Scalable Parallel Communications*

K. Maly S. Khanna C. M. Overstreet R. Mukkamala M. Zubair
Y. S. Sekhar E. C. Foudriat
Computer Science Department
Old Dominion University
Norfolk VA 23529

February 28, 1992

Abstract

Coarse-grain parallelism in networking (that is, the use of multiple protocol processors running replicated software sending over several physical channels) can be used to provide gigabit communications for a single application. Since parallel network performance is highly dependent on real issues such as hardware properties (e.g., memory speeds and cache hit rates), operating system overhead (e.g., interrupt handling), and protocol performance (e.g. effect of timeouts) we have performed detailed simulation studies of both a bus-based multiprocessor workstation node (based on the Sun Galaxy MP multiprocessor) and a distributed-memory parallel computer node (based on the Touchstone DELTA) to evaluate the behavior of coarse-grain parallelism. Our results indicate: (1) Coarse-grain parallelism can deliver multiple 100Mbps with currently available hardware platforms and existing networking protocols (such as TCP/IP and parallel FDDI rings); (2) Scale-up is near linear in n, the number of protocol processors and channels (for small n and up to a few hundred Mbps); (3) Since these results are based on existing hardware without specialized devices (except perhaps for some simple modifications of the FDDI boards), this is a low cost solution to providing multiple 100Mbps on current machines. In addition, from both the performance analysis and the properties of these architectures, we conclude: 1) Multiple processors providing identical services and the use of space division multiplexing for the physical channels can provide better reliability than monolithic approaches. It also provides graceful degradation and low-cost load balancing; 2) Coarse-grain parallelism supports running several transport protocols in parallel to provide different types of service. For example, one TCP handles small messages for many users, other TCPs running in parallel provide high bandwidth service to a single application; (3) Coarse grain parallelism will be able to incorporate many future improvements from related work (e.g., reduced data movement, fast TCP, fine-grain parallelism) also with a near linear speed-ups.

1 Introduction

Motivation

Past research directed towards increasing the speed of communication has focused mainly on the physical media used to link computers together and the media access methods for such media[2, 6, 9, 19, 10] but limited bandwidth on the physical transmission media is no longer the performance bottleneck in communication systems. The development of high speed data networks in the range of Gbps imposes new requirements on the processing of communication protocols at the network node. Our objective in this paper is to report on a study of communication protocol processing that can handle throughput in excess of gigabit per second for a single user application.

The need for such a system can be justified by examining several distributed collaborative computer applications which require gigabit per second services made to be available to an end user. *Full motion video*

*This work is sponsored by CIT (596045), DARPA (N00174-C-91-0119), NASA (NAG187263), and SUN (596044) grants.

for use in video conferencing which needs high bandwidth and puts strict limits on average and variance of the network delay introduced as data are delivered to end application. *Computer imaging* - medical, seismic, and weather - demands low latency for data collection and high throughput for image transfers. *Visual techniques* are also becoming increasingly important as a means of understanding the results of advanced computer models. Breaking down a problem among networked computing resources - *computer steering* - is used for visually oriented modeling. Such needs result in large bandwidth requirements. For example, to drive a 1280x1024 24-bit color display updated 15 frames per second requires 472 Mbps data streams.

Background

Made possible by progress in fiber-optic and VLSI technologies, networks offering increased transmission capacity at decreased error rates are now becoming available. In many applications, the performance bottleneck hence has shifted from the network to the processing required to execute communication protocols in workstations and servers[7, 13, 27, 8, 11, 1]. Owing to this, a single application running on many "standard systems" cannot utilize a reasonable fraction of the bandwidth even on communication networks with data rates in the 10Mbps range. This restriction becomes more acute as networks offering larger bandwidths, e.g. FDDI (100Mbps), are becoming widespread. Thus the design decisions used in developing many existing protocols and computer architectures are inappropriate for the evolving high-speed networks. The three main decisions in question are the use of processing power to reduce transmission costs, addition of processing cost to recover from errors, and the use of relatively simpler flow-control mechanisms [11, 3, 4, 8, 17, 29, 30, 12].

Zitterbart [32] classifies the approaches to speeding up protocol processing into those which modify the seven layer protocol stack and those which provide high speed implementations of standard protocol suites. An example of the first category is discussed by Haas in [14]. In this work, the seven layers of the traditional OSI stack are grouped into three new layers; the lower three layers are combined into the network access layer, the transport, session, and presentation layers are combined into a communication interface layer, and the application layer becomes the new third layer. The tasks in the communication interface layer are decomposed into functions that can be executed in parallel to provide improved throughput. An example of the second category is explored by Van Jacobson in [17] with an optimized implementation of TCP/IP. XTP [5] is a variation of this approach that combines the transport and network layers into a VLSI implementation.

Approach

The central idea or theme of our approach is to use scalable parallelism at multiple levels as a basis for a network architecture. In this context parallelism is defined as the representation of a single user's data as a set of concurrent data streams which can be moved in parallel. This is not to be confused with multi-user concurrency where data streams from several users may be moved concurrently. The number of data streams may vary as they are being processed at various levels of the protocol stack. The degree of parallelism at a given level is determined by the number of processors needed to operate on the streams in parallel without introducing a bottleneck in the overall flow. Scalability is defined both in terms of a system scaling with the number of processors and physical channels as well as the ability of different nodes to use different amounts of bandwidth in the network.

We classify parallelism in protocol processing as seen in the literature at two levels[32]: *(i) fine-grain*, and *(ii) coarse-grain*. Fine-grain parallelism[33, 21] is the decomposition of a protocol task at a given level into several tightly-coupled, parallel, smaller subtasks. In the coarse-grain approach[15, 23], separate identical protocol processing tasks run independently on several processors, each processing packets from one or more data streams.This is illustrated in Figure 2. The key difference between the two approaches is that a process is decomposed in the fine-grain approach and replicated in the coarse-grain one. The two

approaches are complementary rather than mutually exclusive since, for example, in Figure 2(b), any of the TCP processes could use fine-grained parallelism.

Note that the fine-grain approach results in parallelism limited by the number of distinct subtasks in protocol processing, the dependency among subtasks, and the dissimilar computation times of individual subtasks. In the coarse-grain approach, different degrees of parallelism can be applied at different layers to provide services sufficient for that level. Because processors are operating on separate packets, they are largely independent and synchronization overhead is small. The approach is scalable with physical processors providing network services when needed and available to other tasks at other times. We note that fine-grain parallelism alone cannot provide any improvement in reliability.

Coarse-grain parallelism also provides the flexibility of different classes of service to communicating applications based on the individual application's needs and priority. This gives the flexibility of running, for example, several copies of TCP on different processors and copies of UDP on other processors. Further, for TCP support, one processor might handle all of the small packets for several light load user applications and others share the process load for a single "high-bandwidth" application. Thus, even in shared memory machines, this approach could avoid frequent context switches and cache misses for those TCP processors handling the high bandwidth application. In addition, previous work indicates performance improvement from placing traffic with similar attributes on separate channels[22].

The niche for which we see parallel networks most appropriate is the interoperable distribution of gigabit NREN traffic[25]. In Figure 1, the dark line represents a set of concurrent channels (for example, wavelength division or space division); each node is configured to attach to a subset (possibly all) of those channels to access whatever bandwidth they need. Inside a node protocol processing is done in parallel using multiple physical processors if available and when beneficial.

Parallel networks are important for many reasons. They complement other work on high data rate networks; by building on existing parallelism in computer and data storage systems, e.g. RAID[26, 24], and providing a natural mechanism for connecting them to a parallel network; by utilizing and improving transport level parallelism[18] in networks; by improving data-recovery codes[31]; by applying media level parallelism in addition to media-level concurrency. Parallel networks provide an upward migration path from existing networks. They allow scalability and fault tolerance capabilities not achievable with serial networks and they can provide a favorable cost/benefit ratio in certain environments.

Objectives

The long term objective of our project (jointly conducted at Old Dominion University and Virginia Polytechnic Institute and State University) is to demonstrate the feasibility of a gigabit node both from a technological and an economical point of view. We have developed a general model, we have done a performance study involving at least two different classes of nodes, namely, workstations and massively parallel computers, and we have designed a 3-node, 4-channel prototype. We next plan to realize the prototype, use the experimental results of the prototype to validate the predictive power of our model, and complete the design of a gigabit node.

As outlined below, this paper is a performance study of a workstation node and parallel computer node having parallel interfaces to parallel networks. This study shows that parallelism increases performance nearly linearly for small values of n (under 10), where n is the number of protocol processors or media channels. Given the low cost of FDDI cards (currently about $2,000) and the availability of multiprocessor workstations, we can bridge the gap to gigabit networks now using moderately priced components. Similarly, as industry is developing "gigabit workstations", i.e. workstations more suited to interact with gigabit networks, we will be able to multiply the effect of emerging protocol technologies at a future time. This study also shows that the inherent advantages of parallelism, fault tolerance and graceful degradation, can indeed be achieved with feasible designs. Node, processor, or link failure in a parallel system simply means that traffic is shifted to a different processor or link, only reducing available "bandwidth" but not terminating a connection. Parallelism is particularly suited to forward error correction to solve the latency problem in long networks. Coding across parallel channels combines the advantages of serial

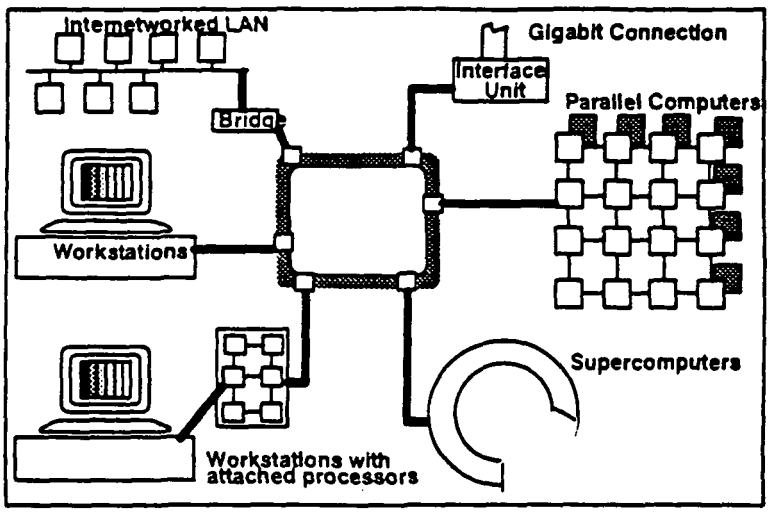


Figure 1: Need for local distribution

forward error corrections and the reliability of a parallel network providing better (latency and reliability) overall performance.

In this paper we study scalability from a performance point of view to see if the approach scales with processors as well as links. We do not address the architectural and routing issues of interconnecting nodes of different throughput capacities to different subsets of available parallel channels; this will be a matter of future papers.

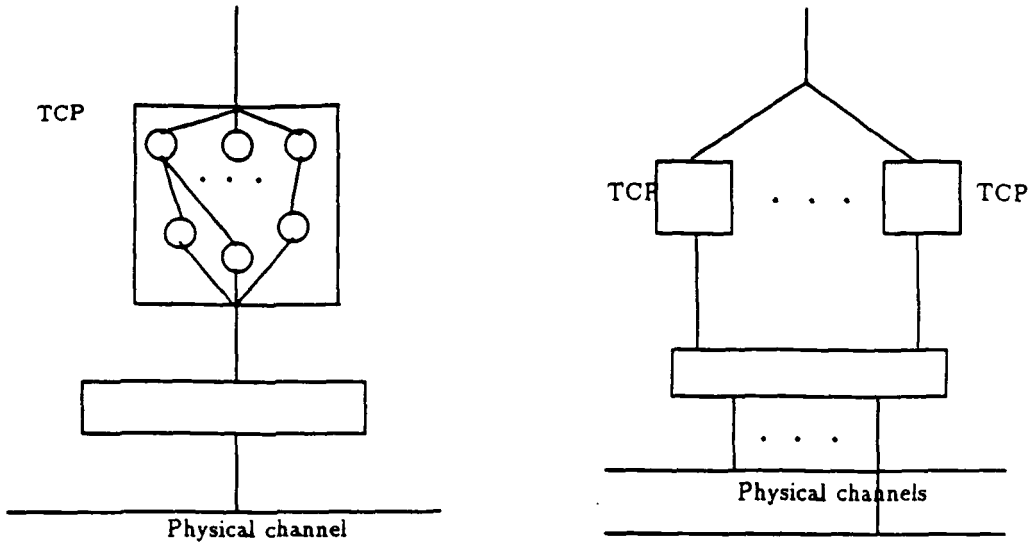
Organization

In Section 2 of this paper we develop a general framework for parallel networks by describing a model and discussing the issues and options posed. Since validation of this approach to high speed low-cost networking must be assessed based on the constraints imposed by real hardware (e.g., memory, bus, and processor speeds), operating system effects and protocol overheads, Sections 3 and 4 instantiate the general model for workstation and parallel computer nodes akin to actual existing machines, and present an analysis of the performance on these architectures. Section 5 summarizes the results of the study.

2 A General Framework For Coarse-Grain Parallel Networks

Communication systems are typically implemented as a hierarchy of protocol layers. Thus processing of a stream of data from a single sender to a single receiver can be done in parallel at different levels both at sender and receiver sites. In addition, if data from a single application are split into multiple streams at some level in the protocol stack, processing can then be performed in parallel on these separate streams. These streams can then proceed separately through additional layers and be rejoined later as appropriate (depending on what works best). If streams are split, this is an ideal application for parallel processing since interactions among the processes working on the separate streams can be very limited, even though they are all working at the same level of the stack. Thus parallelism at a given level is based on replication: separate identical processes (potentially on different physical processors) working on individual data streams.

In addition, this approach is realizable on several existing hardware architectures and provides a general framework for presenting the work that follows. The ideal degree of parallelism both within and among levels depends on physical properties of a particular hardware architecture and which types of network services are most important. The generic model which describes multilevel and replicated parallelism



a) Fine-grain parallelism

b) Coarse-grain parallelism

Figure 2: Different types of parallelism in communications

is intended to provide a framework which allows us to study both these issues which will be generally applicable to the use of parallelism in networking and to study the interaction between a particular hardware structure and the use of parallelism.

2.1 Generic Model Description

Figure 3 is a representation of our model for parallel communication in its most general form. The central idea is to examine the use of parallelism wherever it may be effective; the model presents those places where we expect parallel processing may be useful. We do not expect any effective concrete realization to be this complex but as the model is mapped onto different real architectures, what can effectively be run in parallel will change.

Since we provide for different degrees of parallelism at adjacent levels (and can demonstrate advantages with this approach), we now have the opportunity to schedule work for individual processors as data are moved from one layer to the next.

Figure 3 has 6 levelss, from parallel physical channels at the bottom to the application level (which also may include the presentation and session layers) at the top. Between adjacent layers are one or more schedulers. The model makes no assumption about the assignment of tasks to processes or processes to processors. It includes the idea that several tasks may be performed by a single process and several processes may be assigned to a single processor; effective assignments depends on such items as the degree of interaction among tasks and processes, underlying hardware, the operating system overhead, and processor loads from other tasks. In addition, in a particular hardware architecture, some tasks may be realized in hardware. For example, the scheduling scheme between two levels may be determined by the bus arbitration scheme in the bus hardware.

At the application level, a single user application, operating as a set of processes, A_i, transfers large amounts of data to another user at the remote end. The data units processed and exchanged at application level are called chunks. A chunk may be divided into one or more segments and these segments are exchanged between application A_i and transport T_i layer processes. There may be "n" such transport processes. Since segments must be moved from an application process to many transport processes, scheduler processes S_a are placed logically between application and transport layers to manage this segment

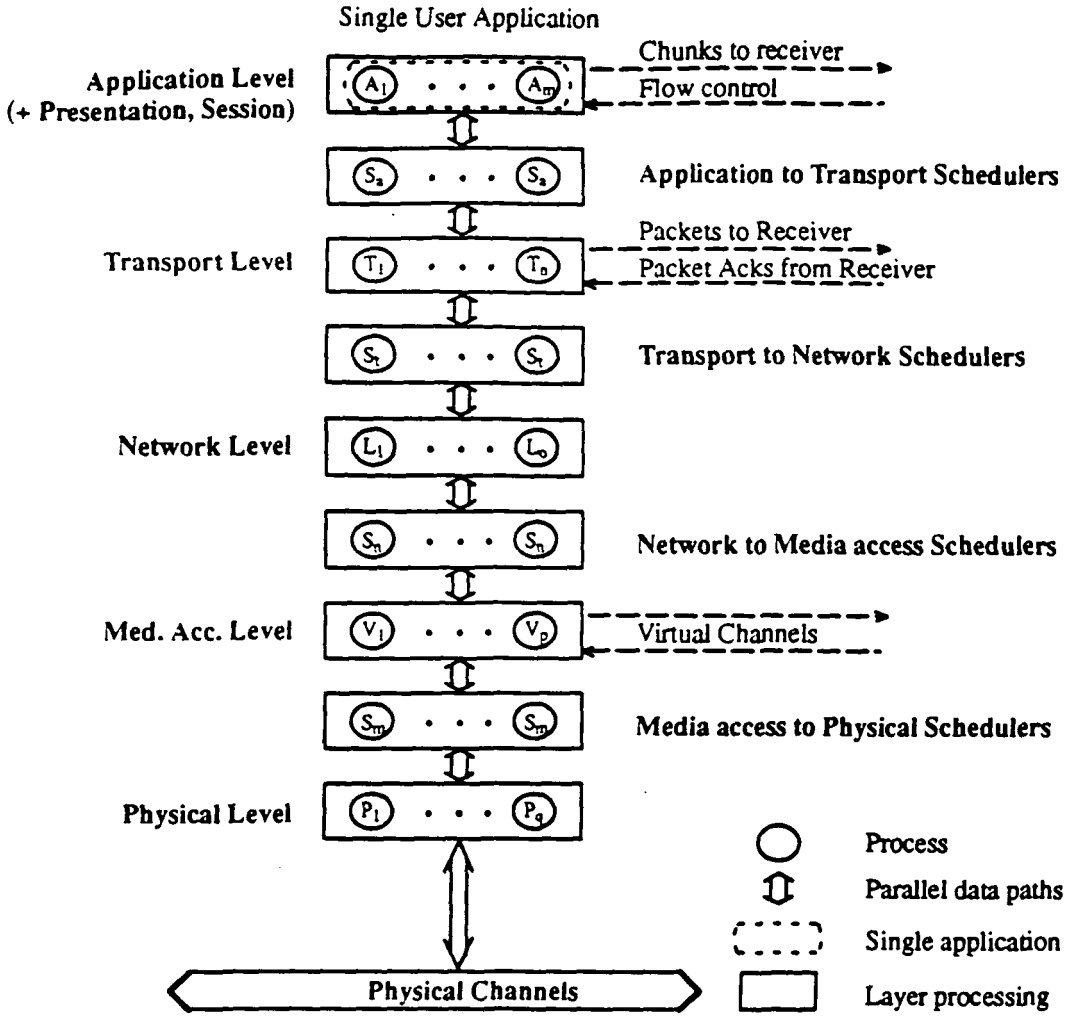


Figure 3: Parallel Communication Logical Framework: A General Concept

allocation. Similarly, scheduler processes S_t are located between transport and network level processes. A scheduler process, S_t, accepts a packet from a transport process and schedules this packet to one of "m" network level processes, L_i. Likewise, a network process, L_i, must select one of the virtual channels. This is achieved through scheduler processes, S_n, located in between the network and media access layers. Lastly, these virtual channels are then mapped onto physical channels by the scheduler processes, S_m. We assume that either many parallel physical channels exist or that one or more high bandwidth physical channels will be multiplexed to provide the appearance of several channels. A S_m scheduler may select the physical channel based on destination and the route selected by network process L_i to reach that destination. A sender and a receiver may have different degrees of parallelism at different layers.

This model assumes the existence of several physical processors. It can be mapped into a variety of realizations. For example, we include the very simplistic approach consisting of single application processor preparing data which are split into two streams. Each data stream is processed independently by a separate TCP/IP processor and sent to a separate FDDI board (for the media access level) and onto physically separate FDDI fibers. These data streams are then received by two separate FDDI boards located in a single node which are connected to two TCP/IP processors, and rejoined to present a single stream to a receiving application process. Thus, once the data streams are separated, they do not interact until rejoined at the receiver. Many variations are possible. For example, the two TCP/IP streams could be

rejoined and transmitted by a single FDDI board.

Because the identified bottlenecks in the protocol processing and absence of standard media access protocols designed for gigabit speeds, we chose to use parallelism at the transport and media access layers. We have one or more fast application processors (APs) connected by some communication fabric (dependent on the particular hardware present in a node) with m protocol processors (PPs) running in parallel. These PPs send and receive data using n physical channels with only one scheduler to load balance the physical channels. Usually the number of PPs and the number of channels differ. Variations on this basic structure are studied in sections 4 and 5.

2.2 Issues and Options

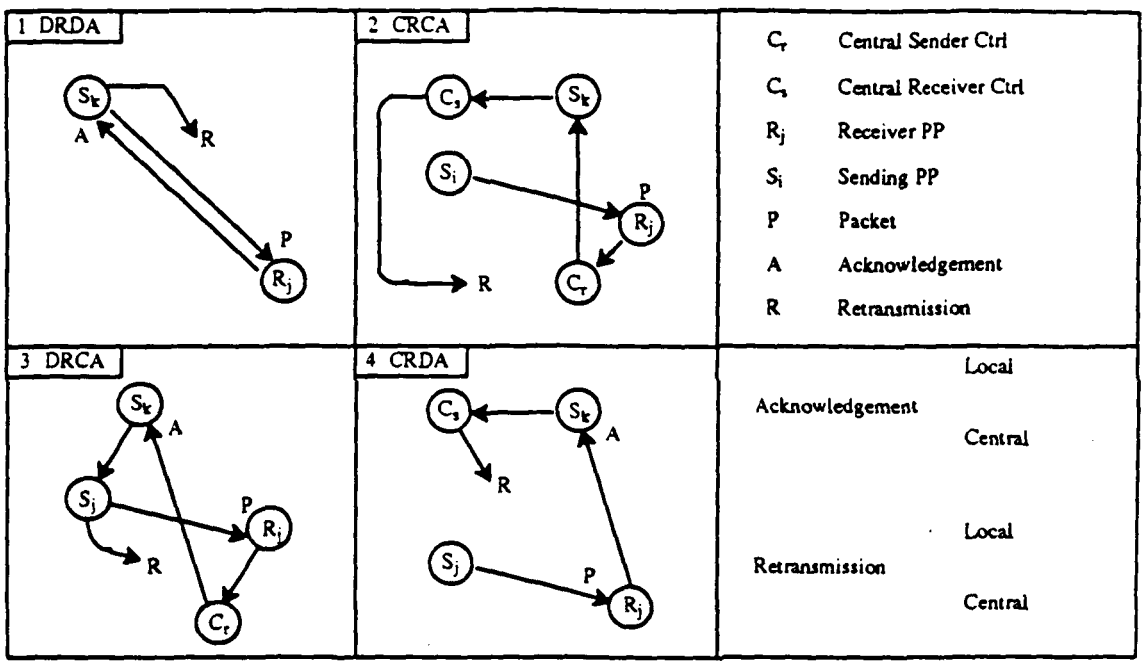


Figure 4: Acknowledgments and retransmissions

In communication systems that use multiple processes at several protocol layers, many issues which effect performance are dependent on properties of hardware present in the system and will be unique to that hardware. However several important issues which would not exist in a serial implementation will be present in almost any parallel solution. This are discussed here.

1. **Scheduling policy.** A major issue is finding an optimal data allocation strategy from an application process to many independent protocol processes and a protocol process to many network interfaces for transmission. The scheduling policy will affect the load balancing potentials among hardware components (processors, buses, memory, and channels, for example) and can have dramatic effects on performance. The data scheduling task can either be adaptive based on information from lower and upper layers or it can be simple such as first-come-first-served or round-robin. An adaptive scheduler may result in better performance utilization at an additional cost of collecting state information: it can make decisions based on information about what is going on below it (queue length at each

processor or expected time of token arrival, for example) or what is going on above it (for example, when the next data will arrive). The cost of doing adaptive scheduling and obsolescence of some types of potentially useful state information are issues which should be used to analyze adaptive scheduling schemes.

2. **Scheduler location.** A scheduling process can run either independently on each protocol (PP) processor (distributed scheduling) per layer or on a single processor (central scheduling) in between two layers. For example the transport to network level scheduler can run locally on each PP or in a central device located conceptually between network layer and MAC layer. This location of a scheduler process will impact the performance and fault-tolerance of the system.

3. **Window management, time-outs and acknowledgments.** One can assume that timers and acknowledgments are processed by the PPs. In this solution, all PPs operate independently and still contribute to the task of transmitting a block of data. This is obviously not the only way to perform these functions and we have identified four basic ways in which they can be done. Each of these are illustrated in Figure 4 and are discussed more below.

4. **Error detection and correction strategies.** Error detection and correction strategies across multiple parallel physical channels can be employed to recover from lost/corrupted data and channel loss efficiently. These strategies can save retransmissions at a cost of additional code bits per packet. If a forward error correction mechanism is employed, all bits transmitted in parallel must be present at the receiver at about the same time so that errors can be corrected quickly. On the other hand not all packets need to be present to compute all correct data packets.

5. **Retransmission timer value.** Retransmission timer value in existing nonparallel transport protocol implementations is computed based on the smoothed round trip time of packets over a single channel. This computation may not be valid when a single transport connection spans over parallel channels which behave independently. Hence, retransmission timer management is an issue in parallel implementations of communication protocols.

6. **Shared memory.** In multiprocessor architectures, distributed shared memory for processors versus local memory per processor is a major performance issue in achieving the higher throughput. Although shared memory makes processor coordination simpler, but it comes at the cost of lower performance. Local memory reduces the load on global memory access, but may require additional data copy operations from one processor's local memory to another processor's local memory.

7. **Memory access, data copying and operating system interrupts.** These issues are equally important as in serial implementations. Their impact though is different because some of the operations can be overlapped and pipelined.

Acknowledgements and Retransmissions Management

Figure 4 illustrates the four choices in handling acknowledgments and retransmissions. The first solution, called DRDA for Distributed Retransmissions Distributed Acknowledgments, assigns the task of timer maintenance to the PP that generated a packet. The receiving PP generates an acknowledgment and sends it to the transmitting processor directly. This is a distributed solution and essentially uses separate transport connections between the sending and receiving PPs.

The second solution, CRCA - Centralized Retransmissions, Centralized Acknowledgments - uses separate PPs on both sides to maintain timers and to generate acknowledgments. In this solution, the PP labeled C_s assigns a packet for transmission to PP labeled S_i and S_i sends it to any receiving PP labeled R_j. On receipt of a packet, R_j notifies the PP labeled C_r which determines if an acknowledgment is required. If one is needed, it is built and sent to any PP on the transmitting side. Received acknowledgments are

forwarded to C_s and the corresponding timer is stopped. This solution is similar to that proposed by Jain et al. in [18].

The third configuration, DRCA - Distributed Retransmissions, Centralized Acknowledgments - uses a central PP on the receiving end, C_r, but each of the sending PPs maintains timers for its packets. When a packet is received, C_r is notified as in CRCA, and it sends an acknowledgment to any PP on the sending side. As before acknowledgments are forwarded to the sending PP and the timer is stopped.

The final solution, CRDA - Centralized Retransmissions, Distributed Acknowledgments, uses a central PP for timer maintenance and packet retransmission as with CRCA, but each of the receiving PPs generates acknowledgments locally and sends them to any PP on the sending side.

Providing a completely decentralized solution (DRDA), like the first one shown in Figure 4, reduces interprocessor communication requirements on the sending side and we can expect throughput to increase. The cost associated with this solution, however, becomes evident when resequencing is performed at the receiving end. The host or receiving application must have enough storage space for many segments of data, rather than packets, because while packets within a segment are sequenced by the sending and receiving processors, the segments must be resequenced by the receiving application host.

A central processor performing acknowledgment generation makes packet resequencing easier and may reduce system latency because only complete packets must be received, rather than segments. It also implies that a single window is maintained for data coming from all processors, so more memory is required. Additionally, increased interaction among processors is required and may reduce overall system throughput.

3 Bus-based Multiprocessor Workstation Architectures

Previous work in parallel computing shows that inappropriate decomposition of a problem results in poor performance rather than naively predicted speed-ups[20]. We feel that the potential for significant improvements both in speed and reliability clearly exists with this parallel approach, but must be demonstrated. Proper evaluation must take real operating system, protocol, and hardware effects into account. In the next two sections, we describe our analysis of parallel network performance on both a Sun Galaxy Multiprocessor and a Touchstone machine. These machines have very different architectures. Before beginning detailed analysis, we felt that the likely performance bottlenecks would be related to processing speeds, memory contention, and bus contention. These machines are of interest because they present different communication fabrics: Sun is bus-based with a single global memory, Touchstone has a mesh interconnection fabric with local memory for each processor. Concerns about memory access and bus contention also led us to consider a variation of the Sun architecture with two buses and local memory for each processor. Results from analysis of these architectures are presented in this and the next section. The work reported here presents positive support of the benefits of coarse-grain parallelism.

Our conceptual model for parallel protocol processing represents processing functions as cooperating processes. To instantiate this model on a given parallel hardware architecture, we need to describe the facilities for interprocess communication, memory architecture and the location, number and types of schedulers and map the application and the protocol processes onto different processors. As a first instantiation of our model we have selected a multiprocessor workstation. We believe for gigabit networks to succeed they cannot rely on a few supercomputers to operate in a distributed fashion but rather gigabit communication must be made available to the large number of workstation users who constitute the vast majority in our computing community. Although current workstations are not designed to handle gigabit traffic, many now have multiprocessors. We will show that these workstations can handle multiple 100 Mbps traffic when conjoined with a parallel network. Within the current technology framework we are restricted to bus-based architectures when instantiating the model for workstations.

Here, we describe two bus-based instantiations to realize the proposed conceptual model for parallel processing. The instantiations have the following common features.

- Each application process is located in an independent applications processor (AP).

- Each protocol process is located in an independent protocol processor (PP).
- Each protocol process does TCP and IP processing. We assume that the application requires reliable stream oriented transmission of data and postpone for later studies the impact of using such protocols such as UDP for high bandwidth applications.
- To allocate the segments generated by the application processor to the protocol processors, we employ a scheduler (called the application scheduler). The scheduler is located in the application processor itself.
- The scheduler to allocate packets from the protocol process to the network interface (called the network scheduler) is located either in each protocol processor or in a special mux/demux device.
- The bus (or buses) acts as the basic communication fabric for interprocess communication.
- If shared memory is present, it can also be used as a medium for interprocess communication.

With this concept of parallel processing in mind, the challenge is to design MIMD (multiple instructions multiple data) like architectures in which processors independently execute TCP/IP on multiple data streams originating from a common application source stream and push processed packets on multiple channels to provide higher end-to-end throughput. Since an application generates data and data are distributed to protocol processors, the data should be available to all processors at a common place which can be a global memory, to prevent multiple copying of the data. This flow across AP and PPs must be controlled. One possible and readily available solution being providing a bus between these two components to serialize the access to the bus. Keeping this as motivation the following feasible architectures are studied for such applications.

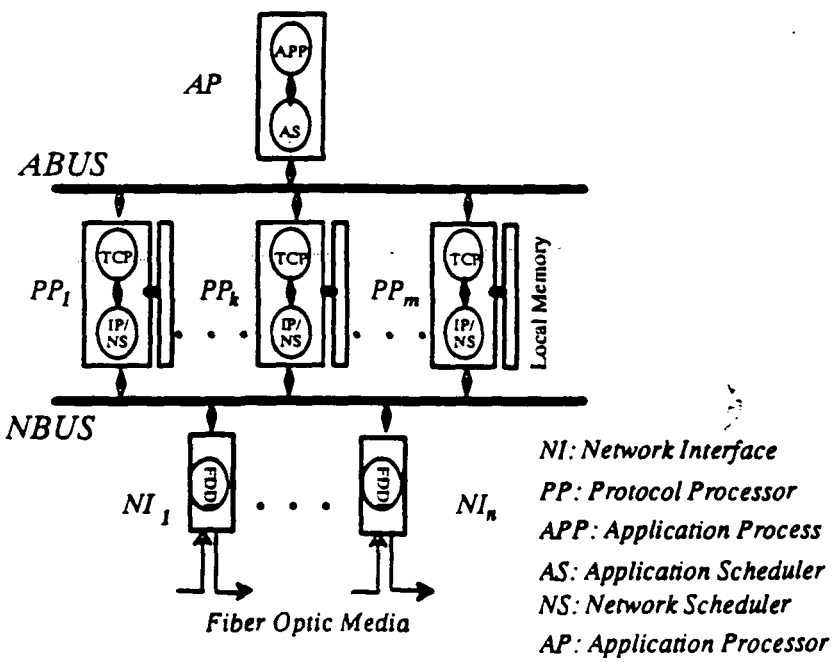


Figure 5: 2-Bus Distributed Scheduler architecture(2-Bus/DS)

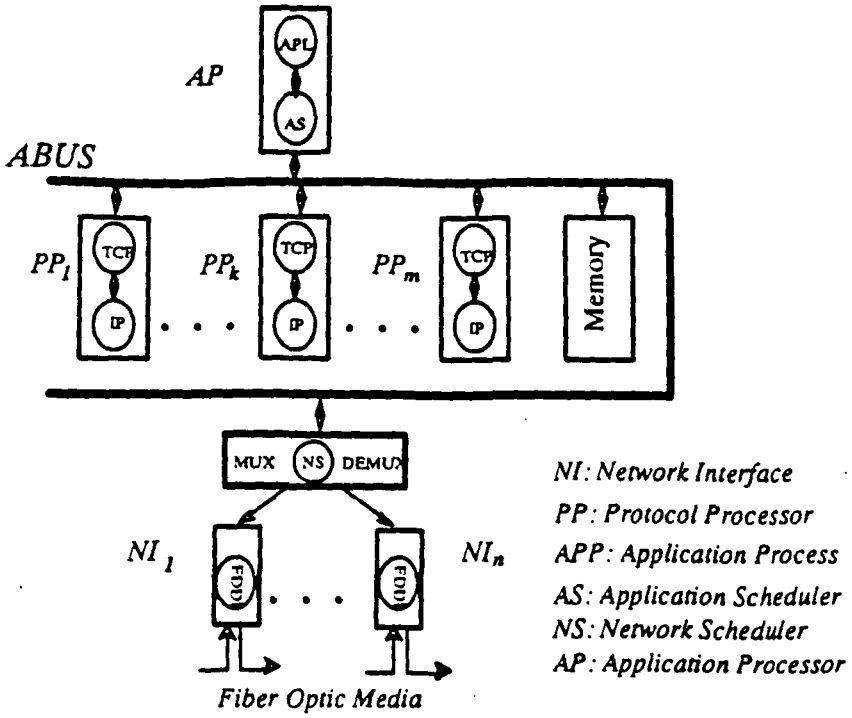


Figure 6: 1-Bus centralized scheduler architecture(1-Bus/CS)

3.1 A Two-Bus Instantiation

Figure 5 shows a two-bus instantiation of the general parallel architecture. The main objective in selecting this architecture is to determine the impact of distributed scheduling and local memory on the overall performance of the system. Here a single high performance processor is running the application and is capable of generating (at sender's end) or consuming (at receiver's end) data at the rate of multiple 100 Mbps. The two buses are referred to as ABUS (or the application bus) and the NBUS (or the network bus). Since a single processor cannot efficiently utilize the available channel capacity, multiple independent protocol processors (PPs) are connected between an ABUS and an NBUS where each PP executes an independent copy of TCP/IP. Each processor is assumed to have sufficient local memory to do processing as well as data storage functions. The application generated data, in the form of segments, are copied from the AP's local memory to the local memory of a PP over the ABUS. The PP is chosen based on a scheduling policy implemented by application scheduler process running on application processor. When a PP gets a segment, it does TCP/IP processing for it and schedules packets for the network interfaces according to a policy implemented by network scheduler. This architecture is a distributed scheduler architecture because network scheduler task is distributed among the PPs.

3.2 A Single-Bus Instantiation

Figure 6 presents an instantiation based on Sun Microsystem's Galaxy Multiprocessor system. The main objective of selecting this architecture is to determine the effect of global memory and central scheduling on the performance of the overall system. Since distributed scheduling is assumed to be more expensive in terms of propagation of information and the use of probably obsolete load information, the centralized scheduler is expected to result in improved performance. The increased performance may be achieved at the expense of reduced fault-tolerance. This architecture, referred to as 1-bus, has a very high speed bus which can support up to sixteen Sparc processors and I/O devices. (The current release, however, supports four processors, three I/O devices, and the memory, and is adequate for our experimentation.) Each processor gets its instructions and data from the shared memory. Since many independent processors get their data

and text from one shared memory, the memory contention problem can be solved temporarily with faster memories and using a "Split Protocol" on the bus[16]. This protocol guarantees that any module requesting memory does not block the bus during the time of service of that request. The network scheduler is located in the Mux/Demux device. The custom-made device Mux/Dmux is definitely a performance bottleneck because it serializes the parallel effort achieved in the transport layer. However, it enables us to analyze the tradeoffs in the use of forward error-correction techniques and a centralized scheduler. With cross channel coding, a form of forward error-correction technique, it is possible to recreate the data stream at the receiver even when l out of m channels are faulty reducing the need for retransmissions[31]. For efficient coding of data on all channels the data stream has to be serialized at some point before it is put on the physical channels.

Various issues which may effect the performance of these multiprocessor architectures when parallel protocol processing concept is implemented on them are detailed in the next section and options for efficiency and higher performance are also proposed.

3.3 Model and Simulation

Here we describe the simulation models developed to evaluate the proposed 2-bus and 1-bus architectures. The major common components in all the architectures are: application process, application scheduler, protocol process, network scheduler and FDDI interface. But the specifications of these common components may vary between the architectures. The discrete event simulator was written in C with every process and hardware component modeled as an object. The model was validated in several ways. Early on, detailed traces were generated to confirm that the proper events where occurring as they should. In addition, the model was run with parameter set to values which should produce results with well know performance features to confirm that the program would replicate those featues. We also determined the run-lengths sufficient to ensure that output statistics are stable.

- **Application process.** An application process runs on the application processor (AP) generating segments of data at uniform rates. It requires an end-to-end transfer of its data in excess of 300 Mbps (arbitrarily assumed in our experiments). The application data are available as a sequence of segments and such segments are allocated to the protocol processors (PPs) by the application scheduler process.

- **Scheduling policies.** The application scheduler process implements a first-come-first-served scheduling policy (FCFS) in allocating segments generated by the application process to the protocol processes. Here, as soon as a protocol process is found to be capable of handling an additional segment (e.g. due to availability of free buffers), that PP is allocated the next segment in line. In our model, we assumed PPs never had to wait for segments from APs since we were concerned with PP processing speeds and not with processing speed at the application level.

  The network scheduler process implements two types of scheduling policies in allocating packets from the protocol process to the network interface. The round-robin or RR policy is the simplest one, where the network Scheduler allocates packets to NIs in a round-robin fashion. The second policy, the adaptive scheduling policy, allocates packets to the NIs based on the minimum value of a function which depends on the queue size at a NI and the last cycle token rotation time of the NI. Clearly, a NI with smaller queue size is preferred to the one with a larger queue size; a NI with smaller last token rotation is preferred to one with a larger token rotation time.

- **TCP/IP connection.** Each sender PP works on an independent TCP connection with a receiver PP. The data stream is handled by a PP according to the RFC 793 [28]. The processor speed is assumed to vary from 1 MIPS (typical of a Sun3/50) to 25 MIPS (typical of a Sparc). Instructions for TCP/IP are assumed to be in cache with a 100% hit rate, but data to be sent has a 0% hit rate.

- **Network interface.** The network interface (NI) is modeled as an FDDI interface connected to a fiber-optic ring with 20 nodes evenly distributed along a length of 20 kms. Among these nodes, two are designated as the sender and the receiver for our experiment. Since we assume the high-bandwidth requiring nodes to coexist with other low-bandwidth nodes in the network, we simulated the latter traffic by introducing the background traffic. Hence, the background traffic is an integral part of the total network traffic and is aggregated as a single entity in our simulations. The background traffic on one ring is independent from the traffic on other rings. We consider both uniform and nonuniform cases of background traffic. Also the background traffic on a ring is not delivered to eiher the parallel sender and parallel receiver; they only process data belonging to the application. If we assume, for example, 30% of total capacity of six FDDI channels as background traffic, then the available channel capacity will be approximately 6*(100-30-token rotation loss) or approximately 300-390 Mbps.

- **Memory.** When local memory is present in a processor board, data are exchanged between processors via the bus (i.e. a memory to memory copy is performed). In case of shared memory, each processor's transmit/receive buffers are managed as separate entities in the memory and hence no explicit data transfer is necessary. To resolve bus contention in the case of local memory and memory contention in the case of shared memory, we use the FCFS policy.

- **Acknowledgments and retransmissions.** Acknowledgments and retransmissions are handled in a completely distributed manner (DRDA). Experiments with other policies are currently underway. Errors on channels are simulated as negative acknowledgments and sender PP starts retransmission of all the packets starting from the error packet in its window till the last packet sent.

- **Mux/Demux.** The multiplexor/demultiplexor device in the 1-bus architecture schedules packets from PPs to NIs in both round robin and first-come-first-served fashion, based on the option selected. It may also encode data packets and pad them with some extra bits to enable complete reconstruction of data at receiver when employing the cross channel coding.

Since our approach to protocol processing is radically different from the traditional serial approach, we have considered the serial architecture for baseline comparisons. For this purpose, a serial architecture is represented as consisting of one application processor, a single protocol processor, communicating on a single bus. We have not made any attempts to optimize the parameters of this architecture to function efficiently at high speeds. Instead, when the serial system is to be compared to a parallel system with four protocol processors, the speed of the serial processor is increased four times. Similarly, other serial system parameters are also set on the same basis. In comparing the performance of the proposed parallel architectures with the baseline serial architectures, our intention is to determine whether parallelism has an advantage over traditional architecture in reliability and cost while not loosing in performance. It may also be true that we may be trading one for the other. We wish to determine the facts with our experiments.

Table 1 illustrates various hardware and software parameters important for simulation and their corresponding values adopted in our experiments. The parameters in Table 1 are classified into three categories: hardware-related, software-related, and control parameters. While the hardware and software related parameter values chosen here represent a sample of reasonable values in the current technology, the values in the control section represent optimal values obtained through experimentation.

3.4 Performance Results

The main objective of our experiments is to determine the feasibility of the proposed architectures to achieve the desired scalability in performance. Especially, we are interested in determining the reliability-roundtrip delay tradeoffs within the architectures. In addition, we are also interested in determining the efficacy of each of the options proposed to solve issues in these architectures. This section attempts to summarize our results and conclusions in this context, without strongly supporting one architecture over the other to build a system. The performance of an architecture is measured in terms of throughput and

	2-Bus/DS	*1-Bus/CS*	*Serial*
Selected Hardware Parameters	Local Memory	Shared Memory	Shared Memory
No of APs	1	1	1
No. of PPs	4	4	1
No. of NIs	6	6	1
MIPS	25	25	200
ABUS speed (Mbps)	800	2400	2400
NBUS speed (Mbps)	800	-	-
Error rate	10e-9	10e-9	10e-9
FDDI speed (Mbps)	100	100	600
Memory Access (nsecs/word)	80	20	5
Selected Software Parameters			
Application Scheduler policy	FCFS	FCFS	-
Network Scheduler policy	RR/Adaptive	RR/Adaptive	-
Selected Control Parameters			
Segment size (Kbits)	36	36	144
Window size (kbits)	216	72	864
Packet size (kbits)	36	36	36

Table 1: Table of simulated hardware and software parameters

round trip delay per segment (i.e., the time a segment has been generated by the application to the time it has been fully acknowledged at the sender application scheduler).

Experiments were done with the simulator to find the optimal values of (i) segment size, (ii) TCP window size, (iii) processor speed, (iv) bus speed, and (v) memory access time for all architectures operating under variable environment parameters such as: (a) background traffic on channels, (b) background traffic on the bus, (c) scheduler policy, and (d) error rate. Our aim is to provide end-to-end throughput around 300 Mbps at minimal round trip delay per segment. We identified 300 Mbps as a throughput level because it is a significant improvement over what is currently available and is also sufficient to raise major issues in use of parallelism.

To make maximum use of a FDDI frame, the TCP packet size was fixed at 4500 bytes. Through successive experimentation, it was observed that at least four PPs and six FDDIs (NIs) are needed to achieve around 300 Mbps end-to-end throughput in all the two architectures. Subsequent subsections present a brief analysis and interpretation of the experiment results.

- **Bus Speeds, Processor speeds and Memory Access Time.** Through experimentation with processor speeds and bus speeds, we conclude the following.

  - For the 2-bus architecture, the processor speed can be anywhere between 1 MIPS (86% utilization) to 25 MIPS (4% utilization) and bus speeds should be 700 Mbps or greater. To avoid overloading of the components, 2 MIPS processor (with 45% utilization) and 800 Mbps bus speed were selected for subsequent experiments.
  - For the 1-bus architecture, bus speed (2400 Mbps) and processor MIPS (25 MIPS) were chosen to resemble the SUN Galaxy MP system. Experiments, where memory access time was varied from 20ηsecs/word to 80ηsecs/word, showed that throughput and round trip delay would deteriorate by 90% for the slower memory. Table 2 shows this behavior. For subsequent experiments, we chose 25 MIPS protocol processor speeds for a fair comparison of the architectures but the bus speed for 2-bus architecture was kept at 800 Mbps.

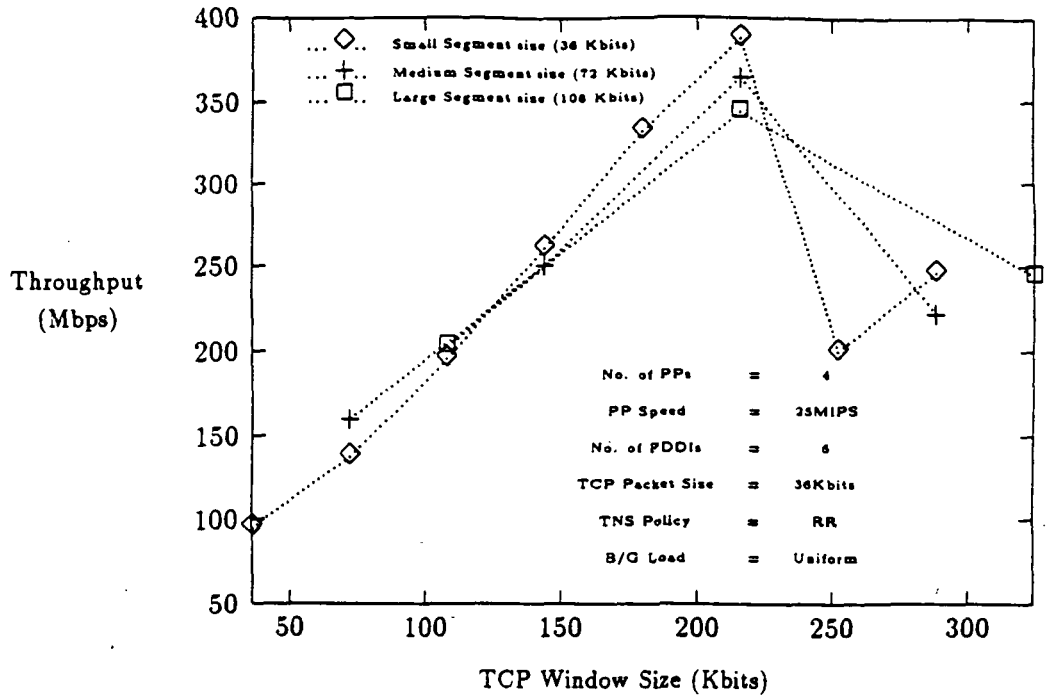


Figure 7: 2-bus architecture: End-to-end throughput vs window size

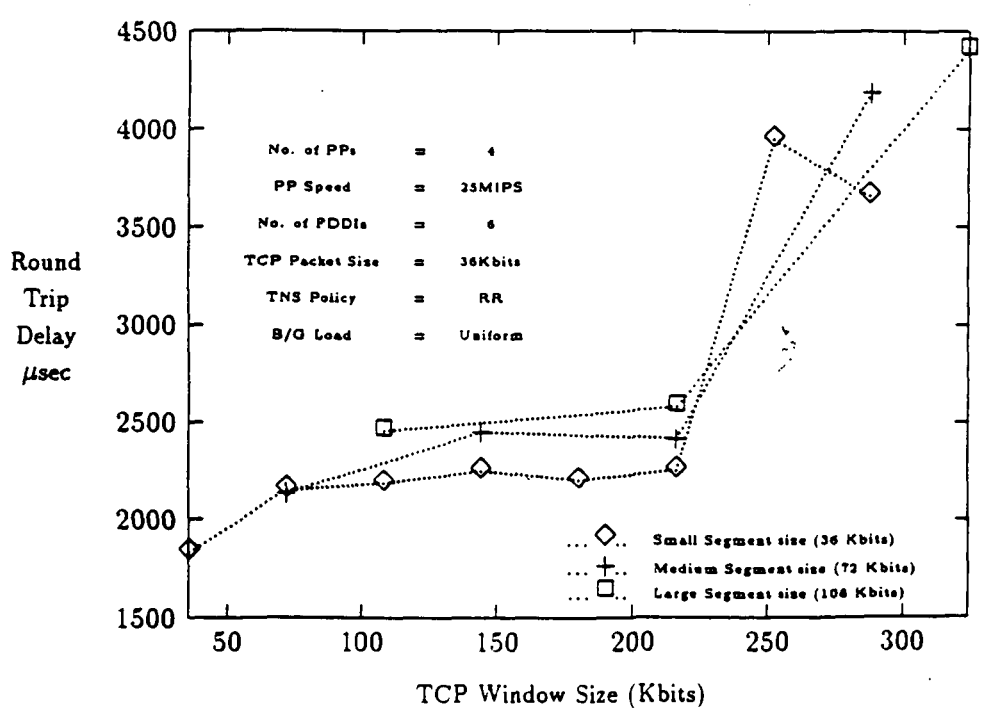


Figure 8: 2-bus architecture: Round trip delay per segment vs window size

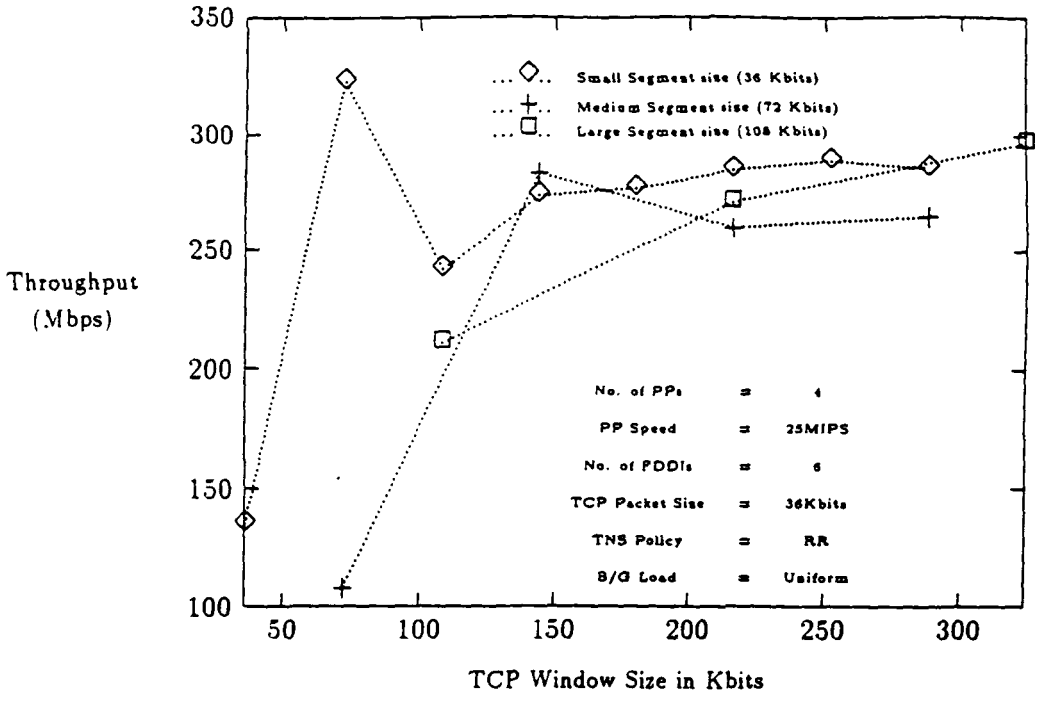


Figure 9: 1-Bus: End-to-end throughput vs window size

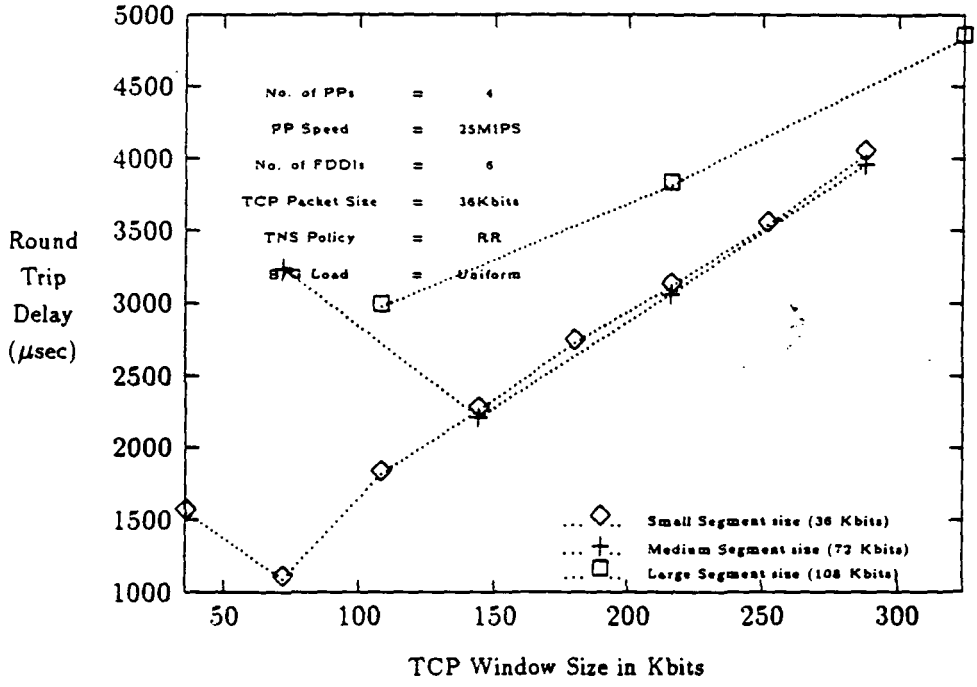


Figure 10: 1-Bus architecture: Round trip delay per segment vs window size

Access time per word ηsecs	Delay per read/write μsecs	System Throughput Mbps	Round Trip time per segment μsecs
20	7.1	323	1092
40	17.46	284	1235
80	54.87	170	2085

Table 2: Comparative performance of 1-bus architecture under varying memory speed

- **TCP Window Size.** To determine the optimal TCP window size which delivers maximum throughput with minimal round-trip delays, three experiments were conducted corresponding to small (36 Kbits), medium (72 Kbits) and large (108 Kbits) segment sizes. Figures 7 through 10 illustrate the impact of window size for the 36 Kbit, 72 Kbits and 108 Kbits segment sizes. From the plots, the optimal window size for the 2-bus architectures is found to be 216 Kbits and for the 1-bus architecture it is 72 Kbits for segment size of 36 Kbits. These results indicate that the window size selection is such that -
  - No protocol processor starves for data to be sent while waiting for the acknowledgments (This helps in achieving higher throughput by proper scheduling.);
  - No packet waits unnecessarily in sender's window. (This helps in achieving lower round-trip delays.)
- **Segment Size.** To avoid introducing additional latency, the segment size for any architecture should be bigger than the TCP packet size and smaller than the TCP window size. Experiments suggest a minimal segment size of 36 Kbits for both the architectures which is the lower bound on the segment size. A larger segment size for any architecture provides similar throughput but at around 7% to 23% higher round trip delay. A larger segment size results in higher round trip delay because the segment has to contend multiple times for bus and memory (for every TCP packet a segment makes). Also every TCP packet of the segment suffers protocol processing delay.
- **Round Trip Times per TCP packet.** Figures 11 and 12 illustrate the round-trip timing distribution for TCP packets sent by a PP under nonuniform channel loads for both the architectures under study. Figure 13 shows the round trip time distribution for TCP packets for a conventional (single processor) architecture. While the average round-trip delay in Figures 11 and 12 should not be directly compared due to architectural differences, the large variations in TCP packets' round trip time in the parallel architectures considered here as compared to the conventional case is apparent. This variation may cause serious concerns in managing TCP timeout timers in multiple channel implementations. If a timeout timer is managed as specified in current TCP recommendations, the TCP connections will timeout for many packets for this large variation of round trip delay. Hence many retransmissions will occur. (To get round trip time distributions, these simulation experiments were conducted with a very high timeout timer values.) To confirm this belief, the distributions were processed according to suggested smoothed round trip time ($srtt = (1 - \alpha) * rtt + \alpha * srtt$) and retransmission timeout timer ($rto = \beta * srtt$) calculations according to the equations given below from [28]. The values of α and β chosen were 0.2 and 2.0 respectively. The number of computed timeouts which could have occurred in each case are depicted in each figure. These variations in round trip times are observed to occur only under nonuniform channel load conditions.
- **Comparison of Architectures.** Table 3 presents a comparative estimate of the performance of two architectures for various policies implemented in network scheduler and various background loads on

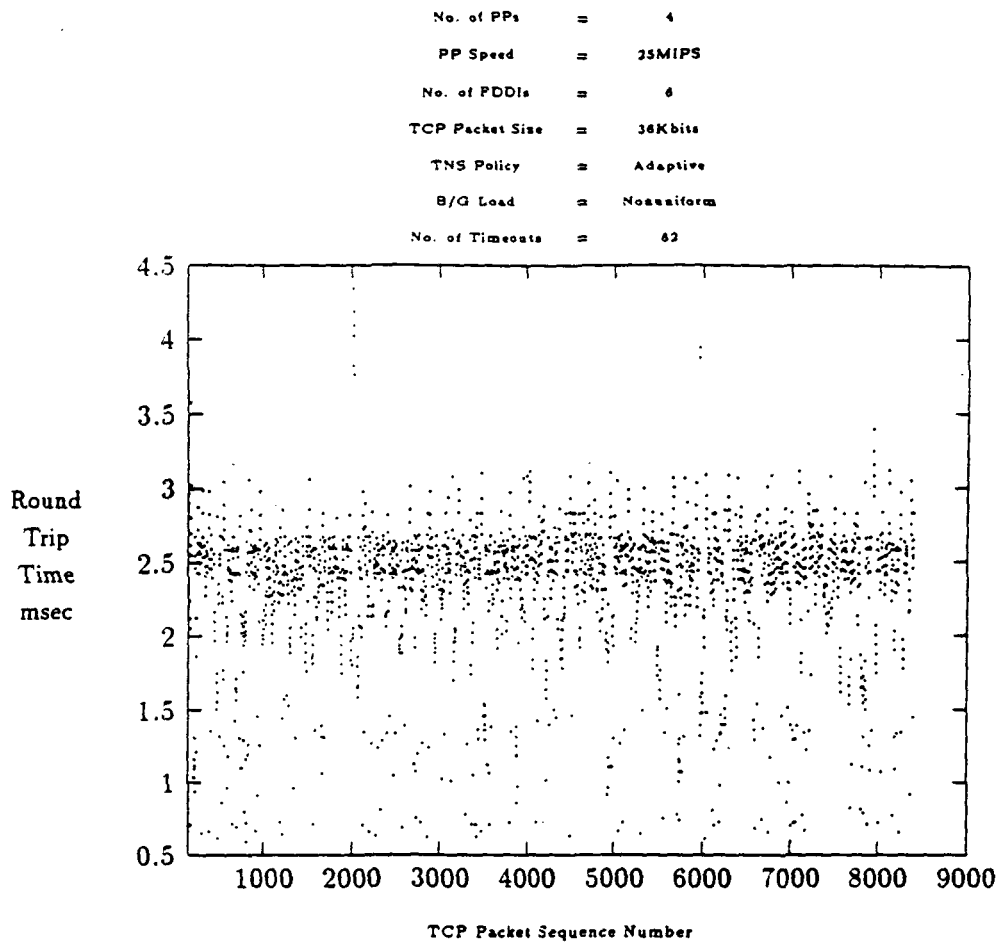


Figure 11: 2-Bus architecture: RTT Distribution for TCP Packets

FDDI channels. In nonuniform case, background load on each channel was varied in steps of $\pm 10\%$ (lower and upper bounds being 10% and 50% respectively) with an initial and final loads of 30%. Hence, nonuniformity starts and ends with a uniform channel loads on all channels. Half of the channels take positive steps and remaining half take negative steps during six such variations evenly distributed over a simulation run. Under all channel conditions (refer Table 3), the 2-Bus architecture performs better than 1-Bus architecture in terms of smaller processor MIPS and slower bus speeds for similar throughput values.

- **Conventional vs. Bus-based Multiprocessor Architecture Performance.** Tables 1 and 3 present a comparison of performance of the two proposed instantiations with the serial architecture. (Recall from Section 3 that a serial architecture is a single processor, single bus system.) As observed from the experiments, the window size needed for TCP is 864 Kbits which is three times TCP window size (4 x 72 Kbits) in the 1-bus architecture. The protocol processor MIPS and memory speed are eight and four times morer respectively. The main reason the performance of the serial architecture is actually less than the "equivalent" parallel system is twofold: one, "equivalent" is not really accurate because we only adjusted the processor's speed, and two, more importantly, in our simulation the serial implementation did not pipeline its memory access and computation cycles. Performance of the serial architecture is still in the same range as the parallel architecture, but requires a hypothetical network interface of the order of 600 Mbps is needed which would be more expensive using current technology. In terms of reliability and fault-tolerance, the serial architecture has no fault-tolerance

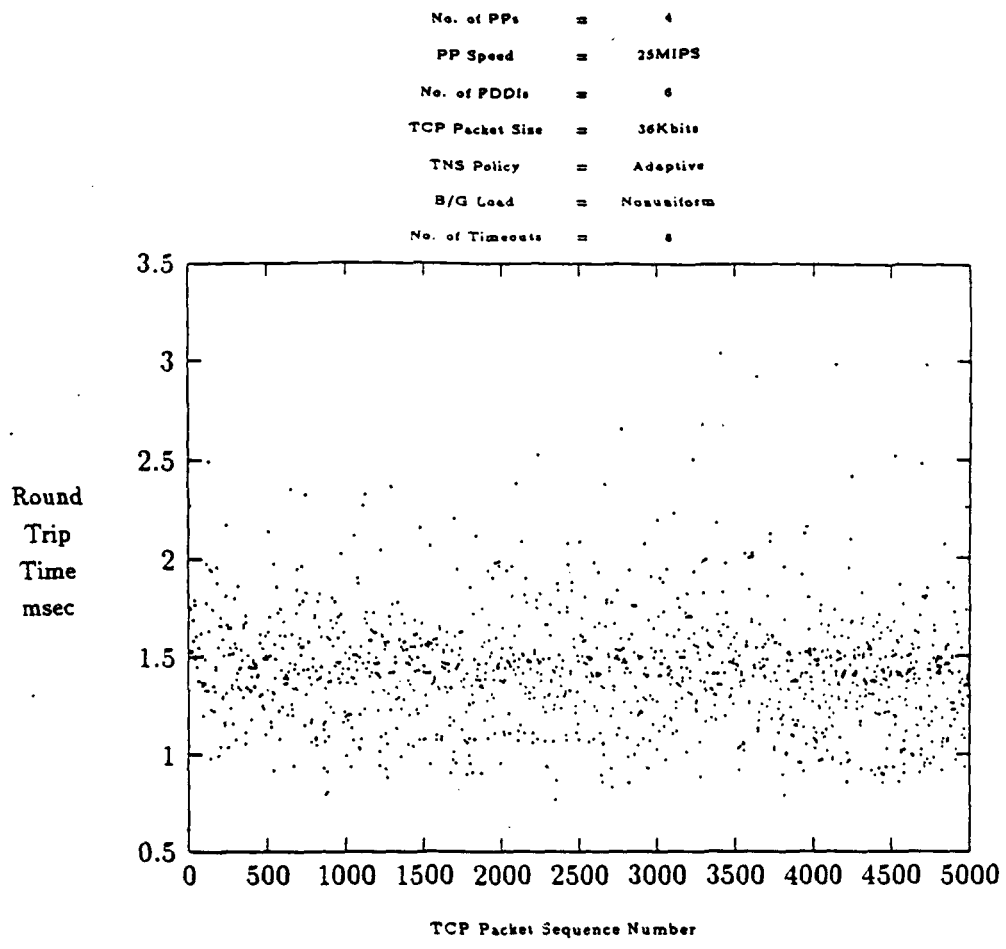


Figure 12: 1-Bus architecture: RTT Distribution for TCP Packets

while the 1-bus architecture can tolerate 3 protocol processor failures, and 5 channel failures. We can conclude that "comparable performance" is juxtaposed to a serial architecture with more cost and less reliability.

3.5 Memory Access and Operating System Interrupts

As mentioned in the introduction, a main objective of our study is to determine the effectiveness of coarse grain parallelism processing in building high-speed networks using the existing technology. It is widely believed that processing speed is the singlemost bottleneck in achieving higher throughput between single application process pairs. While this statement appears to be valid in a nonparallel system, its validity in the parallel case is not obvious. For example, if we consider the TCP/IP protocol processing, some components of the processing need to be done in a strictly serial fashion. Updating windows, for example, is a function that is to be executed serially for a given pair of communicating application processes. Other functions such as computing checksum may be executed in parallel for different packets within the same window. For this reason, we have analyzed the fast-path TCP/IP and categorized its processing requirements in terms of serial and parallel. The time estimates based on the 1-bus shared memory instantiation are summarized in Table 4. For brevity, only the results of our analysis for the *send* call are presented. As shown in Table 4, when the packet size is 500 bytes, for a memory speed of 40 ns, processor speed of 15 MIPS, and the checksum computed in software, then the parallel portion of TCP/IP consumes

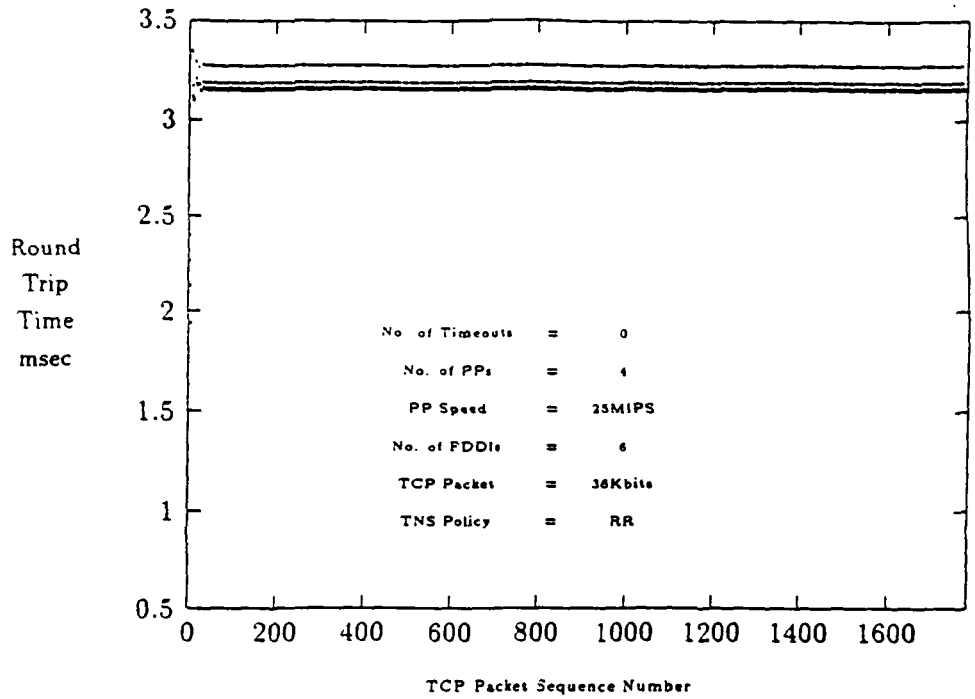


Figure 13: Serial architecture: RTT Distribution for TCP Packets

58.2 μs and the serial portion consumes 12.1 μsec. This indicates that while a serial protocol processor of 15 MIPs would take 70.3 μs to send a single 500-byte packet, a two-processor system could send two packets in (2*12.1+58.2) or 112.4 μs, a speedup of 1.25. Continuing this analogy, the maximum achievable speedup of 5.7 ($\approx$ 70.3/12.1) is possible. Similarly, when the packet size of 4500 bytes is considered, a speedup of up to 3.5 can be obtained for a memory speed of 40 ns.

From this analysis as well as the simulation results discussed earlier, we conclude that parallelism at the protocol processing level can result in a 3-7 fold increase in the network throughput. This gain may be obtained in spite of the inherent limitations such as the existing TCP/IP protocols and the bus speed in bus architectures. Further, if the current efforts of other researchers to develop more efficient protocols succeed (thereby reducing the sequential portion), then we can obtain even higher throughput by employing the proposed parallel architectures.

4 Distributed Memory-Based Parallel Computer Architectures

A second application of parallel networking we believe to be particularly fruitful is where parallel physical networks are connected to a device which already uses parallelism in some fashion, for example massively parallel computers. In this section we study the instantiation of the generic model in which a single application is running as a set of cooperating processes on a large set of individual processors, each with its own memory and connected through a complex communications fabric. The selection of this model was motivated by the structure of the Touchstone DELTA machine. The DELTA is a MIMD machine and can have up to 512 processors and has six I/O nodes (gateway processors). It is suitable for several computation intensive applications. For example, a user at a remote site is interested in the time evolution of, for example, the solution of a computational fluid dynamics problem. This application in addition to the need for intensive computation requires a large amount of data to be moved through the network at a very high speed (Gbps).

Background Traffic			Performance			
A-Bus Traffic	N-Bus Traffic	FDDI Channel	Architectures	Network Scheduler Policy	Throughput (Mbps)	Round Trip Delay (μsecs)
30% uniform	30% uniform	30% uniform	2-bus	RR	388	2253
			1-bus	RR	323	1092
			Serial	-	236	4311
30% uniform	30% uniform	30% non uniform	2-bus	RR	268	2601
				Adaptive	302	2246
			1-bus	RR	228	40997
				Adaptive	270	3449
			Serial	-	236	4442

Table 3: Summary of performance comparison

Packet Size (bytes)	Memory Speed (ns/word)	Processor Speed (MIPS)	Checksum Comput.	Time for Send() call Parallel Part (μs)	Time for Send() call Serial Part (μs)	Maximum Speedup
500	40	15	Software	58.2	12.1	5.8
	20	25		30.4	5.8	6.2
	40	15	Hardware	19.2	3.1	7.2
	20	25	(on the fly)	10.0	1.5	7.6
4500	40	15	Software	164.3	63.6	3.5
	20	25		98.6	39.6	3.4
	40	15	Hardware	14.3	3.6	4.9
	20	25	(on the fly)	8.8	2.1	5.1

Table 4: Potential speedup for send() system call

The instantiation of the model based on Touchstone Delta is shown in Figure 16 with the following features:

- a set of k application processors interconnected through a mesh type interconnection network,
- a set of n protocol (gateway) processors interconnected as a linear array,
- a set of m network units, one for each gateway processor,
- all types of processors are assumed to have local memory, and
- both the schedulers, application scheduler and network schedulers, are mapped onto the protocol processor.

4.1 Model and Simulation

This simulator builds on the simulator used for the workstation node and uses the same modules for the TCP/IP and FDDI simulations and the same DCRA method for handling acknowledgments and

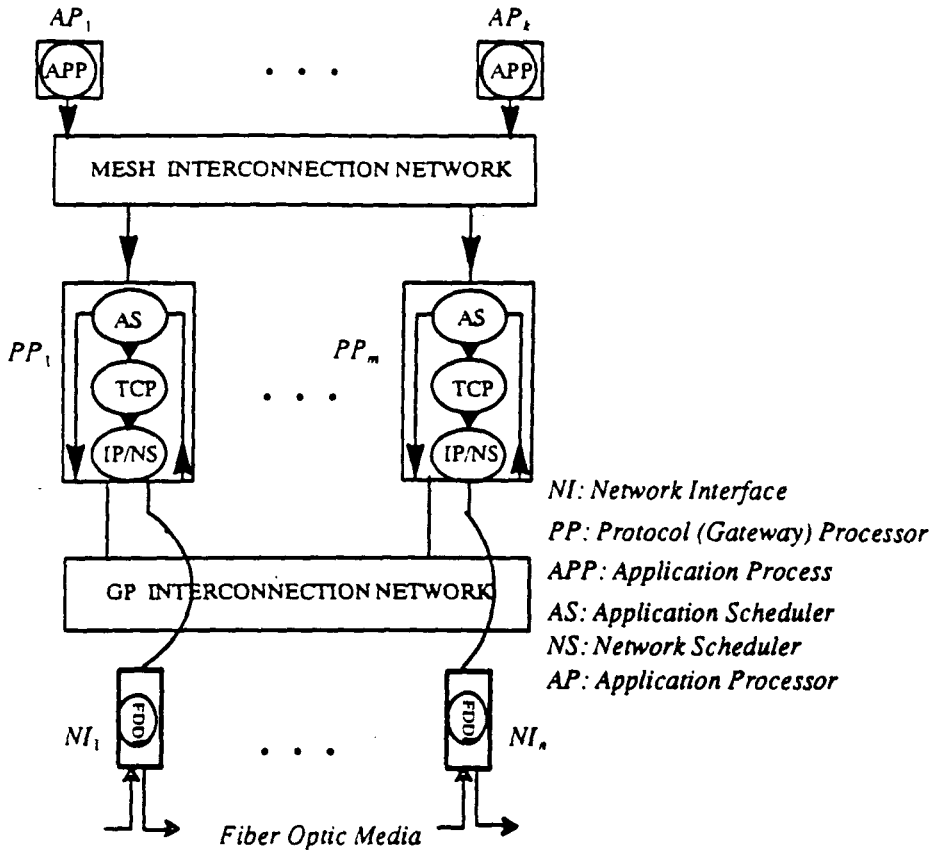


Figure 14: Distributed memory machine model

retransmissions. Some of the features of the model for which we did the simulation runs are summarized below.

- **Independent data generators.** The simulation model does not incorporate the modeling of the application data generation from the various processors. The protocol processors receive data from independent data generators.

- **Fixed mapping.** The number of protocol processors is much less than the number of application processors, thereby implying more than one application processor sends data to a single protocol processor. In general, this mapping of application processors to protocol processors may not be fixed. That is, an application processor with time can change its destination protocol processor. However, for simplicity we assume this mapping to be fixed. By changing the arrival data rate on a protocol processor we can have the same effect.

- **Scheduling strategy.** We simulate *fixed* and *adaptive* strategies for the application scheduler and network scheduler. In the fixed strategy the application scheduler sends data to its TCP/IP process. In the adaptive strategy the application scheduler, using the information from TCP/IP, can choose to send data to an application scheduler residing in the neighboring processor. An application scheduler accepts or rejects a segment depending upon the status of its input queue. If its input queue reaches the "high" threshold value, the application scheduler rejects the segment and passes it to the application scheduler on the right adjacent processor. This application scheduler behaves in

Sched. Policy	Throughput (Mbps)			
	No Background Load	Uniform Background Load (30%)	One Channel Overload	Two Channels Overload
Fixed/ Fixed	104.87	88.18	74.39	73.15
Adaptive/ Fixed	153.83	153.57	151.26	145.97
Adaptive/ Adaptive	154.54	153.86	151.81	146.23

Table 5: Impact of scheduling with data generation rate being 37% of the available capacity at a fixed window size of 64 Kbits and network length of 20 Km

Sched. Policy	Throughput (Mbps)			
	No Background Load	Uniform Background Load (30%)	One Channel Overload	Two Channels Overload
Fixed/ Fixed	224.47	125.18	100.54	88.15
Adaptive/ Fixed	374.29	270.88	266.53	258.04
Adaptive/ Adaptive	374.80	275.30	224.03	202.66

Table 6: Impact of scheduling with data generation rate being 87% of the available capacity at a fixed window size of 64 Kbits and network length of 20 Km

like fashion unless application scheduler queues on all the processors have reached the high threshold values. In this case the data segment will stay with the *last* processor. Here, the last processor is the left adjacent neighbor of the processor which initially had the data segment. To keep track of the last processor, a count byte is appended with the data segment. The count is incremented by a processor before it passes the data segment to the right neighbor. (We assume the linear array interconnection network has wrap around connection.) Similarly in the fixed strategy for network scheduler, data are sent to the network units residing in the same processor whereas in the adaptive strategy data can be send to the network units residing in the adjacent processor.

- **Control flow.** Once a data segment reaches the last processor, a control flow signal is broadcast to all data generators to stop sending data. The data generators are restarted once the application scheduler queue of any processor falls below the low threshold value. Note that we use two thresholds: "high" threshold and "low" threshold for the input queue at application scheduler. The "high" threshold is used for blocking the data generator, and the "low" threshold is used for starting the data generator.

- **Acknowledgments and retransmissions.** The acknowledgment for a packet after coming to a network interface is passed to the network scheduler of the connected processor. The network scheduler diverts the acknowledgment to the proper TCP where the packet was generated. A TCP,

Sched. Policy	Throughput (Mbps)					
	Network Length of 5 Km		Network Length of 20 Km		Network Length of 100 Km	
	Uniform Load	Two Channel Overload	Uniform Load	Two Channel Overload	Uniform Load	Two Channel Overload
Fixed / Fixed	88.65	73.15	88.18	72.15	75.2	60.34
Adaptive/ Fixed	153.82	147.43	153.56	145.97	125.2	120.20
Adaptive/ Adaptive	154.01	147.56	154.20	146.10	146.52	127.58

Table 7: Impact of scheduling on different network lengths with data generation rate being 37% of the available capacity at a fixed window size of 64 Kbits

after collecting all the packet acknowledgments of a segment (which it had broken into packets), sends the segment-ack to the application scheduler.

- **Hardware Characteristics.** In our simulation some of the parameter values such as time taken to move data from one processor to an adjacent processor do not correspond to the actual values of the Touchstone DELTA. But these timings do not affect our conclusions since they are based on relative performances. We also assume that in the future fast lower level protocols such as FDDI will be supported on the system.

4.2 Performance Results

Our approach to evaluate the system mirrors the one described in Section 3.2. The emphasis here is on the impact of scheduling, window size, and network length.

- **Scheduling impact.** The scheduling impact results are summarized in Tables 5 and 6. The first column of these tables give the scheduling strategies adopted for application scheduler and network scheduler.

  For example, an entry *Adaptive/Fixed* implies that we use adaptive strategy for application scheduler and fixed strategy for network scheduler. The second column lists the throughput under no background load conditions. The third column lists the throughput under uniform background load. The fourth and fifth columns list throughput when one and two channels are overloaded. Table 5 gives results when the application offers data at a rate well below the capacity of the parallel network and Table 6 gives the results when the application has to be throttled because the parallel net is saturated.

  Below the performance knee of the delay curve for overall throughput, the scheduling can maintain throughput even when individual channels are overloaded. At saturation level the scheduler helps improve throughput but cannot effectively shift the load. As currently simulated the network schedulers actually degrade performance when the channels are overloaded. The key reason for this behavior is the linear interconnection scheme of the Touchstone I/O nodes. We are developing alternative scheduling policies which, we hope, will solve these problems.

Window Size	Latency (μsec)			
	No Load	Uniform load (30%)	One Channel Overload	Two Channel Overload
4000	76	88	180	1150
8000	53	77	80	220
14000	70	70	70	90

Table 8: Impact of scheduling on window size with data generation rate being 37% of the available capacity at a fixed network length of 20 Km for adaptive/adaptive scheduling

- **Window size and network length.** In Table 7 we observe that throughput is maintained better under overload for longer networks when we use scheduling (although they are not yet optimal). In longer networks queues start building up at the network interface units which are redistributed by the schedulers.

  Table 8 shows the expected impact of latency by increased window size under no load and shows that the effect is exacerbated by overloading of channels. We observed the same randomization effect on round-trip delays over parallel channels which we saw in the workstation node and expect solutions for the window management and retransmissions used in the workstation node to work as well here.

In summary, we have shown that coarse grain parallelism is equally applicable to distributed memory parallel machines. The protocol issues and solutions carry forward from the bus-based machines but scheduling policies have to be adapted to the architectural features of parallel machines.

5 Conclusions

In this paper we have analyzed the use of coarse-grain parallelism (that is, the use of both multiple protocol processors running replicated software and several physical channels) to provide high speed communication services to a single application at a single network node. We developed a general structure for coarse-grain parallelism appropriate for use in a gigabit per second node. Since performance is highly dependent on real issues such as hardware properties (e.g., memory speeds and cache hit rates), operating system interference (e.g., interrupt handling), and protocol performance (e.g. effect of timeouts) we performed detailed simulation studies of both a bus-based multiprocessor workstation node (based on the Sun Galaxy MP multiprocessor) and a distributed-memory parallel computer node (based on the Touchstone DELTA). Mapping of the general model into concrete architectures requires selection of scheduling algorithms and assigning processes (protocol and scheduling) to physical processors. To operate near the potential speeds possible using the particular architecture, this mapping must reflect the communication fabric available in the underlying hardware.

We can draw some general conclusions about the use of coarse-grained parallelism for high performance networking. Some are based directly on the performance studies which we concluded of the two hardware architectures we studied:

- Coarse-grain parallelism can deliver multiple 100Mbps with currently available hardware platforms

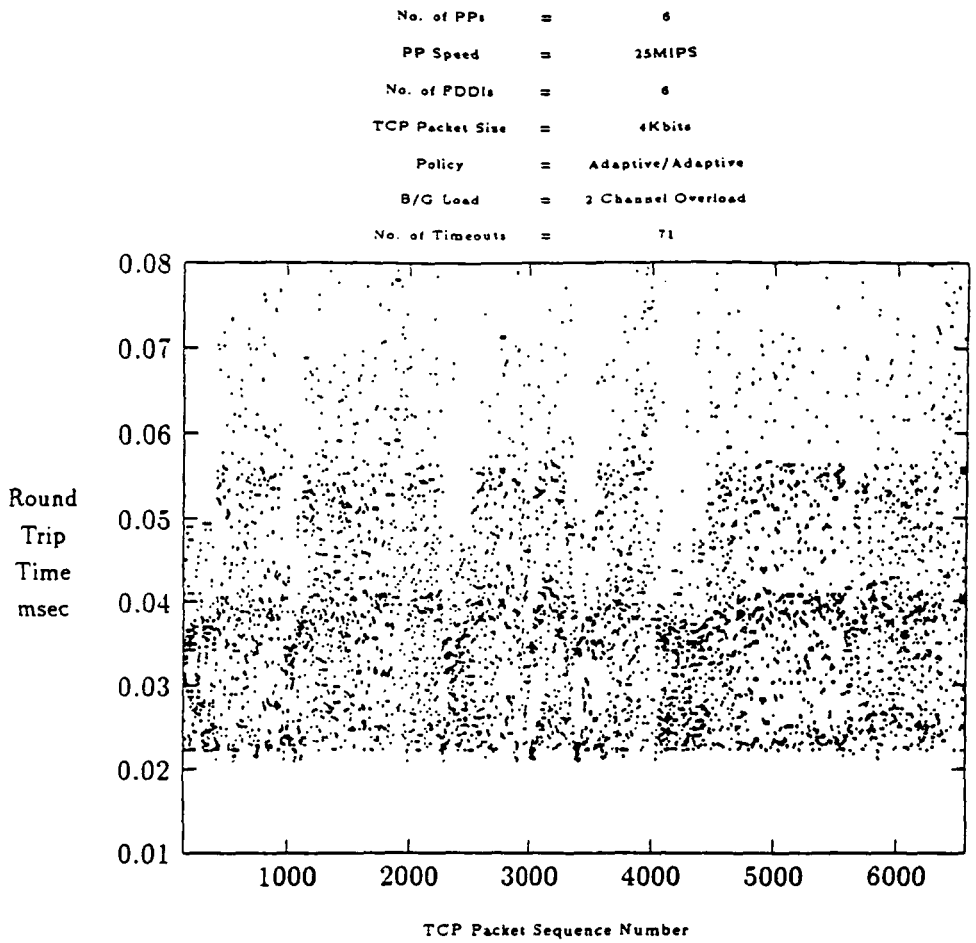


Figure 15: RTT Distribution for TCP Packets

(such as the Sun Galaxy) and with existing networking protocols (using TCP/IP) using parallel FDDI rings.

- Scale up is near linear in the number of protocol processors and channels (at least up to a few hundred Mbps). This is more significant since the analysis is for hardware architectures and existing protocols (TCP) and MAC layer components (FDDI) not designed with high speed network applications in mind.
- Since these results are based on existing hardware (both the Sun and the Touchstone) without specialized hardware (except perhaps for some simple modifications of the FDDI boards), we feel that this is a low cost solution to providing multiple 100Mbps on current machines.

In addition, from both our detailed performance analysis of the physical structures of these architectures based on coarse-grained parallelism, and the components assumed to be present in the architectures, we conclude:

- The use of multiple processors providing identical services and the use of space division multiplexing will provide better reliability than monolithic approaches if properly designed. It also provides graceful degradation and low-cost load balancing.
- This architecture supports running several different protocols (e.g., TCP and UDP) in parallel. This allows, for example, different TCPs to manage network connections with different service requirements

(many small messages for many users handled by one TCP with several other TCPs providing a high bandwidth network connection for a single application).

- The basic architecture will be able to incorporate many improvements from other work (e.g., reduced data movement, fast TCP, gigabit nodes, fine-grained parallelism) again with a near linear speed-ups (at least for a small n) as these improvements become available.

We can also make some detailed conclusions about the particular architectures we studied. These conclusions should also apply to other hardware platforms with similar features.

- The Sun MP Galaxy based parallel processing architecture is capable of delivering throughput in excess of 300Mbps provided sufficiently high speed memory is available.
- A Touchstone-like machine is capable of delivering 300 Mbps network service to a multiprocessor distributed application using FDDI (though as is typical with parallel algorithms, performance depends on how effectively the application uses the Touchstone processors).

Scheduling and scheduler placement has significant impact on performance. Related to scheduler placement and scheduling algorithms:

- The 2-bus distributed scheduler architecture outperforms other bus-based architectures in terms of higher throughput, lower delay, lower processor MIPS and bus speeds.
- The throughput capability of a 1-bus centralized scheduler architecture is limited by the shared memory bandwidth.
- For the bus-based architectures a simple scheduling policy of transport to network scheduler fails to push expected throughput at lower round trip delay under nonuniform channel load conditions.
- On the parallel computer node, the scheduling technique which we implemented (shifting packets among protocol processors) maintains throughput irrespective of load on the channels at lower data generation rates but does not work well in overload conditions.
- On the parallel computer node, the current TS scheduler is more effective for larger network lengths.

Related to properties of protocols when used in a parallel environment:

- The segment size of the data flowing from application to the transport layer must be the same size as MAC frame size to reduce round-trip delay time.
- TCP window size should be such that none of the protocol processors starve for data and no packet waits extra for available TCP send window.
- Efficient implementation of the TCP timeout timers mechanism in case of multiple parallel channels is nontrivial and requires further study.
- As window size increases, latency increases for each architecture which we examined.
- On large diameter networks (with large latencies) the parallel channels can be used with techniques such as cross channel coding to reduce latency (somewhat) and to significantly reduce the need for retransmission of data.

Our general conclusion is that coarse-grain parallelism is effective for increasing the networking capabilities of currently available hardware and is a promising approach for building a true gigabit node. It is compatible with, and complements, much of other work designing gigabit nodes.

We are continuing our analysis and are in the process of designing and implementing a functional hardware prototype capable of providing several 100Mbps to a single application. This will allow us to validate our concepts and to refine our analysis and modeling techniques so that we can design a gigabit node based on this approach.

Acknowledgments

We wish to thank R. Yerraballi, S. Kelkar and H. Srivatsan for their contributions to this work.

References

[1] Bharat Bhargava, Tom Mueller, and John Reidel. Experimental Analysis of Layered Ethernet Software. *IEEE Publication*, pages 559–568, 1987.

[2] W. E. Burr. The FDDI Optical Data Link. *IEEE Communications*, pages 18–23, May 1986.

[3] D. Cheriton and C. Williamson. VMTP as the Transport Layer for High-Perfromance Distributed Systems. *IEEE Communications Magazine*, 27:158–169, June 1989.

[4] G. Chesson. XTP/PE Design Considerations. *Processings of IFIP Workshop on Protocols for High-Speed Networks*, pages 27–33, May 1989.

[5] G. Chesson. XTP/PE Design Conserations. *Protocol Engines, Inc.*, 1990.

[6] I. Chlamtac and A. Ganz. A Multibus Train Communication (AMTRAC) Architecture for High-Speed Fiber Optic Networks. *IEEE Transactions on Communications*, pages 903–912, July 1988.

[7] D. D. Clark. Modularity and Efficiency in Portocol Implementation. *RFC 817, NIC*, July 1982.

[8] D. D. Clark, V. Jacobson, J. Romkey, and H. Salven. An Analyis of TCP Processing Overhead. *IEEE Communications Magazine*, pages 23–29, June 1989.

[9] P. Cochrane and M. Brain. Future Optical Fiber Transmission Technology and Networks. *IEEE Communications*, pages 45–60. November 1988.

[10] J. F. Ewen et al. Gb/s fiber optic link adapter chip set. *10th Annual IEEE Gallium Arsendie Integrated Cricuit Symposium*, pages 11–14. Nov. 6-9, 1988.

[11] William et. al. Survey of Light Weight Transport Protocols for High Speed networks. *IEEE Transactions on communications*, pages 2025–2039, November 1990.

[12] E.C. Foudriat, K.J. Maly, C. M Overstreet, S. Khanna, L. Zhang, and W. Sun. Combining two media access protocols to support integrated traffic on high data rate networks, 14pp. *Proceedings of the 16th Local Computer Network Conference, Minneapolis*, 1991.

[13] A. G. Fraser. Towards a Universal Data Transport System. *IEEE Journal on selected areas in communications*, SAC-1:803–816. November 1983.

[14] Z. Haas. A Communication Architecture for High-Speed Networking. *Proceedings of IEEE Infocom*, 1990.

[15] B. Hoffman, W. Effelsberg, T. Held, and H. Konig. On the parallel implementation of osi protocols. *Proceedings of IEEE workshop on the architecture and implementation of high performance communication subsystems*, February 1992.

[16] R.N. Ibbett and N.P. Topham. *Architectures of High Performance Computers - Volume II.* Springer-Verlag, NY., 1989.

[17] Van Jacobson. Congestion Avoidance and Control. *Processidngs of ACM SIGCOMM'88*, pages 314–329, 1988.

[18] N. Jain, M. Schwartz, and T.R. Bashkow. Transport Protocol Processing at GBPS Rates. *Proceedings of Sigcomm '90*, pages 188 –199, 1990.

[19] K. Kaede. Twelve-channel parallel optical fiber transmission using a low-drive-current 1.3 μm led array and a pin pd array. *OFC'89 Technical Digest*, page 15, Feb. 1989.

[20] S. R. Kunkel and J. E. Smith. Optimal pipelining in supercomputers. *Proceedings of 13th Symposium on Computer Architecture*, June 1986.

[21] T. LaPorta and M. Schwartz. Design, verification and analysis of a high speed protocol parallel implementation architecture. *Proceedings of IEEE workshop on the architecture and implementation of high performance communication subsystems*, February 1992.

[22] K. Maly, E. C. Foudriat, D. Game, R. Mukkamala, and C. M. Overstreet. Dynamic allocation of bandwidth in multichannel metropolitan area networks. *Computer Networks & ISDN*, 1992. To appear.

[23] K. Maly, C. M. Overstreet, R. Mukkamala, M. Zubair, F. Paterra, S. Khanna, Y. S. Sekhar, and R Yerraballi. Design, verification and analysis of a high speed protocol parallel implementation architecture. *Proceedings of IEEE workshop on the architecture and implementation of high performance communication subsystems*, February 1992.

[24] K. Maly, F. Paterra, C. M. Overstreet, R. Mukkamala, and S. Khanna. Remote visualization using parallelism in the context of existing networks. *To appear in Proceedings of International Phoenix Conference on Computers and communications*, April 1992.

[25] NRI. *Toward a National Research Network*. National Research Council, 1988. NRI Review Committee.

[26] David A. Patterson, Garth Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). *ACM SIGMOD Conference*, pages 106–116, June 1988.

[27] Thomas F. La Porta and Mischa Schwartz. Architectures, Features, and Implementation of High-Speed Transport Protocols. *IEEE Network Magazine*, 5:14–22, May 1991.

[28] J. B. Postel. Transmission Control Protocol. *RFC 793*, September 1981.

[29] L. Svobodova. Implementing OSI Systems. *IEEE Journal on Selected Areas in Communications*, pages 1115–1130, September 1989.

[30] R. W. Watson and S. A. Mamrak. Gaining Efficiency in Transport Services by Appropiate Design and Implementation Choices. *Processings of IFIP Workshop on Protocols for High-Speed Networks*, 1987.

[31] J. A. Wiencko. Cross-channel coding implementation guide, Novemeber 1991.

[32] M. Zitterbart. High-Speed Transport Components. *IEEE Network Magazine*, pages 54–63, 1990.

[33] M. Zitterbart. Parallel protocol implementations on transputers - experiences with osi tp4, osi clnp, and xtp. *Proceedings of IEEE workshop on the architecture and implementation of high performance communication subsystems*, page 4, February 1992.