A Performance Analysis of Network I/O in Shared-Memory Multiprocessors

Chandramohan A. Thekkath, Derek L. Eager†,
Edward D. Lazowska, and Henry M. Levy

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

Technical Report 92-04-04

Abstract

Network I/O performance has not kept pace with processor performance in modern multiprocessor workstations and servers. This paper studies this problem in the context of two such systems: a 5-processor DEC SRC Firefly and a 20-processor Sequent Symmetry. We devise an analytic model of network I/O. We parameterize it using measurements of the two systems. We use it to study the performance of these systems as they currently exist, and to explore certain feasible modifications that can enhance performance. Our experiments indicate that network controllers designed for uniprocessors may not be suitable for multiprocessor configurations. They also suggest that a high performance I/O bus can significantly improve performance for larger packets. Finally, they lead us to conclude that unlike uniprocessors, the performance improvement due to simpler transport protocols is comparatively small in multiprocessors. These results arise more from the increased parallelism that is available with multiprocessors than just from the increased processing power.

This work was supported in part by the National Science Foundation under Grants No. CCR-8703049, CCR-8619663, and CCR-8907666, by the Washington Technology Center, by the Digital Equipment Corporation Systems Research Center and External Research Program.

† Derek Eager is with the Department of Computational Science, University of Saskatchewan, Saskatoon, SK S7N 0W0, Canada.

A Performance Analysis of Network I/O in Shared-Memory Multiprocessors

Chandramohan A. Thekkath, Derek L. Eager*, Edward D. Lazowska, and Henry M. Levy

Department of Computer Science and Engineering FR-35
University of Washington
Seattle, WA 98195

April 6, 1992

Abstract

Network I/O performance has not kept pace with processor performance in modern multiprocessor workstations and servers. This paper studies this problem in the context of two such systems: a 5-processor DEC SRC Firefly and a 20-processor Sequent Symmetry. We devise an analytic model of network I/O. We parameterize it using measurements of the two systems. We use it to study the performance of these systems as they currently exist, and to explore certain feasible modifications that can enhance performance. Our experiments indicate that network controllers designed for uniprocessors may not be suitable for multiprocessor configurations. They also suggest that a high performance I/O bus can significantly improve performance for larger packets. Finally, they lead us to conclude that unlike uniprocessors, the performance improvement due to simpler transport protocols is comparatively small in multiprocessors. These results arise more from the increased parallelism that is available with multiprocessors than just from the increased processing power.

1 Introduction

Multiprocessors are gaining widespread use as workstations, file servers and timesharing machines. The performance of network I/O has not kept pace with the capabilities of these systems. Host software and controller hardware are often similar to those found in uniprocessor configurations. In this paper we explore the performance of existing software and hardware architectures and we predict the behavior of certain modifications to the I/O subsystem.

We specifically study the performance of network I/O in two shared-memory multiprocessors: the DEC SRC Firefly [11] and the Sequent Symmetry [7]. We examine the effect on overall performance of host hardware and software as well as the controller-host interface. This level of abstraction allows our results to be applicable to a variety of host, operating system, and network combinations.

*Derek Eager is with the Department of Computational Science, University of Saskatchewan, Saskatoon, SK S7N 0W0, Canada.

This work was supported in part by the National Science Foundation under Grants No. CCR-8703049, CCR-8619663, and CCR-8907666, by the Washington Technology Center, and by the Digital Equipment Corporation Systems Research Center and External Research Program.

Previous work [2, 1] on uniprocessors has indicated that software overhead and controller latency are significant in limiting network performance. We wish to study the relative effects of these factors in multiprocessor systems. We have characterized the essential hardware and software components by measuring service demands for the various resources in the two systems mentioned above. The measurement data is used to parameterize a queuing model. We use this model to derive some performance bounds, for example, the maximum rates of network traffic that can be accommodated in these systems, and to predict the effects of changes to the protocol processing overhead, the I/O bus bandwidth, and the controller-host interface. These changes are feasible given most modern operating systems and network controllers.

Analytic queuing models are ideal tools for such a study for several reasons. First, they are relatively efficient and accurate; these models are easily defined, parameterized and evaluated. Second, modeling is cheaper and easier than experimentation for exploring a design space; some of the proposed design changes are difficult to perform. Finally, using a model enhances our understanding and intuition. Analytic models have been used in the past to study network-related performance. For example, Lazowska et al. study the performance of diskless workstations connected by an Ethernet [5], concentrating on the effects of file system parameters such as block size on the performance of remote file systems. Meister, Janson and Svobodova report on the performance of file transfer in local area networks; their paper investigates the performance impact of different implementations of link level protocols on file transfer [8]. Garodia and Vernon assess the effect of network interface units (NIUs) on inter-machine communication [4]; an NIU is capable of off-loading protocol processing from the host and the emphasis of the paper is to study the effect of partitioning the protocol overhead between the host and the NIU.

This paper is organized as follows. Section 2 describes the analytic queuing model and discusses the measurement of the service demands. Section 3 provides a qualitative study of the baseline systems with a view towards capturing some general principles. Section 4 presents a quantitative analysis and studies the effect of the design changes. Section 5 summarizes our main conclusions. The appendix presents the derivation of a queuing network equation that was developed to handle this particular modeling problem.

2 The Systems and the Performance Model

Our study focuses on the DEC SRC Firefly and the Sequent Symmetry. These systems are representative of current shared-memory multiprocessors. In addition they are readily available to us for measurement.

We first give a brief functional description of the two systems. Then we describe the components of our performance model. In Section 3, we study some details of the systems, using the model as a framework.

2.1 An Overview of the Systems

The DEC SRC Firefly is a small-scale multiprocessor with 5 processors: 4 CVAX processors and a single MicroVAX-II. Each CVAX is rated at around 3 MIPS and the MicroVAX-II is rated at 1 MIPS. In the measured environment a cluster of 10 Fireflies are connected by a 10 Mb/sec Ethernet. The Firefly hosts the Topaz operating system. As in many multiprocessor systems, lightweight processes called threads provide concurrency. Remote procedure call (RPC) is the predominant mode of interprocess communication. Network I/O is performed when threads demand or provide services (such as file service) via RPC to threads on other Fireflies.

To transmit a packet, a thread must first acquire a network buffer for its data. Topaz networking software is structured so that only a single buffer is allocated independent of the size of the packet to be sent. Buffers are shared between user and kernel to minimize copying of data between the two levels. Network buffer acquisition is done under a lock to provide mutual exclusion among multiple threads that might be simultaneously trying to send packets. After the buffer is acquired, the thread performs some amount of protocol processing and copies the data into the buffer. The software processing is composed of three parts. First, there is a marshaling cost; this varies with the amount of data to be marshaled. Second, time is spent initializing protocol-specific headers and performing checksums. Third, the packet must be prepared for queuing on the controller. This work includes checking the argument validity, locating the correct network link, and arranging for the packet buffer to be reclaimed. Typically there are sufficient processors in the measured system that threads do not have to delay contending for processors.

The thread then places the network buffer in a shared-memory queue for the network controller module and activates the network controller to begin the network transfer. Access to the network controller is mediated by a lock to ensure mutual exclusion among threads. The controller uses DMA (direct memory access) to move the data over the I/O bus into a silo or staging area from which the data is sent out when the network is free.

Similarly, when a packet is received, the controller DMAs the data from the network to a shared queue of network buffers. The host then makes this data available to the destination thread. This involves additional computational overhead. The shared queue of transmit buffers is distinct from the queue of receive buffers.

The Sequent Symmetry is a shared-memory multiprocessor with 20 Intel i386 processors. Each processor is rated at about 3 MIPS. The Sequent Symmetry supports the Dynix operating system, a derivative of UNIX. Interprocess communication is provided using the standard UNIX socket interface. In the measured environment there is a single Symmetry that remotely mounts files from other machines connected to an

Ethernet. Most network I/O in our system is caused by these remote file accesses.

In general, the message passing mechanism on the Sequent is similar to that of the Firefly described above. However, some of the specifics differ. First, Dynix allocates buffer memory in two sizes: small fixed sized parcels of memory (*mbufs* in the parlance) are allocated if the packet is small. Larger packets are allocated in bigger units (called *clusters*) that can accommodate a maximal Ethernet packet. When the larger size clusters are unavailable, however, medium and large packets will require multiple allocations of mbufs. Multiple visits to the lock that protects the memory pool are required to allocate multiple mbufs. Second, network buffers are not shared between user and kernel; data must be copied between the two levels.

2.2 The Performance Model

To study these systems, both as they currently exist and under various modifications, we model them as queuing networks. The key components to be specified for a queuing model are the service centers, the service demands at these centers, and the workload classes and intensities. The service centers represent the hardware and software resources that we wish to model. The workload classes and intensities describe the types and rates of requests. The service demands represent the average amount of service that each request needs at each service center. The choice of the service centers and the workloads dictate to a large extent the type of modifications that can be studied with the model. After the model is specified it is solved for different values of the parameters. Each solution yields predicted residence times (i.e., the time spent queuing for the service plus the time spent in service) at each center, and thus the overall performance of the system.

2.2.1 The Service Centers

Figure 1 shows the queuing models we used for the Firefly and the Sequent. The center marked *buffer lock* is the lock that the thread acquires before it can obtain a network buffer. The center marked *controller lock* is the lock that the thread must acquire before the packet buffer(s) can be enqueued and the controller activated. The service time at this center is determined primarily by the interface supported by the controller. This makes it convenient to study the effects of host-controller interface modifications. In general, the total amount of time spent in service at these centers depends on the size of the packets since the number of buffers needed may vary.

We use a simple delay center to represent software processing overhead. Protocol header processing, copying, checksumming, and bookkeeping are the major contributors to this cost. Here again we have to model differing service requirements for various network packet sizes; in this case to account for certain

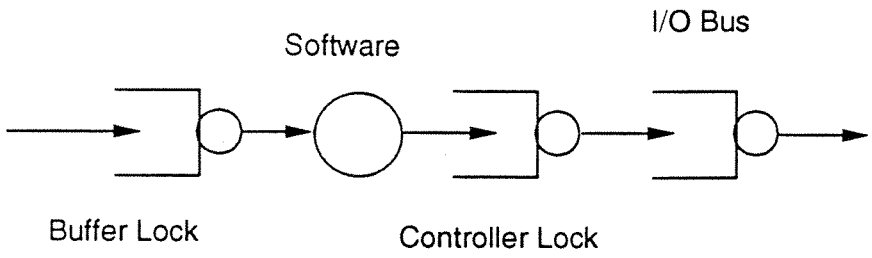

Figure 1: Queuing model

aspects of software processing like checksumming that are sensitive to packet size. An implicit assumption we make in this characterization is that there are sufficient processors in the system so that threads do not have to contend for processors. Constraints on the number of available processors could cause the software overhead to be more significant than what the model predicts.

To model the DMA operation between the network buffer memory and the network controller, we use a queuing center to represent the I/O bus that needs to be acquired and held for the duration of the transfer. This characterization is not strictly correct for the Sequent bus. Unlike the I/O bus on the Firefly, the Sequent bus allows split transactions (i.e., following a read or write request, the bus is available for other transactions even before the the reply to the request is ready). However, to sustain a bandwidth of 10 Mbits/sec on the Ethernet, the 53 Mbytes/sec bus needs to use only one out of every 50 cycles. This implies that, effectively, there is no queuing at the bus in any case.

There are certain aspects of network I/O that our model does not account for. First, we do not explicitly model received packets. Received packets cause contention at the buffer lock and at the I/O bus. This can be accounted for by inflating the service requirements for the transmitted packets at these centers. The controllers we consider have separate send and receive queues, and thus received packets and transmitted packets will not contend for the same queue. Second, typical network controllers introduce some delay in getting packets from their staging area to the link, even when the network is unloaded. This delay could be a function of the packet size or it could be a constant if the controller uses pipelining. Our model does not account for this delay nor does it account for the delay in accessing the network which could vary considerably depending on the medium access protocol. For instance, for FDDI networks each node added to the ring causes additional delay in passing the token and hence in sending out data.

2.2.2 Workload Characterization

The "customers" in the model are the network packets that make their way from the address space of a thread to the network controller. En route, they encounter various delays at the centers because of service time and contention. We represent the workload as a transaction workload; i.e., we specify the rate at which network packets are generated on a machine. Since the size of the packet affects the requirements at each center we use multiple customer classes, with each class representing a particular packet size.

The distribution of packet sizes is obtained by measurement; system performance for various rates of packet generation is then studied given this distribution. Measurements of packet size in the two systems indicate the most frequent packet sizes are in four distinct ranges. Correspondingly, we chose four packet sizes—74, 150, 650 and 1514 bytes—as representative of these ranges, giving us an open model with four classes. The arrival rate of a class represents the generation rate of packets of the corresponding size.

The packet was chosen as the unit of work, rather than a high level request like an RPC call or a socket send operation, because these are inherently programming environment specific and cannot easily be generalized across various operating systems. Given the performance measures in terms of packet residence times it is possible to study a specific communication paradigm such as RPC.

We measured the relative frequencies of the packet size ranges of interest in the two systems. In our model evaluations, we keep the arrival rates for the various classes in the same fixed ratios as those measured.

2.2.3 Service Demands

Service demands specify the service requirements of a customer of each class at each service center. Typical queuing models require only the aggregate service demand (summed over all visits) at each center. However, our model has a transaction workload with four customer classes (one for each packet size) with different service requirements per visit at several of the centers, which are scheduled FCFS. Thus, the standard queuing model solution is not applicable [6]. A modified approach is required to approximately evaluate the model and is presented in the appendix. This solution technique requires the per-visit service requirement and the number of visits at each center. For the special case when the per-visit service requirement at each center is identical for all classes, this reduces to the standard solution for a multi-class open model with FCFS scheduling.

To obtain the service demands for the Firefly, we made RPC calls with various packet sizes. We instrumented the kernel as well as the user level RPC system to obtain the elapsed time at the various centers: the buffer lock, the network I/O procedures, and controller queue lock. The Firefly network software is struc-

Packet Size (bytes)	Service Demand			
	Buffer Lock	Software Overhead	Controller Lock	I/O Bus
74	46	495	537	70
150	46	591	537	109
650	46	734	537	368
1514	46	1061	537	815

Table 1: Firefly service demands (in μseconds)

Packet Size (bytes)	Service Demand			
	Buffer Lock	Software Overhead	Controller Lock	I/O Bus
74	34	793	311	3
150	34	846	311	6
650	54	993	311	24
1514	54	1299	311	57

Table 2: Sequent service demands (in μseconds)

tured so that each packet makes one visit to each of these service centers per transmission. Determining the service requirements for the Firefly I/O bus by direct measurement proved more difficult. Instead, we used previously published figures [9] and interpolated for packet sizes of interest. The service demands for the Firefly are shown in Table 1.

Table 2 summarizes the service demands for the Sequent. To arrive at the service demands we made socket calls of various sizes. We modified the kernel to measure the elapsed times, the number of times mbufs were requested, and the number of times clusters were requested for each packet size. We measured the buffer lock overhead per visit to be 17 μseconds for mbufs and 37 μseconds for the larger clusters. In the model we do not consider the case when a packet's data does not fit into one mbuf and the system has run out of clusters. Since a single mbuf is allocated for the protocol header independent of packet size, the service demand is 34 $(17 + 17)$ μseconds for small packets, and 54 $(17 + 37)$ μseconds for large packets. As in the case of the Firefly, the time spent on the bus was estimated from the manufacturer's specifications for the bus.

2.2.4 Summary

This paper concerns system design. The model is used simply as a means to that end—as a vehicle for structuring our study of the system, and for extrapolating our understanding to configurations that cannot be measured directly. The model itself is conservative, and will not receive much attention.

3 The Systems in Greater Detail

In this section we study the two systems with a view towards extracting some general principles. Our objective is not so much to contrast the two systems as it is to find common points. There are basically four features that affect performance in these systems: the memory allocation policy, amount of data copying done, the design of the network controller/driver software, and the I/O bus bandwidth. These performance factors are not peculiar to multiprocessors; much of what we say here is applicable to uniprocessor environments as well. The treatment in this section is intended to be a qualitative examination of the two systems; a quantitative analysis is deferred until the next section.

3.1 Memory Allocation

Recall from subsection 2.1 that we differentiate between network packets and network buffers—one or more fixed sized memory regions that happen to contain a network packet. On the Firefly, buffer allocation is simplified because it is done once per packet with the restriction that the packet size cannot be larger than the maximal Ethernet transmission unit size. Larger packets are broken up at user level and require multiple calls to the network I/O subsystem. Thus, the Firefly buffer allocation scheme tends to overallocate memory given the fact that most packets are quite small. In our model, we have ignored the fact that the number of buffers available in the system is finite. Conversely, the average number of customers in the system is an output of the model which can be used to estimate the number of buffers required. Realistically, under high load one would expect buffers to be a scarce resource on the Firefly. The Sequent scheme conserves the total amount of buffer memory. But, as mentioned earlier, multiple buffers may have to be allocated even for a packet smaller than the Ethernet maximum transmission unit size. Thus, if the proportion of clusters to mbufs is not chosen judiciously, medium and large packets could suffer increased delay due to multiple visits to the buffer lock center.

A related issue is the way headers are allocated. Topaz allocates space for headers and data at the same time while Dynix allocates protocol headers at different times. This has the effect of making the Dynix software processing more complex, which adversely affects latency under light load.

3.2 Protection and Copying

The Firefly is intended to be a personal workstation where the user is trusted. The Topaz operating system trades protection for performance by mapping the buffer memory into kernel as well as user space. The only copying overhead in the Topaz system is the cost of marshaling the data into the packet. The network controller has access to processor memory and it directly DMAs the packet to its memory.

Dynix, in contrast, is a traditional timesharing operating system with strict notions of user and kernel spaces. Thus, copying from the user's buffer to the kernel's mbufs is done. In addition, the Ethernet driver program performs another copy from these mbufs to an area of memory from which the controller can access it for DMA. Copying has a particularly adverse effect on performance because it is a time cost that is roughly proportional to the amount of data being copied.

There are some alternatives to be considered where the cost of getting the buffer to the controller can be kept down without compromising protection. One approach is to use a scatter/gather facility if the controller provides it. This allows the controller to read the header from kernel space and the data from user space. The efficacy of this approach is limited by the bus and by the speed of the controller. Now DMA transfers have to be set up twice and it is also likely that the controller is slower than one that does not support this facility because of the increased complexity in the controller. Another option is to allocate the header and data contiguously but across two adjacent virtual pages: one containing the header and the other the data. Protection can then be achieved by suitably mapping the two page table entries.

3.3 Host-Controller Interface

The interface supported by the network controller dictates the structure of the host device driver. On the Sequent, the low level details of the controller are hidden from the device driver. Even so, there are several points of similarity between the Firefly and Dynix device drivers.

Both drivers use a single ring of descriptors to describe the packets to be sent, and another ring to describe the buffers to be received into. These rings are shared between the host and controller. On the Sequent a single lock mediates processor access to the ring. On the Firefly the situation is slightly complicated. There is a single designated processor that is allowed to communicate with the controller with which it shares the ring. Thus other processors have to activate the specialized processor under a lock.

In a multiprocessor environment, we believe that benefits would arise in having multiple send rings per controller and by giving each processor equal access to the controller. Ideally the number of rings could be configurable by software at controller initialization time. Thus the host could have as many rings as there are processors. Each processor would then get its own ring which can be accessed without a lock. Mutual exclusion between processor and controller is provided with simple memory interlock. This scheme results in both better latency under light load as well as better performance under heavy load. We consider this in a quantitative setting in Section 4.

3.4 I/O Bus

The Firefly uses a 16 Mbit/sec Q-Bus without split transactions to transfer data between processor memory and the controller, whereas the Sequent uses a 53 Mbyte/sec pipelined bus designed to handle the traffic from its 20 i386 processors. In general for both systems a slow bus would limit the rate at which packets can be transferred from the processor's memory to the controller. For small packet sizes the bus is not a bottleneck. For bigger packet sizes and under load, one would expect that the Q-Bus on the Firefly would be more of a limiting factor.

4 Performance Results

In this section we use the model developed in Section 2 to study the performance of the baseline systems, and the effects of some I/O system design alternatives.

4.1 Bounding Analysis

We first consider some simple bounds for our model of the baseline systems. We concentrate on the common and important case of small packet sizes. First we consider the performance under light load. Under light load, the packet transmission latency (i.e., the sum of the service requirements) is 1148 μseconds for the Firefly and 1141 μseconds for the Sequent. Since the Sequent has much more processing power than the Firefly, one might expect the software processing delay to be smaller on the Sequent, leading to a smaller overall latency. However, this is not the case because the Dynix kernel performs significantly more software processing. In fact, for the smallest packet size, software processing on Dynix is more than 1.5 times that on the Firefly. Further, in this case, software overhead for these packets accounts for nearly 70% of the total time on the Sequent. On the Firefly, it is only 43% of the total. As packet size increases the performance of the Sequent gets progressively better relative to that of the Firefly although its software overheads are still higher. The good performance of the Firefly given its processing speed is largely due to its network I/O software, which was specially structured to achieve good latency with multiprocessors, while the Dynix kernel (like the 4.2BSD kernel) tries to preserve flexibility at the expense of speed.

At the other extreme, under high load the rate determining center is the queuing center with the largest service demand. For both systems the service demand at the Ethernet controller lock is the highest. The reason for the high service demand at this center is different in the two systems. On the Firefly, a distinguished I/O processor has to be notified to read and write the I/O space before the Ethernet controller can be prodded into action. On the Sequent, on the other hand, the large service requirement is mostly due to the software

structure of the device driver.

4.2 Performance Under Intermediate Load

Recall that the measured packet size distribution determines the relative arrival rates of the various classes in the model. We solve the model for 10-90% of the maximum possible overall rate. Figure 2 graphs the performance for the Firefly and Figure 3 for the Sequent. The Y axis is the residence time in μseconds for

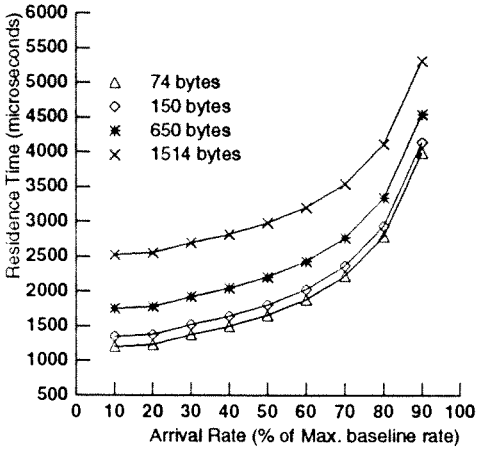

Figure 2: Projected performance under load on the Firefly

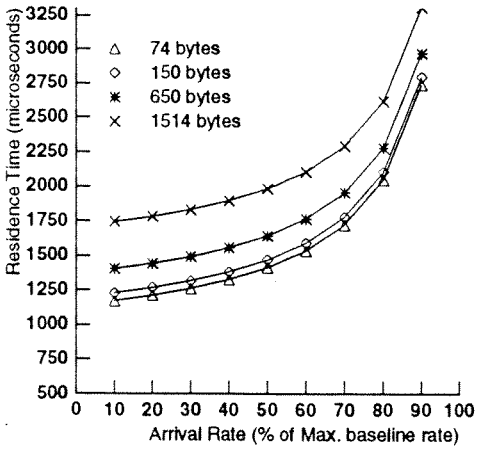

Figure 3: Projected performance under load on the Sequent

each packet size when there is a mix of job classes. The X axis shows the arrival rate as it is varied between 10% of the maximum to 90%. A direct comparison of performance numbers between the two systems would be unfair because the processors, the architecture, the scale and the use are very different. However, we

would like to study some overall trends in the two systems.

The first observation is that at a given load, residence time on the Firefly is more sensitive to packet size than it is on the Sequent. That is, large packets suffer relatively more delay on the Firefly than on the Sequent. This is due to the large delay imposed by the I/O bus on the Firefly for the larger packets. At light loads, for small packet sizes, both systems have similar performance. Another characteristic of both these graphs is that performance degrades fairly rapidly under load. At about 60% of the maximum load, there is significant degradation in residence time in both systems.

In the next subsections, alternative designs of some of the key components are explored with a view of improving performance under load. The design alternatives considered are: (1) using a simplified protocol in both systems, (2) replacing the I/O bus on the Firefly with a faster bus, and (3) modifying the host-controller interface to reduce contention among processors.

4.3 Protocol Overhead

Both systems under consideration use IP/UDP as their transport protocol. Using IP allows the packets to cross local network boundaries, i.e., the sender and receiver can be located on different networks. The cost of using these protocols is primarily the overhead of adding additional headers and performing byte operations like checksums on the packet. As an alternative, one could consider a simpler protocol that trades some generality for ease of processing. For instance, it is possible to use Ethernet datagrams for communications that originate and terminate on the same network. Thus it might be possible to eliminate the overhead of IP/UDP in the common case. In our model, a change in the protocol alters only the amount of service required at the delay center. The new values for the service demands were estimated from the baseline measurements. The model was solved for these new values and results plotted in figures 4 and 5.

Figure 4 shows the effect on the residence time for the Firefly. The primary observation to be made is that this modification reduces residence time slightly for all loads. The reduction is proportional to the size of the packet, because in general, the amount of protocol processing increases with the length of the packet. There is no discernible improvement in tolerance to high load. As before, at and past loads of about 60% of the maximum arrival rate, the residence time starts degrading. This is as expected, because, as seen from Table 1, under heavy load it is the controller lock that is the bottleneck. In another experiment, we (arbitrarily) decreased the total software processing overhead by as much as 50%. The gain in latency was not commensurate with the change in the software overhead. A 50 percent reduction in processing resulted in about 18 percent savings for the smallest packet size. For the largest packet size the saving was about 19 percent. These results are plausible because the software overhead accounts for only a fraction of the total

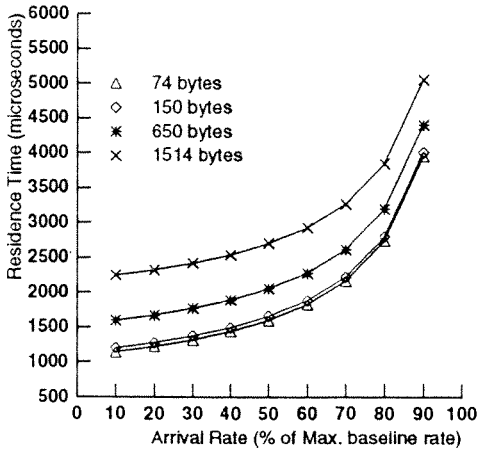

Figure 4: Effect of reducing protocol overhead on the Firefly

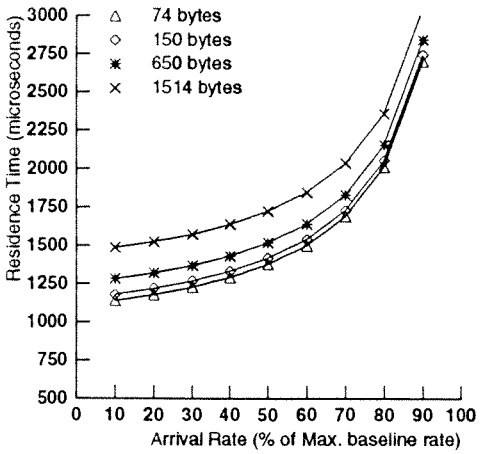

Figure 5: Effect of reducing protocol overhead on the Sequent

delay. Figure 5 shows the effect of protocol overhead on the Sequent. Once again, the reduction in residence time is modest. However, the marginal increase is more than in the case for the Firefly. When the software processing overhead was drastically reduced to 50% of its old value, there was a reduction of about 29% in the residence time at light loads. At heavier loads this effect is less. Unlike the Firefly, software is a bigger overhead relative to the other costs. Thus, reduction of software costs yields better performance than in the case of the Firefly. At higher loads the effect is less because of the insignificance of this overhead in relation to the queuing delay for locked resources.

4.4 I/O Bus Changes

The Q-Bus used on the Firefly can become a bottleneck if large packets are transmitted. The Ethernet controller DMAs the data from system memory to the controller over this bus. This is a source of contention during high load or when transferring large packets because the bus is tied up. This is not a problem with the Sequent because its bus is designed for high bandwidth and allows split transactions. A faster bus has been proposed by an earlier study as a way of reducing latency [9]. We wish to study the effect on performance of a new bus that has twice as much bandwidth as the Q-Bus. We model this by changing the service requirements from Table 1. We reduce the bus requirement by 50 percent and solve the resulting queuing model. The results are shown in Figure 6.

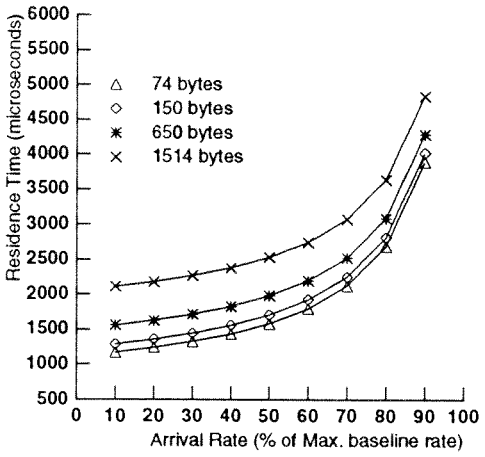

Figure 6: Effect of a faster bus on the Firefly

The smaller packets are largely unaffected by this change; there is no significant effect on the residence times even under high load. The explanation for this lies in the fact that the time spent in service at this bus is not a limiting factor to these packet sizes. For larger packet sizes there is about a 20% decrease in residence time for all loads. Once again, the tolerance to load is largely unaffected; comparison with Figure 2 shows that the performance begins to degrade at roughly the same load as before. These effects can be explained as follows: even with the proposed fast bus, the controller lock continues to be the queuing service center with the largest service demand except for packet sizes of 1514 bytes. Thus, the performance under load is still largely dictated by the controller lock's service requirement.

4.5 A Different Ethernet Controller

In general, commercial Ethernet controllers are designed for use with uniprocessors. The interface presented by the controller to the host is a reflection of this [10, 3]. Typically these controllers have some amount of internal pipelining. This has the good effect that controller-introduced latency is insensitive to the size of the packet. However, the fact that the controllers are designed for uniprocessor environments causes contention in accessing and programming them in a multiprocessor environment. For instance, on both the Firefly and the Sequent a large fraction of the time is spent in activating the Ethernet controller. The time spent in this activity becomes a serious bottleneck because it is done under a single lock. There are various reasons for this on each machine. On the Firefly, there is one processor that is "special"; since all Ethernet controller activity has to be initiated from this processor, some form of locking is required. Further, the Ethernet controller has only a single queue that it scans to find descriptions of packets to be transmitted. On the Sequent the controller and the processors share a single pool of buffers. Processor activity on different CPUs is kept synchronized by one lock. As an alternative, one could partition these buffers equally among the processors so that groups of processors share a single lock instead of all processors sharing the same lock, or even to allocate each processor its own buffer pool.

An alternative suitable to multiprocessor environments would be to have the controller scan multiple queues. The number of queues and the base of each of the queues would be set by host software at initialization. This design has the effect of reducing the contention under load. Low latency is also achieved in this case because the processor does not have to spend the time doing the locking operation. Having a per processor queue to reduce contention is not novel, but it has not been applied in this setting. We study one particular form of this modification by considering the case when the controller implements two queues instead of one. We model this by creating two centers for the controller lock, each with half the aggregate service demand. Figures 7 and 8 show the effect of the modification for the Firefly and the Sequent respectively.

While the latency reduction under light load is modest for the Firefly, this modification causes a significant change in the performance under load. This is not surprising since the Ethernet controller was the bottleneck. By reducing the service demand at this center we have reduced the queuing delays. Under heavy load, the effect is a discernible reduction in residence time. A similar effect is seen on the Sequent. Both these graphs argue for the alternative structure for the controllers, which provides very good performance under load without compromising latency.

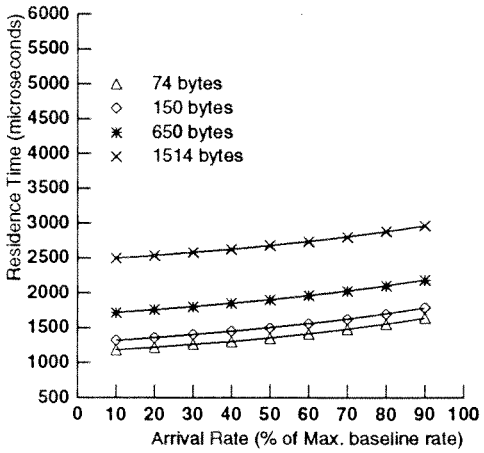

Figure 7: Effect of a different controller on the Firefly

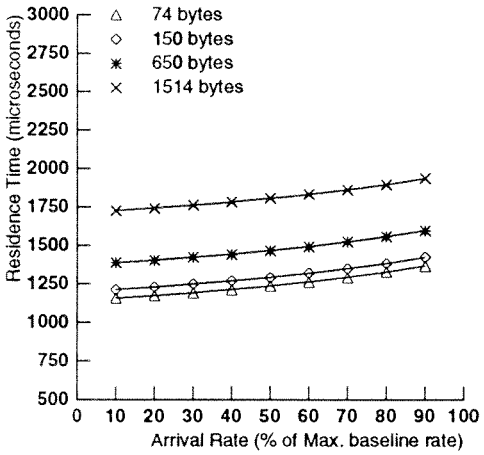

Figure 8: Effect of a different controller on the Sequent

5 Conclusions

We have analyzed the network I/O performance of two small-scale multiprocessors: the DEC SRC Firefly and the Sequent Symmetry. Our analysis shows that:

1. Software processing is a major contributor to latency under light load. However, reducing software processing alone does not buy commensurate performance gain at higher loads. Thus using a simplified transport protocol may not be as advantageous as one might initially think, especially at high loads.

2. I/O bus performance is important to performance under high load and for large packets. For instance, in our particular environment, a faster bus or a split transaction bus may increase performance by

roughly 20% for large packets.

3. By far the most important feature for high performance at heavy load is the structure of the network controller. Traditional controllers designed for uniprocessors may not be appropriate in a multiprocessor environment. Having a single controller data structure shared by the multiple processors gives rise to excessive congestion at high loads. Effective controller interfaces may be designed to achieve very high performance on multiprocessors under heavy loads. For instance, processors could configure the number of queues that the controller services. This arrangement results in less contention and latency in accessing the network.

The design of controllers for multiprocessor network I/O is an area that merits attention for performance oriented environments.

6 Acknowledgements

The authors thank John Zahorjan for his readings of earlier drafts of this paper and his insightful comments which added greatly to its clarity.

References

[1] Luis-Felipe Cabrera, Edward Hunter, Michael J. Karels, and David A. Moser. User-process communication performance in networks of computers. *IEEE Transactions on Software Engineering*, 14(1):38–53, January 1988.

[2] David R. Cheriton and Willy Zwaenepoel. The distributed V kernel and its performance for diskless workstations. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles*, pages 129–140, December 1983.

[3] Digital Equipment Corporation. *DEQNA ETHERNET–User's Guide*, September 1986.

[4] Tarak Goradia and Mary K. Vernon. A model for quantitative analysis of network interface units. In *IEEE Infocom '87, Proceedings of the 6th Annual Conference*, pages 1063–1073, March 1987.

[5] Edward D. Lazowska, John Zahorjan, David R. Cheriton, and Willy Zwaenepoel. File access performance of diskless workstations. *ACM Transactions on Computer Systems*, 4(3):238–268, August 1986.

[6] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Englewood Cliffs, N.J., 1984.

[7] Tom Lovett and Shreekant Thakkar. The Symmetry multiprocessor system. In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 303–310, August 1988.

[8] B. Meister, P. Janson, and L. Svobodova. File transfer in local-area networks: A performance study. In *Proceedings of the 5th International Conference on Distributed Computing Systems*, pages 338–349, May 1985.

[9] Michael D. Schroeder and Michael Burrows. Performance of Firefly RPC. *ACM Transactions on Computer Systems*, 8(1):1–17, February 1990.

[10] M. Stark, A. Kornhauser, and D. Van-Mierop. A high functionality VLSI LAN controller for CSMA/CD network. In *Compcon '83, 26th IEEE Computer Society International Conference*, pages 510–517, Spring 1983.

[11] Charles P. Thacker, Lawrence C. Stewart, and Edwin H. Satterthwaite, Jr. Firefly: A multiprocessor workstation. *IEEE Transactions on Computers*, 37(8):909–920, August 1988.

Appendix

Here we derive the residence time for a multiclass queuing model with a transaction workload where the queuing discipline is FCFS but the per visit service requirement is not the same across all classes.

For a queuing center k, the residence time of a customer of class i is the sum of the total time spent in service and the total time spent in queue waiting for the customers ahead of it to get their service. This latter term is $\sum_{j=1}^{C} S_{j,k}\hat{A}^{i}_{j,k}(\vec{\lambda})$, where $S_{j,k}$ is the service time of a class j customer at center k and $\hat{A}^{i}_{j,k}(\vec{\lambda})$ is the average number of class j customers at center k seen by an arriving class i customer, C being the total number of classes. Thus, if a class i customer makes $V_{i,k}$ visits to center k, its total residence time at the center is:

$$R_{i,k}(\vec{\lambda}) = V_{i,k}\left[S_{i,k} + \sum_{j=1}^{C} S_{j,k}\hat{A}^{i}_{j,k}(\vec{\lambda})\right] \tag{1}$$

Here we make an intuitively appealing assumption (and one often used with success) that, in an open system, the average number of class j customers seen by a class i (i and j may be the same) customer at center k is the time averaged queue length of class j customers at that center. We denote this by $Q_{j,k}$. Applying Little's law we get

$$Q_{j,k} = \lambda_j R_{j,k}$$

Substituting this into equation 1 yields the following expression for residence time:

$$R_{i,k}(\vec{\lambda}) = V_{i,k}\left[S_{i,k} + \sum_{j=1}^{C} S_{j,k}\lambda_j R_{j,k}(\vec{\lambda})\right] \tag{2}$$

Because the summation term is independent of the customer class i and is only a function of center k, we denote the summation by T_k. We can then rewrite the above formula as:

$$R_{i,k}(\vec{\lambda}) = V_{i,k}\left[S_{i,k} + T_k\right] \tag{3}$$

Now consider the residence time $R_{j,k}$ of a class j customer at center k. Using equation 3 we can write this as:

$$R_{j,k}(\vec{\lambda}) = V_{j,k}\left[S_{j,k} + T_k\right] \tag{4}$$

Eliminating T_k between equations 3 and 4 leads to the following relation between $R_{i,k}$ and $R_{j,k}$:

$$R_{j,k}(\vec{\lambda}) = V_{j,k}\left[S_{j,k} - S_{i,k} + \frac{R_{i,k}(\vec{\lambda})}{V_{i,k}}\right]$$

We substitute this expression for $R_{j,k}$ into equation 2 and obtain

$$R_{i,k}(\vec{\lambda}) = V_{i,k}\left[S_{i,k} + \sum_{j=1}^{C} S_{j,k}\lambda_j V_{j,k}\left[S_{j,k} - S_{i,k} + \frac{R_{i,k}(\vec{\lambda})}{V_{i,k}}\right]\right]$$

Simplifying this we get

$$R_{i,k}(\vec{\lambda}) = V_{i,k}\left[S_{i,k} + \sum_{j=1}^{C} S_{j,k}\lambda_j V_{j,k}\left[S_{j,k} - S_{i,k}\right]\right] + R_{i,k}(\vec{\lambda})\sum_{j=1}^{C} S_{j,k}\lambda_j V_{j,k}$$

The term $S_{j,k}\lambda_j V_{j,k}$ is the utilization $U_{j,k}$ of class j at center k. When summed over all classes, the result is the total utilization, U_k, at center k. Making this substitution above and simplifying, we finally derive the expression for the residence time:

$$R_{i,k}(\vec{\lambda}) = \frac{V_{i,k}S_{i,k}}{1-U_k} + \frac{V_{i,k}}{1-U_k}\sum_{j=1}^{C} S_{j,k}\lambda_j V_{j,k}\left[S_{j,k} - S_{i,k}\right] \tag{5}$$

Note that the first term on the right hand side of equation 5 is the usual expression for response time for a multiclass model where the per visit service requirement is assumed to be identical across classes. The second term represents an additional factor which vanishes as the service requirement per visit of the various classes converge to the same value. That is, the second term is 0 if $S_{i,k} = S_{j,k}$ for all i,j.