A Customizable Simulator for Workstation Networks *

Mustafa Uysal, Anurag Acharya, Robert Bennett, Joel Saltz
Computer Science Department
University of Maryland, College Park, MD 20742
{uysal,acha,robertb,saltz}@cs.umd.edu

Abstract

We present a customizable simulator called `netsim` for high-performance point-to-point workstation networks that is accurate enough to be used for application-level performance analysis yet is easy enough to customize for multiple architectures and software configurations. Customization is accomplished without using any proprietary information, using only publicly available hardware specifications and information that can be readily determined using a suite of test programs. We customized `netsim` for two platforms: a 16-node IBM SP-2 with a multistage network and a 10-node DEC Alpha Farm with an ATM switch. We show that `netsim` successfully models these two architectures with a 2-6% error on the SP-2 and a 10% error on the Alpha Farm for most test cases. It achieves this accuracy at the cost of a 7-36 fold simulation slowdown with respect to the SP-2 and a 3-8 fold slowdown with respect to the Alpha Farm. In addition, we show that the cross-traffic congestion for today's high-speed point-to-point networks has little, if any, effect on application-level performance and that modeling end-point congestion is sufficient for a reasonably accurate simulation.

1 Introduction

The performance of workstation clusters with high-performance interconnects has improved to the point that they are gradually replacing the traditional tightly-coupled dedicated multicomputers as the platform of choice for parallel computation. Most contemporary commercial and research parallel platforms fall into this category (e.g. the IBM SP-2, the DEC Alpha Farm, the Berkeley NOW [4], the Wisconsin COW [14], the CESDIS Beowulf [20]).

Unlike multicomputers, each processing node of a workstation cluster is a complete machine with its own operating system, often with multiple users and significant I/O resources. Analyzing the performance of applications on such machines is significantly harder than doing so on the traditional multicomputers. More so since access to hardware has to pass through many layers of system software none of which can be looked at by a non-privileged user. We can testify first-hand about the six months we spent trying to track down performance problems in a set of communication-intensive programs with irregular communication patterns[1] [2].

A fast and reasonably accurate simulator would significantly simplify this task. However, building a simulator for the communication subsystem, let alone the entire machine is not feasible for the average user. In addition to the enormous time and effort, a non-privileged user does not

*This research was supported by ARPA under contract No. #DABT63-94-C-0049, Caltech Subcontract #9503, by NASA under contract No. NASA #NAS5-32337, USRA/CESDIS Subcontract #555541 and by grants from IBM Corporation and Digital Equipment Corporation

[1] And about the large number of conditional compilation statements which were quite difficult to clean up later.

have access to detailed information about the hardware and the operating system. Furthermore, whenever the application of interest is ported to a new platform, a new simulator would be needed.

We are led, then, to consider several questions. Is it possible to build a customizable simulator for workstation networks which is accurate enough to be used for performance analysis yet is easy enough to customize so that it is worth doing so even for a single application? Is it possible to do so without using any proprietary information, that is, using information that is either publicly available or can be determined using test programs? Would such a simulator be fast enough to be practical?

There is some *a priori* reason to believe that at least some of these questions can be answered in the affirmative. There is a convergence in design of both the interconnects used for workstation networks and the messaging software used on these networks. Modern workstation network interconnects are designed using switching elements and point-to-point links with a regular, low-dimension topology and aggressive cut-through routing and flow-control [11, 21]. Data rates are high and the error rates are low. Communication is either packet-based, with an upper bound on the packet size, or cell-based with a fixed packet size. Many network adapters provide *outboard buffering* where the adapter buffers are large enough to function as retransmit and receive buffers. DMA is almost universally supported for transfers between host memory and adapter buffer. Few systems provide protocol processing on the adapter, leaving the protocol overheads to software [19]. On the messaging software side, standardization efforts has produced the Message Passing Interface (MPI) standard which has been largely accepted by users and vendors [8]. While it is possible that different vendors could implement the interface in completely different ways, the common software interface and relatively similar networking hardware (as described above) indicates that most implementations on workstation clusters can be expected to be not significantly dissimilar.

In this paper, we address these questions by describing our experience building and evaluating `netsim`, a customizable network simulator for workstation networks. `Netsim` models point-to-point dedicated links, network adapters with an outboard buffer and a DMA engine and buffered communication software. The network is assumed to be lossless. `Netsim` models the connection between any pair of hosts as a dedicated link and ignores congestion due to cross-traffic. It does, however, model end-point congestion which occurs when several nodes try to communicate with a single node. It has six *hardware parameters*, which specify the characteristics of the interconnect and the adapter, and five *software parameters* which specify the characteristics of the memory and the messaging software. The hardware parameters can be easily obtained from information made public by the manufacturer; the software parameters can be determined by a small set of controlled experiments on the selected platform.

We have customized `netsim` for two different platforms: the IBM SP-2 with the IBM High Performance Switch, the i860-based communication adapter and IBM's `MPL` message-passing library; and a cluster of DEC Alpha 2100 4/275 four-processor workstations with the GIGAswitch/ATM network, the ATMworks 750 adapter and the portable `MPI-CH` message-passing library from the Argonne National Lab. We believe that these are important platforms and are currently in use at a large number of sites worldwide. We would also like to point out that these two systems differ in many aspects – host architecture (uniprocessor/SMP), network architecture (multistage/crossbar, packet-based/cell-based), I/O peripherals bus (MCA/PCI) and communication software (native/portable). If `netsim` is able to achieve reasonable performance for both platforms, this would be evidence for its customizability.

To evaluate the customized simulators, we used a suite of microbenchmarks representing common low-level network operations. Our results show that `netsim` is able to achieve reasonable accuracy for both platforms. For the SP-2, `netsim` was successfully able to model the application-level bandwidth across a seven orders of magnitude difference in message size. The error for most message sizes was 2-6%, the maximum error being 12%. For the Alpha Farm, `netsim` was able to model the application-level bandwidth within an error of 10% for message sizes up to 1 MB (for message sizes larger than 1 MB, the performance of our MPI-CH installation drops sharply and unexpectedly). This accuracy is achieved at the cost of a 7-36 fold slowdown for the SP-2 and a 3-8 fold slowdown for the Alpha Farm. To put this in context, Benveniste and Heidelberger [6] report that a detailed sequential simulator takes 1 day on a workstation to simulate 1 second of simulation time for a 128 node SP-2.

As an important aside, our experiments also show that for high-performance point-to-point networks, modeling end-point congestion is sufficient for a reasonably accurate simulation and that cross-traffic congestion contributes little, if any, to application-level performance.

`Netsim` has been developed as part of the HOW[2] project whose goals is to evaluate architectural and OS policy alternatives for data-intensive tasks on workstation clusters. `Netsim` has been integrated into a larger simulator, `howsim`. `Howsim` simulates I/O devices (storage and network) and the corresponding OS software at a fairly low level and the processor at a fairly high level. `Howsim` is currently operational and simulates the interconnect, network adapters, disk devices, peripheral buses, disk controllers, the file system and the OS scheduler. In the immediate future, we plan to use `howsim` for application-driven studies of I/O architectures for workstation clusters and cluster-wide scheduling policies.

2 Description of `netsim`

`Netsim` models point-to-point interconnection links, network adapters with an outboard buffer and a DMA engine, and buffered communication software. Figure 1 shows the configuration modeled for a pair of nodes.

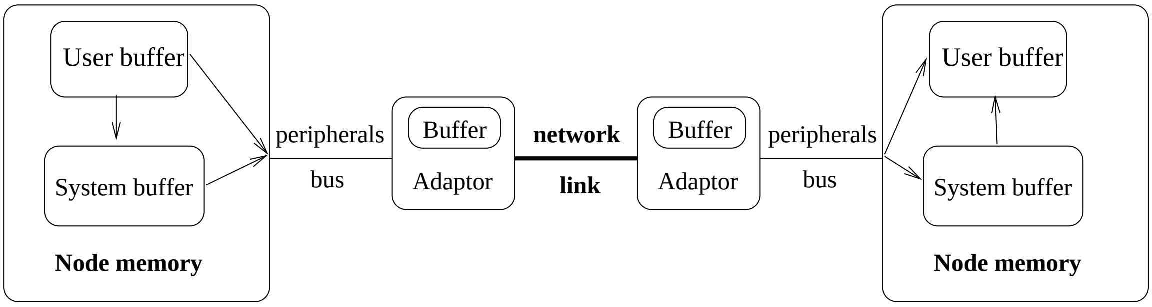
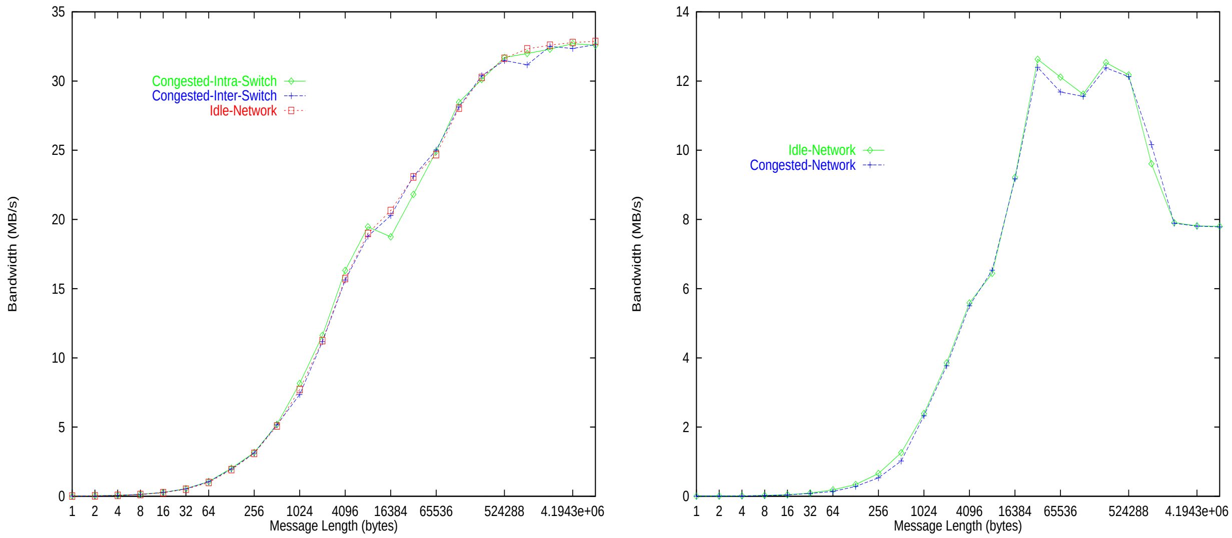


Figure 1: `netsim` network model for a pair of nodes.

The network is assumed to be packet-switched and each node is assumed to be connected to all its peers by dedicated point-to-point links. Network links are modeled by a simple latency-bandwidth model. Time to transfer a packet of size L over the wire is assumed to be $T = \alpha + \beta \times L$ where α and β represent the wire latency and bandwidth, respectively. As a result, cross-traffic congestion is not modeled but the effects of end-point congestion are included. We conducted a series of experiments on various SP-2 and Alpha Farm configurations to verify the validity of this

[2] HOW stands for *horde of workstations*. With the help of the Berkeley NOW and the Wisconsin COW, we hope to complete the rollcall of *how now brown cow*.



(a) IBM SP-2

(b) DEC Alpha Farm

Figure 2: Effect of cross-traffic congestion.

assumption. In these experiments, two selected nodes exchange a sequence of messages and the remaining nodes flood the network by repeatedly sending 8 MB messages to each other. Remaining nodes also change their communication peers after every iteration in a round-robin fashion so as to apply the maximum load on the interconnection network. On the SP-2, we considered two cases based on the distance between the selected nodes – nodes attached to the same switching element(*intra-switch*), nodes connected to different switches but within the same frame (*inter-switch*). Results in Figure 2 show that for high-performance point-to-point networks cross-traffic congestion contributes little to application-level performance. We obtained similar results from an experiment which tried to measure the effect of congestion for nodes located in different switch frames in a 128-node SP-2 configuration at NASA Ames. We did not detect any significant effect.

Network adapters are assumed to be bi-directional, dual-ported devices with an outboard buffer. The *DMA port* of a network adapter is used to copy network packets between the processor and the network adapter, and the *network port* is used for communicating with other network adapters. The unit of transfer in either port is a *packet*. A packet consists of a packet header, containing routing information, and a payload. Both ports can be simultaneously active, but only one packet can be in transit on each port at any given time. The link between the adapter and the host memory (used by DMA) is assumed to be characterized by bandwidth alone.

The simulator also models several software layers: (1) a synchronous messaging library layer which copies data to and from system buffers, initiates sends and selects the appropriate message for a receive call, (2) the flow-control layer that maintains buffers corresponding to different peers and schedules message sends, and (3) the interaction layer that controls the interaction between the messaging library and the adapter.

2.1 Customization

`Netsim` has six hardware parameters and five software parameters (see Table 1). The hardware parameters can be easily obtained from information made public by the manufacturer; the software parameters are to be determined using a suite of controlled experiments on the selected platform.

We have developed three programs, `bcopy`, `send` and `recv`, for determining the values of the software parameters. The first of these is used to determine the in-memory copy bandwidth using the *bcopy()* function. Together with the packet size, the copy bandwidth is also used to compute the *packet copy cost* parameter.

The `send` and `recv` programs are used to determine the other four parameters. We determined the *system buffer size* as follows: (1) we ran `send` for increasing powers-of-two message sizes till the point where the application-level bandwidth drops, (2) we use the message size just before this happens as the *system buffer size*. This works because for messages smaller than *system buffer size*, a *send()* operation returns after copying the message to the system buffer whereas for messages larger than this size, some packets need to be transferred to the adapter, a significantly slower operation.

The protocol processing costs for a message are assumed to fit a linear model with a fixed component corresponding to the cost of entry into the messaging layer and a variable component that depends on the number of packets. We determined *OS send cost* as follows: (1) we ran `send` for all powers-of-two message sizes smaller than the packet size, (2) for each message size, we computed the average time to return from the *send()* call, and (3) using these numbers, we computed the average time to complete a *send()* call for messages smaller than the packet size. We used this value as an estimate of the *OS send cost*. The *OS recv cost* was determined in a similar fashion.

We determined the *packetization cost* in the following way: (1) we ran `send` for all multiples of the packet size that are less than the combined capacity of the system buffer and the adapter buffer; (2) for each message size, the *OS send cost* is subtracted from the time to complete the send operation and the remaining quantity is divided by the number of packets in the message; this yields the overall per packet cost; finally (3) the in-memory copying cost per packet is subtracted from the overall per packet cost to yield the *packetization cost*.

3 Evaluation

In order to evaluate `netsim`, we used a set of three network operations commonly used in distributed and parallel applications as microbenchmarks. The first microbenchmark, `point-to-point` sends a sequence of messages from a source to a sink and computes the average time spent at both ends waiting for communication calls to complete. This is the simplest possible messaging benchmark and provides the baseline numbers for other benchmarks. The second microbenchmark, `exchange`, exchanges a sequence of message between a pair of nodes and computes the average round-trip time. This benchmark provides a measure of the application-level bandwidth and latency. It is also is a primitive building block of most collective communication operations. The final microbenchmark, `many-to-one`, sends messages from multiple sources to a single sink. This corresponds to a client-server scenario in distributed systems and a hotspot node in a parallel application. It also allows us to measure the effect of end-point congestion as the incoming bandwidth of the sink node can usually be saturated by one, at most two, source nodes.

We selected two systems for evaluating `netsim`'s modeling accuracy and simulation speed. The first was a 16-processor IBM SP-2 with the High Performance Switch, the i860-based communication adapter and IBM's `MPL` message-passing library and the second was a cluster of ten DEC Alpha 2100 4/275 four-processor workstations with the GIGAswitch/ATM network, the ATMworks 750 adapter and the `MPI-CH` message-passing library from the Argonne National Lab.

For all our experiments, we varied the message size from 1 byte to 8 MB. For the `many-to-one` benchmark, the number of source nodes was varied between 2 and 4. All experiments were repeated

Hardware Parameters

Wire latency	the latency between two network adapters. We ignore the internal structure of the network and assume that the latency is the same for all host-pairs.
Wire bandwidth	the bandwidth between two network adapters. The bandwidth is assumed to be the same for all host-pairs.
DMA bandwidth	the bandwidth between the adapter buffer and host memory.
Packet size	the packet size can be a constant (for cell-based networks) or variable with an upper bound (for packet-based networks).
Packet header size	the header size can be a constant or it can be variable with a lower and upper bound.
Adapter buffer size	size of the outboard buffer.

Software Parameters

OS Send cost	fixed time spent in the messaging layer for every send call.
OS Recv cost	fixed time spent in the messaging layer for every receive call. The recv cost is usually higher than the send cost as it includes the cost of searching messages to match an incoming message and the cost of interrupts from the network adapter.
Packetization cost	cost of allocating and managing the buffer space for each packet. This cost is applied only till enough packets have been created to fill the packet pipeline from/to processor.
Packet-copy cost	cost of an in-memory copy for each packet. This cost is paid when the data is copied between the system and user buffers. This cost is applied only for messages that fit into the system buffer. Larger messages are assumed to be transferred directly to/from the user buffer, a typical optimization commonly found in high-performance communication software.
System buffer size	the system buffer is assumed to be pinned in memory.

Table 1: Parameters for `netsim`.

Microbenchmark	**IBM SP-2**		**Alpha Farm**	
	native	simulator	native	simulator
exchange	0.99	31.78 (32.1)	3.96	31.78 (8.0)
point-to-point	0.51	18.45 (36.2)	4.03	18.45 (4.6)
many-to-one (2 senders)	3.0	42.99 (14.3)	8.03	42.99 (5.4)
many-to-one (3 senders)	6.16	54.79 (8.9)	11.88	54.79 (4.6)
many-to-one (4 senders)	10.23	72.94 (7.1)	24.46	72.94 (3.0)

Table 2: Comparison of native execution speed and simulation speed (in seconds). In the simulator columns, the number in the parenthesis is the relative slowdown of `netsim` compared to the native system.

500 times and the average value was taken as the measure to be computed. To avoid contamination due to unrelated intermittent network activity, data points which differ more than 3 times the standard deviation from the median are classified as outliers and eliminated. Standard deviation was used as a measure of error in measurement. To avoid *cold-start* effects, if any, measurements were taken only after running the experiment 100 times.

The simulation slowdown for the two platforms is shown in Table 2. The numbers in the columns labeled *SP-2* and the *Alpha Farm* are the total time for all nodes in each experiment. The results indicate that `netsim` achieves its accuracy at the cost of a 7-36 fold slowdown for the SP-2 and a 3-8 fold slowdown for the Alpha Farm. To place this result in context, Benveniste and Heidelberger [6] report that a detailed sequential simulator takes 1 day on a workstation to simulate 1 second of simulation time for a 128 node SP-2 configuration.

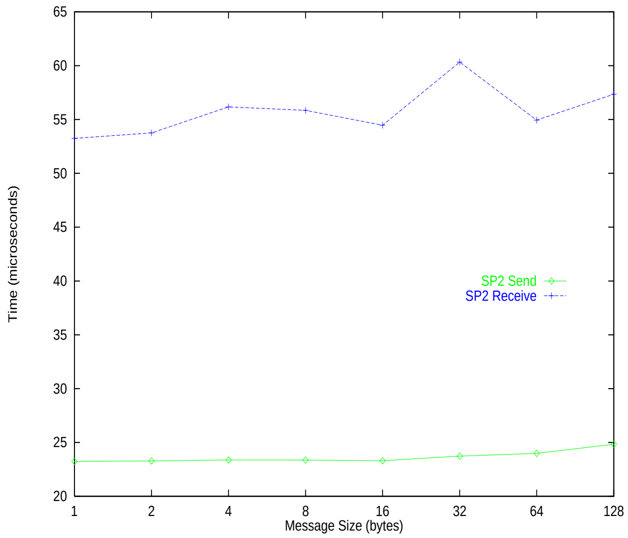
3.1 Case Study - I (IBM SP-2)

The SP-2 network is constructed from 8-input-8-output switching elements which can forward packets from any input port to any output port. These elements are organized as 4×4 bidirectional switches and are grouped into 16-processor units called *frames* which contain 2 layers of bidirectional switches. The bisection bandwidth of this network scales linearly with the number of processors. In this study, we used a 16-node SP-2 running AIX 3.2.5. All the nodes in this machine are the so-called *thin nodes*.

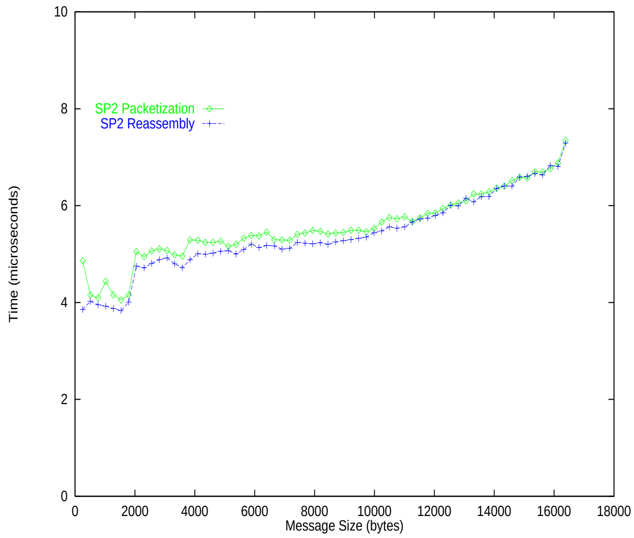
3.1.1 Customization

We obtained the hardware parameters for the IBM SP-2 from two articles in the *IBM System Journal vol 34, number 2* – Stunkel *et al*'s report on the SP-2 network[21] and Snir *et al*'s description of the messaging software [17].

Figure 3 shows the results of the `send` and `recv` programs and Figure 4 illustrates the memory bandwidth characteristics of the SP-2. Note that the packetization and reassembly costs are quite close to each other. For small messages, MPL makes two copies, one from user buffer to system buffer (called *the pipe input buffer*), and the other from the system buffer to virtual switch interface which contains the pinned pages for the DMA interface [17]. In order to obtain the costs of packetization, we subtract the cost of two copies (2 μs) from the mean per packet overhead to arrive at 3μs. The system buffer size is determined from the `send` bandwidth profile as the point of



(a) Messaging Costs



(b) Packetization (per packet)

Figure 3: Messaging send/recv and packetization costs for IBM SP-2

Hardware Parameters		Software Parameters	
Wire latency	1.3 μ s	Packetization cost	3 μs
Wire bandwidth	40 MB/s	OS Send cost	23 μs
DMA bandwidth	80 MB/s	OS Recv cost	55 μs
Packet size	$\leq$ 255 bytes	packet-copy cost	1 μs
Packet header size	$\geq$ 6 bytes	System buffer size	8 KB
Adapter buffer size	4 KB		

Table 3: Simulator Parameters for the IBM SP-2.

inflection where a significant bandwidth drop happens (see Figure 5(a) for corroboration). Values of the `netsim` parameters for the SP-2 are shown in Table 3.

3.1.2 Evaluation

Results from `point-to-point` are presented in Figure 5. As can be seen from the graph, the simulator is able to model the waiting time fairly accurately across a seven-orders-of-magnitude increase in the message size. Similarly, the results for `exchange` show that the simulator is able to model the application-level bandwidth with an error rate of 2-6% for almost all message sizes and a maximum error of 12%. The results for `exchange` are shown in Figure 6.

The results for `many-to-one` are presented in Figures 7, 8 and 9. For the graphs showing sender bandwidth, processors are sorted in the order of decreasing bandwidth, so that the processor with the highest bandwidth in the experiment is labeled "processor 1", next highest as "processor 2" and so on. This is done to eliminate the non-deterministic processor ordering effects (barrier completion times, network scheduling, network state etc) due to which each processor can achieve significantly

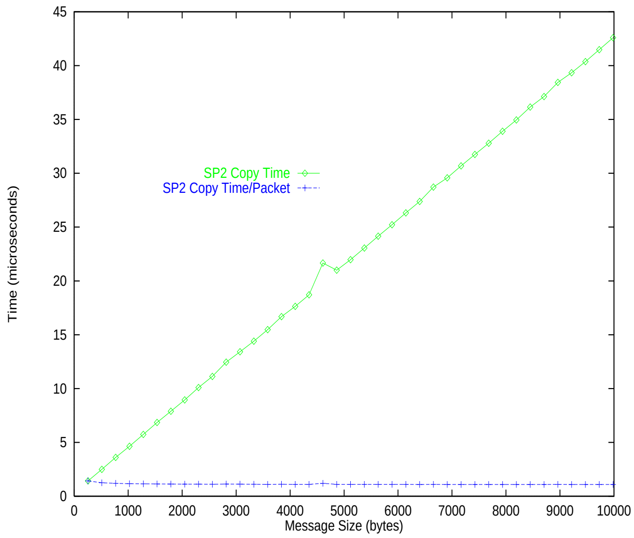


Figure 4: Memory bandwidth of IBM RS6000/390 and per packet copy cost

different performance in different iterations of a sequence of experiments. "Processor 1", in this case, is the fastest processor, "processor 2" is the next fastest and so on.

Even though **netsim** models the SP-2 software and memory latencies quite well, it slightly underestimates the bandwidth for multiple senders before the network bandwidth saturates. This effect is caused by the presence of shared buffers in each of the switching elements, which the simulator does not model. Not modeling the switch buffers has the opposite effect on the reciever bandwidth. Connections are established in **netsim** whenever a packet in the source adapter is ready to transmit. After a connection is established, the simulated receiver adapter favors the receipt of packets from the last connected adapter. In the SP-2 network, however, switch elements make the decisions independently for each packet. As the network gets flooded, the switch buffers (and the adapter buffer) contain packets not only from the currently transmitting node but also from other nodes. As a result, the simulator overestimates the buffer availability resulting in a small overestimate of reciever bandwidth.

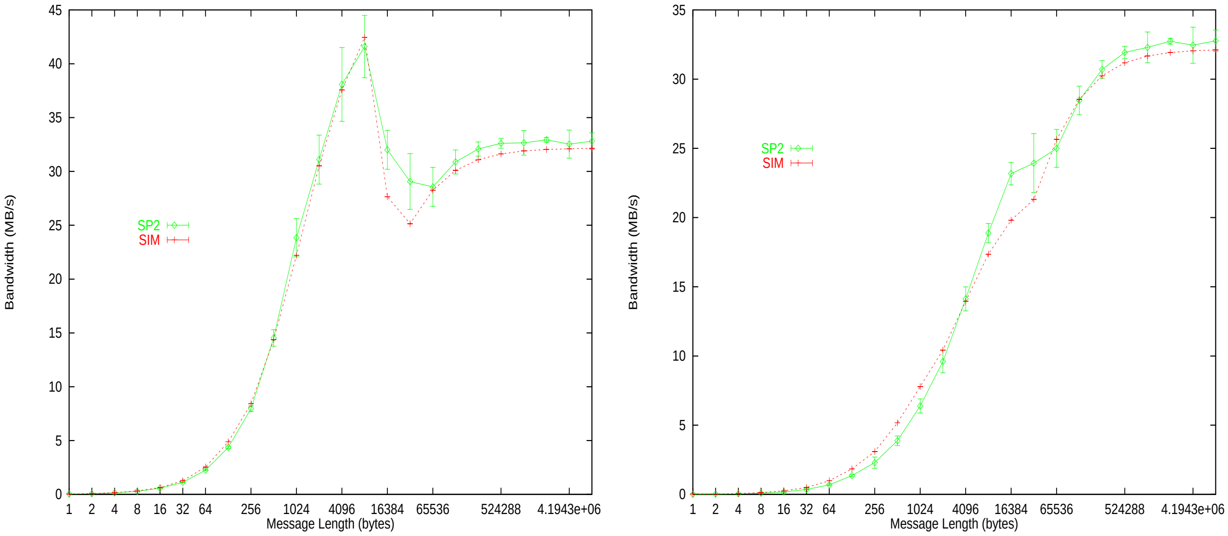
3.2 Case Study - II (Alpha Farm)

The Alpha Farm used in the study consists of ten DEC Alpha 2100 4/275 four-processor SMP workstations connected by a Digital GIGAswitch/ATM. For messaging, we used MPI-CH, a portable implementation of MPI from the Argonne National Labs [13]. The workstations run Digital Unix V3.2D-1.

3.2.1 Customization

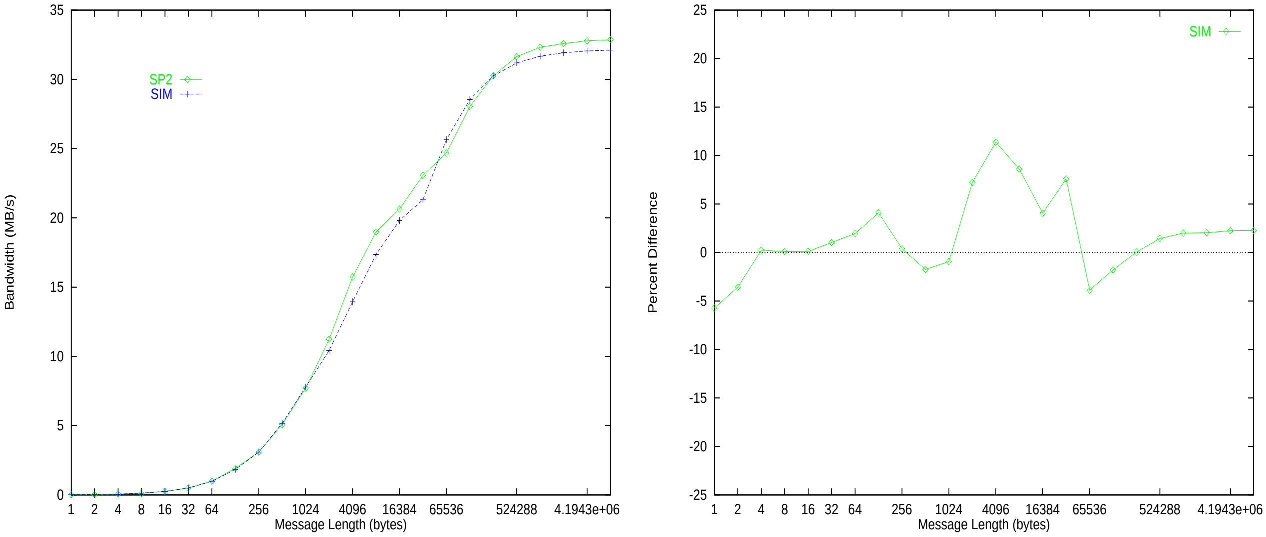
We obtained the hardware parameters for the GIGAswitch/ATM from [18] and from Digital's web site `http://www.networks.digital.com:80/dr/gigaatm/descrip/gigaatm.ps`. We used a different switch latency ($2\mu s$) than the $10\mu s$ mentioned at the web site. With a 10 μs latency per packet, a 155 Mbit/s network can yield no more than 4 MB/s, far lower than the bandwidth actually measured.

Figure 10 shows the results of **send** and **recv** programs, illustrating the costs of the messaging layer. We determined from Figure 10(b) that the mean packetization cost is 3 μs, send cost is 265



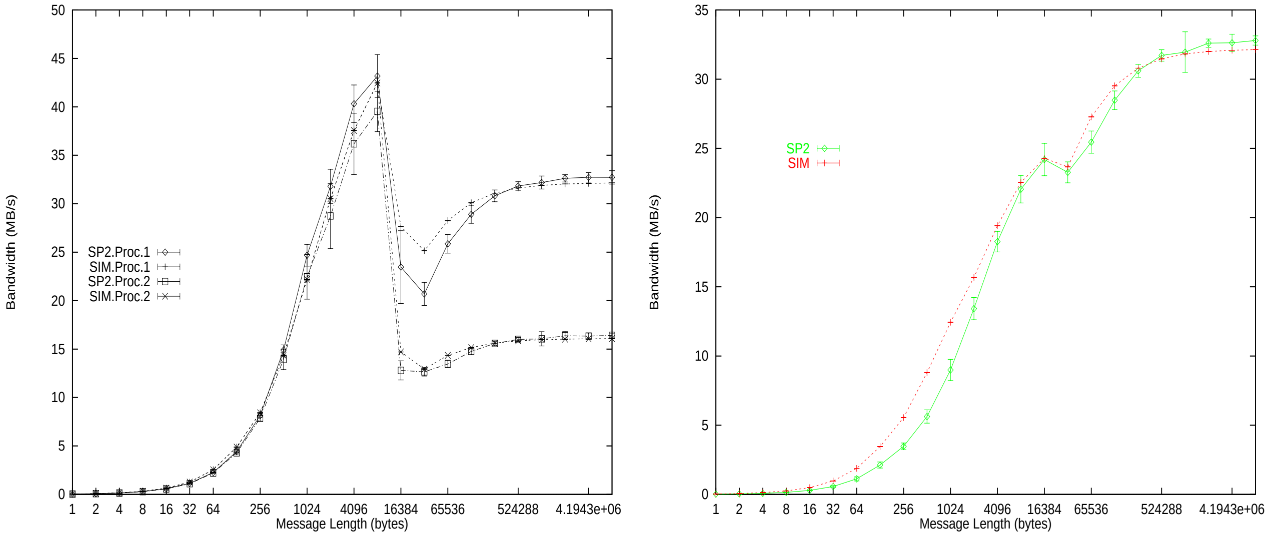
(a) Sender Bandwidth (b) Receiver Bandwidth

Figure 5: `point-to-point` results for the SP-2



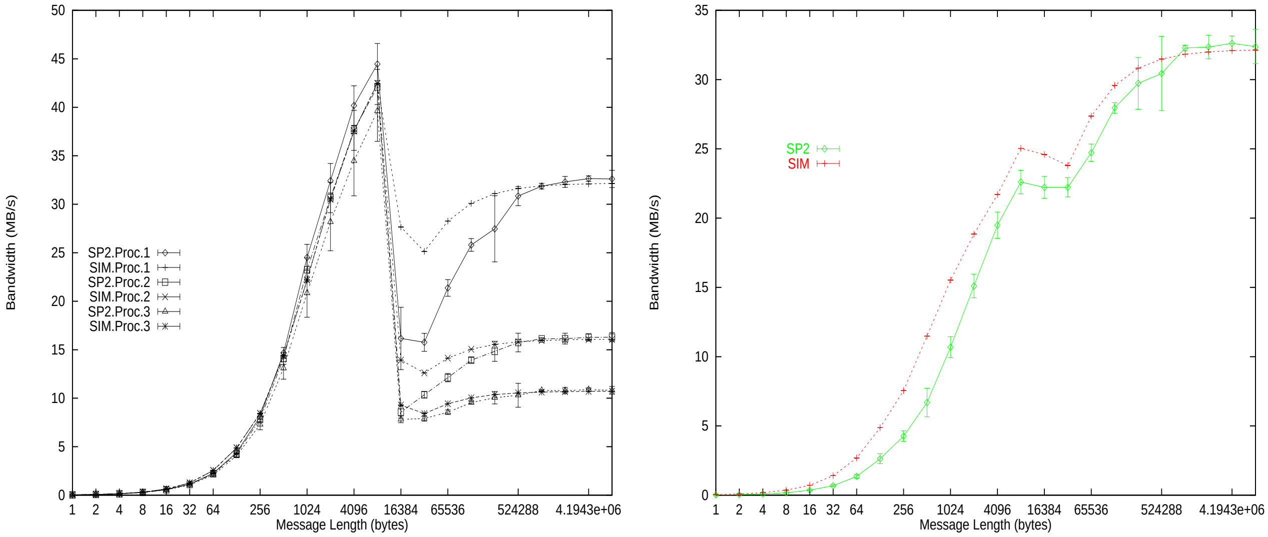
(a) Bandwidth Profile (b) Relative Error

Figure 6: `exchange` results for the SP-2



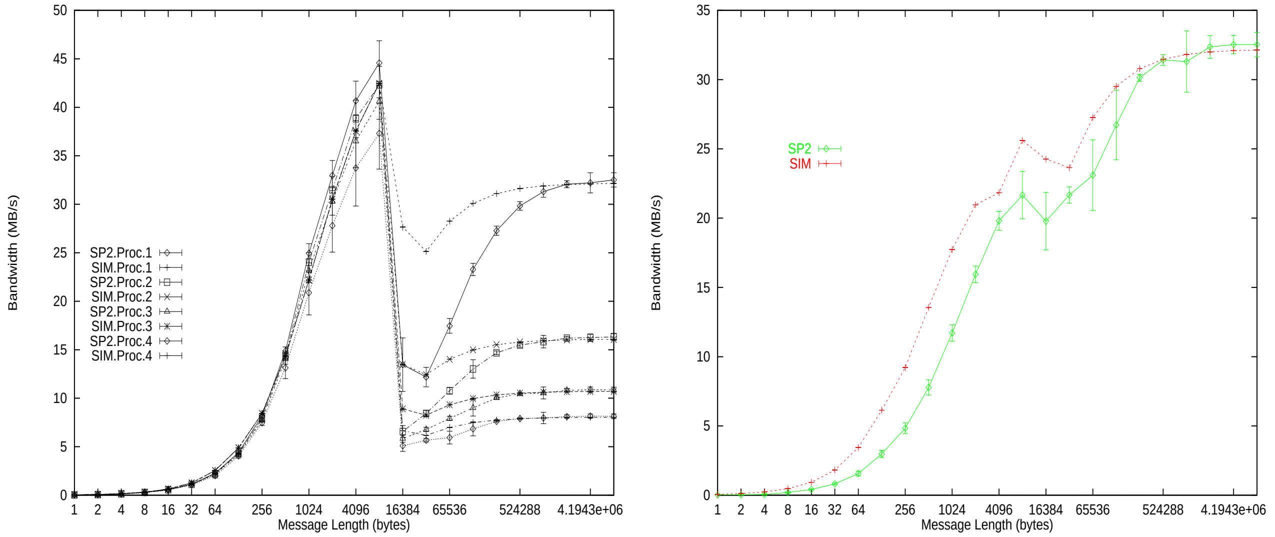
(a) Sender-side (b) Reciever-side

Figure 7: `many-to-one` results for the SP-2, 2 senders.



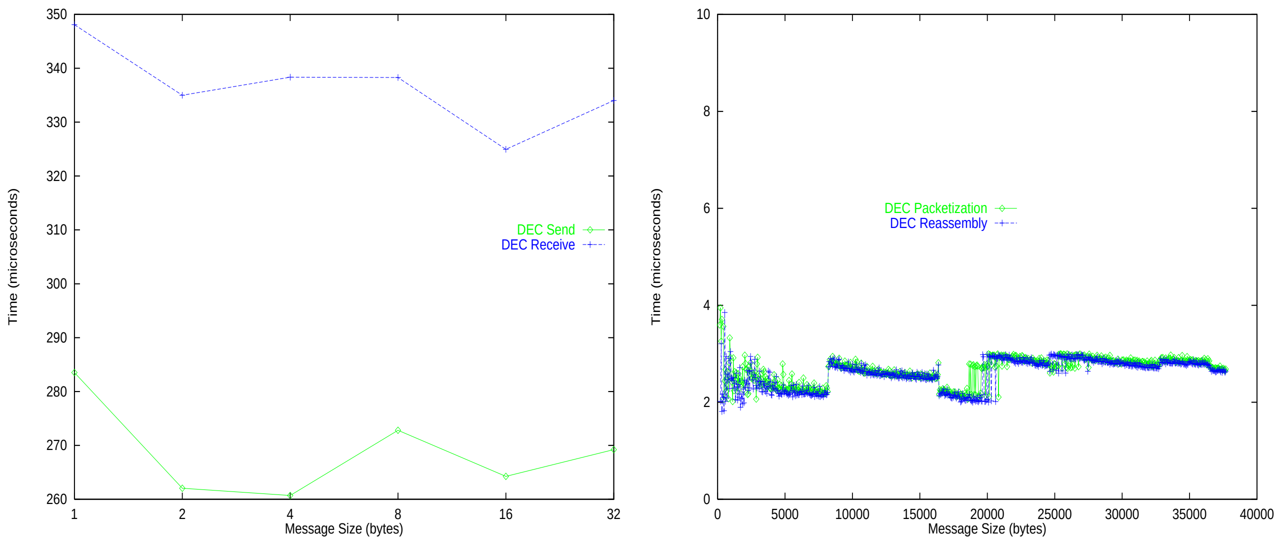
(a) Sender-side (b) Reciever-side

Figure 8: `many-to-one` results for the SP-2, 3 senders



(a) Sender-side (b) Reciever-side

Figure 9: `many-to-one` results for the SP-2, 4 senders.



(a) Messaging Costs (b) Packetization (per packet)

Figure 10: Messaging send/recv and packetization costs for DEC Farm

Hardware Parameters		Software Parameters	
Switch Latency	2 μ s	Packetization ovhd	3 μs
Network Bandwidth	155 Mbit/s	OS Send Overhead	265 μs
DMA Bandwidth	155 Mbit/s	OS Recv Overhead	335 μs
Packet Size	53 Bytes	packet copy ovhd	0.2 μs
Packet Header	5 Bytes	Processor Buffer	16 Kbytes
Adapter Buffer	16 Kbytes		

Table 4: Simulator Parameter Customization for Alpha Farm.

μs and recv cost is 335 μs. Memory copy cost per packet for the Alpha 2100 4/275 was determined to be 0.1 μs using `bcopy` (Figure 11). Simulation parameters for the Alpha Farm are presented in Table 4.

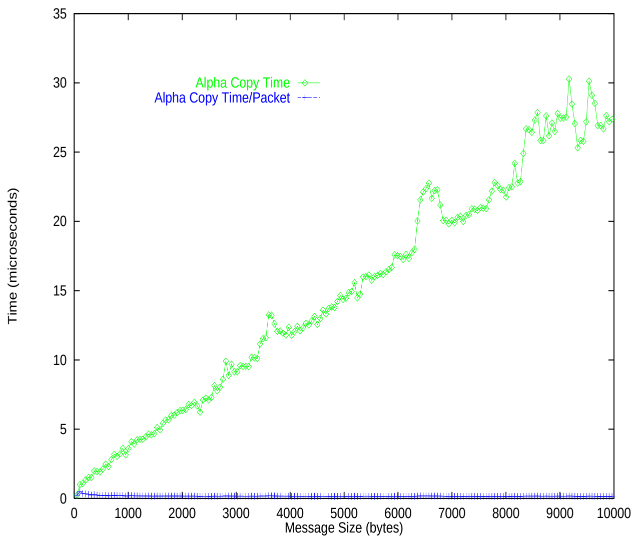
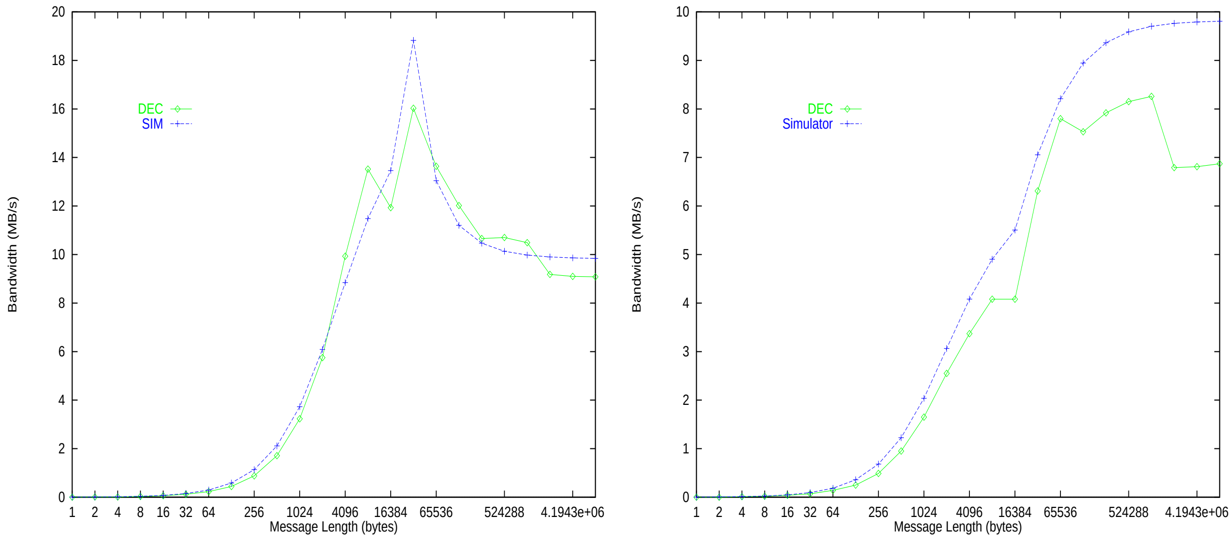


Figure 11: Memory bandwidth of Alpha 2100 4/275 and per packet copy cost

3.2.2 Evaluation

Results from `point-to-point` are reported in Figure 12. The agreement between the simulator and the actual execution is not as good as that for the SP-2. Nevertheless for message sizes smaller than 1 MB, the simulator is able to model the behavior with relatively small error. For message sizes larger than 1 MB, the performance of MPI-CH drops sharply and unexpectedly – as can be seen in the dip in the bandwidth curve at the right end of the graph. We believe that this is a performance bug specific to MPI-CH on Alpha Farms and is not a property of MPI *per se*. Our belief is based on the fact that on the SP-2, MPI-F (IBM's native MPI) does not show this effect and is able to achieve bandwidths close to those achieved by MPL. We expect that a vendor-supplied MPI implementation on the Alpha Farm will not show such behavior and that results for such an implementation would be in better agreement with the simulator.

The results for `exchange` show a similar pattern (see Figure 13). For message sizes less than



(a) Sender Bandwidth (b) Receiver Bandwidth

Figure 12: `point-to-point` results for Alpha Farm

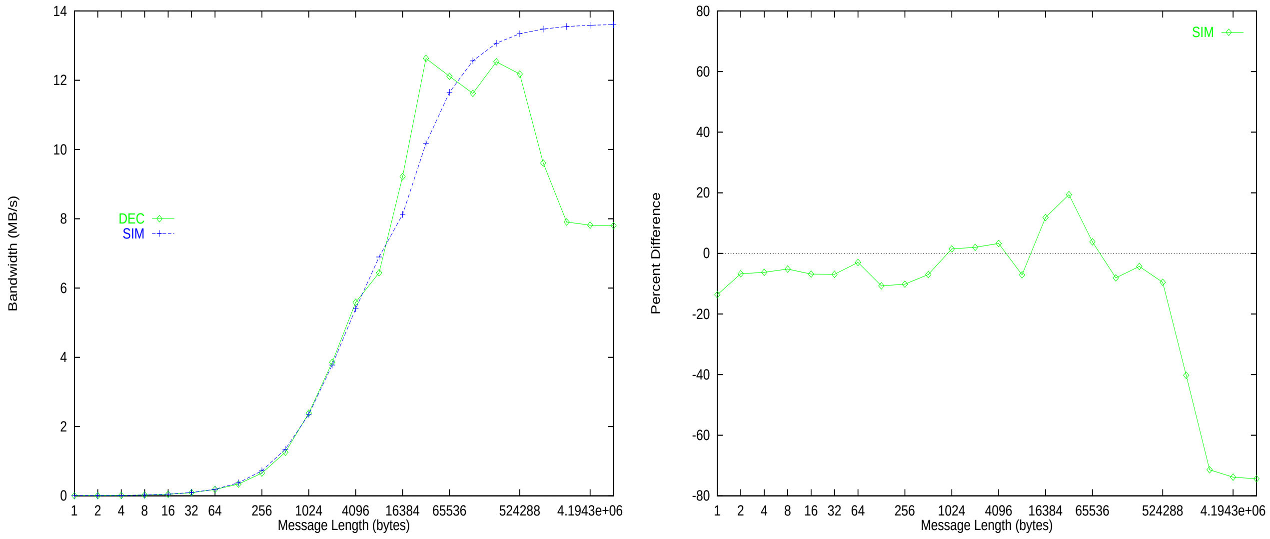
512 KB, the simulator is able to model the Alpha Farm to within 10% error. For larger messages, the MPI-CH performance bug causes the graphs for the simulator and the actual execution to diverge.

Results for `many-to-one` are presented in Figures 14, 15 and 16. The numbers for the sender-side bandwidth are sorted (as mentioned in the previous section). A curious effect to note is that the bandwidth of a congested sender using MPI-CH *increases*. This is unexpected. As congestion increases, achievable bandwidth should drop (which is, in fact, what happens for MPL on the SP-2 – see section 3.1). We speculate that this increase is due to source buffering for large messages. This speculation is partially supported by the fact that for all three cases (2, 3 and 4 senders), the bandwidth of the slowest node is close to the application-level bandwidth reported by `exchange` and that the bandwidth reported by other processors grows with the number of senders. `Netsim` is geared towards optimized libraries which do not make use of source buffering for large messages. As a result, `netsim` does not capture the sender behavior as well as it does on the SP-2.

4 Related Work

Many analytical models have been developed for analyzing network interconnects [1, 3, 9, 16]. Even though it is more convenient to use analytical models for the analysis, it is very difficult to obtain accurate models for complex systems. Analytical models are usually complemented with simulation for the missing accuracy. We resort to simulation in our work, because modeling a full workstation cluster, including network and I/O subsystems and their associated software is a daunting task.

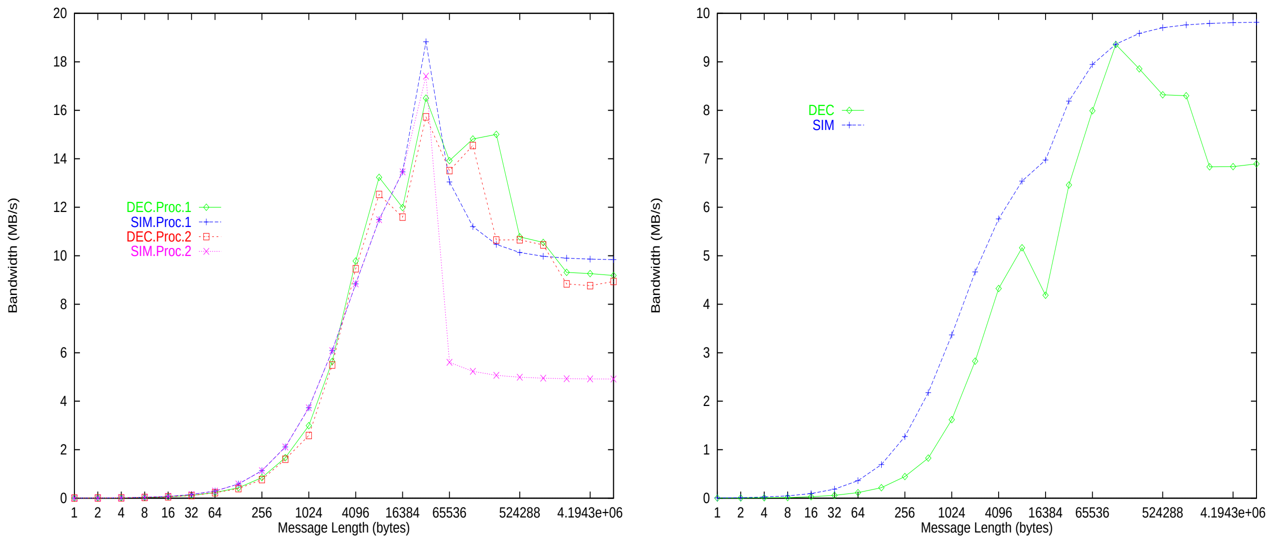
Other simulation models for network interconnects have been built at various levels of detail, accuracy and speed [5, 6, 15]. All these simulators are specific to a single architecture and perform detailed hardware simulation. `Netsim` differs from these simulators in that it is a high-level simulator and that it models both the hardware and the software. Furthermore, it is customizable with a small number of parameters that can be easily determined by simple tests and from hardware



(a) Bandwidth Profile

(b) Relative Error

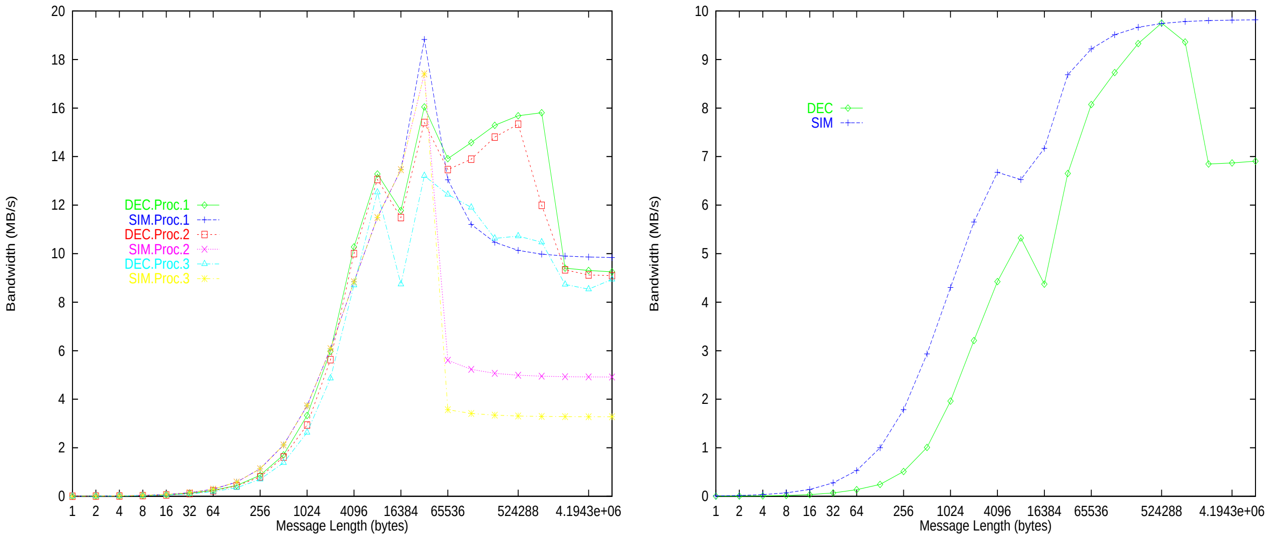
Figure 13: `exchange` results for Alpha Farm



(a) Sender-side

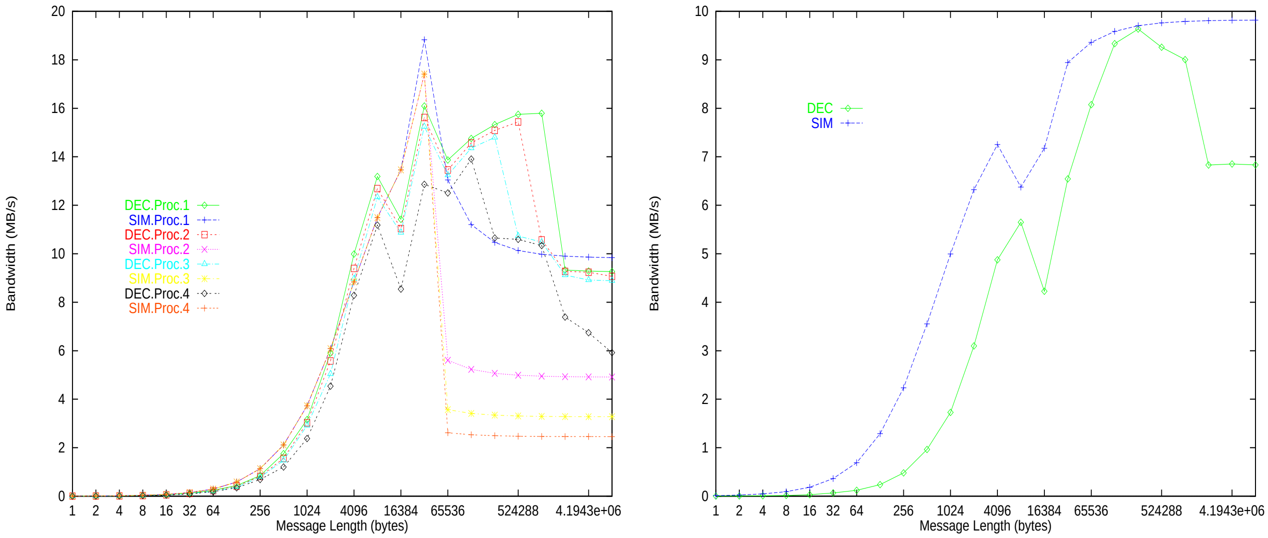
(b) Receiver-side

Figure 14: `many-to-one` results for the Alpha Farm, 2 senders.



(a) Sender-side (b) Receiver-side

Figure 15: `many-to-one` results for the Alpha Farm, 3 senders.



(a) Sender Side (b) Receiver Side

Figure 16: `many-to-one` results for the Alpha Farm, 4 senders.

specifications.

Customizable simulators are not a new idea. Customizable simulators, such as Proteous [10] and Tango [7] have been built previously. The difference between Proteous and `netsim` is that the former is an execution-driven simulator for k-ary hypercubes that can be customized by programming the architecture in the simulator whereas `netsim` is built for point-to-point high-speed workstation networks and can be easily customized by setting a small set of parameters. Tango simulates shared memory multiprocessors.

Fast and accurate network simulators are a desirable commodity and various techniques have been proposed to speed up complex simulations, such as parallel and distributed simulation [6, 12]. Literature in this area has been focused on techniques for parallelization and synchronization of simulation events on large parallel machines, achieving reasonable speedups for most of the cases. At this point, we do not consider using a parallel simulation infrastructure for `netsim`.

5 Conclusions

In this paper we described `netsim`, a customizable simulator for modern packet-switched workstation networks that is accurate enough to be used for application level performance analysis yet is easy enough to customize for multiple architectures and software configurations. `Netsim` can be rapidly customized, even by application programmers. Customizing `netsim` for a new platform requires the user to determine the values for six hardware and five software parameters. The hardware parameters can be obtained from information made publicly available by the vendor and the software parameters can be determined by running a small number of test programs.

We presented two customization case studies: a 16-node SP-2 with a multistage switch and a 10-node four processor Alpha workstation farm having a cross-bar ATM switch. We evaluated the customized versions of `netsim` using a suite of low-level network microbenchmarks commonly used for building higher levels of networking software. Our results suggest that `netsim` is accurate enough for application-level performance analysis, successfully modeling the two test platforms with a 2-6% and a 10% error rate, respectively, for most test cases. We also show that it is of practical use, with a 7-36 fold slowdown for SP-2 simulations and a 3-8 fold slowdown for Alpha farm simulations.

As an important side result, we showed that for high-performance point-to-point networks, modeling end-point congestion is sufficient for a reasonably accurate simulation and that cross-traffic congestion contributes little, if any, to application-level performance.

Acknowledgments

We would like to thank Alan Sussman for his comments on a previous version of this paper and Jeff Hollingsworth for useful discussions on this and related research.

References

[1] Gheith A. Abandah and Edward S. Davidson. Modeling the Communication Performance of the IBM SP2. In *In Proceedings of 10th International Parallel Processing Symposium*, pages 246–257, April 1996.

[2] Anurag Acharya, Mustafa Uysal, Robert Bennett, Assaf Mendelson, Michael Beynon, Jeff Hollingsworth, Joel Saltz, and Alan Sussman. Tuning the Performance of I/O-Intensive Parallel Applications. In *Proceedings of the 4th IOPADS*, pages 15–27, Philadelphia, PA, May 1996.

[3] Anant Agarwal. Limits on Interconnection Performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[4] T.E. Anderson, D.E. Culler, and D.A. Patterson. The Berkeley Networks of Workstations (NOW) project. In *Digest of Papers, COMPCON'95. Technologies for the Information Superhighway*, pages 322–6, March 1995.

[5] C. Benveniste and Y. Hsu. Performance Evaluation of Central Queue Arbitration Policies for the Vulcan Parallel System. In *Proceedings of the 1994 Summer Computer Simulation Conference*, 1994.

[6] Caroline Benveniste and Philip Heidelberger. Parallel Simulation of the IBM SP2 Interconnection Network. Research Report RC 20161, IBM, August 1995.

[7] H. Davis, S.R.Goldshmidt, and J. Hennessy. Multiprocessor Simulation and Tracing Using Tango. In *Proceedings of ICPP*, August 1991.

[8] J.J. Dongarra, S.W. Otto, M. Snir, and D. Walker. A message passing standard for MPP and workstations. *Communications of the ACM*, 39(7):84–90, July 1996.

[9] D. Culler et. al. LogP: Towards a Realistic Model of Parallel Computation. In *Proceedings of ACM Symposium on Principles and Practice of Parallel Programming*, pages 154–164, February 1993.

[10] E. A. Brewer et. al. PROTEOUS: A High-Performance Parallel-Architecture Simulator. Technical Report MIT/LCS/TR-516, MIT, September 1991.

[11] Nanette J. Boden et. al. Myrinet - A Gigabit/second Local-Area Network. *IEEE Micro*, February 1995.

[12] Q. Yu et. al. Time-driven Parallel Simulation of Multistage Interconnection Networks. In *Distributed Simulation*, pages 191–196, 1989.

[13] William Gropp et. al. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. http://www.mcs.anl.gov/mpi/index.html, 1995.

[14] Mark Hill. The Wisconsin COW. http://www.cs.wisc.edu/ wwt/cow.html, Feb 1995.

[15] C. P. Kruskal and M. Snir. The Performance of Multistage Interconnection Networks for Multiprocessors. *IEEE Transactions on Computers*, C-32:1091–1098, December 1983.

[16] R.W.Hockney. Performance Parameters and Benchmarking of Supercomputers. *Parallel Computing*, 17:1,111–1,130, 1991.

[17] Marc Snir, Peter Hochschild, D. D. Frye, and K.J. Gildea. The Communication Software and Parallel Environment of the IBM SP-2. *IBM Systems Journal*, 34(2), 1995.

[18] Robert J. Souza, P.G. Krishnakumar, Cuneyt M. Ozveren, Robert J. Simcoe, Barry A. Spinney, Robert E. Thomas, and Robert J. Walsh. GIGAswitch System: A High-performance Packet-switching Platform. *Digital Technical Journal*, 6(1):9–22, 1994.

[19] Peter A. Steenkiste. A Systematic Approach to Host Interface Design for High-Speed Networks. *IEEE Computer*, pages 47–57, March 1994.

[20] T. Sterling, D.J. Becker, J.E. Dorband, D. Savarese, U.A. Ranawake, and C.V. Packer. Beowulf: a parallel workstation for scientific workstation. In *Proceedings of the 24th Internationa Conference on Parallel Processing*, pages I/11–14, August 1995.

[21] Craig B. Stunkel, Dennis G. Shea, Bulent Abali, Mark Atkins, Carl A. Bender, Don G. Grice, Peter H. Hochschild, Douglas J. Joseph, Ben J. Nathanson, Richart A. Swetz, Robert F. Stucke, Michael Tsao, and Philip R. Varker. The SP2 High-Performance Switch. *IBM Systems Journal*, 34(2):185–204, 1995.