order of magnitude faster for virtual cut-through and packet switching experiments by modeling only the head and tail flits in each packet;

wormhole switching simulations typically execute 50% faster, since the v-router does not model low-level timing details. The support provided by **Net**, **Ralg**, and **Workload** allow pp-mess-sim to run the same experiments on both router models.

6.2 Sample Experiments

The simulator's flexibility enables a broad range of experiments evaluating multicomputer router designs. Figure 2 shows the performance of various switching schemes using static routing on an 8×8 square mesh of v-router **Nodes** carrying 16-flit packets; experiments using the SPIDER model showed similar trends. The graphs plot the mean packet latency as a function of the average link utilization for two traffic patterns. As expected, virtual cut-through switching consistently outperforms packet switching, since virtual cut-through traffic often avoids buffering delay at intermediate nodes. At low loads, wormhole switching performs extremely well for both traffic patterns.

However, the relative performance of virtual cut-through and wormhole switching varies significantly between Figure 2(a) and 2(b). Under uniform random traffic, the two switching schemes exhibit comparable performance at low loads; however, network contention limits wormhole throughput at 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.

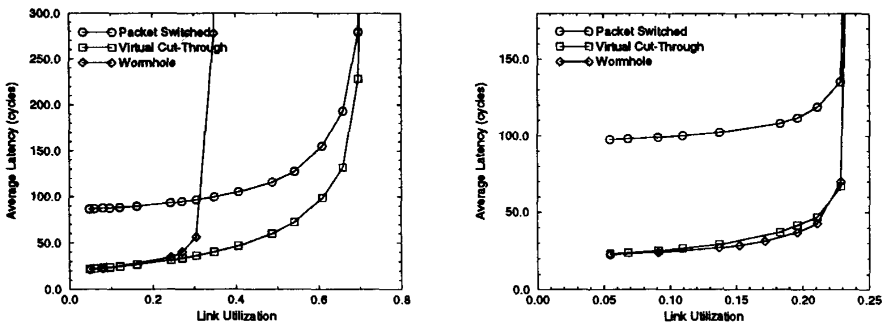
Despite channel contention, wormhole switching excels for the matrix-transpose permutation. This effect occurs because matrix-transpose traffic, coupled with dimension-order routing, limits harmful contention between packets heading to different parts of the network. In a square mesh, the matrix-transpose permutation requires node (c, d) to communicate with node (d, c). With dimension-order 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. Similarly, traffic patterns significantly impact the performance of routing algorithms, as shown in Figure 3. These pp-mess-sim experiments evaluate wormhole switching under both dimension-order and adaptive routing; virtual cut-through simulations showed the same qualitative trends. The adaptive algorithm is a fully-adaptive minimal routing scheme that requires two virtual channels per link to prevent network deadlocks [17]; in these experiments, both routing algorithms employ a pair of virtual channels to enable fair performance comparisons. The dimension-order routing algorithm uses the extra virtual channel to reduce contention between packets traveling on the same link [11].

Contrary to intuition, static routing consistently outperforms adaptive routing in Figure 3(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 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 3(b). This experiment considers bursty traffic, in contrast to the traditional Poissonian packet arrival process in Figure 3(a). The source nodes generate bursty traffic using a two-stage normal distribution of packet interarrivals [4]. Packet interarrivals stem from two independent normal distributions, with different means; sources randomly select 80% of interarrivals from the distribution with the small mean.

In Figure 3(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



(a) Uniform random traffic

(b) Matrix transpose traffic

Figure 2: Comparing switching schemes under dimension-order routing

handful of related messages. Figures 3(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 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. The simulator's flexibility enables such experimentation with routing-switching schemes, router models, and communication workloads.

7 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 schemes, 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 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 investigating new routing-switching algorithms, using the various **Ralg** routing instructions and **Net** selection functions.

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 schemes, 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.

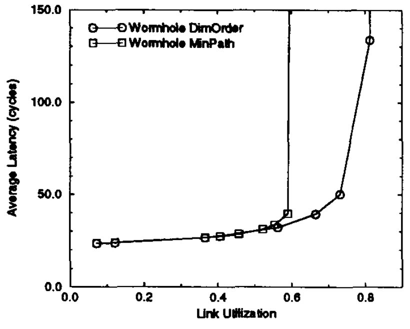
References

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

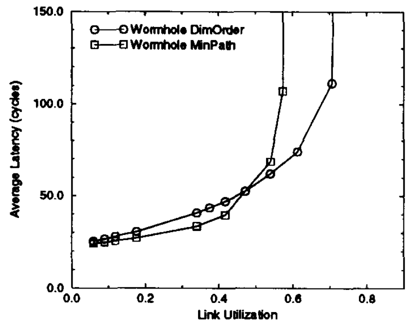
[2] X. Zhang, "System effects of interprocessor communication latency in multicomputers," *IEEE Micro*, pp. 12–15,52–55, April 1991.

[3] 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.

[4] J.-M. Hsu and P. Banerjee, "Performance measurement and trace driven simulation of parallel



(a) Poisson arrival process



(b) Bursty arrival process

Figure 3: Comparing routing algorithms under wormhole switching and bit-complement traffic

CAD and numeric applications on a hypercube multicomputer," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, pp. 451–464, July 1992.

[5] 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.

[6] 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.

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

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

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

[10] 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.

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

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

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

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

[15] 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.

[16] 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.

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

[18] 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 *Proc. Int'l Conf. on Distributed Computing Systems*, pp. 574–580, June 1994.