

The Avalanche Myrinet Simulation Package

— User Manual for V2.0 —

Chen-Chi Kuo
Department of Computer Science
University of Utah

May 3, 1997

Contents

1	Introduction	3
2	Configuration Files	3
2.1	System Parameters File	4
2.2	Topology File	7
2.3	Routing Table File	7
3	Interfaces with the Upper Level Simulation Codes	8

1 Introduction

This is the user manual for the configurable Myrinet¹ simulation package that has been developed for the Avalanche project at University of Utah. This package requires the use of the PAINT architecture simulator, which was evolved as part of the Avalanche effort from the University of Rochester's MINT simulator. To use this Myrinet simulation package, you must link with the PAINT library and use PAINT to drive the simulation itself.² Please refer to the Avalanche project home page at <http://www.cs.utah.edu/projects/avalanche> for more details about the PAINT simulation and to acquire a copy of PAINT.

Version 2.0 of the Myrinet simulation package was designed to allow a high degree of configurability of the modeled network. Version 1.0 modeled only simple square mesh topologies with 4-port switches, and users could specify only a limited number of switch parameters. As Myricom released larger and faster versions of their Myrinet switches, the V1.0 simulation model became obsolete. Users of the V2.0 package can specify arbitrary network topologies composed of Myrinet switches with different number of ports. For example, 4-port and 32-port switches can be used in a single system. Because the V2.0 model supports arbitrary topologies, simple X-then-Y source routing is no longer sufficient to model the required routing. Thus, users of the V2.0 package must specify the routing table themselves, as described in Section 2. In addition, to track improvements to the circuit technologies used in the Myrinet switches, the clock rate, latency and bandwidth have been parameterized. Users can change the parameters in order to meet their simulation needs.

The remainder of this user manual is organized as follows. In Section 2, the formats of the system configuration files are explained through a series of examples. Section 3 describes the interface between the Myrinet simulation package with PAINT, so that users can integrate their Myrinet network model into their PAINT architecture model.

2 Configuration Files

Users of this package must provide three configuration files:

- a system parameter file that describes the performance parameters of the switches and links in the system, as well as a small number of global parameters,
- a network topology file that describes how the switches in the system are interconnected (i.e., what ports are connected to what other ports), and
- a network routing file that describes how to route from every processor to every other processor.

The names of these parameter files can be specified in the PAINT command line using the `-k`, `-t`, and `-r` flags. For example, `sim -n 16 -s 0x1800000 -- -k sfile -t tfile -r rfile kernel -s ksetup`³

¹*Myrinet* is a trademark of Myricom, Inc. For detailed information on Myrinet technology, see the Myricom home page at <http://www.myricom.com>.

²PAINT is designed to model HP PA-RISC based multiprocessors, while MINT is designed to model MIPS-based multiprocessors. Although this package requires the use of PAINT, a port to the MINT system should be feasible with a limited amount of effort. If you perform this port, please send it back to us for inclusion in our release for others to use, and we will (of course) give you full credit for the port.

³*kernel* is a subset of the BSD 4.4 kernel. You can acquire a sample kernel with the *PAINT* package. *ksetup* is a setup file in order to run in the MPMD mode. Please refer to the *PAINT* package for more details.

indicates that the system parameter file is called `sfile`, the topology file is called `tfile`, and the routing file is called `rfile`.

The required format of the three configuration files are explained in Sections 2.1 through 2.3 using the example topologies illustrated in Figures 1 and 2. Figure 1 illustrates a simple mesh topology composed of four and eight node switches, while Figure 2 illustrates a chordal ring topology composed of only four-node switches.

2.1 System Parameters File

The System Parameters File specifies the configuration of the Myrinet switches in the system. An example is given below, with comments to explain the meanings of each parameter:

```
# Simple Mesh System Parameters File

# Total number of the processors in the system
numOfProcessor 16

# Maximum number of ports on any single switch in the system
maxNumOfPorts 8

# Total number of switches in the system
numOfSwitch 7

# Link propagation delay, in cycles where 1 cycle == 10 ns
propDelay 4

# Time to perform taxi translation and cross bar setup
# for different switch sizes, measured in system cycles
fallThruDelay4 26
fallThruDelay8 27
fallThruDelay16 30
fallThruDelay32 35

# Ratio between CPU and Myrinet switch clock rates
# For example, if the processor speed is 100MHz and the modeled
# Myrinet system clocks at 50MHz, the SpeedFactor is 2
SpeedFactor 2

# Myrinet switch slack buffer sizes (see Myrinet technical specs
# for discussion of the kg, h, and ks values in the buffer)
buffer_kg 32
buffer_h 16
buffer_ks 32
```

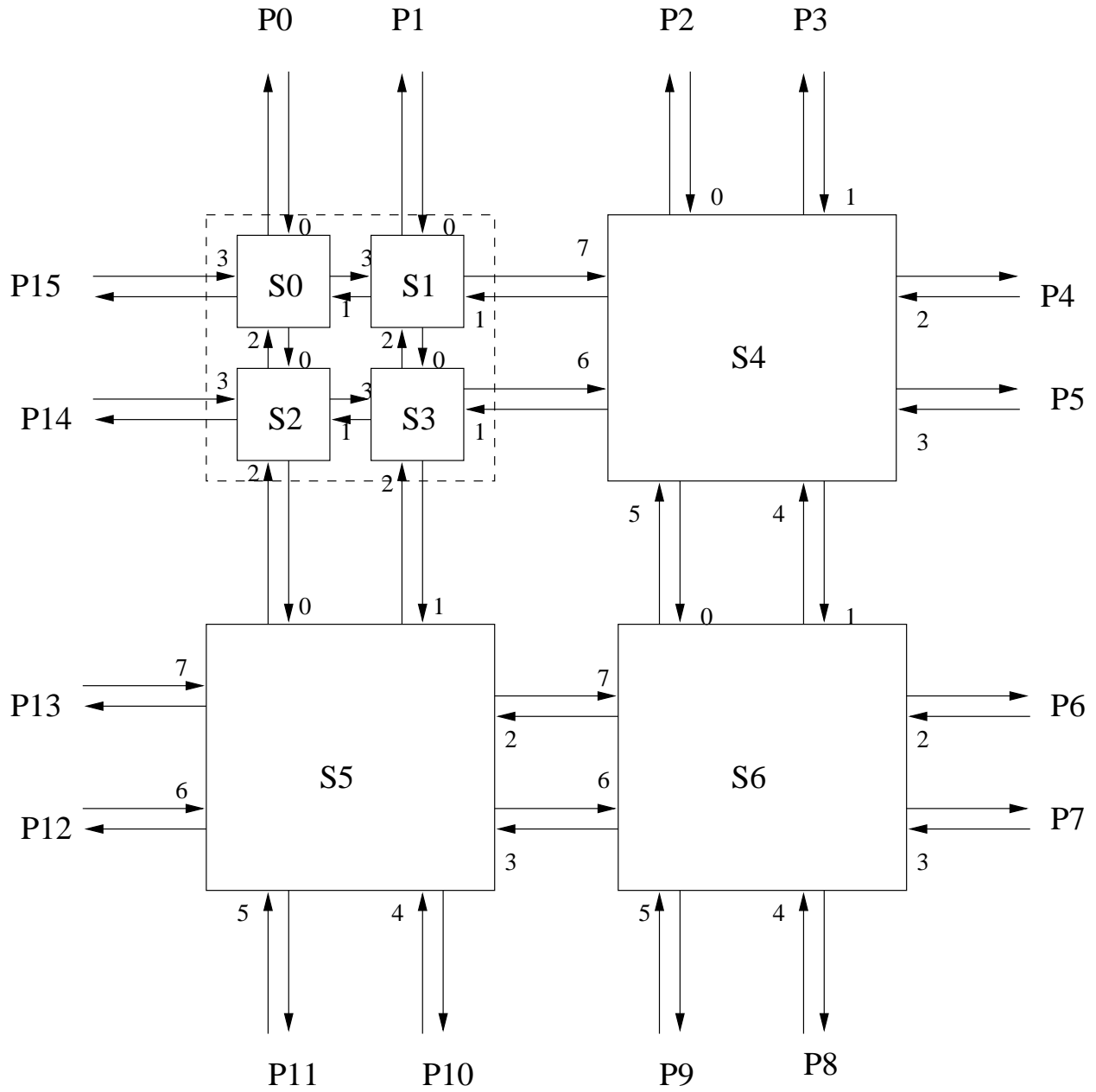


Figure 1: Example Topology: Simple Mesh

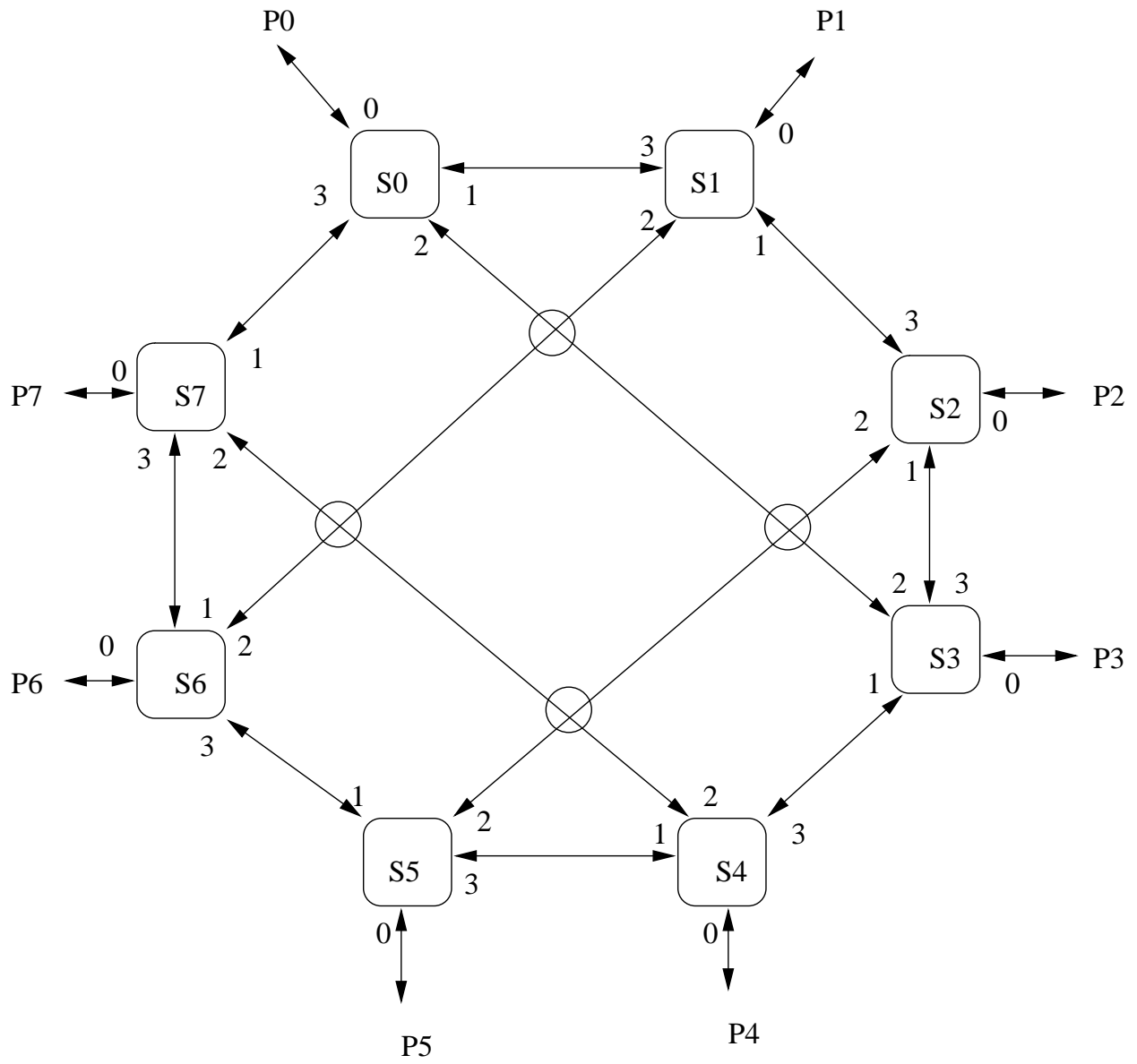


Figure 2: Example Topology: Chordal Ring

2.2 Topology File

The topology file specifies the interconnections between individual switches in the system. It is used to define the overall system topology. The following example file is the topology specification for the simple mesh in Figure 1. The topology file consists of one line per switch that designates where each of that switch's ports are connected (either to ports on other switches or to processors). Each line should have one entry for each port (i.e., the topology entry for a four-port switch must have four entries, while that for an eight-port switch must have eight).

```
# Simple Mesh Topology File
```

```
# Some definitions:
```

```
# S0.1 means port 1 of switch number 0
```

```
# P0 means processor number 0
```

```
# D means dangling line
```

```
S0: P0 S1.3 S2.0 P15
```

```
# Meaning: Port 0 of Switch 0 is connected to Processor 0
```

```
# Port 1 of Switch 0 is connected to Port 3 of Switch 1
```

```
# Port 2 of Switch 0 is connected to Port 0 of Switch 2
```

```
# Port 3 of Switch 0 is connected to Processor 15
```

```
S1: P1 S4.7 S3.0 S0.1
```

```
S2: S0.2 S3.3 S5.0 P14
```

```
S3: S1.2 S4.6 S5.1 S2.1
```

```
S4: P2 P3 P4 P5 S6.1 S6.0 S3.1 S1.1
```

```
S5: S2.2 S3.2 S6.7 S6.6 P10 p11 P12 P13
```

```
S6: S4.5 S4.4 P6 P7 P8 P9 S5.3 S5.2
```

The following example file is the topology specification for the chordal ring.

```
# Chordal Ring Topology File
```

```
S0: P0 S1.3 S3.2 S7.1
```

```
S1: P1 S2.3 S6.2 S0.1
```

```
S2: P2 S3.3 S5.2 S1.1
```

```
S3: P3 S4.3 S0.2 S2.1
```

```
S4: P4 S5.3 S7.2 S3.1
```

```
S5: P5 S6.3 S2.2 S4.1
```

```
S6: P6 S7.3 S1.2 S5.1
```

```
S7: P7 S0.3 S4.2 S6.1
```

2.3 Routing Table File

Myrinet technology uses a static source routing mechanism. For simple mesh topologies composed of symmetric switches, a simple X-then-Y routing mechanism suffices to route packets between

input and output ports. However, because the V2.0 simulation package supports arbitrary network topologies and heterogenous switch sizes, X-then-Y routing is no longer sufficient. Users must specify the static source routing tables explicitly to specify to the simulation how to compose the packet headers. A complete routing table file must include $N * N$ routing directions in an N processor system, one entry for each processor pair. Note that the routing need not be symmetric, meaning that packets from port X to port Y can take a different path than packets from port Y to port X. A partial sample routing table is given below. For space purposes, only the routes from one processor are given. Please refer to the simulation package itself for a complete example.

The syntax of the routing file is as follows. For each processor pair, there must be one line specifying the order of switch output ports that a message traveling from the source to the destination must take. Port numbers are designated via a single character, ranging from 0-9 (for the first ten ports) and then a-z (for the next 26 ports). For the V2.0 product, this results in a maximum switch size of 36 ports (or, realistically, 32 ports).

In the example routing table below, for Processor 0 (P0) to send a packet to Processor 8 (P8), the packet will go through port 1 of S0, port 1 of S1, port 4 of S4, and finally port 4 of S6. This route is directed by the entry in the routing table P0 P8 with the sequence 1144.

#Simple Mesh Routing Table File

#sender	receiver	portnumber-sequences
p0	p0	0
p0	p1	10
p0	p2	110
p0	p3	111
p0	p4	112
p0	p5	113
p0	p6	1142
p0	p7	1143
p0	p8	1144
p0	p9	1145
p0	p10	224
p0	p11	225
p0	p12	226
p0	p13	227
p0	p14	23
p0	p15	3

3 Interfaces with the Upper Level Simulation Codes

Traditional PAINT architecture simulations consist of models for the CPU, cache controller, directory controller (for scalable DSM models), network interface, system bus, and other components. This Myrinet simulation package provides a portion of the router interface to create source-routing headers, route packets between nodes, model the network delays due to internal buffering con-

straints, etc. It does not, however, model input or output buffering within a node (i.e., between the system bus and the network device). This level of buffering must be modeled in the architecture simulation. The network simulation package models packet delivery and flow control at a cycle-by-cycle level of precision.

To inject packets into the Myrinet fabric, the architecture simulation should invoke the *Send* function, which has the following type signature:

```
Send(task_ptr ptask, int src, int dest, int payload, void *msg_addr, int info_size, void *usr_ptr, int do_mem_costs)
```

The meaning of the parameters is as follows:

- **ptask**: the PAINT task that will be scheduled by the Myrinet simulation after the last flit of the current packet is injected into the interconnect. At that point, the architecture simulation can issue another *Send* to ship the next packet. If other pieces of the simulation need to respond to the event of a packet transmission completing, users of this package must signal this event within the *ptask* routine.
- **src**: the processor id of the sending processor
- **dest**: the processor id of the destination processor
- **payload**: the length of the user message body in bytes
- **msg_addr**: the memory address of the user message body, used to perform DMA transfers
- **info_size**: length of the user message header in bytes
- **usr_ptr**: an arbitrary pointer to be used by the communicating peers of the upper level simulation, which can be to pass information useful for controlling the simulation
- **do_mem_costs**: a flag to indicate if this *Send* call needs to DMA the packet data from the memory, which may cost some delay. A user-define function *Memory_read*, which is explained below, will be invoked when *do_mem_costs* is set.

If the **do_mem_costs** is set, the function *Memory_read*, defined by the users, will be invoked in order to simulate the delay caused by DMAing the packet data from the memory. The type signature of the *Memory_read* function is as follows:

```
Memory_read(task_ptr ptask, int src, void *msg_addr, int payload)
```

The meaning of the parameters is as follows:

- **ptask**: the PAINT task that *MUST* be scheduled in the *Memory_read*, so the Myrinet simulation can finish shipping this packet.
- **src**: the processor id of the sending processor
- **msg_addr**: the memory address of the user message body, used to perform DMA transfers
- **payload**: the length of the user message body in bytes

Once the packet arrives at the destination processor, the *Receive* will be invoked by the Myrinet simulation. Users of this package must implement this *Receive* function. When the upper level simulation has consumed the packet, it *MUST* schedule the ptask or the Myrinet simulation will not deliver more packets. The type signature of the *Receive* function is as follows:

```
Receive(task_ptr ptask, int src, int dest, int payload, void *msg_addr, int info_size, void *usr_ptr)
```

The parameters of *Receive* are the same as the parameters in *Send*.