Old Dominion University
ODU Digital Commons

Computer Science Theses & Dissertations | Computer Science

Summer 1993

Multilevel Parallel Communications

Sanjay Khanna
Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/computerscience_etds

Part of the Computer Sciences Commons

Recommended Citation
Khanna, Sanjay. "Multilevel Parallel Communications" (1993). Doctor of Philosophy (PhD), dissertation, Computer Science, Old Dominion University, DOI: 10.25777/4eky-1111
https://digitalcommons.odu.edu/computerscience_etds/110

This Dissertation is brought to you for free and open access by the Computer Science at ODU Digital Commons. It has been accepted for inclusion in Computer Science Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.

MULTILEVEL PARALLEL COMMUNICATIONS

by

SANJAY KHANNA
B.E. July 1987, Delhi University, India
M.S. December 1991, Old Dominion University, USA

DISSERTATION
Submitted to the Faculty of Old Dominion University in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY
COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
August 1993

APPROVED BY
SUPERVISORY COMMITTEE:

Supervisor: _____

_____

_____

_____

_____

_____

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

MULTILEVEL PARALLEL COMMUNICATIONS

Sanjay Khanna
Old Dominion University, 1993

Director: Kurt Maly

The research reported in this thesis investigates the use of parallelism at multiple levels to realize high-speed networks that offer advantages in throughput, cost, reliability, and flexibility over alternative approaches. This research specifically considers use of parallelism at two levels: the "upper" level and the "lower" level. At the upper level, N protocol processors perform functions included in the transport and network layers. At the lower level, M channels provide data and physical layer functions. The resulting system provides very high bandwidth to an application. A key concept of this research is the use of replicated channels to provide a single, high bandwidth channel to a single application. The parallelism provided by the network is transparent to communicating applications, thus differentiating this strategy from schemes that provide a collection of disjoint channels between applications on different nodes. Another innovative aspect of this research is that parallelism is exploited at multiple layers of the network to provide high throughput not only at the physical layer, but also at upper protocol layers. Schedulers are used to distribute data from a single stream to multiple channels and to merge data from multiple channels to reconstruct a single coherent stream. High throughput is possible by providing the combined bandwidth of multiple channels to a single source and destination through use of parallelism at multiple protocol

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

layers. This strategy is cost effective since systems can be built using standard technologies that benefit from the economies of a broad applications base. The exotic and revolutionary components needed in non-parallel approaches to build high speed networks are not required. The replicated channels can be used to achieve high reliability as well. Multilevel parallelism is flexible since the degree of parallelism provided at any level can be matched to protocol processing demands and application requirements.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Copyright
by
Sanjay Khanna
1993

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To my parents
Vishwanath and Prem Lata Khanna,
and my brothers
Karan and Udeek Khanna.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgements

I must thank Professor Maly and other members of my committee for the continuous support and encouragement. I could not have finished this work without their guidance and valuable suggestions all through my research work. I must also thank Dr Zubair for his encouragement all these years. I will be failing in my duties if I do not thank Ajay Gupta and his systems group. They have been very patient with my demands about various software and hardware requirements for my experiments. I also thank all those who directly and indirectly have helped me all these years. I must also thank my parents for their blessings and my brothers for their unending moral support. My special thanks go to my wife, Dheerja, who endured my late night and early morning working schedule. Last but not the least, my hearty thanks to the almighty God for His graciousness all these years.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table of Contents

Acknowledgements iv

List of Tables vii

List of Figures viii

Chapter 1. Introduction 1

Chapter 2. Background and Related Research 10
2.1 Comparison with Prior Parallelism 10
2.2 Advances in Media Access Control Protocols 14
2.3 Advances in Protocol Processing for High Speed Networks 17

Chapter 3. Multilevel Parallel Communications 24
3.1 Generic Model Description 25
3.2 A Special Case of General Framework 29
3.2.1 Application-Transport Layer Interface 30
3.2.2 Segment Processing at the Transport/Network Layer 32
3.2.3 Transport/Network and Media Access Interface 34
3.3 Issues and Options 35
3.3.1 Window Management Schemes 38
3.4 Network I/O Performance of a Typical Workstation 40
3.5 Features for Parallel Communications Systems 49

Chapter 4. Multiprocessor Instantiations 53
4.1 Instantiations 54
4.2 Architecture 1 57
4.3 Architecture 2 59
4.4 Architecture 3 61
4.5 Model and Simulation 62

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.5.1 Components 63
4.5.2 Types of Components 63
4.5.3 Action Routine 67
4.5.4 Packets 69
4.5.5 Event manager 69
4.5.6 Major Assumptions 70
4.5.7 Parameters of Interest 71
4.5.8 Selected Metrics for Performance Measurements 72

Chapter 5. Results and Measurements **74**
5.1 Simulation : Verification and Validation 75
5.2 Simulation Results 79
5.2.1 CPU Speed 80
5.3 Application Scheduling and Number of PPs 83
5.4 Network Scheduling and Number of FDDIs 85
5.5 Memory Speed 87
5.5.1 Transport Window Size 91
5.5.2 Application Segment Size 93
5.5.3 Scheduler Location 93
5.5.4 Comparison of Architectures 96
5.5.5 Acknowledgment and Retransmission 96
5.5.6 DRDA vs. CRCA 98
5.6 Experimental Measurements for Network Scheduling 103
5.6.1 Details of the Experimental Testbed 104
5.6.2 Experiment Measurements and Results 109

Chapter 6. Conclusions **115**
6.1 Future Directions 119

Bibliography **122**

Vita

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Tables

4.1 Major architectural characteristics of architectures 61
4.2 Table of simulated parameters 72

5.1 Major parameters for early experiments 82
5.2 Memory Speeds and related memory bandwidths 87
5.3 Final architectural parameters 90
5.4 Comparative performance of three architectures 95
5.5 Architecture 3 : DRDA vs. CRCA (Lossless Channels) 98
5.6 Architecture 3 : DRDA vs. CRCA (1% Loss on Channels) . . . 99
5.7 Performance of AA Scheduling Under Bursty Loads on Ethernets 111
5.8 Performance of IAA Under Bursty Loads on Ethernets 112
5.9 Parallel Ethernet Performance table for uniform and continuous loading of Ethernets 113
5.10 Parallel Ethernet Performance table for Bursty loading of Ethernets 114

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

1.1 Motivation 7

2.1 Physical Interface for HiPPI 16
2.2 Classification of High Speed Network Research 18

3.1 Multilevel Parallel Communications: A Generic Framework . . 26
3.2 Multilevel Parallel Communications: A Case Study 31
3.3 Acknowledgements and Retransmissions 38
3.4 Sparc Station 1 Architecture Based on SBus 41
3.5 TCP/IP Processing Time as a Function of Packet Size 45

4.1 Architecture 1 58
4.2 Architecture 2 59
4.3 Architecture 3 60

5.1 Batch variance as a function of batch size 78
5.2 Throughput as a function of CPU speed 81
5.3 Protocol processor utilization as a function of CPU speed 81
5.4 Round trip delay as a function of CPU speed 83
5.5 Throughput as a function of number of protocol processors . . . 84
5.6 Round trip delay as a function of number of protocol processors 84
5.7 Throughput as a function of number of FDDIs 86
5.8 Queueing delay at FDDI as a function of number of FDDIs . . . 86
5.9 Throughput as a funcion of memory speed 88
5.10 Round trip delay as a function of memory speed 88
5.11 Throughput as a function of TCP window size for Arch. 1 . . . 91
5.12 Round trip delay as a function of TCP window size for Arch. 1 91
5.13 Throughput as a function of TCP window size for Arch. 2 . . . 92
5.14 Round trip delay as a function of TCP window for Arch. 2 . . . 93
5.15 Throughput as a function of TCP window size for Arch. 3 . . . 94

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.16 Round trip delay as a function of TCP window for Arch. 3 . . . 94
5.17 Experimental testbed for Parallel Ethernets 106
5.18 Illustration of the relationship between SLOT and Sample IP Packets for Adaptive Algorithm 108
5.19 Performance Comparison of a Single Ethernet versus Parallel Ethernets 111

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

Introduction

As computer communications advance further in to optical networks technology, more importance and expectations in terms of throughput is placed on data communications. Fiber optics networks (e.g. FDDI - Fiber Distributed Data Interface) supply much higher bandwidth (100 Mbps) to users than the current networks (e.g. Ethernet - 10 Mbps). The promise of these new technologies has led to prototyping and development of true high bandwidth applications. These applications with high bandwidth and strict delay requirements place additional performance requirements on communication protocols. These applications include full motion video for use in teleconferencing which needs high bandwidth and puts tight bounds on the network delay and computer imaging - medical, weather and seismic - which demands low latency for data collection and high throughput data transfers. Visual techniques are also becoming increasingly important to understand the results from advanced computer models and simulations. Distributing a problem among networked computing resources, and computer steering are used for visually oriented modeling. Such needs result in large bandwidth requirements. For example, to drive a 1280x1024 24-bits color display updated 15 frames per second requires 472 Mb/s.

Currently available low-end (single CPU) workstations, such as SUN Sparcstation 1, are now capable of processing network I/O in the excess of 10 Mb/s. But to sustain higher throughput, say 500 Mbps, the network interface of the workstation must process packets at the rate of approximately one packet

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

every 2 μs for 100 bytes packets or every 20 μs for 1 Kilobyte packets. This implies that if a processor capable of executing one instruction every 50 ns is used, only 40 or 400 instructions are allowed for each packet. This strict processing requirement has several solutions. The traditional approach is to use a faster processor. Alternatively, several slower processors can be used in parallel.

The other aspect of the problem of supporting hundreds of Mb/s of throughput needed for modern applications is the availability of the physical media which can carry data faster. The currently available inexpensive LANs can carry data at 10 Mb/s (Ethernet or Token Ring).Also, faster solutions like 100Mb/s FDDI, 150Mb/s DQDB and 155Mb/s ATM LANS are available. But cost may become a factor in their selection. From current market estimates, an Ethernet card is available for less than $100 whereas an FDDI card is more than $1500. The choice of a particular network technology is always driven by the requirements of the network designers and cost is an important factor in their design. From above numbers, there is an indication that employment of parallel and inexpensive LANs can be a cheaper alternative to purchasing an expensive, and faster serial LAN. But one must not ignore the fact that there are other costs involved in intelligently using the parallel channels. Also, there is a possibility of this trend in costs to continue as technology advances (cost of one fast channels vs. many slow channels).

The research reported in this thesis investigates and demonstrates the use of parallelism at multiple protocol levels to realize very high speed networks, providing multi-hundred Mb/s bandwidth, that offer advances in throughput, cost, reliability, and flexibility over alternative approaches.

This research specifically considers the use of parallelism at two levels — the *upper* level, and the *lower* level. At the upper level, "n" protocol processors perform functions of OSI transport and network layers. At the lower level,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

"m" channels provide data and physical layer functions. The system provides a very high bandwidth to an application and is termed the "Multilevel Parallel Communications System".

A central idea of this research is the use of replicated channels to provide a single high bandwidth channel to a single application. The parallelism inherent in the network is transparent to communicating applications, thus differentiating this strategy from schemes that provide a collection of disjoint channels between applications on different nodes. An innovation of this approach is that parallelism is exploited at multiple layers of the network to provide high throughput not only at the physical layer, but also at upper protocol layers.

Very high throughput is possible by providing the combined bandwidth of multiple channels to a single source or destination, and through the use of parallelism at multiple protocol layers. This strategy is cost effective since systems are built using standard technologies that benefit from the economies of a broad application base. The exotic and revolutionary components needed in monolithic approaches to high speed networks are not required.

Replicated channels can be used to achieve high reliability as well as performance. Faults may degrade the performance but operations can continue, thus providing graceful degradation which is otherwise impossible in a single channel network. The multilevel parallelism approach is flexible since the degree of parallelism provided at any level can be matched to protocol processing demands and applications requirements.

Demonstrating both the benefits and limitations of incorporating parallelism as a central concept in the design of LANs is the purpose of this research. In our parallel approach, multiple copies of existing hardware and software components are used in parallel to provide for concurrent transmission and reception of single application's data. Here, parallelism is defined as the representation of a single user's data as a set of concurrent data streams

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

which can be moved in parallel. The degree of parallelism at any level is determined by the number of processing elements or channels needed to operate on the data streams in parallel without introducing a bottleneck in the overall flow. This technique of protocol processing is called **Multilevel Parallelism.** The number of data streams may vary as they are being processed at various levels of the protocol stack. Scalability of this approach is defined 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 bandwidth levels in the network. I believe that the use of parallelism in network node equipment and on the network itself will provide a mechanism for large increase in communication performance. This improvement of network is similar to the way that parallel computing techniques which have been used to achieve dramatic advances in the computational power of the computers. The question of what parallelism provides in terms of performance, reliability, and cost is intimately tied to how parallelism is implemented. Issues which arise include data decomposition, process replication, channel selection and control, and end-to-end control. Many of these are similar to issues which exist in parallel computing.

There are many way to construct a parallel protocol processing structure. One way is to construct as a pipelined structure. In various stages of this pipeline, multiple packets can be processed simultaneously for various fields in the packet structure. Alternatively, independent processing of several fields of a packet can take place in different stages of pipeline simultaneously. Due to the packet formats and placement of checksum, however, it is difficult to parallelize the processing of a single packet (i.e. process several fields simultaneously). Also, replicated parallel structures can be constructed which can perform independent protocol processing on several packets. Each of the replicated structure must have very little resources and information to share with other structures for best performance. An important requirement in this scheme is the division of application data into multiple data streams at the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sender's end and then merging them at the receiver's end. This approach depends upon the scheme used to connect the replicated structures. In this research, I have followed this approach as it is flexible and a natural evolution of existing software and hardware components.

A two level classification for parallelism in protocol processing is found in [103]: *functional parallelism* and *data parallelism.* Functional parallelism is the decomposition of a protocol task at a given level into several tightly-coupled, parallel subtasks. In the data parallelism approach, identical protocol processing tasks run independently on several processors; each handling packets from one or more data streams. (This is the approach used in multilevel parallelism.) The key difference between the two approaches is that the protocol process is decomposed in the first approach and replicated in the second.

Note that functional parallelism is limited in its degree of parallelism by the number of distinct subtasks in the protocol and the dependence among subtasks. In the data parallelism, different degrees of parallelism can be applied to different layers to provide services sufficient for that level. Because, in data parallelism, processors are operating on separate packets, they are largely independent and synchronization overhead is small. Both, but to a greater extent the latter approach, are scalable with physical processors providing network services when needed and available to other tasks at other times.

Data parallelism also provides a performance advantage when differing classes of service are required for different communicating applications. This gives the flexibility of running, for example, several copies of TCP on one set of 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 avoids frequent context switches and cache misses for processors handling high band-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

width applications. In addition to this, previous work indicates a performance improvement is realized by placing traffic with similar attributes on separate channels[64].

One advantage of parallelism which needs no demonstration is increased reliability [48, 60]. One can exploit the redundancy of multiple resources to achieve graceful degradation if individual resources fail. A second advantage is flexibility. Parallelism does not have to be restricted to homogeneous networks and systems. No intrinsic reason prohibits from sending packets concurrently over different media access boards such as Ethernet or FDDI cards. Similarly, one can have heterogeneous nodes, such as workstations and parallel computers, communicating over the same parallel net. Finally, multiple applications can use parallel communication simultaneously. A third major advantage, and the one which I seek to study, is performance gain. With parallel resources, gains are generally available over a broad range of conditions if some form of dynamic load balancing is used. Parallelism's potential can be a near linear gain at least for several processors and channels. At some point adding more processors will not significantly add to performance because the overhead of parallelization increases (due to the cost of additional resources and load control).

The motivation for this research is the inability of the current protocol implementations on existing architectures to support high throughput to a single user application. This problem of preservation of throughput at higher OSI layers is more critical when network like FDDI (100 Mb/s) is employed. A very small percentage ($\approx$ 17%) of the FDDI bandwidth is available at the transport layer and approximately half of the bandwidth available at the transport layer is observed to be used by any one user application. This loss of bandwidth as one moves up the OSI stack is illustrated in Figure 1.1. As seen from this figure, the ideal case will be when the entire bandwidth available at the physical layer can be used by the user applications. But the conventional implementations show a marked deterioration from the ideal case. To solve this problem of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

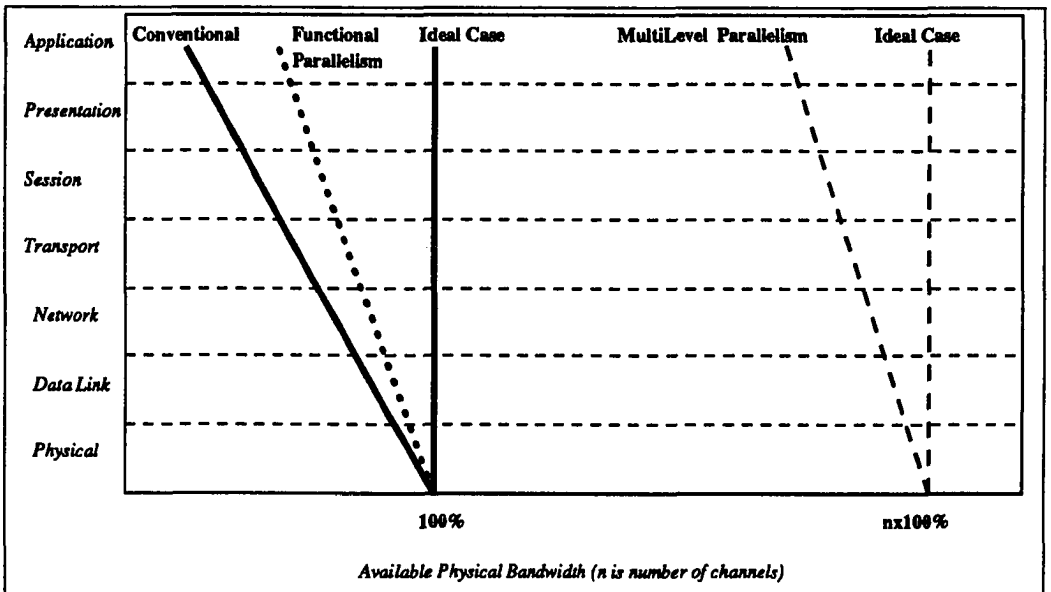

Figure 1.1: Motivation

inability of preservation of bandwidth at the higher ISO levels, many research efforts have been carried out in the past. There have been proposals for the extensively redesigned transport protocols, newer protocol stacks (with layers removed/merged), implementations of the protocols in silicon and parallel implementation of the protocol processing. The research presented in this thesis is entirely based on a parallel approach to the solution of this problem. Other approaches are presented in Chapter 2.

The ISO seven layer stack model of communication may not be workable for multi-hundred Mb/s network implementations due to extra overhead associated with maintaining the layer segmentations. Nevertheless it still serves as a good model to discuss the potential introduction of parallelism into a system. I have introduced a very general form of parallel communications in this research. A major part of this research explores the possible use of multilevel parallelism for high performance communications. Parallelism at the transport and network layers can provide high bandwidth service while a single transport like connection service to any application. At the data link or media access

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

layers, parallelism can be introduced by having separate processors (media access cards e.g. FDDI cards) for each channel. At the physical channel level, a number of alternatives exists. On one hand, a simple space multiplexing of separate optical fibers to handle individual streams of data can be used . On the other hand, wavelength division multiplexing of all streams onto a single fiber can also be used.

How can one integrate all these various possibilities into a single system and what other questions arise as a result of integration? This research work has made an attempt to answer this. Scheduling policies which utilize physical channels effectively are studied in detail. Protocol processing issues arising out of scheduling and their impact on overall performance are studied. In particular, I am interested in identifying scheduling policies, window management techniques, timeout and acknowledgment handling policies and retransmissions. Based on the results of this study, different network scheduling policies (to effectively balance load on parallel channels) are tested on a parallel Ethernet network.

With this background about the problem and the possible solution approaches, let us refer to Figure 1.1. As a comparison to the conventional protocol processing and its performance, expected performance curves for data parallelism have been drawn. For some degree of parallelism, the ideal case in performance would be a vertical line. In real world, it is expected that performance will be worse than the ideal case. A greater fraction of the bandwidth, however, is expected to be preserved at the higher levels in comparison to what is preserved in today's network architectures.

In this thesis, I explore issues in multilevel parallel communications. This exploration is extended to determine what performance gains can be expected from currently available architectures. In Chapter 2, various efforts researchers have done to implement high performance communications are de-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

scribed. An attempt to categorize different efforts in the classification of various research activities in the area of high speed networks has been made. In Chapter 3, the generic concept of multilevel parallel communications, various issues and their related options, and a special case study of the generic model which becomes the model of all later discussions are presented. In Chapter 4, three multi-processor instantiations of this model are introduced. A description of the simulation model is also provided. Performance results from simulations are reported in Chapter 5. I also performed experiments on a parallel network testbed. Results of these experiments are also reported here. Finally, I summarize our conclusions in Chapter 6.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

Background and Related Research

Parallelism is commonly in use in several research areas and products relating to electrical engineering and computer science. The ideas motivating the use of parallelism in these areas suggest the need for, and the benefits of parallelism in network systems. Parallelism in network systems can be employed at various levels of the protocol stack. In this chapter, an introduction to the parallel systems already in use in computations, computer systems and telephony is presented first. Advances in the media access protocols which are key elements in building a high performance LAN are also reported. Growing interest in protocol processing resulted in new developments in recent years. A great deal of research has been directed towards developing new transport protocols, porting existing and new protocols to silicon, and developing Transputer based protocol processing systems. A survey of such activities is presented as well.

2.1 Comparison with Prior Parallelism

Parallelism has been employed in computing for over 20 years [2, 33, 102]. The major reason for using parallelism in computers has been to increase computational speeds by increasing the overall processor power in a computer [72, 58, 5]. Continued work in parallel computing is based on supposition that increasing computational speeds to sufficient levels is either not possible for serial structures, or it is more effectively done by the use of replicated processor structures

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for reasons of cost, and development time.

A secondary reason for using parallelism in computers has been fault-tolerance. The replicated processor structures is used to achieve some degree of fault-tolerance, although hardware parallelism is not the only technique used to achieve fault-tolerance. Historically, fault-tolerant computing has been of interest to niche markets, with application types including general-purpose commercial systems, high availability systems, systems with long life needs (especially space-borne systems), and systems providing critical computations (especially real-time systems). Many of the computers designed for these applications achieve their fault-tolerance in part by the use of parallel, and redundant processing elements [89].

Parallelism has also been employed in backplane buses. Micro, mini, and main-frame computers have used parallel connections for transmission of address and data signals. For example, VME, a bus standard commonly in use now, uses two separate 32-bit bus lines for address and data [1]. As with computing, it is possible to use the parallelism inherent in buses to achieve fault-tolerance, although many efforts todate at including fault-tolerance in buses have focussed on use of parity bits, redundant arbitration procedures, and redundant clocks[73]. Research work in re-arrangeable networks (for example, shuffle exchange networks) has resulted in techniques to achieve fault-tolerance in buses by switching incorrectly operating channel out of the bus [81].

Electronic interface links have commonly employed parallelism. In some ways, they can be thought of as extensions of parallel buses. The centronics parallel interface used to connect printers to computers is common example. A more recent example is the high performance parallel interface (HiPPI). A standardization effort by ANSI task group. It implements parallelism using 25 meters of twisted pair copper wire. Research work has been done in the area of parallel disks, or Redundant array of inexpensive disks (RAID)[75]. Initial

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

results indicate a potential for both cost and fault-tolerance improvements in data storage technology.

Parallelism has been employed in local area networks, but generally only for fault-tolerance. Arcnet-based networks with a redundant twinax bus structure were developed for common industrial use in control systems as early as 1984 [42]. An Ethernet compatible system with counter rotating fiber rings was developed in 1985-1987 and is currently in widespread use [99, 98]. It contains automatic and transparent reconfiguration in the event of single or multiple station or cable failures. The fiber distributed data interface (FDDI), a new and popular ANSI standard that also utilizes self reconfiguring counter-rotating fiber optics rings, is currently gaining widespread use among a large number of vendors. Within the FDDI committee in ANSI, there has been discussion of the use of the redundant pathway in an FDDI network as an additional pathway for data transfer, but as yet there is not a part of the standard.

Within metropolitan and wide area networks, parallelism has been used to increase throughput between two points as well as to increase fault-tolerance through diverse routing. The most common example of the use of geographically diverse parallelism is the long distance telephone network [49]. When parallelism is used in telephony for extra throughput, however, it is generally to provide multiplexing capability; the resulting large bandwidths are not available to be used monolithically, for single sets of users. Plans exist to use source routing bridging techniques to transfer up to sixteen parallel channels of information across local and metropolitan, and potentially wide area networks; with this technique, a single set of users can simultaneously use diverse paths within a multiple-connected network to achieve a high data rate throughput capability.

Parallel networking is employed in the newly developed asynchronous

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

transfer mode (ATM) system [92]. Here parallel networks are obtained through the lower virtual circuits (Synchronous Transfer Mode or STM) support for high data rates. Bandwidth can be increased by request to the underlying network control but the ATM system has no mechanism within its own operations to utilize the parallelism to its advantage. In addition, the parallelism provided by the STM results in out of sequencing of the arriving packets (or cells in ATM) and hence there is a need for having a controller at each end which must periodically test the circuit in order to enable correct sequencing [7].

Finally, some initial work on employing parallel channels directly on optical fibers has been done. Based on current trends and theoretical limits on the speed of electronic circuitry, the vast bandwidth capability of fibers (on the order of Terabits per second) is unlikely to be utilized by electronic time division multiplexing (TDM) alone. Indeed, there are strong arguments suggesting that optical processing be employed to multiplex a number of moderate speed electronically multiplexed TDM channels. This techniques is more feasible than further pushing the single channel speed given the limitations of electronic circuitry [47].

Wavelength division multiplexing (WDM) is another technique to obtain parallel channels on an optical fiber. Tunable lasers [56] may be used for wavelength generation, but this necessitates optical heterodyne techniques for detection. Suitable choice of modulation (for example, wide-deviation FSK) can lessen requirements on transmitter and local oscillator stability. Alternatively, wider wavelength separation may be used allowing use of passive optical filters. Considerable progress has been made in improving resolution of such filters [14, 88], and in implementing them in fiber-like structures [82].

Collectively, these efforts relating to parallel computing, parallel buses, parallel links, parallel disks, and self-healing local, metropolitan, and wide area networks demonstrates that parallelism, when properly employed, can provide

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

significant advantages relating to performance and system cost.

2.2 Advances in Media Access Control Protocols

Networks can be divided into two categories: those using point-to-point connections and other using broadcast channels. Media access protocols are important in those networks which use multiaccess channels as the basis of communication. From experience with static channel allocation, Ethernet[4] and Token Ring LANs, significant understanding for low bandwidth, high latency protocols has been gained. It is preferable to talk of relatively new media access control (MAC) protocols such as FDDI, DQDB, CSMA-RN, and HiPPI. Development of such high performance media access protocols will result in the building of high speed LANs. Their importance in parallel networks will become obvious when they are commonly used and cheaper to build. Their wide-spread acceptability will result in their cheaper integration to parallel networks. For these reasons, it is important to mention the developments in this area.

Perhaps the most significant accomplishment of FDDI (Fiber Distributed Data Interface) [52, 53, 59, 85, 93, 94, 29, 6, 24, 41] is its interoperability with existing networks and processors. Its design provides for implementation for back-end peripherals to processors, back-bone interconnection of networks, and as a network for interconnection of a variety of workstation types. The basic design is a dual counter-rotating ring. A variety of traffic types are accommodated including isochronous, synchronous, and asynchronous traffic. Each of the rings are a version of the token ring format where each node transmits according to a timed token rotation protocol to allow processing of synchronous traffic. The design also includes fault tolerance capabilities. A single link can be severed without disconnecting the nodes and the network becomes a single ring rather than a dual ring. As with other token rings, the sending node is required to remove packet it places on the ring. Recent development of copper

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

based FDDI or CDDI has caused a renewed interest in FDDI. CDDI is a copper based cheap alternative to fiber optics. CDDI is less expensive because the use of opto-electronics is avoided.

Similar to Fasnet[62, 84], DQDB uses a dual unidirectional bus architecture. The node at the head of the bus inserts frame markers on the bus and handles reservation of frames for synchronous traffic. This approach does not issue tokens, but instead pumps empty frames onto the bus for use by nodes with queued packets. Like Fasnet, DQDB places traffic on channels on the network based on the upstream or downstream physical position relative to the sender. The interesting aspect of DQDB is the distributed queueing algorithm the developers claim to be a perfect scheduler. In addition to being more equitable in allocation of bandwidth to nodes on the bus, DQDB is designed with recovery procedures to enable the network to reconfigure itself in the event of a single cable cut. If a second link is severed, the network is physically disconnected. DQDB also provides for prioritization of packets. Due to the more efficient placement of packets by the distributed queueing algorithm of DQDB, one would anticipate better throughput (packets/sec) and better delay performance. Throughput could be slightly improved by taking advantage of packets which have been reserved for downstream use, but will be emptied before reaching the sending node. This would require that the upstream node's destination is prior to the downstream node that will reuse the packets.

CSMA-RN[36, 37, 34, 55] is a carrier sensed multiple access protocol for high data rate ring networks. This protocol takes advantage of the fact that, at high data rates, networks can contain multiple messages simultaneously over their span, and that in a ring, nodes need only to detect the presence of a message arriving from the immediate upstream neighbor. When an incoming signal is detected, the node truncates the message it is presently sending instead of aborting it. The service time is basically a function of the network rate; it changes by a factor of 4 between no load and full load. Wait time, which is zero

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for no load, remains small for load factors up to 70% of full load. Response time, which adds travel time while on the network to wait and get serviced, is mainly a function of network length, especially for longer distance networks. Destination removal on average increases network load capacity by a factor of 2. A scaling factor based upon message to network length demonstrates that CSMA-RN is applicable to wide area networks too. When the assumption of uniform destination selection on the ring of gigabit speeds is dropped, the problem of ring hogging and fairness may become serious issues in local area networks.

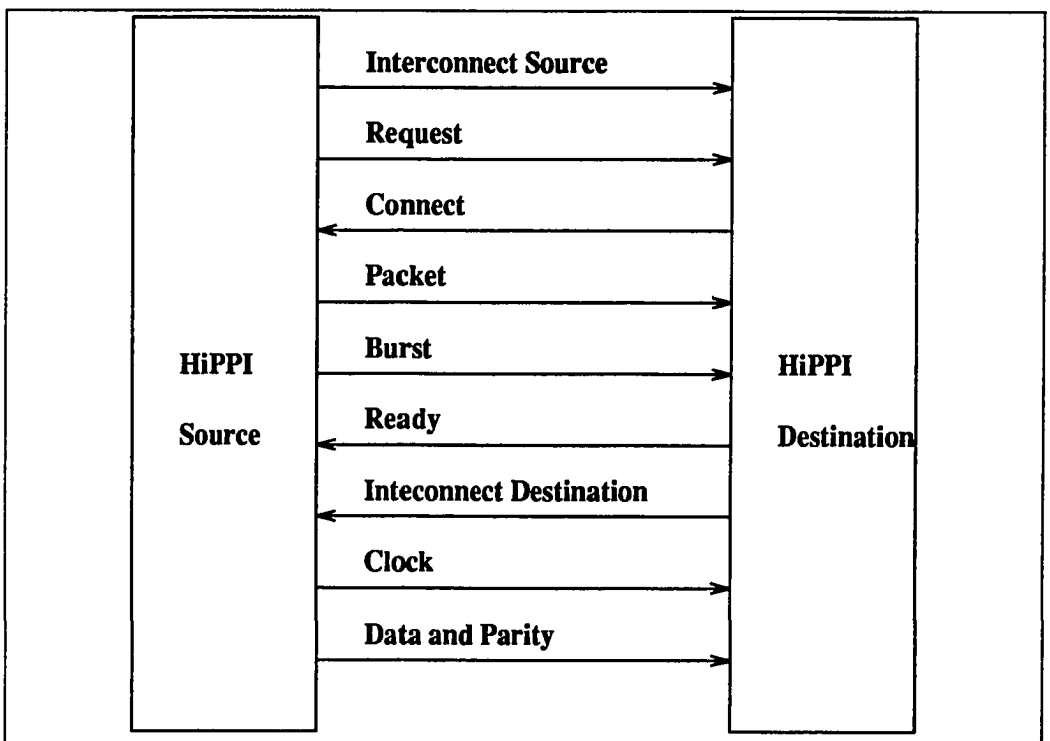

Figure 2.1: Physical Interface for HiPPI

HiPPi[101] is a physical and link layer protocol used to transport data at a rate of 800Mbps as detailed in ANSI X3T9.3 standard. Each HiPPI link is a simplex electrical link consisting of 32 data bits, 4 parity bits, 7 control signals, and 1 clock signal. The peak data throughput that can be achieved is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

793Mbps. For a full duplex HiPPI connection, two independent simplex HiPPI connections are required. Figure 2.1 illustrates the physical interface used for HiPPI connections between source and destination machines. The interconnect source and destination signals indicate that hosts are powered up and the network is connected. The request signal is used by the source to establish connection with the destination. The source places a 32 bit I-field on the data bus when the signal is asserted. This I-field may be used for routing and addressing functions. The destination HiPPI responds to a request signal by asserting connect signal. The packet signal is used to delineate packet boundaries. It has no limit on the size of a packet. Each packet is composed of bursts, and bursts are normally 1KBytes. The clock is 25MHz clock with 50% duty cycle. The ready signals are used for flow control. The error detection consists of a parity bit, associated with each byte of data, and a length/longitudinal redundancy code(LLRC). This combination is guaranteed to detect all three-bit errors. Congestion control and single point of failure are the two weak points about HiPPI interface. When two HiPPI hosts are connected over Internet, the intermediate gateways and routers will get swamped with data.

2.3 Advances in Protocol Processing for High Speed Networks

Made possible by progress in fiber-optic and VLSI technologies, networks offering increased transmission capacity at decreased error rates are becoming available. New applications can benefit from this bandwidth but software protocol processing rates have not kept up with available raw transmission speed available at the hardware level. The performance bottleneck has hence shifted from the network to the processing required to execute communication protocols in workstations and servers[15, 39, 77, 17, 31, 3]. Owing to this, a single application cannot utilize a reasonable fraction of the bandwidth of a communication network, even on readily available networks with data rates in the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

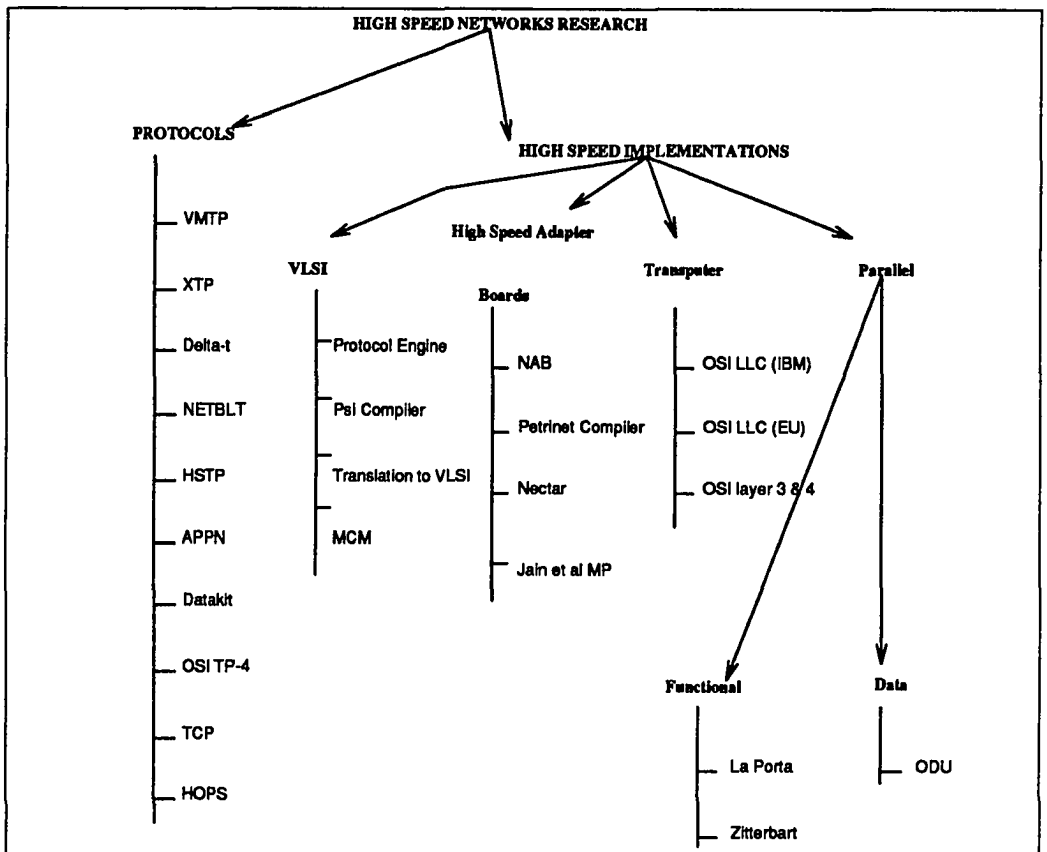

Figure 2.2: Classification of High Speed Network Research

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

10Mbps range. This restriction becomes more acute when when networks offering larger bandwidths e.g. FDDI (100Mbps) become widespread. Thus the design decisions used in developing many existing protocols are inappropriate for the evolving networks. The three main decisions in question are the liberal use of processing power to reduce transmission costs, addition of extra processing cost to recover from errors, and the use of relatively simpler flow-control mechanisms. The efforts in high speed networks (especially, protocol processing) can be classified [103] as shown in Figure 2.2. Various independent efforts were adopted by researchers. A large number of them proposed new protocols for transport layer [9, 11, 10, 32, 96, 57, 97, 18, 71, 25, 46, 66, 86, 30, 13, 39, 40, 15].

XTP, one of such newly developed transport protocol, has gained some acceptability for high speed networks but it is still far from widespread acceptance. Since Internet's TCP has a very large existing base, there is a great deal of inertia in the community against accepting new protocols. Most of the proposes protocols have been designed for use with a particular problem domain in mind (for example, VMTP was designed for transaction oriented communications). There are doubts about their performance in a general environment.

TCP[8, 16, 20, 21, 80] relies on the Internet Protocol (IP) to provide a unified network-wide datagram service, independent of many subnetworks that make up the Internet. The emphasis on survivability in the presence of failed nodes and desire of independence from particulars of the underlying networks have led to development of the connectionless network service, readily be implementable on large numbers of inexpensive routing nodes. This service also recognized the different requirements of end-to-end data transport service. One of these is the fully reliable, connection-oriented, byte stream data transmission provided by the Transmission Control Protocol (TCP). TCP suffers from the problem of retransmission ambiguity: when an acknowledgment arrives for a datagram that has been retransmitted, there is no indication of whether the acknowledgment is for the original or retransmitted packet. This

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

may result in error in the estimate of round trip times. According to Clark's analysis[17], approximately 400 instructions are required to implement a TCP receiver. This includes 154 lines of common code for entering and exiting procedures, 59 instructions to perform protocol processing tasks, 35 instructions for buffer management for packets that have been transmitted and are awaiting acknowledgment and 57 instructions for IP processing. To implement TCP on a RISC machine, the analysis assumes an extra 33% overhead bringing the final total to 400 instructions. Clarks's analysis also showed that 235 instructions were required to transmit a packet. The great amount of analysis performed on TCP protocol and its widespread use motivated us to consider it as a potential protocol for parallel networks.

Alternatives to OSI protocol stack for computer communications have been suggested. An important contribution, the development of Horizontally Oriented Protocol Structure (HOPS), is reported by Haas in [44]. The main idea behind HOPS is the division of the protocol into functions instead of layers. The functions, in general, are mutually independent in the sense that the execution of one function can be performed without knowing the results of the execution of another. Thus intercommunication between layers is substantially reduced. This reduces the latency of the protocol and improves throughput. HOPS can be implemented as a collection of custom designed hardware and general purpose software.

There have been attempts to implement protocol engines in silicon. These VLSI implementations result in very inflexible protocol processing. Once developed in VLSI, it will be very difficult to change protocol parameters to match changes in environment. The goal of using VLSI chips is to find another way of translating protocol specifications into implementations. The approach taken by such implementations mostly deal with the protocol state machine and do not necessarily improve the performance of a protocol layer. Other problems of flexibility of VLSI implementation include the lack of support for

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

multiple connections on a single chip. The PSi Compiler and work reported in [26] deals with design of VLSI implementation out of protocol specifications. Additionally, within the XTP project[12] an architecture for a VLSI chipset for XTP protocol is described. The Modular Communication Machine (MCM)[68] aims at a modular system design based on functional units (e.g. protocol functions) that are programmable for special protocol requirements. If high performance is possible with a programmable element using general protocols, then this is a more desirable approach rather than the inflexibility and cost of VLSI implementation.

Another research effort has been concentrating on developing network adapter boards which can process protocols at higher speeds. This effort is based on separating host processing from protocol processing. Integration of such adapter boards into the host systems may be an issue. The Network Adapter Board[54] developed at Stanford University is optimized for working on VMTP messages; for example, it supports the calculation of the checksum on the fly. The other concept of realizing a special adapter board is based on the description of the protocols with Petrinet[87]. Special hardware is needed to realize petrinet based adapters. Carnegie Mellon University has built a high speed local area network called Nectar that uses programmable communication processors as host interfaces [28]. They have implemented the TCP/IP protocol suit and Nectar-specific communication protocols on the communication processor. Steenkiste, et al. [90] look at a host interface architecture which streamlines the execution environment for protocol processing. In particular, a Communications Accelerator Block (CAB) is provided which minimizes data copies, reduces host interrupts, supports DMA, hardware checksum and network control access. The system is mapped to both an iWarp parallel machine and to a DEC workstation with a TURBOchannel bus. Jain et al.[50] propose an architecture based on parallel processing to achieve Gbps rates for end-to-end protocol processing. The key concept is that of processing packets on

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

distinct processors in parallel. The protocol processing task is presumed to be accomplished in front-end system which is dedicated for this purpose. They assume suitable hardware support to perform any necessary lower layer protocol processing. Using global data structures for the window context records increases the probability of contention in this architecture. In [83], Ramakrishnan addresses a similar problem of interfacing a network with 100 Mb/s performance. This work concentrates on partitioning functions between the network interface and the host software. A simple model is provided in this work which looks at the performance of the network I/O and predicts user perceived throughput which is the most important parameter in the evaluation of overall network system operation. A similar model has been adopted by us to estimate I/O capabilities of a typical workstation.

A small group of researchers have explored the use of Transputers to do the protocol processing. Some Transputer based implementations[22, 27] could be classified as adapter based solutions as well. Using parallelism and a general purpose solution are main design issues for Transputer approaches. The University of Erlangen and IBM implementations focus mainly on the implementation of OSI LLC protocol. They do not support parallelism inside the protocol state machine, but have built a global memory for Transputers which seems to be necessary for high performance protocol implementation. University of Karsruhe implementation supports parallelism based on level of protocol functions, including global memory concept[103]. Due to software emulation of global memory in Transputer based solutions, it may not be an efficient solution to high performance protocol processing.

Another important research effort has been the exploration of use of parallelism for high speed protocol processing. In [76], La Porta reports an architecture for parallel implementation of TCP transport protocol. The design is based on dividing the general protocol functions, such as connection management or reliable data transfer, into subtasks which can be performed in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

parallel. Some of these tasks were performed by dedicated processors and others are distributed over several processors. Their performance analysis shows that the parallel architecture may provide up to 60% higher throughput than the serial implementation of TCP.

Zitterbart in [104] presented another approach towards high performance communications platform based on a parallel protocol implementation on Transputers. The important design issues in this implementation are the protocol subdivision in send and receive, global memory for data send and receive and global memory for control information. The global memory concept has been emulated in software because Transputers do not support physical global memory. The major conclusion of this study was that the parallelization effort is highly protocol dependent and has to be based on a thorough knowledge of data dependencies between protocol functions. Most of the work discussed above has concentrated on the separation of protocol processing functions and performing them in some parallel fashion. We, on the other hand, considered a different kind of parallelism. In our approach, multiple data streams were handled for protocol processing by various processors. We also considered parallelism at physical channels level. The functional parallelism discussed above compliments in our data parallelism. Moreover, the bulk of parallel network work [79, 43, 71] employs parallel processing mainly to improve performance at the transport/network level but little work analyzes the use of parallel physical channels. In [61, 65, 70, 69, 35], authors have examined strategies for use of parallel network channels at the media access level as well.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

Multilevel Parallel Communications

In communications systems the processing of a stream of data from a single sender to a single receiver is done at different levels of protocol hierarchy both at the sender and the receiver sites. This view lends itself to a pipelined approach in which processing can occur at different levels in parallel. In addition, if data from a single application are split into multiple streams at some levels in the protocol stack, processing can be performed in parallel on these separate streams. These streams can then proceed separately through multiple levels and be rejoined later as appropriate. By judicious choice of stream splitting and hierarchic levels, one can exploit multiple levels of protocol processing with process replication at each level: separate identical processes (potentially on different physical processors) working on individual data streams.

This approach is realizable on several existing hardware architectures and provides a general framework for presenting the research reported here. 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 parallelism is intended to provide a framework which allows us to study issues which will be generally applicable to the use of parallelism in networking and to study the interaction between particular hardware structures when using parallelism.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In this chapter, a generic framework for parallel protocol processing is presented. In a network architecture, major communication components are at the transport, network, media, and physical layers. These layers are primarily responsible for information transfer. Major research interests lie in exploring parallelism at these layers. Thus, a special case of this framework is elaborated which involves parallelism at transport and its lower layers. Many issues related to data scheduling among multiple streams and protocol processing (for example, flow control) arise as a result of the use of parallelism in communications. These issues and various solutions are also explored in this chapter. In order to determine how existing host architectures respond to the network I/O needs, a benchmark study was performed on a typical workstation. This study helped in identifying major bottlenecks present in the processing of network I/O. Lastly, a classification of major features considered important for high performance communications are reported as result of the benchmark study. This classification will form the basis for instantiations of multiprocessor architectures used for parallel networks.

3.1 Generic Model Description

Figure 3.1 is a representation of our model for parallel communications in a very general form. The central idea is to examine parallelism wherever it may be effective; the model presents those places where we expect parallel processing to be useful. We do not expect any effective concrete realization to have a large number of parallel levels but the model is capable of handling them.

As a general model, Figure 3.1 needs further clarifications. In this model, the first three layers have been assumed to be merged into one layer which provides for their functionalities. This is done so because these three layers have a little contribution in supporting end-to-end communication. In the model, it is assumed that data from one or more applications are present

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

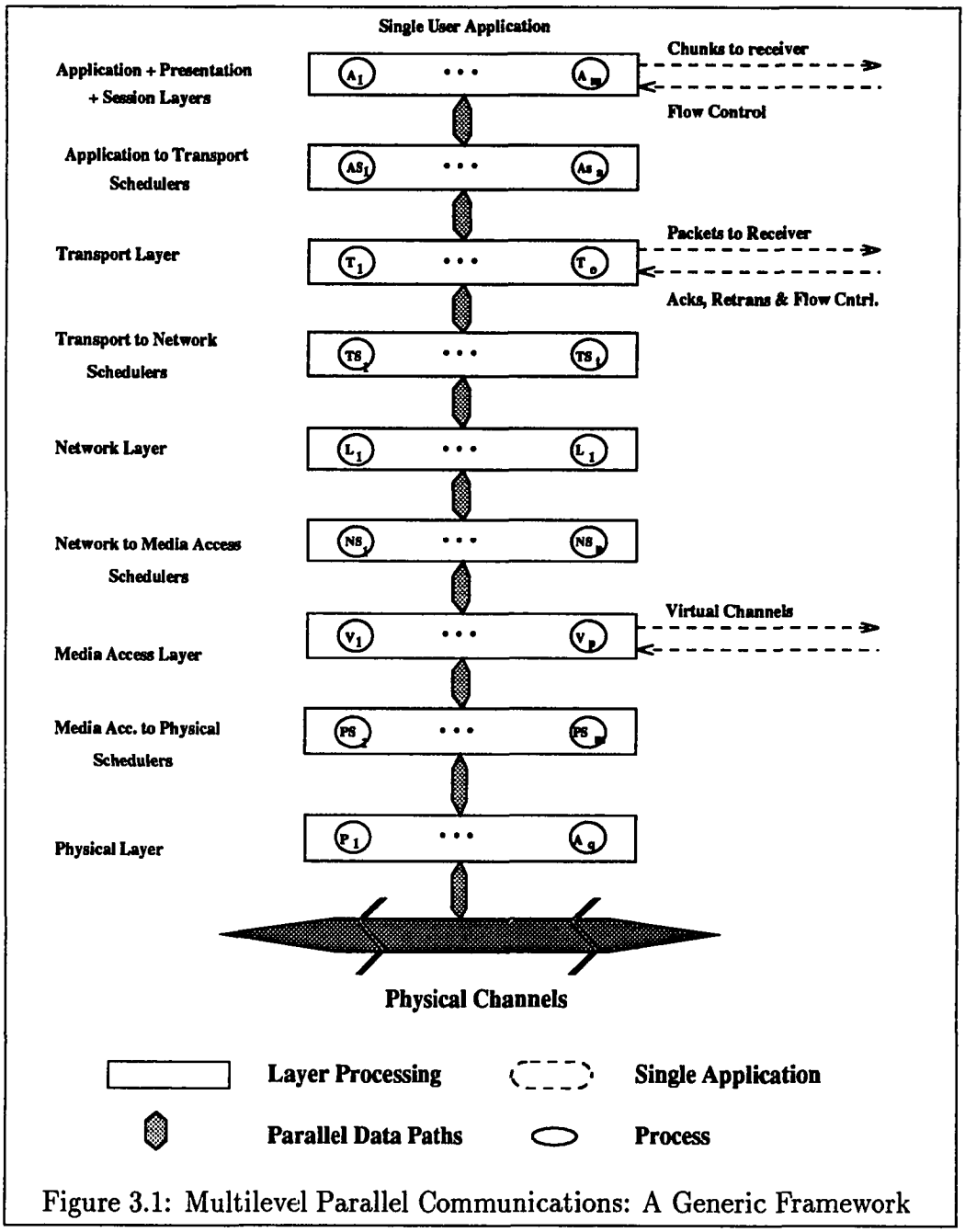

Figure 3.1: Multilevel Parallel Communications: A Generic Framework

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

at an intermediate layer, i. Data (represented as multiple streams) are processed at protocol layer i by many processors and pass through to next layer with the help of scheduling processes. Necessary information is assimilated in a scheduler between layers $i-1$ and i so that it can distribute workload in an intelligent manner to the processes available in layer i. From layer i, data are transferred to the subsequent layer $i+1$ via another scheduler where the similar data stream scheduling operations are repeated. Eventually, the data are transferred to the possibly parallel physical channels connecting the sender and receiver. A similar structure exists at the receiver where data are processed and transferred to the receiver application. It should be noted that in receiver operations, the scheduler between any two layers perform similar scheduling of incoming data as is done for the outgoing data.

As illustrated in Figure 3.1, at the topmost user application level, an application's data is input to the application scheduler ($AS_1 ... AS_a$). The data from application ($A_1 ... A_m$) may be a single stream or multiple streams generated by its concurrently executing threads[1]. Based on criteria of load balancing and flow control, the application scheduler assigns the application data chunks to the transport layer processes ($T_1 ... T_o$). It should be noted that number of parallel streams at application layer and transport layer need not be the same. The procedure of merging varying number of parallel streams at various layers is done by interlayer schedulers. Care should be taken such that the schedulers themselves do not become bottlenecks in the overall performance of the system. A scheduler can exist between every two layers whenever there is a possibility of having multiple processes in those layers. Transport schedulers ($TS_1 ... TS_t$) are between transport and network layers, network schedulers ($NS_1 ... NS_n$) are between network and data link layer and media access schedulers ($PS_1 ... PS_m$) are between media access and physical channels. Usually there is a one to one

[1]In operating systems which support multi-threading, a thread is a unit of execution and a process may consist of more than one such concurrently executing threads.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

mapping between the physical channels and media access layers, i.e. for every channel there is one instance of media access. But when optical communication is used, wavelength division multiplexing (WDM) can be employed on a single fiber to allow the system to send multiple streams of data on a single fiber.

Since the model provides for different degrees of parallelism at adjacent layers, we now have the opportunity to schedule work for individual processes as data are moved from one layer to the next. The model, which at any level assumes a scheduler followed by parallel processing, makes no assumption about the assignment of tasks to processes or processes to processors. It supports the idea that several tasks may be performed by a single process and/or several processes may be assigned to a single processor. Effective assignment depends on factors such as the degree of interaction among tasks and processes, underlying hardware, operating system overhead, requirements placed on the overall communications system, processor loads from other tasks, and most certainly on the requirements placed on the network performance by the user application(s). In a particular hardware architecture, some tasks may also be realized in hardware. For example, the scheduling scheme between two levels may be determined by the bus arbitration scheme in the bus hardware. Thus, Figure 3.1 is the illustration of a very general model of parallelization based on the OSI stack of communications software levels. Parallelism may not be used at all levels; it is used only where effective. In addition the concept of levels motivates the use of pipelining. In a particular implementation, it is unlikely that the actions at different levels and the number of levels will exactly correspond to the ISO stack model some layers may be merged. Any model using parallelism at any number of levels should fit into this generic model.

Because of the identified bottlenecks in the transport protocol processing and absence of standard media access protocols designed for multi hundred megabit speeds, we have chosen to study a system with parallelism at the transport and media access layers. We have assumed one application process

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

connected by some communications fabric (depending on the particular hardware present in a node) with n protocol processes running in parallel. These protocol processes send and receive data using m physical channels with a network scheduler to balance load on the physical channels. No parallelism is assumed at the application level because the emphasis of this study is on use of parallelism in communications protocol processing. Since multiple protocol processes exist in this reduced model, a data scheduler will still be required between application and protocol processes. This scheduler will assign data segments generated by application to protocol processes in some fashion. This reduced model is of importance to the discussion because entire research work and performance studies presented here are based on it. Issues and options evident in this work have been identified and studied. Moreover, several multiprocessor instantiations were designed based on the model.

3.2 A Special Case of General Framework

The general framework discussed in the last section offers parallelism at all OSI layers of study. To successfully design a practical implementation, a reduced parallel framework was studied in greater details and is presented in this section. The three main components of this framework (refer Figure 3.2) are - an application processor (AP) that generates data to be transmitted, N protocol processors (PPs) to perform transport and network layer processing, and M network interface units (NIUs) to connect to the physical media. The NIUs are media access controllers and can be FDDI or Ethernet interfaces.

A mechanism to allocate application data to protocol processors is needed because multiple protocol processors are used. Figure 3.2 shows this as a scheduler process located in the application processor and called *Application/ Scheduler(AS)*. Regardless of scheduling algorithm used by application scheduler, its basic job is to allocate data "segments" of one or more packets to protocol pro-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

cessors. It could use feedback, such as the application's estimate of required bandwidth, the current load on the protocol processors, or other information when determining the allocation of data to protocol processors. It can also use simple policies like, first-come-first-served (FCFS) or round-robin (RR).

The *protocol processors (PPs)*, in addition to performing the transport and network layer processing, must schedule packets from data segments to the NIUs. As with the application scheduler, the scheduling algorithm used can vary. But the basic task is to supply the network interface units (NIUs) with packets for transmission. This task, labeled *NS or Network Scheduler* in Figure 3.2, could also be located in a special purpose switch (not shown in Figure 3.2) which is placed logically between the network layer and media access layer.

The subsequent subsections explain the various interfaces between layers, processing inside a protocol processor, and other issues related to parallel protocol processing at various layers.

3.2.1 Application-Transport Layer Interface

Data buffers generated by an application may consist of many segments. Each such segment is associated with specific locations within that buffer. With respect to the communication system, the data segments within a buffer are assumed to be independent of each other. The expectation from the communication system is to transfer those segments to a receiver application. How this objective is achieved is transparent to the application.

The transport and network layers are major components of any communication system. These layers are implemented on a set of $\hat{N}$ processors, called *Protocol Processors or PPs.* Data generated by the application are divided among PPs by the application scheduler (AS). The physical location of application scheduler is a design decision. If application scheduler resides in the application processor, it controls allocation of data segments to PPs in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

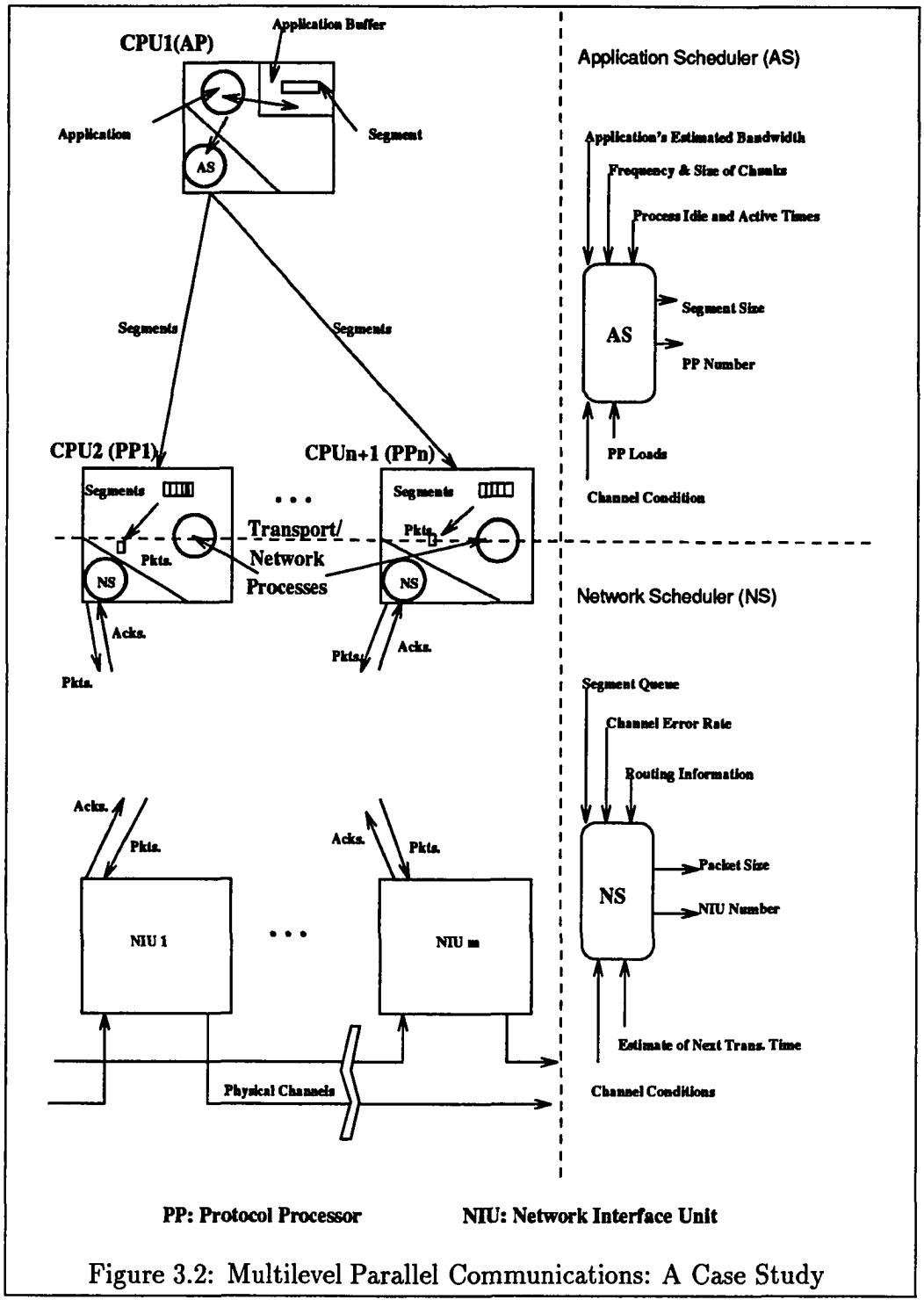

Figure 3.2: Multilevel Parallel Communications: A Case Study

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a *centralized* manner. It could also be implemented in a *distributed* manner where agents of application scheduler are placed on each of the PPs. All the distributed agents would need to cooperate on the scheduling task.

The type of scheduling used by application scheduler has an impact on system performance. One can examine the performance of a system using simple schedulers comparing first-come-first-serve (FCFS) to round robin (RR). With first come first served policy, the application scheduler allocates data segments to PPs as soon as they acknowledge transmission of their previous segment. With round robin, the application scheduler allocates data segments to PPs in a cyclic fashion without overflowing the buffers. Interaction between the scheduler and PPs is reduced in round robin because there is no need to notify the application scheduler when a transmission is complete.

The size of the segment passed from the application processor to the PPs is also an issue. For example, consider a scenario where very large and equal segments are assigned to non-uniformly loaded PPs. When a PP receives its segment and then becomes significantly slower than the others, total system latency will increase and total throughput will decrease. Alternatively, small segments mean more work for application scheduler and more interaction between the PPs and the AP. This could also result in lower throughput because of the processing and data transfers needed for PP to AP communication. In addition to fixed segment sizes, variable segment sizes may be considered when loads are non-uniform. With variable segment sizes, the segment size given to each PP can change as that PP's ability to process data changes.

3.2.2 Segment Processing at the Transport/Network Layer

The transport/network layer is viewed as a set of processes, called TPs, executing on separate PPs. Once a segment is allocated to a sender protocol process, it is the responsibility of that process to deliver the data to the trans-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

port/network layer at the receiving end reliably.

At the transport layer, the management of window, packet acknowledgments and retransmissions are major design issues. Even though their implementation is well understood in a single process case, their extension to parallel processes is not obvious.

In a parallel transport layer, acknowledgments can be processed by either the sender process (TP) or a central process. The central process may run on an totally independent processor. When PPs process their acknowledgments, the network/transport layer interface is responsible for assigning the acknowledgment to a correct protocol process. When a central acknowledgment process is used, all acknowledgments must be assigned to it. These processes must also do transport window management. This can be done in distributed or centralized manner. In distributed case, every protocol process connects to its counterpart protocol process on the receiver side. All window management operations and flow control are done between them (independently of other processes). In a centralized case, one process at each end does window management on behalf of all protocol processes. Therefore, all receive and send operations on packets will have to be done through this centralized process.

A sender protocol process divides a segment into a number of data packets (if segment size is greater than transport packet size), prepares them for transmission, and delivers them to the network layer. The network layer can be implemented on a single process or a set of processes. I have assumed network layer processing to be associated with transport layer processing. They both together form an integral part of protocol processing. Hence, every protocol process does transport and network layer processing.

PPs generate data packets and send them to the media access layer. Since I assume a one-to-one relationship between transport and network layer processing (both residing on a same physical processor), many of the design

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

issues in network layer are the same as a traditional network layer. However, these processes still need to be interfaced with the media access layer. I refer to the processor that handles the MAC layer functions as an network interface unit (NIU), and assume m such units.

3.2.3 Transport/Network and Media Access Interface

Each PP has to select a network interface unit to transmit packets. The task of selecting a network interface unit is done by a *NS or Network Scheduler.* The design decisions for network scheduler are similar to those in the application scheduler design, namely location and scheduling. The network scheduler may be located at each o PP or in some centralized device. The scheduling algorithms can be adaptive or simple. Adaptive scheduling algorithm can be based on channel load, channel latency, NIU's queue size or channel's error rate. An adaptive scheduler will be able to use the total channel capacity more efficiently under non-uniform load conditions. An adaptive algorithm requires state information about each channel. The gathering of state information should be done frequently such that decisions can be made based on fresh information. At the receiver end, network interface units receive packets that must be delivered to PPs. The allocation of data packets to PPs depends on the transport window management policies. Such allocation techniques vary based on distributed and centralized policies.

Several issues may arise when a parallel communication protocol processing model is used. These issues are not existing in the presently available serial implementations. There are several options to handle these issues. In the next section, several of these issues and their related options are discussed.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.3 Issues and Options

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. However, several important issues, which would not exist in a serial implementation, will be present in almost any parallel solution. These issues are discussed here.

1. **Scheduling Policy:**

   Finding an optimal data allocation strategy from an application process to many independent PPs and from a PP to many network interfaces for transmission is a major issue. 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 neighboring layers, or it can be simple, such as first-come-first-served or round-robin. An adaptive scheduler may result in better performance (and utilization) at an additional cost of collecting state information; it can make decisions based on information about what is going on below it (for example, queue length at each processor or expected time of token arrival of a FDDI channel) or what is going on above it (when the next set of data will arrive). The cost of doing adaptive scheduling, obsolescence of some types of potentially useful state information and type of scheduling policy are issues which should be considered when analyzing adaptive scheduling schemes.

2. **Scheduler Location:**

   A scheduling process can run either independently on each PP (distributed scheduling) or on a single processor (central scheduling). For example, the transport to network scheduler can run locally on each PP

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

or in a central device located between network and MAC layers. The location of a scheduler process will impact the performance and degree of fault-tolerance of the system.

3. **Window Management, Time-outs and Acknowledgments:**

   One approach is to have timers and acknowledgments processed by the PPs. In this solution, all PPs operate independently yet still contribute to the task of transmitting a block of data. This is obviously not the only way to perform these functions and I have identified four basic ways in which they can be done. Each of these are illustrated in Figure 3.3 and are discussed in section 3.3.1.

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 efficiently. These strategies can save retransmissions at a cost of additional code bits per packet. If a forward error correction mechanism[100] is employed, all of the bits transmitted in parallel need not be present at the receiver to construct all correct data packets.

5. **Retransmission Timer Value and Packet Loss on Channels:**

   Retransmission timer value in existing non-parallel 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 independent parallel channels meaning that retransmission timer management is an issue in parallel implementations of communication protocols. In case of lossy channels, constructing original data at the receiver application may become a difficult task. Consider a scenario where data from a sender application is sent to a receiver application over multiple channels. Due to packet

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

loss on channels, some of the packets sent from a particular transport window are lost. All protocol processes will continue to transmit their data except for the one process from whose packet is lost. That particular process will block after the complete window is transmitted and will unblock when it times out for the acknowledgment. Meanwhile, other protocol processes may have sent an enormous amount of data since they do not stop sending. This will result in a large out of order queue at the receiver (if the application is provided with ordered delivery of segments). This queue will continue to grow until that lost packet is retransmitted and received by the receiver. An outcome of this will be added delay in receiving data and long out of order data queues at the receiver. This problem may aggravate further when wrong estimates of RTT are used to fire retransmission timers.

One solution that can be used is to send an acknowledgment for every TCP packet received. Transport protocols use positive acknowledgment with retransmissions (PAR). This implies that receiver has received all the data up to the sequence number specified in the acknowledgment. If any packet is lost in this sequence, all subsequent acknowledgments will carry sequence number prior to that lost packet. In this case, a sender PP will receive more than certain number of such acknowledgments and conclude that packet is lost. This way the wrong estimate of round trip times over parallel lossy channels can get compensated with negligible overhead.

6. **Memory Architecture:**

In multiprocessor architectures, distributed shared memory for processors versus local memory per processor is a major performance issue. Although shared memory makes processor coordination simpler, but it comes at the cost of lower performance. Local memory reduces the load

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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 here as they are in serial implementations. A CPU is interrupted when data arrives at the network interface; it causes a context switch of currently executing processes and the execution of interrupt specific code. This is a fixed overhead per receive. This overhead can be amortized over a multiple receives. That is, the processor can be interrupted when a certain number of packets have been received. Also current operating systems copy data from user memory area to system memory area and then to network interface. This copying of data is a big percentage of the protocol service time. Lastly, to maximize the memory bandwidth, the memory and cache architecture should be such that one word is extracted per memory cycle.

3.3.1 Window Management Schemes

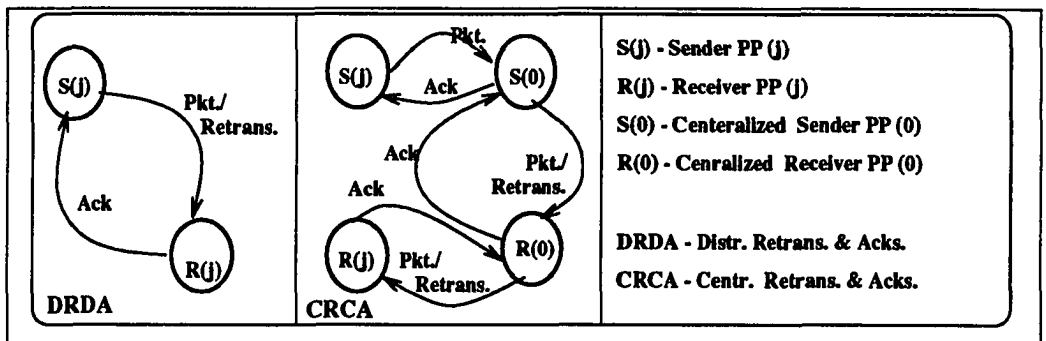

Figure 3.3: Acknowledgements and Retransmissions

Figure 3.3 illustrates the major choices in handling window management operations for acknowledgments and retransmissions. The first solution, called

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Distributed Retransmissions Distributed Acknowledgments (DRDA), assigns the entire task of window management to the PP that generated a packet. The receiving PP generates an acknowledgment and sends it to the transmitting process. This is a distributed solution that essentially uses separate transport connections between the sending and receiving PPs. Exceptions, timeouts and retransmissions are handled by individual PPs.

The second solution, CRCA – Centralized Retransmissions, Centralized Acknowledgments – uses separate PPs on both sides of the connection to maintain timers and to generate acknowledgments. In this solution, the PP labeled S(0) assigns a packet for transmission to PP labeled S(j) from a global transport window and then sends it to receiving PP labeled R(0). On receipt of a packet, R(0) notifies a PP labeled R(j) to do protocol processing. R(j) builds an acknowledgment and refers process R(0) to send it to S(0) process. Received acknowledgments are forwarded to S(0) and the corresponding timers are stopped. This solution is similar to that proposed by Jain et al. in [50]. Hence a single transport window is shared by the PPs at the sender and receiver side.

The third configuration, DRCA – Distributed Retransmissions, Centralized Acknowledgments – uses a central protocol process for acknowledgments and distributed mechanism for retransmissions. This mix of centralized and distributed scheme needs a common window for send/receive operations but the timers for retransmission are managed by individual protocol processes. The last solution, CRDA – Centralized Retransmissions, Distributed Acknowledgments, uses a central process for timer maintenance and packet retransmission. Each PP generates acknowledgments locally and sends them to a PP on the sending side.

Providing a completely decentralized solution (DRDA), like the first one shown in Figure 3.3, reduces interprocessor communication requirements on the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sending side and one expects throughput to increase. The cost associated with this solution, however, becomes evident when resequencing is performed at the receiving end. The receiving application must have enough storage buffers for incoming out of order segments. These segments must be resequenced for the receiving application. In centralized case, increased interaction among processors is required and may increase overall system latency. It may happen that central process turns out to be a bottleneck.

With the background knowledge of various issues which may impact the performance of a parallel communication system, I performed a study on workstation computers to estimate their network I/O performance. The idea behind this study was to determine various features which may be necessary for parallel communications architecture. SUN Sparcstation 1 workstations were selected for study. These workstations are typical example of current technology for implementing various subsystems like- processor, memory and network I/O. Major conclusions drawn out of this study will drive our design decision for multiprocessor workstations for parallel protocol processing.

3.4 Network I/O Performance of a Typical Workstation

I have investigated protocol processing on the Sun Sparc 1 Workstation with the Solaris 2.0 operating system in order to determine *in situ* the influence of its major hardware and software components on performance. The results are taken from actual observations using instrumentation designed to have a negligible affect on processing and timing.

Figure 3.4 illustrates the architecture of the Sparc 1 workstation[38]. The placement of SBus controller in the Figure 3.4 reflects the fact that it gives higher priority to CPU than to other devices. Basic performance features of the components in our configuration are:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

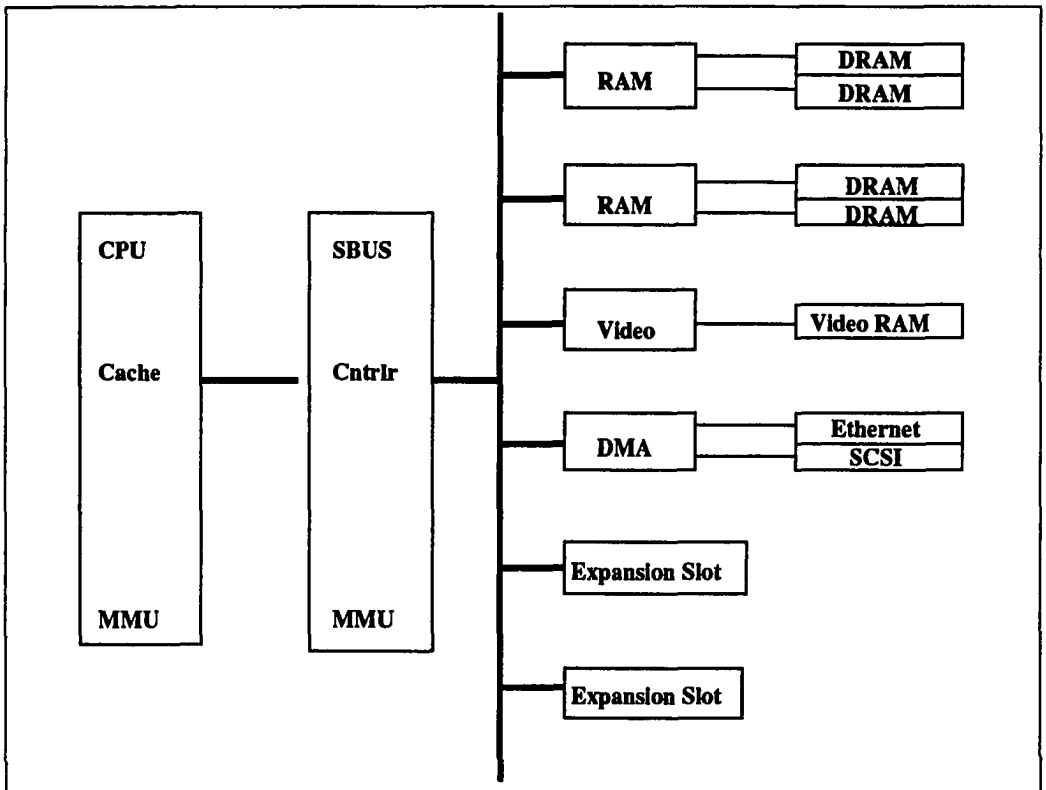

Figure 3.4: Sparc Station 1 Architecture Based on SBus

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- a 100 Mbyte/sec. backplane bus (SBus) with a 32 bit bus width;
- a 20 MHz CPU clock;
- a CPU with both integer and floating point arithmetic;
- a 64 KBytes cache for data, 32 Kbytes each for data and instructions;
- a virtual write through cache policy, i.e., writes go through the cache main memory as individual words in order to maintain main memory - cache memory consistency;
- a cache miss cost of 13 CPU cycles and the operation is blocked while the cache is being loaded.
- a main memory access time of 80 ns per long word (32 bits).

The performance numbers I develop are conservative and in most cases do not reflect best case scenario. Most of the basic numbers are either computed or borrowed from Sun white papers and performance studies [38, 83]. The SBus controller is the main interface to all the memory components and its MMU is used to map virtual into real addresses. The first major step is to evaluate the effects of basic memory operations. The workstation in our study has memory rated at 80 nanoseconds per access. However, the faster cache allows for faster execution of the programs. For repetitive execution of instructions, for the stream of instructions which do not take major jumps, and for data which lies in a block such as a packet, the CPU may hit more often in the cache and will not have to initiate a read from slower main memory. The miss cost in the cache system is 13 CPU cycles during which CPU is completely stopped and waits for the cache line to be loaded from memory. Also, costs are attached to copying of data from cache to main memory and copying between main memory segments. It takes even longer to copy between main memory and the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

I/O buffers of the network device (Ethernet in our case). In the Sparcstation 1, this latter copy is done via DMA. The DMA device steals cycles from CPU and copies the bytes to the I/O device straight from main memory. Hence, the I/O throughput of the system is limited by the throughput of the DMA. Also DMA needs to translate virtual addresses to physical addresses so that a transfer can take place. It uses the MMU for such translations. The whole process of I/O transfer becomes highly serialized in such a way that the CPU operations and the I/O transfer may not be able to occur simultaneously. To alleviate some of the bus congestion problems, Sun in the Sparc station 10 series, provides a DMA which has an additional MMU unit for I/O transfers.

The times required for memory operations are:

1. reading from I/O space = 8 memory cycles / long word;
2. writing to I/O space = 16 memory cycles / 4 long words;
3. reading from main memory = 17 memory cycles / 4 long words;
4. writing to main memory = 9 memory cycles for 4 long words (5 cycles for setup and 1 cycle per word).

Using above values I can compute the following:

1. Reading from main memory and writing to I/O space = (assuming cache miss for reads) 2.64 μsec. for 4 long words;
2. Reading from I/O space and writing to main memory = (one word at a time) 4.48 μsec. for 4 long words;
3. Memory to memory copy = 2.08 μsec. for 4 long words.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Note that these are for events which include setup time. For operations which are pipelined, the setup time can be avoided. However, from this information, I get the distinct impression that memory operations are a serious bottleneck even though the bus is capable of 100 Mbytes/sec. I conclude that memory to memory copy is 1.3 times faster than memory to I/O copy. and 2.2 times faster than I/O to memory copy. Later observations will provide more definitive information on exactly how long copy operations actually take, but I see from these numbers that transfers can be a major bottleneck in the overall performance of the system. While faster than memory to I/O copying of buffers, memory to memory copy has a major impact on the performance. In TCP/IP two copy operations take place - one from user area to kernel area and another from kernel area to I/O device buffers. If these overheads can be reduced somehow, the I/O throughput of the system can be increased considerably.

The above information provides a basic understanding of the performance of the Sparcstation 1 architecture. To determine the impact of various hardware and software components on the protocol processing, I provide a simple model of all major activities which take place. The purpose of this model is to estimate the service time a typical packet accumulates at various layers (from user application level to the network interface level) and the various OS activities that impact a packet's processing. The throughput achievable is inversely related to the service time since most operations in the workstation are highly serialized. The total service time per packet can be broken into three major components:

- Per packet service time which is fixed;
- Per byte service time which varies with the packet size;
- Operating system overheads.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To determine the cost of the transport and network layer processing (TCP/IP in our case), several experiments were performed. In these experiments, the CPU time for sending a large volume of data between two Sparcstations connected to a Ethernet was measured. This measurement includes only the cumulative time TCP/IP code is executed and do not include driver or kernel time prior to or after the TCP/IP segment. These measurements were done for a series of packet lengths (64, 128, 256, 512, and 1024 bytes). CPU time per packet is plotted against the packet length in Figure 3.5. From the results, the time of TCP/IP processing independent of packet length can be extrapolated. The Y-intercept of this plot gives us the approximate per packet service time of TCP/IP processing (independent of packet size) on a Sparcstation 1. From the plot, it can be observed to be approximately 150 μs.

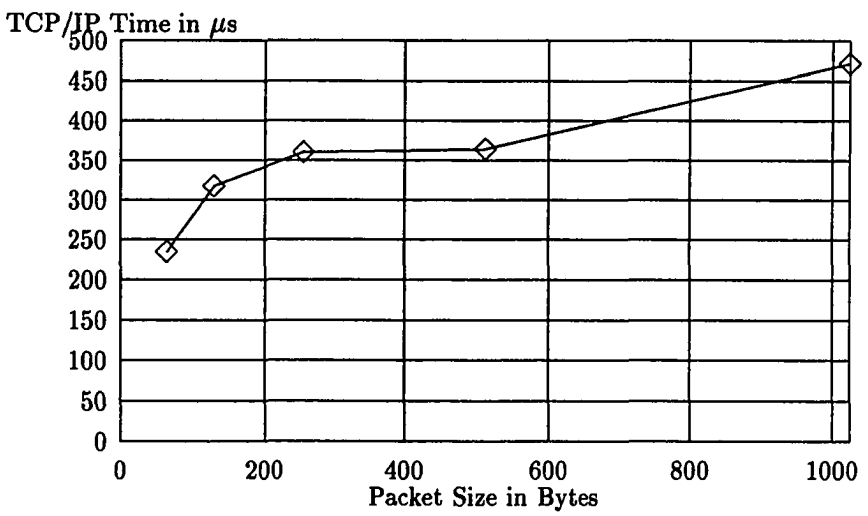

Figure 3.5: TCP/IP Processing Time as a Function of Packet Size

A Sparcstation 1 can route 6000 frames/second from one Ethernet card to another[19], where the frames are of 64 bytes length. This fact gives us the performance rating of the Ethernet processing. If the memory dependent

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

timing is removed from this rate, the service time per frame at the data link layer (802.3 in our case) is 157 μs.

Based upon the above data, the maximum I/O throughput obtainable from a Sparcstation 1 can be computed for the maximum packet size for Ethernet (1514 bytes). The frame size independent service time is 157 μs per frame. With this, the device itself can manage a 77 Mbps[2] throughput rate (with the assumption that it had a fast enough processor on board). When the service time of copying from main memory to Ethernet adaptor is added ($\approx$ 250 μs for 1514 bytes), the I/O throughput rate drops to 29 Mbps. This unavoidable copy causes the throughput rate to reduce by more than 50%.

All the transfers are done over the SBus and a delay occurs when setting up the SBus for transfers [38]. If these service times ($\approx$ 52 μs for transfer and $\approx$ 430 μs for worst case latency for setup) are added to the total time per packet, the I/O throughput drops to 14 Mbps. Hence, the DMA setup and MMU translations of the data to be transferred from main memory to I/O buffers further reduces the throughput by half. I note that the latency which has been used assumes that all slots on the SBus backplane are full since setup latency depends to some degree on SBus slot occupancy.

If TCP/IP service time is now added ($\approx$ 150 μs for every packet + 72 μs for checksum), the I/O throughput rate becomes 11 Mbps. One major observation from the computations so far is the time spent in computing checksum. Considering only TCP/IP processing time ($\approx$ 150 μs), the time for checksum ($\approx$ 72 μs) is almost 50% of the time spent in processing TCP/IP protocols. However, when considering all other operations, the checksum computation is mere 5% of the total service time per packet. Thus, removing checksum from the TCP/IP will not provide major benefits in terms of throughput. The check-

[2]A word of caution: this is not the rate which can be put on the cable since Ethernet is rated at 10 Mbps.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sum computation may be a major factor only in implementations of TCP/IP in a pipeline architecture. In Solaris 2.0, an additional copy of the bytes from user memory area to the kernel memory area occurs. The time for this memory to memory copy is 200 μs. Thus, the network I/O throughput is reduced to 9 Mbps, which is the maximum throughput rate that can be expected from a Sparc 1 operating with Solaris 2.0. If the cost of data cache miss time ($\approx$ 62 μs) is included every time a new packet is sent, this maximum will drop to a conservative value of 8.8 Mbps.

For different packet sizes, the cost of per byte times, like memory copy and checksum computation, may be reduced but other delays will gain importance since the overhead associated with smaller sized packets is large. Some of these times are buffer management processing, traps and interrupt service times, and data movement across software structures of the protocol processing.

I have dealt with the sender times in the above analysis. The receiver involves similar steps with additional overheads of context switching and interrupt handling. The difference between receiver and sender processing is that the receiver must service an additional interrupt when packets are received. Interrupts occur both when the packet is received and when DMA is completed whereas in the sender the DMA transfer interrupt is not required. However, when the packet receive interrupt is taken, there several packets may have accumulated and all are transferred at one time. Hence the overhead associated with traps and interrupt service could be reduced or increased depending on actual packet delivery.

To validate our analysis, I experimented with a Sparc 1 running Solaris 2.0. This involved altering the structure for the protocol stream. The validation step involved measuring the maximum throughput at the application layer when the communication stream is terminated at IP and when it is terminated at the Ethernet layer. As a part of the validation, the stream of data

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

were terminated at the `ip_wput()` routine of the `ip.c` module. Thus, rather than sending datagrams down to the `le` module in `ip_wput()` function, the datagrams were dropped. When the benchmark application `ttcp` was run with UDP[3] as the protocol selection, the maximum achievable throughput measured was 11.3 Mbps. Then I restored the original `ip_wput()` routine in `ip.c` and changed `le_start()` service routine in the `le.c` module. In the `le_start()` function, after the data were copied from the kernel buffers (`mblks`) to the `le` device buffers, the frames were freed rather than sent on the wire. The same `ttcp` benchmark under UDP/IP measured 8.9 Mbps. Hence, these measurements validated the I/O throughput computed with our assumptions.

Finally, I wish to comment on our results relative to the studies noted in the background section. The major factor effecting our results is that almost all actions are serial. Certainly, we [63, 65, 61] and others have shown [79, 78, 103, 50] that parallel (pipelining) operations can provide a major benefit. However, true parallelism using several devices may be difficult to actually achieve when many memory events are serialized because they must be transferred over a single backplane bus. Reducing this serialization may not be effective since serialization enables distributed memory elements to maintain consistency. Further, I observe that memory operations are, by far, the most critical factor in protocol performance and they can have the greatest impact in improving overall protocol performance as seen by the user. However, note that a major operation which can be effectively parallelized is the checksum computation. As a result of this work, we conclude that the current workstation when used as is will not be able to support high speed LANs. This means that investigation into the architectures which support the requirements of our parallel communications is needed. The subsequent section explains the fea-

[3]I selected UDP since TCP connection establishment and acknowledge phases could not operate if IP was not actually delivering packets. Please note that changes in the protocol code would not have a substantial effect on timing.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tures which will enable host architectures to process network I/O at higher rates.

3.5 Features for Parallel Communications Systems

In a conventional bus based computer system, an incoming packet is moved from lower layers into a buffer in the system memory. This is done either through the CPU or via DMA. The bus is occupied during this transfer. If DMA is used for this transfer, no CPU processing can not take place if memory access is required. Once the packet is in system memory buffers, it is examined for its headers and tail. The transport connection tables, linked lists and timers are updated and checksums are computed. During this second phase, CPU remains busy and bus is occupied to supply CPU with data and instructions. Lastly, data inside the packet from system buffers is moved to upper layers. During this memory copy operation, the CPU is blocked and bus is occupied. The reverse of these phases are true for sending packets. It is obvious from this that I cannot expect higher performance I/O with this serialized computer system operations. I need to redesign the system architecture such that I can use parallel processing to provide each executing task with individual access to memory. Also, there should be concurrency supported in various system transfers/executions. Network devices and DMA controllers may also need to be redesigned[83]. In this section, various features which are considered important for computer systems and workstations of high speed LANs are described. I have classified them into architectural and protocol related features.

1. **Architectural features:** These features involve the host, its processing and network architecture.

   (a) Multiple CPUs to support parallel execution of tasks. These CPUs may be connected to each to each other through a communication

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

fabric. A valid choice for this communication fabric is a high speed bus interconnect. This bus will connect all CPUs and memory (if shared) together.

(b) The bus should support a high performance cache coherency between multiprocessor caches.

(c) Memory can be local per CPU or shared among CPUs or both. In either case, a better and efficient memory architecture featuring a wider (say 128 bit or more) memory may be required. The memory management should be such that the maximum bandwidth can achieved from the memory subsystem.

(d) The DMA architecture for I/O should be redesigned. The DMA controller should be designed in a way that CPU can continue to process while DMA is transferring data from or to memory. Use of multiported memories is recommended for such purposes. DMA should have a capability to handle linked lists because packets in system memory are arranged as linked list of small buffers.

(e) The cost of context switching should be negligible. The memory paging architecture should allow for wiring of certain pages of memory to prevent swapping and flushing of memory pages. This prevention of swapping can greatly improve the system performance.

(f) Symmetric multiprocessing must be used to allow for independent and concurrent execution of various activities by multiple CPUs.

(g) Use of multiple channels is imperative in the realization of parallelism at channel level. The architecture should support multiple network interfaces, but the decision about the number of channels should be based on target applications. Also a choice has to be made between having a few fast channels (for example, 100 Mb/s FDDI) or many slow channels(for example, 10 Mb/s Ethernets).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. Protocol related features are:

   (a) Efficient buffer management techniques should be used for allocation and deallocation of system buffers.

   (b) Checksum calculation involves all bytes of the packets to be addressed, but could be computed using simple 1's or 2's complement adders and XORs.

   (c) The multiple timers are managed as linked lists in memory. This may involve extensive linked lists search. Using Hash on active timers connection identifier should improve the performance of timer management. It is recommended to use a single timer per connection.

   (d) Acknowledgment and retransmissions handling is one of the most important issues which differentiates one implementation from another. Here the choice is between a distributed mode and a centralized mode. The distributed mode assumes that each sending and receiving transport protocol process pair maintains separate data streams and each performs its own acknowledgment and retransmission processing. A centralized mode assumes that any transmitting process can send to any receiving process and that acknowledgments and retransmissions are handled by separate processes.

   (e) The transport window size value which will provide the maximum throughput is another issue. Very large windows indicate a large number of buffers holding the data for no considerable gain in throughput. Very small window can reduce throughput to very low levels.

   (f) The size of the application data segments submitted to the protocol processes will also effect the system performance in terms of longer end-to-end delays for larger segments. Very small segment sizes will increase the protocol overhead and hence lower throughput.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(g) Different scheduling policies may have different impact under various channel load conditions. A good policy which efficiently distributes the stream of data should be selected. An adaptive scheduler policy may be more effective as compared to simpler policies (say FCFS, RR) under non-uniform conditions.

(h) To an application, the services provided by the underlying parallel communication system are of primary importance. Service types include real-time response, reliable out-of-order delivery, and reliable in-order delivery of data.

These features can serve as important guidelines for future host system design for high speed LANs. Already a great deal of work has started in improving memory management techniques[67]. Alternatives are being explored to maximize the memory bandwidth[23]. High speed bus interconnects like MBus from Sun Microsystem [91] have been realized. Network interface with better capabilities are being developed [83]. It is extremely important to consider these features as essential to the host architectures of tomorrow. The performance study discussed here was performed on several architectural instantiations that assumed some of these features. These instantiations and the simulator model are discussed in the next chapter.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

Multiprocessor Instantiations

In the parallel network model, multiple levels of parallelism are considered as shown in figure 3.2. Parallelism is employed at the transport and network layers (upper level), and data link and physical layers (lower level). The strategy employed at each level will influence the performance, reliability and cost of the entire system. Performance studies are necessary to demonstrate the correct operation of each level and to evaluate alternative approaches for each level. In addition, models and results for each level must be integrated to permit end to end, i.e. application-to-peer application, evaluations of the most promising approaches. The end-to-end studies will determine overall performance and verify the interface requirements between parallel segments.

Parallelism can be used to significant advantage at the upper layers to provide high performance implementations of transport and network layer protocols. This resolves a problem common to many networks namely, that upper layer protocols are at least as restrictive to good performance as the lower media access and physical levels. Studies in [95] have indicated the performance limitations of TCP/IP operations in Ethernets networks. Even though the speed of TCP/IP has been improved [17], the demands placed on the upper layers will only increase as physical data rates and throughput reach the multiple hundred Mb/s range and beyond.

In this chapter, multiprocessor instantiations of a parallel network model are presented. The architectures presented in this chapter are designed primar-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ily on the various features presented in the previous chapter. The instantiations are based on the concept of multilevel parallelism. Mapping of model processes to the physical resources in every architectures has been done. These machines are of interest because they present different memory architectures, network scheduler locations, network device management, and bus hierarchies. Within the framework of current technology, the instantiations are restricted to bus based architectures. System modeling and simulation is the central tool which will be used to evaluate the performance of parallel network system. Each of the presented architecture are modeled. Such models will provide useful insight into the operation of parallel networks. The end-to-end studies are used to verify overall system performance, to evaluate tradeoff relating to the distribution of functionality among layers, and to observe interactions between parallel channels. A detailed description of simulation model is presented in this chapter. Parameters and metrics of interest are also defined.

4.1 Instantiations

Our conceptual model for parallel protocol processing represents processing functions as cooperating processes. To instantiate this model on a given parallel hardware architecture, one needs to describe the facilities for interprocess communication, memory architecture, and the location, number and types of schedulers, and to map the application and the protocol processes onto the different processors.

As a first step towards instantiation of the parallel system on these architectures, one needs to define the degrees of parallelism that need to be observed at various protocol levels. For our study we chose to apply parallelism at transport, network and media access layers, since these layers are major components of a communication system. We selected the Internet suite of protocols, TCP and IP, for transport and network layers respectively and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

FDDI for the media access layer. An application process is assumed which needs to send huge amounts of data to another application at the receiver end. Various processes identified in the previous chapter were ported to the physical resources in the platforms discussed in this chapter. For convenience of the reader, a brief description of the system model is repeated below again.

The application process (AP) produces the *segments* of data which are to be passed on as a complete segment to the receiver in the correct order. An application scheduler (AS) running at a logical level below AP accepts these segments (coming as one data stream from the sender application) and schedules them over multiple TCP connections which exist between the sender and receiver hosts. It should be noted that such parallel TCP connections between protocol processes on both sides have to be in place before data transfer is initiated. Such connection initiation phase is a pre-consulted phase between protocol processes and, once established its cost is amortized over duration of the data transfers). The processes which handle TCP connections between the peer ends are called protocol processes. These processes do the TCP (at the transport layer) and IP (at the network layer) protocol processing on the data segments, send/receive window management, timer management, TCP checksum, and IP header checksum computation. IP fragmentation is avoided at this level of study by assuming the maximum segment size (MSS of TCP) to be less than MTU (Maximum transfer Unit- 4500 bytes) of FDDI. Since there will be at least one connection existing between each sender process and receiver process, there will be a stream of data packets coming out of the each sender process. Each protocol process may run independently on a separate processor called protocol processor (PP). Since there will be a one to one mapping between protocol processes and processors, acronym PP will be used for protocol processing and processor both. With the availability of multiple channels, the problem of assignment of TCP packets to correctly load balance the physical channels is very critical. This assignment is achieved through scheduling at

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

or below the network layer. This is called network scheduling (NS). Once transmitted on the physical channels, the data packets arriving at the receiver FDDI are passed over to one of the receiving PPs. Based on connection identification, a specific PP can be chosen to handle these incoming packets. After completion of TCP and IP processing of the arriving packets, they are passed on to the receiver application scheduler once a segment is completely assembled. Such arriving segments are passed by the receiver application scheduler to the receiving application in the same order as they are sent by the sender.

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. We describe three bus-based instantiations to realize the proposed conceptual model for parallel protocol processing. The instantiations have the following common assumptions:

1. The bus acts as the basic communication fabric for interprocessor communication. Bus is chosen as the interconnect medium for the processors to reflect the trend in existing multiprocessor workstation development. It should be noted here that bus based architectures have a limited scalability in terms of number of processors that can be connected.

2. Each of the these multiprocessor architectures incorporates symmetrical multiprocessing.

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

4. Each protocol process (PP) is located in an independent protocol processor.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5. Each PP does TCP and IP processing. We assume that the application requires reliable and ordered stream oriented data service.

6. To allocate the segments generated by the application process to the PPs, we employ an AS located in the processor same as AP.

7. NS is located either in each protocol processor, or in a special switching device.

8. FDDI is the network interface considered in these architectures.

9. An overlap is assumed between the processor execution and memory access such that a pipeline is created. This improvement in bus and memory architectures can insure continuous execution of instructions by a processor without having to wait for the future memory accesses to complete in entirety.

10. Use of an advanced memory management unit (MMU) can ensure that the memory transfer rate will be one word per memory cycle. Also, the ability to share the regions of memory between processes as regions of virtual memory can improve sharing of the data.

4.2 Architecture 1

Figure 4.1 shows a two-bus instantiation of the multilevel parallel communications model. The two buses are referred to as the $ABUS$ (or the application bus) and the $NBUS$ (or the network bus). Multiple independent CPUs are connected between the $ABUS$ and $NBUS$ and each executes an independent copy of TCP/IP. Each CPU is assumed to have sufficient local memory to protocol processing as well as data storage functions. The inter-processor communication takes place by copying buffers from one CPU's memory to another. This copying takes place over the $ABUS$. The $FDDI$ devices are connected

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

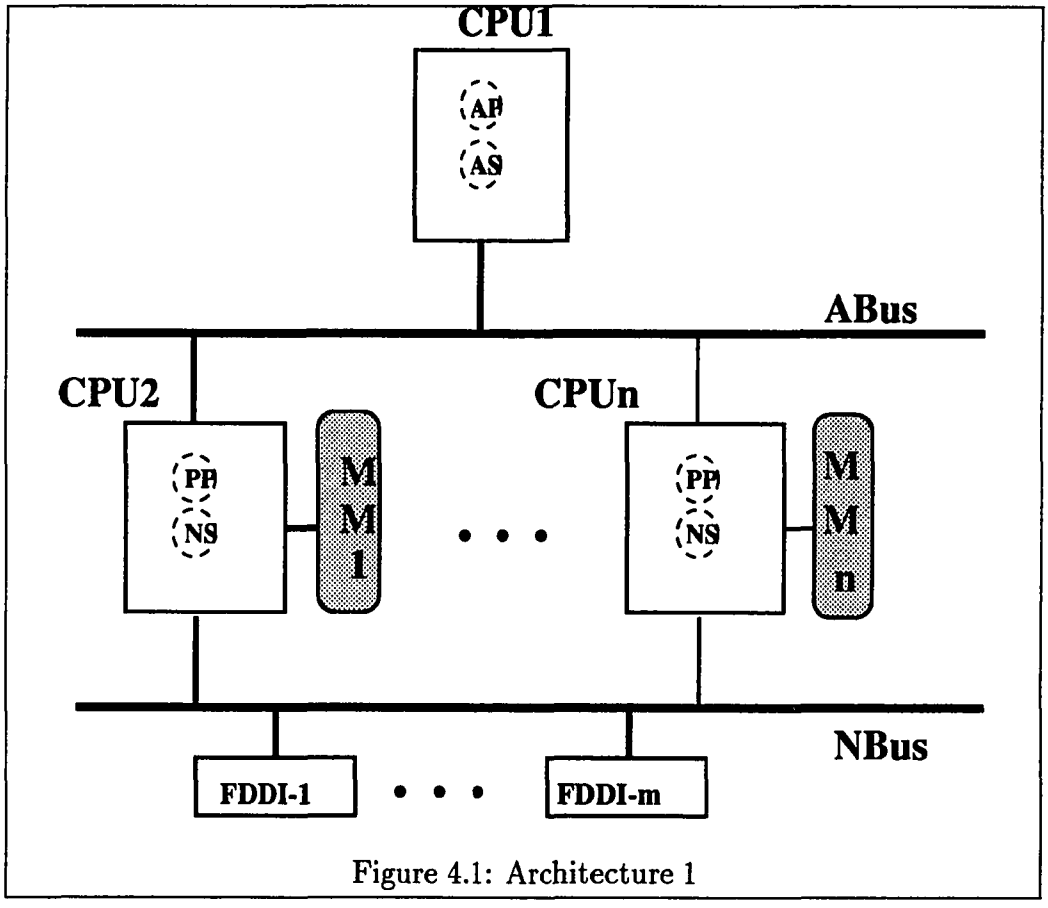

Figure 4.1: Architecture 1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

to the $NBUS$ which is the I/O bus in this architecture. All packets are copied over the $NBUS$ to the $FDDI$ buffers. The main objective in selecting this architecture is to determine the impact of distributed network scheduling and memory on the overall performance of the system.

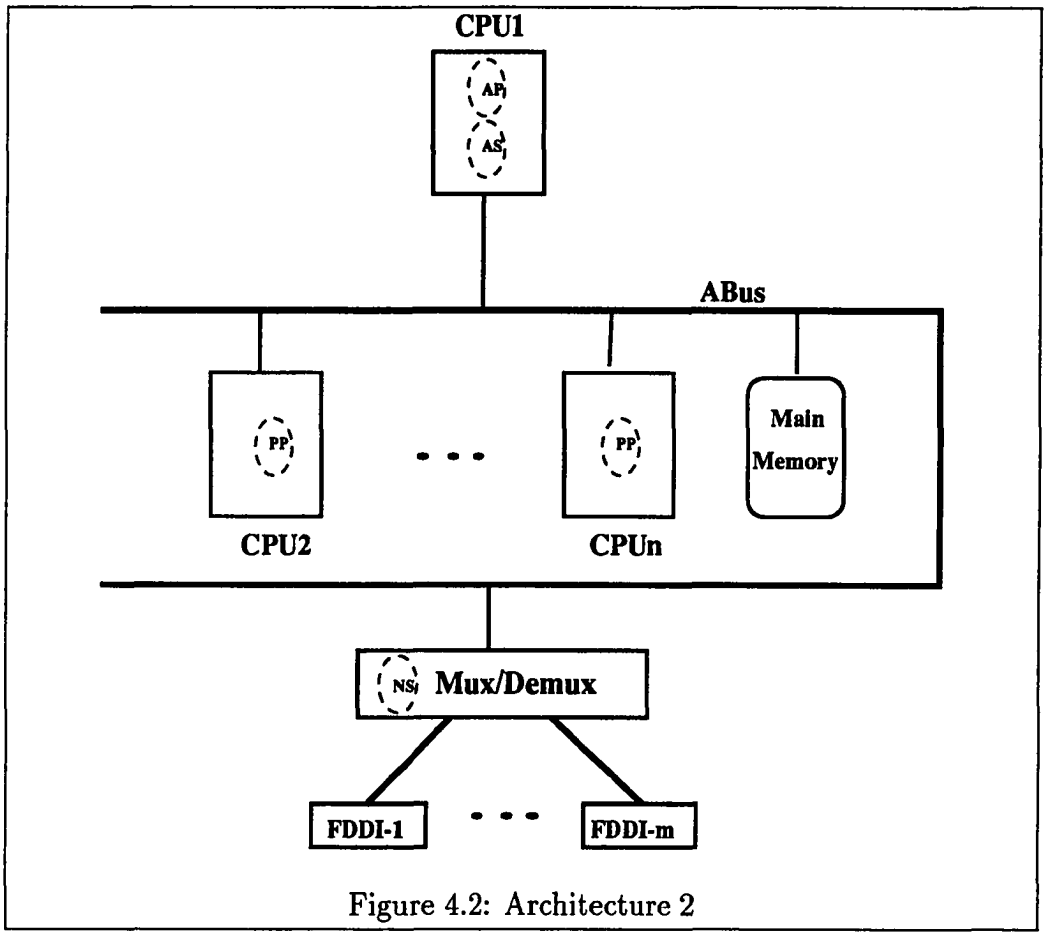

Figure 4.2: Architecture 2

4.3 Architecture 2

Figure 4.2 presents an instantiation based on Sun Microsystem's MBus interconnect. This architecture is selected to determine the effect of global memory and central network scheduling on the performance. Each processor gets its instructions and data from the shared memory. Since many independent pro-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

cessors get their data and text from one shared memory, the memory contention problem can be solved temporarily with faster memories and a wider memory bus. This architecture employs a special switching device to switch streams of data packets coming over the bus to the network interfaces and vice-versa. This switch is called multiplexor/demultiplexor (or Mux/Demux). The outbound packets over the bus are demultiplexed onto available FDDI devices and inbound packets are multiplexed into one stream and put into the shared memory. The processors later process these arrived packets. This device acts as the centralized device performing network scheduling. In future, the Mux/Demux device can also provide forward error correction.

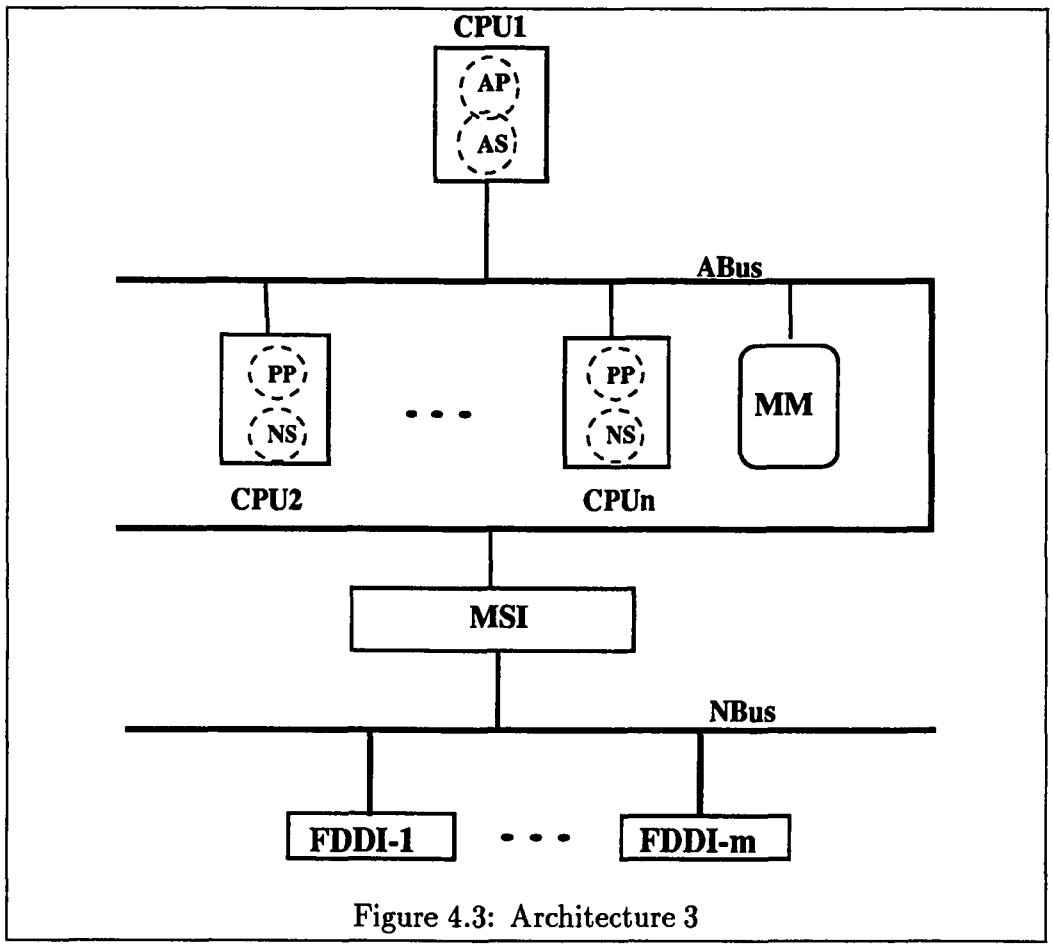

Figure 4.3: Architecture 3

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.4 Architecture 3

Figure 4.3 presents an architecture based on Sun Sparc-10 Multiprocessor, a four CPU shared memory machine. This architecture employs a split bus architecture, i.e. separate memory and I/O buses. Currently, FDDI devices hang off a bus slower than *ABUS*. This bus is called *NBUS* and an interface unit,the MSI, exists between these two buses. The MSI interface between memory and I/O bus has its own memory management unit to perform address translations whenever transfers are taking place over the *NBUS*. The motivation behind selecting this architecture is to test the performance of the parallel communications on an existing architecture. Another objective is to compare the performance of two major acknowledgment schemes - distributed (*DRDA*) and centralized (*CRCA*).

Feature	Architecture 1	Architecture 2	Architecture 3
Number of Bus	2 (ABus & NBus)	1 (ABus)	2 (ABus & NBus)
ABus Speed	800 Mb/s	2.4 Gb/s	800 Mb/s
NBus Speed	800 Mb/s	-	800 Mb/s
Memory	Local per CPU	Shared	Shared
FDDI Interconnect	NBus	Mux/Demux	NBus
Number of APs	1	1	1
I-Cache Hit Rate	High	High	High
D-Cache Hit Rate	Low	Low	Low
AS Location	CPU1	CPU1	CPU1
NS Location	CPU1..CPUn	Mux/Demux	CPU1..CPUn

Table 4.1: Major architectural characteristics of architectures

Table 4.1 lists the major characteristics of the three architecture instantiations of the parallel communications framework. Major differences lie in the location of the schedulers and the FDDI interconnect used in each of the architectures. Also, use of shared versus local memory is another difference in these architectures. The peak rates listed for various buses are taken from Sun

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Microsystem's open standard bus interconnect documents. An 800 Mb/s SBus is available today and we have assumed it to be the communication fabric in architecture 1. In architecture 2 and 3, a recent high speed bus interconnect announced by Sun Microsystem has been assumed. Called MBus, this bus is rated at 2.4 Gb/s peak throughput. The features listed in Table 4.1 are the ones that remain fixed during the entire performance evaluation. Other parameters were varied and are discussed later in this chapter.

Simulation models were developed to analyze these instantiations and to determine the comparative performance of the alternative approaches to multilevel parallelism at the lower four ISO OSI model layers. The main goal of this study is to obtain an understanding of the design tradeoff and the manner in which they effect the network performance. Modeling both, within node and network wide, behaviors is performed. To achieve this goal, a simulation model developed initially at MIT [45] was adapted according to our design needs. A description of the simulation model is provided in this second half of this chapter.

4.5 Model and Simulation

This section is an overview of the operation of the parallel network simulator program and the description of various components in the simulator. The simulator can simulate anything that can be modeled by a network of components that send messages to one another. The program provides the means to load the network configuration from an input file and save the results to an output file.

The components schedule events for one another to cause things to happen. The model being simulated and the action of the components is entirely determined by the code controlling the components, not by the framework of the simulator. The simulator itself only provides the means to schedule events

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and to communicate with the user.

The rest of the simulator consists of the event manager, the I/O routines, and various tools (lists, queues, hash tables, etc.) that can be used to build network components.

4.5.1 Components

The *component* is the basic building block of the simulator. A component consists of a data structure and an action routine (a C function). There are different types of components (for example, FDDI, memory, bus, protocol processor in the current implementation). All components of the same type share the same action routine. This routine is called for each event that happens to a component. Each instance of a component maintains a data structure that is used to store any information needed by the components, as well as a collection of standard information needed by the simulator for every component.

4.5.2 Types of Components

A component has a type. A particular type of component can contain many different instances of component. For example, there can be many FDDI interfaces in a host. To create a new type of component, a new action routine must be written and a new data structure for the component must be defined.

1. **Application Process (AP)**: The AP at the sender end generates data segments at an exponential or uniform rate based on the throughput required. It informs AS of the availability of segments ready to be sent. The AP at the receiver end passively receives the data segments. A representative example of this kind of communication is data transfer from a high performance computer to a graphics display computer such as is needed for remote scientific visualization.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. **Application Scheduler (AS)**: At sender end, the AS receives events from AP indicating that segments are ready to go and from TCP acknowledging the successful receipt of the segments. Incoming segments from the AP are scheduled to various TCP/IP PPs according to the FCFS policy. The segments are assigned to the PPs according to the order in which they return acknowledgments for the receipt of the segments. End-to-end throughput and end-to-end delay are important metrics measured at this level. At the receiver end, the AS receives segments from the receiver PPs and passes them in the correct order (same as transmit order) to the receiver AP.

3. **Bus**: The bus in any of the architectures is modeled as a first come first served queue of bust transfer requests. A request for bus transfer is queued at the end of the input queue of the bus and, as the bus gets serviced, the request comes to the head of the queue. The queuing delay involved can be taken as a measure of the system performance. To take care of transfers which are not related to the network communications, we arbitrarily assume that, on an average basis 30% of the bus capacity is background transfers. Bus utilization and delay in bus transfers are important measures of the system state.

4. **Memory**: In shared memory architectures, architectures 2 and 3, memory transfer requests are modeled as a stream of requests (read or write) which are handled in first come first served order. The requests are queued and may suffer queueing delay until serviced. Access time per word defines the memory speed and controls the time spent by a request in the memory. The number of times memory references are done per transmitted packet can be a good measure of the effective memory bandwidth required.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In architecture 1, local memory in PP is assumed. That is, a PP can access memory locally over a private bus (not modeled) rather than going over common bus. Any transfer of packets from one processor to another will involve copying from local memory of one to another over the bus.

5. **Protocol Processes (PP) :** TCP/IP protocol processing is the most significant part of every simulation. For TCP protocol processing, time for send or receive processing, checksum computation, and memory read/write of headers constitute important activities. The send and receive window management, timer controls, and retransmissions and acknowledgments constitute important functionalities of send and receive operations. In our models, we have avoided IP fragmentation and reassembly by selecting maximum TCP packet size such that it can fit in 4500 bytes of FDDI frame. The routing control is passed on to the NS. Hence, IP protocol processing for send and receive signifies IP header processing. Processor utilization, TCP end to end delay per TCP packet, TCP throughput per PP, and percentage bandwidth lost in retransmissions are important metrics of interest.

   The NS has different locations in different architectures and can implement different policies for scheduling of TCP packets to various FDDIs. In architecture 2, it is logically placed centralized with respect to PP and FDDI interfaces. In the remaining architectures, network scheduling is distributed as part of the IP processing. Based on the load on various FDDI channels, the NS may select a from round robin or adaptive scheduling policies. The adaptive scheduling is based on transmit queue length and token rotation time. It schedules next TCP packet to an FDDI interface which has minimum values for the above defined parameters.

6. **FDDI**: The NIU is modeled as an FDDI interface connected to a fiber-optic ring with 20 nodes evenly distributed along a length of 20 kms.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Among these nodes, two are designated as the sender and receiver for our experiment. Since we assume the high-bandwidth requiring nodes coexist with low-bandwidth nodes in the network, we simulated the latter traffic by introducing 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 balanced and unbalanced cases of background traffic. Also the background traffic on a ring is not delivered to either the parallel sender and parallel receiver; these nodes only process data belonging to the AP. If we assume, for example, 30% of total capacity of three FDDI channels as background traffic, then the available channel capacity will be approximately 3*(100-30-token rotation loss) or approximately 120 - 135 Mbps. Since there are many channels simulated, the combined effect of the (balanced or unbalanced) load and network scheduling policy on these channels will determine the effective usage of the network capacities. The network utilizations and transmit queueing delay are important state variables. The FDDIs are simulated in their simplest forms. Only token arrival, data arrival, data send, and token release in a non-exhaustive environment are simulated[1].

7. **Mux/Demux:** The multiplexor/demultiplexor device in the architecture 2 schedules packets from PPs to NIUs in both round robin and first-come-first-served fashion, based on the option selected. One future use of this device can be to encode data packets and pad them with extra bits to enable complete reconstruction of data at the receiver when employing the cross channel coding [100].

[1]Non-exhaustive means only one FDDI frame is transmitted even if the token holding time allows for more.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.5.3 Action Routine

When an event for a component fires, the component's action routine is called. Components can send any type of events to one another. However, in order to a allow the simulator to do various housekeeping functions, every action routine must respond to a fixed set of commands. The action routine is called usually *not* by the event manager, but rather directly by the simulator, to perform these commands. A synopsis of the action routine and the commands is as follows:

```
/** All of these include files may not be needed, but they are the
    common ones.  */
#include <sys/types.h>
#include <stdio.h>
#include "sim.h"
#include "log.h"
#include "q.h"
#include "list.h"
#include "comptypes.h"  /* The types of components */
#include "packet.h"
#include "eventdefs.h"  /* Types of events & commands defined here */
#include "event.h"
#include "this_component_type.h"


caddr_t
action(src, comp, type, pkt, arg)
Component *src;   /* Component that sent this event.  Null for cmds. */
Component *comp;  /* Component to which this event/cmd applies. */
int type;         /* Type of event or cmd that is happening. */
Packet *pkt;      /* A packet. */
caddr_t arg;      /* Whatever */
{
       /* Usually a big switch statement on the event type */
}
```

All components accept the following commands:

1. **EV_CREATE**: (Create a new instance of a component) The action routine allocates the correct amount of memory for the new component's

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

data structure, creates its (empty) neighbor list, and creates any packet queues. This command must also initialize all the private data in the component.

2. **EV_RESET**: (Reset the state of the component) The action routine clears out any packet queues and deletes any packets being processed.

3. **EV_START** : (Start simulation going) The action routine must start the simulation after receipt of this command. For example, the *AP* will start generating segments. For many components, this is a no-op.

4. **EV_NEIGHBOR**: (Attach another component as a new neighbor) A component allows only legal neighbors to added in its neighbor lists. For example, in architecture 2 (and 3) memory is a legal neighbor of bus and not the PPs. This definition of a neighbor is important for passing events between the components.

5. **EV_STOP**: (Stop the simulation) A component prints the statistics it has gathered during the simulation run to an output file. This command is sent to all components when the simulation time is over.

Currently, there are three classes of event:

- commands (such as EV_CREATE);
- regular events (such as EV_RECEIVE); and
- private events.

Private events are meant for events that components send to themselves. For example, the timeout events for the PPs. Regular events are ones that cause the system simulation to proceed and are passed from one component to its neighbors. For example, receive event for a FDDI component signifies arrival of a packet for transmission on fiber.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.5.4 Packets

Since the simulator is designed to simulate packet-switched networks, a packet data type has been defined. A packet is merely a data structure. An event may include a packet and most events that have to do with the simulation (as opposed to housekeeping commands) do. In addition, there are modules to handle the allocation and deallocation of packets. These modules keep track of all the packets, so that when the simulator is reset all packets can be freed in one step.

The packet data structure is not constrained to be any particular format. A packet can contain any data. In the current implementation, the packet structure consists of a group of variables used by the components to send packet through the network, and structures for the TCP connection. The same packet structure is used when the *AP* generates segments and passes them to the *AS* for scheduling.

4.5.5 Event manager

The simulator is event-driven. Components send each other events to communicate and to send packets through the network. The event manager provides a general facility to schedule and send events. The simulator time is maintained by the event manager in units of "ticks". Currently, tick resolution is ten nanoseconds. The only other event-related function that a component needs to know about is `ev_enqueue()`. This feature creates a new event and places it on the event queue to be fired at the proper time. One may schedule events at the current time or at any time in the future. Events scheduled at the same time are not guaranteed to fire in any particular order.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.5.6 Major Assumptions

The simulation models have a number of assumptions. These assumptions are needed to avoid the unnecessary complications in the modeling process. These assumptions represent the actual scenarios in the system and do not oversimplify the entire process of performance study. The major assumptions are:

1. The AP and AS process run alternately on a processor. That is, when one process becomes idle for some reason (e.g. not enough buffers to send), the other process is switched to run. Sometimes both the processes may be idle and not running. No context switching overheads are accounted for switching the idle processes to run. This is assumed because the modern processors have more than two hardware contexts built into them to reduce the context switching costs.

2. The AP at the sender's end is the source of the continuous stream of data segments. The AP at the receiver end is the sink of this stream of segments.

3. Every bus has a maximum peak throughput rate, but a percentage of its peak capacity is assumed to be consumed in other bus activities like bus arbitration and setup for transfers. In our simulations, almost one third of the bus capacity is assumed to be lost for such transfers.

4. The PPs are assumed to be independent of other processing and related overheads. This assumption is important in knowing the performance of the whole system without complicating the model. Also, the instruction cache hit rate is assumed to be hundred percent for protocol processing. This assumption holds because all modern day processors are available

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

with onboard instruction cache (mostly of the order of 64 KBytes) meaning that the instructions needed to do TCP/IP can easily be cached in instruction cache for the entire protocol processing.

5. The environment for modeling is assumed to be local area networks. This assumption eliminated the need for modeling the routing and fragmentation issues which are very common in wide area networks.

4.5.7 Parameters of Interest

Table 4.2 illustrates various hardware and software parameters important for simulation and the corresponding values adopted in our experiments. The parameters in Table 4.2 are classified into three categories; architecture-related, scheduling-related, and protocol-related parameters. While the architecture related parameter values chosen here represent a sample of reasonable values in the current technology. The values in the protocol section represent range of values that are used to obtain improved performance through experimentation. Lower end-to-end delay and higher throughput can be a measure of a good performance.

The time to execute TCP/IP related code on 20 MHz Sparc Station 1 was measured to be 150 μs per packet. This measurement has been reported in chapter 3. Since same TCP/IP code is assumed to be executed on platforms with different CPU speeds, the corresponding time can be easily computed (e.g. 75 μs for 40 MHz CPU speed and 300 μs for 10 MHz CPU speed). The bit error probability is also translated to the packet error rate. For example, 10^{-6} bit error probability can be easily used to evaluate probability of packet in error. For a 4500 bytes packet, it amounts to approximately 2%. The remaining parameters and their importance has been discussed in great length in chapter 3. The motivation behind selecting these parameters is to stick to major issues and options envisioned in parallel communications system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Selected Architectural Related Parameters	Range of Values
Number of PPs	1 through 8
Number of FDDIs	1 through 8
CPU Speed in MHz	10 through 80 Mhz
Memory Speed in nsec/word	20 through 80 nsec/word
Background Load on Bus	30% of Capacity
Selected Scheduling Related Parameters	**Range of Values**
Location of NS	Centralized or Distributed
Scheduling Policies	RR or Adaptive
Selected Protocol Related Parameters	**Range of Values**
TCP Window Size	4500x(1 through 10) bytes
Application Segment Size	450, 4500, and 9000 bytes
TCP MSS Size	4500 bytes
Background Load on Channels	Balanced or Unbalanced
Application Data Arrival	Exponential or Uniform
TCP/IP Processing Time on 20 MHz CPU	150 microsec.
Bit Error Rate on Channels	No Error or 10.0e-6

Table 4.2: Table of simulated parameters

4.5.8 Selected Metrics for Performance Measurements

Before we proceed with the discussion of the performance results, it is useful to define the metrics of performance comparison. The purpose for selecting these metrics is to determine the feasibility of the proposed architectures in achieving the desired performance levels. We are interested in determining various tradeoff and gaining better performance levels. The major metrics are:

1. **End-to-end Throughput:** This is the measure of the rate of data transfer which an application can achieve when communicating with another application over a network. The end-to-end measure adds the guarantee that the achieved throughput includes the delivered bits to the receiver only.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. **End-to-end Delay:** This is the round-trip time of a data segment. This time includes the time for sending and receiving both data and acknowledgment. This is measured at the AP.

3. **Wait Time at AS:** This is the time a segment of data waits in the input queue of AS before getting assigned to one of the PPs. This wait time is a measure of the effectiveness of the AS.

4. **Wait time at FDDI:** Every packet arriving at FDDI interface goes into a queue of buffers. When the token arrives, the packet at the head of this queue is transmitted and token is released. This results in a wait time for the packets waiting in the queue. Measurement of this wait time can be a good measure of efficiency of the NS.

5. **Number of Timeout and Retransmissions:** In case of errors or packet loss, timeout and retransmissions have been used as a recovery mechanism in TCP. There can be large number of such retransmissions if channels lose packets or acknowledgments get delayed due to heavy load on channels. These values can be a good indication of how effectively the scheduling policies adapt to the network conditions.

6. **Reorder Delay and Reorder Queue Size:** This represents the delay caused by the out-of-sequence reception of TCP packets. It also includes the delay involved in propagating received and assembled segments to the application in the same order in which they were actually sent. A segment is considered complete and assembled once all packets constituting it are received. A received segment is buffered at the AS until all segments prior to it are received. The delay involved is the time a segment has to remain buffered owing to prior segments not being received yet.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

Results and Measurements

Relating to performance, two key issues in parallel networking are the decomposition of work, specifically data streams and processing tasks, and an effective distribution of this work among the parallel elements. Embedded in these issues are such factors as:

1. The protocol data unit or packet size that is to be processed at each level;
2. Effective methods and techniques for controlling the distribution of data to be transmitted based on available resources and input demands;
3. Distribution of workload between processors;
4. Use of alternative distribution of workload among processing elements in the event of load changes and failures;
5. Recombination and redistribution of data and processing activities if other sets of parallel elements exist;
6. Effective and/or special handling of particular services as identified by various data types or uses; and
7. Use of alternative policies to distribute packets over multiple channels.

Of these issues indigenous to parallel network systems, the most significant ones are related to the distribution of protocol processing workload among

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

multiple PPs. Channel assignment strategy can be based on dynamic or static operations, choice of strategy, and the ability to perform load balancing.

Simulation models have been developed to evaluate parallel network instantiations. These instantiations and their major characteristics are discussed in chapter 4. These models were implemented to gain insight into the applicability of currently available architectures to the parallel network framework. A two step methodology is adopted to achieve this objective of evaluating the performance of these architectures and study the issues concerning parallel networks. Initially simulation models are used to estimate the bounds on the performance of the multiprocessor architectures. Impact of various issues is also studied. Secondly, a multi-channel prototype is explored. This prototype only integrates at the lower level, that is, media and physical layers. Prototyping upper level parallelism at transport/network layers requires development of more complicated software and availability of multiprocessor architectures. Therefore, only prototyping effort at the lower level is reported. Lower level prototyping is used to study network scheduling in the multiple Ethernets. The most important factor in any simulation study is verification and validation. A discussion of this process is presented.

5.1 Simulation : Verification and Validation

The goodness of any simulation model is measured by the closeness of the model output to that of the real system. Since a number of assumptions concerning the behavior of the system were made when developing the model, two steps are employed to determine the goodness of the model. The first step is **verification**. Verification can also be called debugging. That is, ensuring that the model does what it is intended to do. A number of techniques can be used for debugging. As a first step to avoid bugs, modular programming techniques are employed. The modules have well defined interfaces and they

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

communicate with each other through these interfaces. Modularity thus allows the verification of the model to be broken into smaller problems of verifying modules and interfaces. Additional checks in the outputs in the programs are used to identify bugs. Counts of packets are checked from the source to the destination such that total number of packets sent is same as total number of packets received, and that they match the total number in the system. Event traces, procedure traces, and variable traces are used for additional verification.

The second step is **model validation.** Validation refers to ensuring that the assumptions used in developing the model are reasonable. Mostly the assumptions, input, and output values are validated using expert intuition. Some real system measurements are also used to validate the model. We used measurements to validate our models as follows. As stated earlier, a typical uniprocessor workstation can generate around 9 Mb/s network I/O with existing 80 ns/word memory subsystem and Ethernet network. A similar memory processing subsystem is modeled as an alternative using our simulator. The simulation output results show 8.5 Mb/s as end-to-end throughput achievable by an application. This way first validation of our model is achieved and successful approximation of the system processing is confirmed in terms of measured throughput.

As a next step towards validation, heuristic methods are employed to remove transients from the simulation results. Long runs are employed to ensure that the presence of initial conditions does not affect the results. Also, initial data is not considered in determining the overall averages. This is done to avoid introducing the transients into the averages. Once the system is in steady state the averages will not change much as the observations are thrown. A method of **batch mean** has been used to study the variance and transient removal.

In batch mean method, the simulation run is divided into several parts

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of equal durations. Each such part is called **batch**. The mean of observations in each batch is called **batch mean**. This method studies the variance of the batch means as a function of batch size. The method used is as follows:

1. For each batch compute batch mean.

$$\overline{x_i} = \frac{1}{n}\sum_{j=1}^{n} x_{ij}$$

where,

j = 1,...,m

n = batch size

2. Compute overall mean.

$$\overline{\overline{x}} = \frac{1}{m}\sum_{i=1}^{m} \overline{x_i}$$

where,

m = $\lfloor N/n \rfloor$ number of batches

N = Number of observations

3. Compute the variance of the batch means.

$$Var(\overline{x}) = \frac{1}{m-1}\sum_{i=1}^{m} (\overline{x_i} - \overline{\overline{x}})^2$$

Increase n and repeat steps 1 and 3 for n = 3, 4, 5, ...

Then variance is plotted as a function of the batch size. The length of the transient interval is the value of batch size at which the variance definitely starts decreasing. Once such plot has been illustrated in Figure 5.1. For architecture 3, the count of packets (TCP) arriving at the MSI interface (see Figure 4.3)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

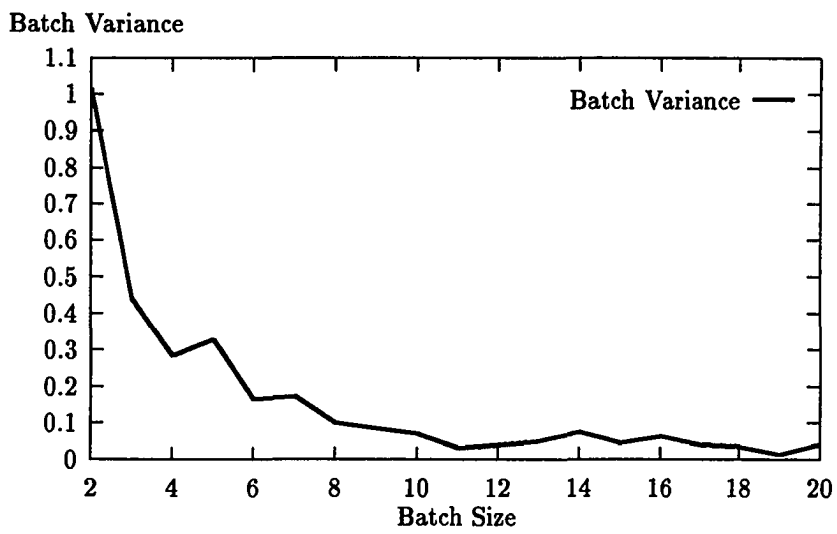

Figure 5.1: Batch variance as a function of batch size

are taken for 100 equal durations of the simulation. The batch mean procedure of elimination of the transients is employed. As illustrated in the plot, after the batch size of 10 the variance changes very little. The rationale behind this method is as follows[51]:

Suppose the length of the transient period is T. If the batch size n is much less than the T, initial batches bring the overall mean towards the initial batch mean and the variance is small. As the batch size is increased, the variance increases. At n larger than T, only the first batch mean is different, other batch means are approximately equal. This results in decrease of the variance. Note that in using this method, one should ignore the peaks on the variance curve that are followed by an upswing. From Figure 5.1, the initial increase of variance with respect to small batch size is not visible. Only the falling part of the variance curve is visible. This may be due to the fact that the initial transient period is much smaller than the one hundredth of the simulation run assumed (i.e. 1/100th of 1 second run). Still for statistics collection purposes,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

initial 1/10th of the duration of simulation is not considered.

It is also important that the length of the simulation is properly chosen. If the simulation is too short, the results may be highly variable. Another validation is, the confidence intervals for the number of TCP packets arriving at the MSI interface. The same observations are used in determining the confidence interval as were used in the computation of batch mean. The confidence interval for 95% confidence (mean is 34.32 packets) is ± 0.1342 packets. This narrow width of the confidence interval suggests that the duration of the simulation run is satisfactorily chosen.

Now that model verification and validation is complete, the discussion of the results and various observations on the behavior of the system is presented. The next section summarizes the performance of the various architectures under study.

5.2 Simulation Results

The overall approach of this research is to rely on the modeling study in the initial phase and use the results as input to the prototype development effort. Specifically, the modeling study will provide the information for a comparison of our parallel approach with other high speed network research efforts. It will also be used to evaluate the alternatives with in the parallel approach. These studies will allow the analysis of latency, buffer size, data loss, throughput, and delay as a function of different hardware devices, placement of functionalities, architectural alternatives, and amount of information passed between layers.

Early experiments were done with the simulation models to determine the values of the following parameters that give better performance in terms of throughput and round trip delay: (Please note that both these metrics were measured on end-to-end basis)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. CPU Speed;
2. Number of PPs;
3. Number of FDDI interfaces;
4. Memory Speed;

   Once all of the above are determined, the rest of the experimentation will be done to determine the values for the parameters:

5. Application Segment Size; and
6. TCP Window Size.

The parameters, once selected, would define various hardware and software attributes of the architectures under study. In this early evaluation phase, bus speed is not assumed to be a parameter because every bus is assumed to be a representative of currently available bus systems. In the last phase of this study, totally configured architectures are studied for various alternatives in scheduling, processing requirements, and channel selection strategies.

5.2.1 CPU Speed

The first hardware parameter which is determined is CPU Speed. The processors in these architectures will be shared for protocol processing and other user applications. We need to find a CPU speed for which the protocol processing is not consuming the entire processor time, leaving a major percentage of the CPU for other user activities. Throughput and round trip delays should not suffer when such a criterion is used for selecting the CPU speed. A slower CPU means that less share of the CPU utilization is available for the user applications when priority is given to the protocol processing.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

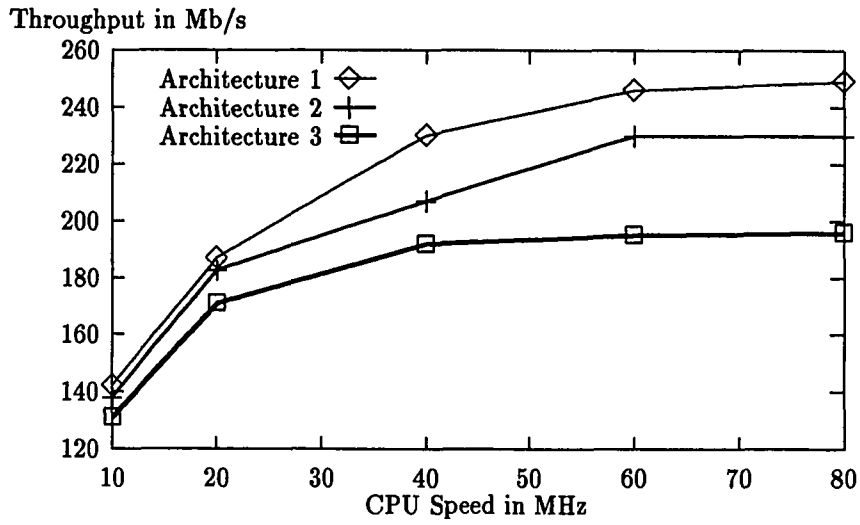

Figure 5.2: Throughput as a function of CPU speed

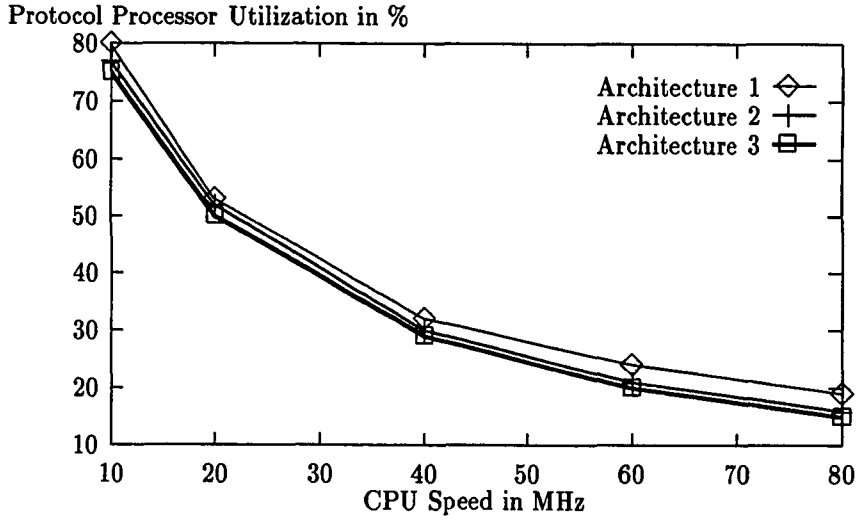

Figure 5.3: Protocol processor utilization as a function of CPU speed

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Results of the experiments with the architectures to determine the suitable CPU speed are reported in Figures 5.2 through 5.4. CPU clock frequency (in MHz) is used as the measure of its speed. In Section 3.4, experimentally determined TCP/IP processing time on a 20 MHz CPU has been reported as 150 μs (no checksum computation). Based on this measurement, processing time for 40 MHz CPU will be 75 μs. Similarly, we computed TCP/IP processing time for other CPU clock frequencies. To create an environment free of bottlenecks that can impact performance results, fast memory (20ns/word), and a large number of FDDIs (eight because NBus is rated at 800 Mb/s peak) are selected. Table 5.1 lists all the major parameters and their values used.

Parameter	Architecture 1	Architecture 2	Architecture 3
No. of CPUs	8 PPs + 1	8 PPs + 1	8 PPs + 1
No. of FDDIs	8	8	8
Memory Speed	20 ns/word	20 ns/word	20 ns/word
ABus	800 Mb/s	2400 Mb/s	2400 Mb/s
NBus	800 Mb/s	-	800 Mb/s
Load on Bus	0%	0%	0%
Load on Channels	0%	0%	0%
TCP Window Size	4x4500 bytes	4x4500 bytes	4x4500 bytes
Appl. Segment Size	4500 bytes	4500 bytes	4500 bytes
FDDI Frame Size	4500 bytes	4500 bytes	4500 bytes

Table 5.1: Major parameters for early experiments

Figures 5.2 through 5.4 show throughput, utilization, and round trip delay respectively as function of CPU speed for all three architectures. From these plots, an initial increase in CPU speed almost doubles the throughput. But after 40 MHz CPU speed, there is not much improvement in throughput or round trip delay. From these figures, 40 MHz is selected as the CPU speed for all later experiments.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

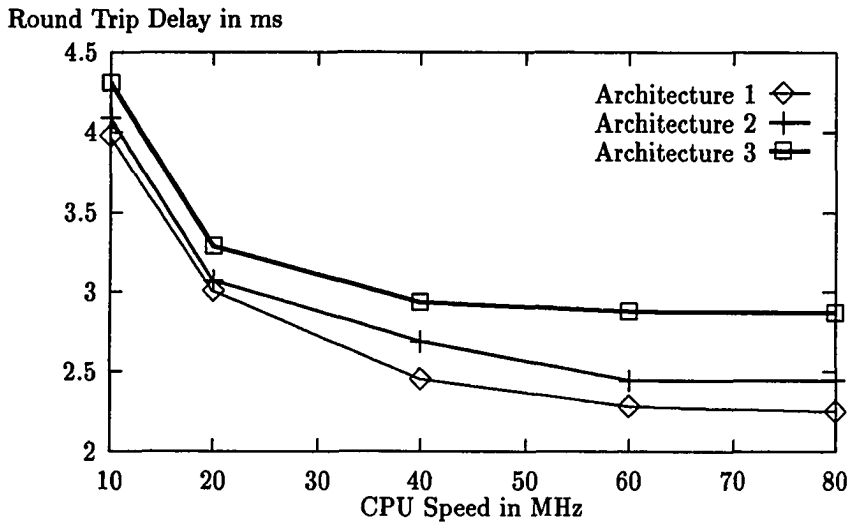

Figure 5.4: Round trip delay as a function of CPU speed

5.3 Application Scheduling and Number of PPs

Experiments are repeated for PPs. The objective of these experiments is to determine the number of PPs such that throughput is maximized and round trip delay is minimized. Configuration of the architectures is the same as what is listed in Table 5.1. The difference is that the number of PPs are varied from one through eight and the CPU speed is fixed at 40 MHz.

Figures 5.5 and 5.6 illustrate observed throughput and round trip delay respectively as a function of number of PPs. Initially, throughput increases with an increase in the number of PPs. These architectures, however, are unable to maintain this increasing rate in throughput with continuing increase in the number of processors. This occurs because the bottleneck now shifts to limited bus capacity in case of architecture 1, and memory bandwidth in case of architectures 2 and 3. As observed from these figures, the optimal performance in terms of higher throughput and lower round trip delay occurs for three protocol processors in case of architectures 2 and 3. For architecture

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

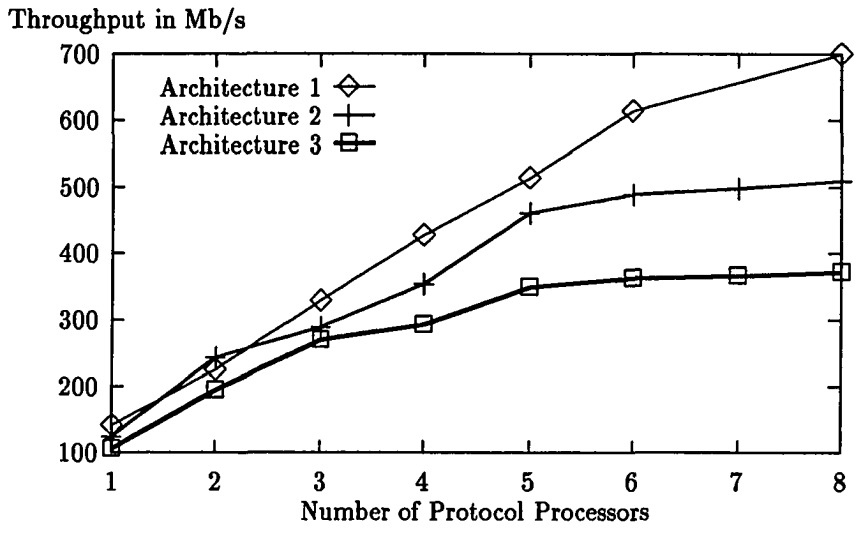

Figure 5.5: Throughput as a function of number of protocol processors

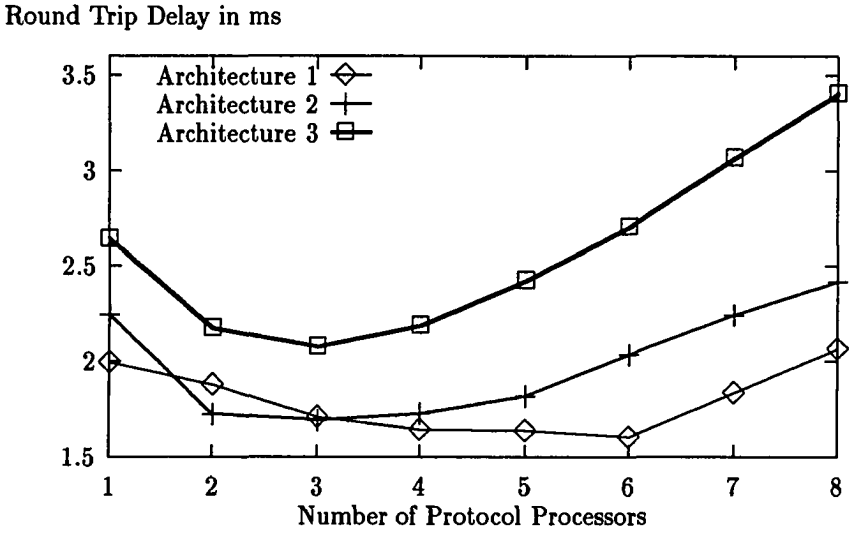

Figure 5.6: Round trip delay as a function of number of protocol processors

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1, throughput continues to increase with the increase in number of protocol processors. But this increase gradually tapers for number of protocol processors greater than six. Round trip delay also continues to decreases until six PPs are used and then it takes an upward swing. This is due to the fact that bottleneck is now in bus speed which is rated at peak of 800 Mb/s (as compared to the observed throughput of 600 Mb/s and more for larger number of PPs). Architecture 1 shows better performance because local memory per PP proves as an advantage over shared memory in the other two architectures.

To estimate the impact of increase in number of PPs on the scheduling, we also observe the average queueing delay per segment at the AS. For architecture 1, queueing delay decreases for every added PP. The rate of decrease in this delay slows down after four PPs. In architecture 2, after four PPs the queueing delay remains stationary at approximately 450 μs since adding additional PPs does not result in increase in throughput. Similar behavior is observed in architecture 3. In its case, queuing delay is around 600 μs. Hence, based on the performance plots, three PPs is selected as the suitable value for configuration of all the architectures.

5.4 Network Scheduling and Number of FDDIs

For these three architecture, the impact of increasing number of FDDIs on performance is illustrated through Figures 5.7 and 5.8. Parameters listed in Table 5.1 are kept same except the number of FDDIs is varied from one through eight. There is a linear increase in throughput observed with increasing number of FDDIs (refer Figure 5.7). Saturation in throughput after a certain number of FDDIs in shared memory architectures (five for architecture 3, and seven for architecture 2) is observed. A linear increase is seen in case of architecture 1 until eight FDDIs but for greater number of FDDIs it behaves similar. Weighted average of queueing delay per packet at FDDI interfaces was also measured.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

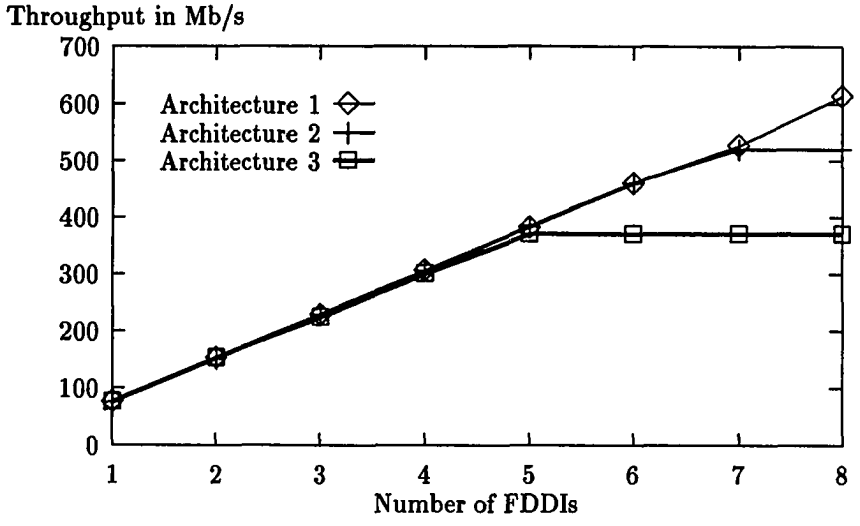

Figure 5.7: Throughput as a function of number of FDDIs

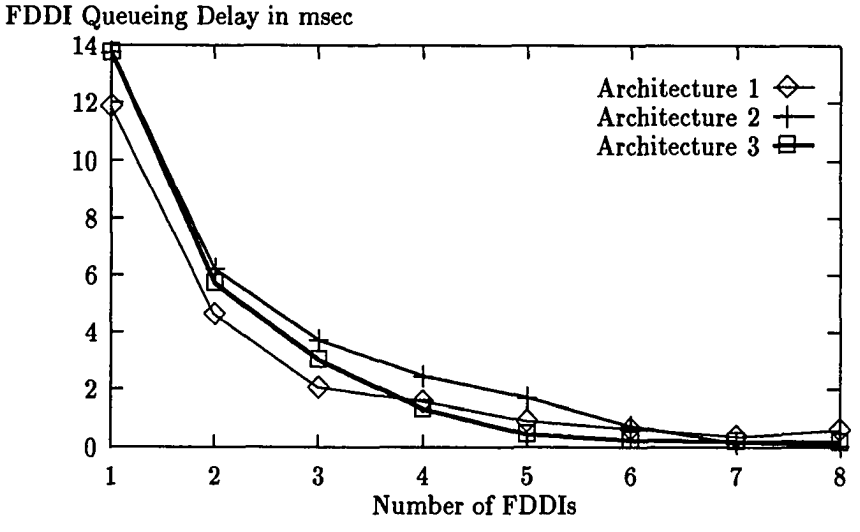

Figure 5.8: Queueing delay at FDDI as a function of number of FDDIs

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This is shown in Figure 5.8. An inverse relationship between queueing delay and number of FDDIs is observed for all the architectures. From Figure 5.8, the knee of the curves is observed at three FDDIs. Beyond three FDDIs, there is not significant improvement in queueing delay. Hence, three FDDIs are selected as the value for the later phase of the experimentation.

5.5 Memory Speed

Impact of memory access time and type of memory on the performance of each of the architectures is illustrated in Figures 5.9 and 5.10. The most commonly used memory today is 80 ns/word, with the super-computers like Cray using 20 ns/word memory. Such fast memory is very expensive as compared to 80 ns/word memory. Table 5.2 lists the peak memory bandwidth and effective memory bandwidth available for various memory speeds. When computing effective memory bandwidth for network I/O, three memory accesses per word of data transmission are assumed (application write, TCP data read + TCP checksum write, FDDI packet read through DMA). Hence, for a particular memory speed, effective memory bandwidth is the upper bound on throughput at the application level in shared memory architectures[74].

Memory Speed (ns/word)	Peak Bandwidth (Mb/s)	Effective Bandwidth (Mb/s)
20	1600	525
30	1067	350
40	800	250
50	650	200
60	525	175
70	450	150
80	400	125

Table 5.2: Memory Speeds and related memory bandwidths

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

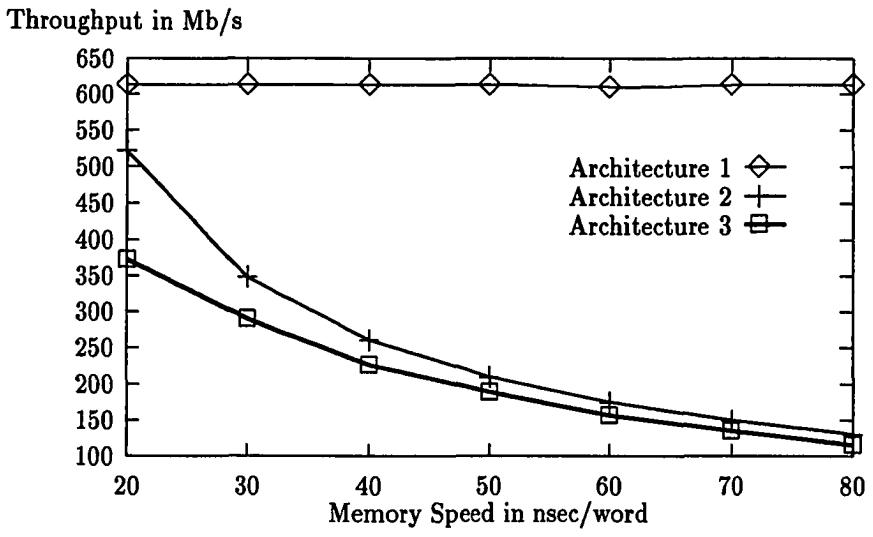

Figure 5.9: Throughput as a funcion of memory speed

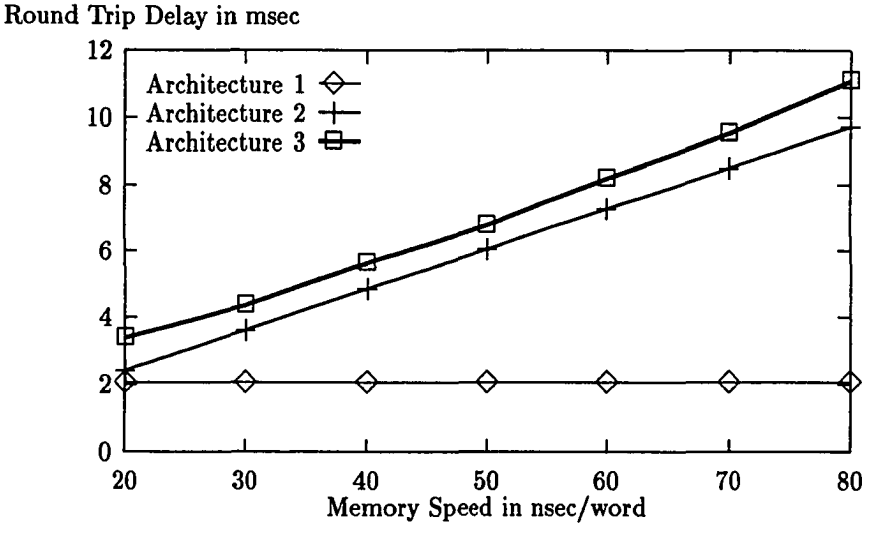

Figure 5.10: Round trip delay as a function of memory speed

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The results reported in this study use the same configuration as listed in Table 5.1. Performance of Architecture-1 is independent of the memory speed since every processor has its local memory. Architectures 2 and 3 have different behavior from architecture 1 because the CPUs in these architectures share a common memory. The overall throughput approximately reduces to the effective memory bandwidths listed in Table 5.2. If there are other memory intensive tasks going on in these shared memory architectures, then throughput is expected to reduce further. Throughput shows an inverse relationship with the memory speed while round trip delay displays a linear relationship (refer Figures 5.9 and 5.10).

From these initial experiments, the final configuration of three architectures is illustrated in Table 5.3. Along with values of major hardware parameters, background load classifications is also presented in this table. The background load on FDDI channels is classified into no load, balanced load and unbalanced load. A 30% (of FDDI capacity) uniform load is put on each FDDI channel to test for balanced load conditions. To test the response of parallel communications architectures to varying conditions on channels, three unbalanced load conditions are tested. In first case, different loads (20%, 30% and 40%) loads are applied on each channel[1] such that their overall average is 30%. In second and third cases, the constant loads are varied at different rates over the entire run of the simulation. To model slow variation of load on each channel, the load is varied eight times (variations are spread uniformly over the entire simulation) and every time load was incremented by 10% with a limit of 70% as maximum load. Fast variations are modeled as 200 such variations during the entire simulation. Every variation lasts equal duration. The objective behind simulating these varying load conditions is to check for the effectiveness of the network scheduling of packets to parallel channels.

[1]There are three FDDI channels in every architecture.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Parameter	Architecture 1	Architecture 2	Architecture 3
CPU Speed	40 MHz	40 MHz	40 MHz
Number of APs	1	1	1
Appl. Load	Exponential	Exponential	Exponential
AS Location	CPU 1	CPU 1	CPU 1
AS Type	Centralized	Centralized	Centralized
AS Policy	FCFS	FCFS	FCFS
ABus Speed	800 Mb/s	2400 Mb/s	2400 Mb/s
NBus Speed	800 Mb/s	-	800 Mb/s
Memory	Local per CPU	Shared	Shared
Memory Speed	80 ns/word	80 ns/word	80 ns/word
Number of PPs	3	3	3
NS Type	Distributed	Centralized	Distributed
NS Location	CPU2, CPU3, CPU4	Mux/Demux	CPU2, CPU3, CPU4
NS Policy	RR/Adaptive	RR/Adaptive	RR/Adaptive
FDDI Interconnect	NBus	Mux/Demux	NBus
Number of FDDI	3	3	3
FDDI Speed	100 Mb/s	100 Mb/s	100 Mb/s
FDDI Load Types			
(1)	No Load	No Load	No Load
(2)	Balanced 30% Load	Balanced 30% Load	Balanced 30% Load
(3)	Unbalanced Load	Unbalanced Load	Unbalanced Load
(3a)	Fix - (20, 30, 40%)	Fix - (20, 30, 40%)	Fix - (20, 30, 40%)
(3b)	8 Variations	8 Variations	8 Variations
	(20%+10%) mod 70%	(20%+10%) mod 70%	(20%+10%) mod 70%
	(30%+10%) mod 70%	(30%+10%) mod 70%	(30%+10%) mod 70%
	(40%+10%) mod 70%	(40%+10%) mod 70%	(40%+10%) mod 70%
(3c)	200 Variations	200 Variations	200 Variations
	(20%+10%) mod 70%	(20%+10%) mod 70%	(20%+10%) mod 70%
	(30%+10%) mod 70%	(30%+10%) mod 70%	(30%+10%) mod 70%
	(40%+10%) mod 70%	(40%+10%) mod 70%	(40%+10%) mod 70%
Bus Load types			
(1)	No Load	No Load	No Load
(2)	Balanced 30% Load	Balanced 30% Load	Balanced 30% Load

Table 5.3: Final architectural parameters

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.5.1 Transport Window Size

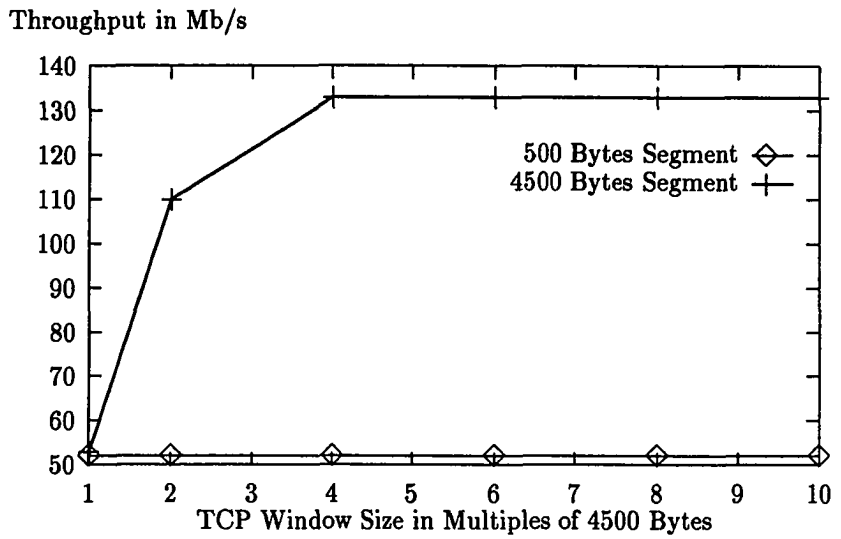

Figure 5.11: Throughput as a function of TCP window size for Arch. 1

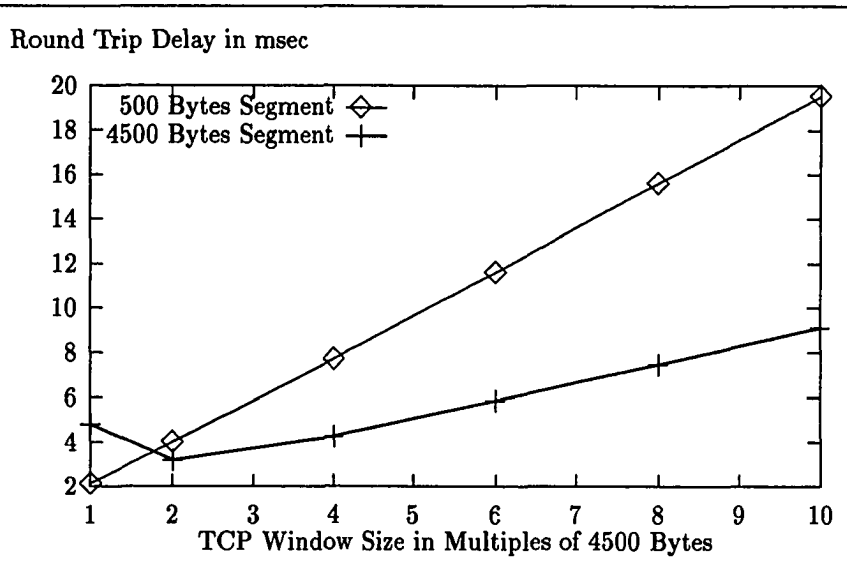

Figure 5.12: Round trip delay as a function of TCP window size for Arch. 1

To determine TCP window size for which the parallel network system delivers maximum throughput with minimum round trip delay, experiments were con-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ducted for varying window sizes. These experiments were repeated for different application segment sizes. Figures 5.11 through 5.16 illustrate the impact of window size for small (500 bytes) and large (4500 bytes) segment sizes for all the architectures. From these plots, the results indicate that the window size selection should be such that:

1. No PP starves for data to be sent while waiting for the acknowledgments. This will only hold if TCP send window is still open. This condition helps in achieving higher throughput by proper application scheduling.

2. No packet waits unnecessarily in sender's window. This condition helps in achieving lower round trip delays. A large send window in a communications system implies larger buffer space for accepting bytes from sender application. Since same PP is doing send and receive functions, the delay per byte in TCP window buffers will increase for large window.

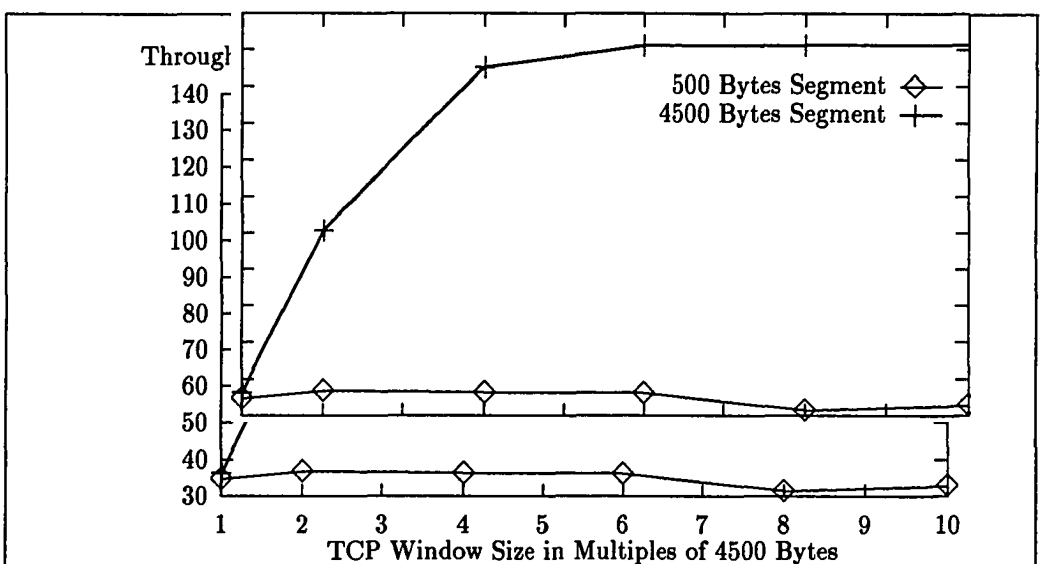

Figure 5.13: Throughput as a function of TCP window size for Arch. 2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

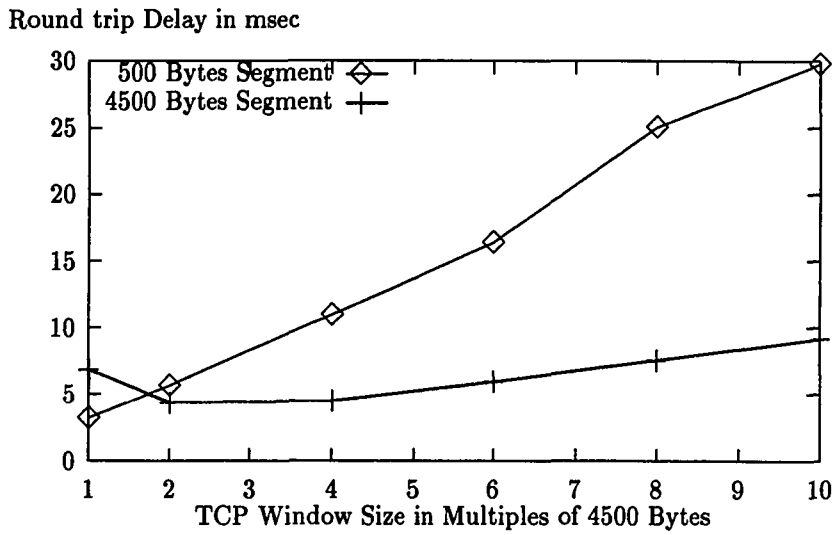

Figure 5.14: Round trip delay as a function of TCP window for Arch. 2

5.5.2 Application 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 that a small segment size (500 bytes) results in low throughput and high round trip delay. But for 4500 bytes segment there is a considerable improvement in performance (e.g. 50 Mb/s for 500 bytes segments and 110 Mb/s for 4500 bytes segment). There is a 100% increase in throughput when application segment of FDDI frame size are used. For application segment sizes larger than FDDI frame size, the PPs will generate multiple TCP packets from a segment. This is so because FDDI frame size is the maximum transfer unit when FDDI networks are used.

5.5.3 Scheduler Location

Table 5.4 illustrates the comparative performance of the NS when used in centralized and distributed mode. Although the architectures in consid-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

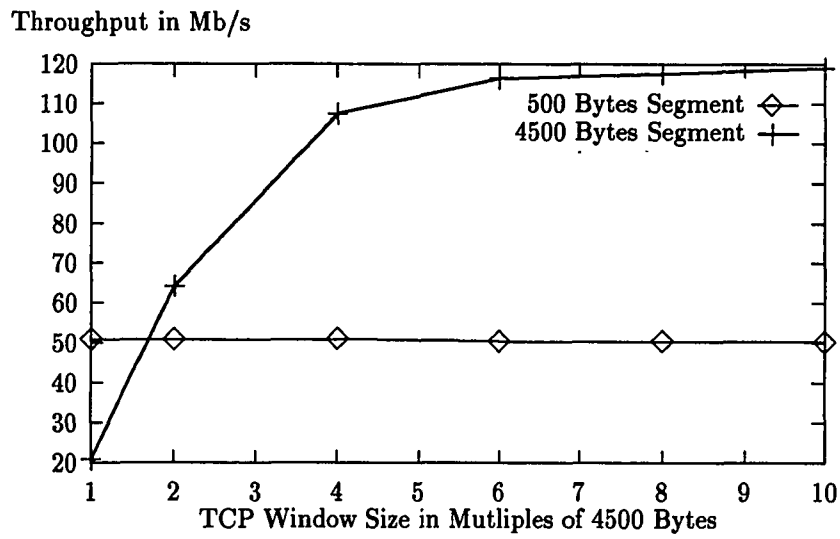

Figure 5.15: Throughput as a function of TCP window size for Arch. 3

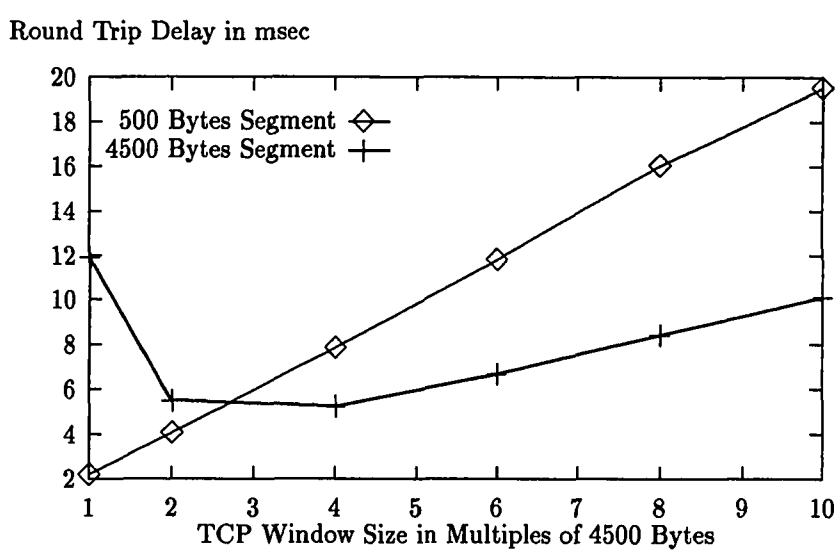

Figure 5.16: Round trip delay as a function of TCP window for Arch. 3

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Load on Bus	Load on FDDI	NS Policy	Metric	Distr. NS Arch. 1	Centr. NS Arch. 2	Distr. NS Arch. 3
No Load	No Load	Round Robin	Throughput (Mb/s)	230	130	117
			RT Delay (ms)	2.453	4.345	4.818
		Adaptive	Throughput (Mb/s)	226	131	114
			RT Delay (ms)	2.494	4.324	4.949
30% Load	30% Balanced	Round Robin	Throughput (Mb/s)	133	131	116
			RT Delay (ms)	4.268	5.956	6.698
		Adaptive	Throughput (Mb/s)	132	120	109
			RT Delay (ms)	4.286	4.715	5.169
30% Load	Unbalanced Constant	Round Robin	Throughput (Mb/s)	92	112	106
			RT Delay (ms)	6.173	5.041	5.354
		Adaptive	Throughput (Mb/s)	130	119	110
			RT Delay (ms)	4.336	4.751	5.139
30% Load	Unbalanced Slow Varying	Round Robin	Throughput (Mb/s)	53	56	57
			RT Delay (ms)	10.829	10.062	10.011
		Adaptive	Throughput (Mb/s)	76	69	95
			RT Delay (ms)	7.537	8.248	5.917
30% Load	Unbalanced Fast Varying	Round Robin	Throughput (Mb/s)	76	75	71
			RT Delay (ms)	7.500	7.609	8.005
		Adaptive	Throughput (Mb/s)	84	77	105
			RT Delay (ms)	6.740	7.362	5.419

Table 5.4: Comparative performance of three architectures

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

eration differ, the activities of NS are similar. Under varying unbalanced load conditions, distributed network scheduling (architecture 3) outperforms the centralized network scheduling (architecture 2). Specifically, when similar adaptive scheduling policy is used in both the cases, distributed network scheduler results in higher throughput (105 Mb/s) as compared to centralized network scheduling (93 Mb/s). Under balanced load conditions on channels and no load conditions, however, centralized scheduling (Architecture 2) performs better than distributed scheduling (130 Mb/s vs. 117 Mb/s). But, this result is due to the nature of the architectures used rather than scheduling algorithm.

5.5.4 Comparison of Architectures

Table 5.4 presents a comparative illustration of the performance of the three architectures for various network scheduling policies and background loads on channels. Under no load and balanced load conditions, architecture 1 outperforms other two architectures by almost 1.7 times in terms of throughput and 1.9 time in terms of round trip delay. For constant unbalanced load on channels, adaptive network scheduling policy shows a marked improvement in performance. This improvement is more significant in terms of reduced round trip delay. For varying load conditions, architecture 3 out performs the other two architectures in adapting to the dynamic load conditions. Architecture 2 is the most sluggish to fast varying conditions.

5.5.5 Acknowledgment and Retransmission

In parallel network systems, an important issue is the number of ways transport window can be managed and retransmissions and acknowledgments can be processed. This issue is non-existent in serial systems, since there are not very many options existing relating to the mapping of such activities to the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

processors. As discussed in Chapter 3, TCP window can be managed in an entirely distributed way or in an exclusive centralized fashion. On architecture 3, a comparative study of CRCA and DRDA is performed. Architecture 3 is selected for this performance comparison because it resembles closely to an available architecture (Sun Microsystem's Sparc-10) and in prototyping phase it is feasible to implement one of these possible schemes based on the outcome of this study.

In centralized mode, one protocol processor is assigned a special purpose task of implementing a shared TCP window for other PPs. All packets transmitted and received are first referred to this special processor for TCP window updates and related decision making about TCP processing. Acknowledgments and retransmissions out of the shared TCP window are handled by this processor. The distributed window management is the usual case in which every protocol processor implements its own TCP window and every operation is done independent of any other protocol processor. For a fair comparison of the two schemes, following architectural configurations are considered:

1. Distributed management: 3 Protocol Processors, 8x4500 bytes TCP window per processor
2. Distributed management: 2 Protocol Processors, 8x4500 bytes TCP window per processor
3. Centralized management: 2 Protocol Processors, 16x4500 bytes shared TCP window

The rest of the configuration remains the same for all these cases (see Table 5.3.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.5.6 DRDA vs. CRCA

Tables 5.5 and 5.6 summarize the effects of centralized and distributed modes of window management on the performance of the system. Here, the performance is measured in terms of the achieved throughput (Mbps), the end-to-end delay between the TCP entities, the utilization of PPs, percentage of retransmissions due to timeout, and the delay at the receiver end between the TCP entity and the application due to out of order reception of packets and segments respectively. These were measured under the conditions of lossless channels (Table 5.5) and lossy channels (Table 5.6).

Measured Parameter	CRCA				DRDA				DRDA			
	3 PPs, 16x TCP Window				3 PPs, 8x TCP Window				2 PPs, 8x TCP Window			
	No Load	Bal 30%	UnBal 30%		No Load	Bal 30%	UnBal 30%		No Load	Bal 30%	UnBal 30%	
	RR	RR	RR	ADAP	RR	RR	RR	ADAP	RR	RR	RR	ADAP
Throughput (Mb/s)	125	106	107	103	117	108	63	120	117	115	80	101
End-to-End Delay (ms)	7.4	9.2	11.3	11.7	5.8	7.2	15.1	5.9	3.9	4.8	7.9	5.2
PP Utilization (%)	40	24	28	26	16	13	11	16	24	19	16	21
TCP Reseq. Delay (ms)	0.4	0.7	1.4	1.3	0.2	0.4	3.2	1.1	0.2	0.4	3.1	1.1
Sched. Reseq. Delay (ms)	1.0	10.6	14.1	12.3	0.1	0.5	0.7	0.5	0.1	0.5	0.7	0.5

Table 5.5: Architecture 3 : DRDA vs. CRCA (Lossless Channels)

1. **Throughput:** With unbalanced background traffic, the adaptive assignment of packets to channels (ADAP) results in higher throughput than the round-robin (RR) policy. While its effect is insignificant in the centralized window management, it is effective in the distributed mode. For example, in the lossless case, with 3 PPs each with a window size of 8 packets, the throughput with ADAP is 1.6 times that of RR. With 2 PPs, however, the throughput with ADAP is only 1.5 that of RR. These

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Measured Parameter	CRCA				DRDA				DRDA			
	3 PPs, 16x TCP Window				3 PPs, 8x TCP Window				2 PPs, 8x TCP Window			
	No Load	Bal 30%	UnBal 30%		No Load	Bal 30%	UnBal 30%		No Load	Bal 30%	UnBal 30%	
	RR	RR	RR	ADAP	RR	RR	RR	ADAP	RR	RR	RR	ADAP
Throughput (Mb/s)	103	99	64	62	101	100	61	99	99	95	60	63
End-to-End Delay (ms)	8.1	12.1	14.4	14.7	6.2	7.9	13.6	6.6	4.2	5.5	8.3	5.7
PP Utilization (%)	41	28	30	30	16	13	10	15	24	18	16	20
TCP Reseq. Delay (ms)	0.9	1.4	2.1	2.1	1.6	1.4	6.9	1.6	1.2	1.1	3.3	1.4
Sched. Reseq. Delay (ms)	8.3	11.7	15.0	12.4	1.6	1.3	2.6	1.	0.8	0.7	0.9	0.8

Table 5.6: Architecture 3 : DRDA vs. CRCA (1% Loss on Channels)

observations are also valid in the case of the lossy channels. These results support earlier studies which identified the key performance factors in parallel ring networks, and justifies the use of adaptive policies in assigning messages to alternate channels to maximize performance [70, 69]. The effect on throughput from increasing the number of processors, and thereby increasing the effective TCP window size is clear from the results. While the effect is insignificant under balanced background traffic, it is quite significant under unbalanced background load. For example, with the lossless channel and adaptive assignment of packets, the throughput with 3 PPs is 1.2 times that with 2 PPs; this comparison is also applicable to channels with loss. When we consider the round-robin policy, however, the effect is not so apparent. For example, with the lossless channel and unbalanced background, the throughput with 3 PPs is only 0.8 of the 2-PP system. Similarly, for the system with loss on channels, the throughput of the 3-PP system is only 0.9 of the 2-PP system. This is counterintuitive. We attribute the reduction in the throughput of the 3-PP system to the RR policy rather than the number of PPs. In other words, even though the increase in the number of PPs (and hence the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

effective window size) improves the system throughput under unbalanced background load cases, the increase is overshadowed by the decrease in throughput with the RR policy when the number of PPs is increased. In summary, under unbalanced load conditions it is observed that (i) increasing the number of PPs will decrease the throughput under the RR policy, and (ii) increasing the PPs will improve the throughput with adaptive assignment of packets.

2. **End-to-end Delay.** TCP end-to-end Delay is the delay between TCP connections at the source and receiver. With no background traffic on the channels, the distributed window management (with RR) achieves lower delay than the centralized management (with RR). In fact, the 3-PP distributed mode has a delay which is 0.8 of the 3-PP centralized mode. Within the distributed case, the 2-PP system has a delay which is 0.8 of the 3-PP system. This can be explained as follows. As the number of PPs is increased, so does the total effective window size. Thus, given the same number of physical channels, an increased contention occurs for the channels under the 3-PP system than the 2-PP system. Hence additional delay will occur at the channels. The higher delay with RR is due to its naive assignment of packets to channels. This also supports our earlier results on the effects of assignment policies on parallel channels [70, 69]. These observations are also valid under balanced and unbalanced background load conditions.

3. **PP Utilization.** The results of PP utilization as seen in Tables 5.5 and 5.6 indicate both the overhead of centralized mode as well as the effect of memory bottleneck. Considering the no-background traffic case, even though the throughput is the same in the centralized and the distributed case, the processor utilization is quite different: the 3-PP centralized system has 40% utilization, and 3-PP distributed one has 16% utiliza-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tion. We attribute the higher processing overhead in the centralized case to the additional communication and control processing involved in the centralized processing. The memory bottleneck is apparent from the distributed 2-PP and 3-PP systems: (i) both result in the same throughput (163 Mbps), (ii) the total processing to yield this throughput is equally divided among the processors: 16% PP utilization for the 3-PP case, and 24% utilization in the 2-PP case. If memory were not the bottleneck, we would have expected higher throughput and hence higher utilization per PP in the case of the 3-PP system.

4. **Retransmission Percentage.** To study the impact of loss of packets or acknowledgments in channels in the parallel system we measured retransmissions as percentage of the total packets transmitted for the cases listed in Tables 5.5 and 5.6. The results (not shown in Tables 5.5 and 5.6) indicate that the centralized mode suffers more retransmissions as compared to distributed mode (20% as compared to 12% of the total packets transmitted). We attribute this behavior to a greater staleness of round trip time in calculating timeout timers and a single processor deciding about the loss of a packet or an acknowledgment. One major conclusion from this study is that for a loss of packets or acknowledgments as low as 1% of the total packets that go on the medium, a large percentage of retransmissions (10-20%) result. This indicates a serious inadequacy in determining the exact time to retransmit a packet and estimation of the packet loss in current TCP protocol. In other words, it illustrates the importance of using the latest value of round trip time (which correlates to the most recent state of the network channels) in reducing the retransmissions due to timeout.

5. **TCP Resequence Delay.** This represents the delay caused by the out-of-sequence reception of TCP packets. The inefficient assignment of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

packets to channels by the RR policy is reflected in this delay. If packets are assigned to a channel with heavy load, the receiving TCP has to wait for the packets that are stuck in slower channel. While other out of sequence received packets are placed in the resequencing queue. Hence, the TCP resequencing delay for out of order packets accumulates. In all cases, the resequence delay is higher with the RR policy. In addition, the delay is higher with the 3-PP system than with the the 2-PP system. This is due to the increase in the number of processes doing the wrong channel assignment of packets.

6. **Scheduler Resequencing Delay.** This represents the delay involved in propagating received and assembled segments to the application in the same order in which they were actually sent. A segment can be considered complete and assembled once all packets constituting it are received. A received segment is buffered at the application scheduler until all segments prior to it are received. The delay involved is the time a segment has to remain buffered owing to prior segments not being received yet. It can be seen from the tables that this delay is an order of magnitude more in the centralized mode than in the distributed.

In summary, simulation modeling has enabled investigations into the architectural advantages of parallelism in network system. These models have enabled us to perform a comparison of various differing design strategies and to predict accurately the performance of parallel systems. Among other things, these models have enabled the investigation of performance improvement mechanisms such as load balancing and elimination of temporally serial operations in network systems. Also, these models have been instrumental in determining if high data rate nodes using all channels of parallel network can coexist with other nodes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.6 Experimental Measurements for Network Scheduling

Even though a number of design related issues have been resolved through modeling and analysis, there are a number of other issues that can only be resolved through a prototype. A prototype is also useful in validating the results from modeling and analysis. A prototype for parallel communications systems can be used to address following issues:

1. The impact of parallelism at different communications layers on overall system performance;
2. The overhead of additional interfaces (schedulers) between communication layers;
3. Determining system bottlenecks including limitations due to the speed of protocol processing, speed of interfaces (e.g. contention for bus and memory) and buffering; and
4. Limitations on the applications due to the proposed architecture.

The prototype development will help in understanding the implications and limitations of using current technology in building multi-hundred Mb/s networks.

In this section we report about a parallel network prototype built using uniprocessor workstations. Non-availability of multiprocessor workstation at the time of prototype study forced us to study parallelism at the lower level (media access and physical layers) only. Nevertheless, the problem of assignment of packets to channels is still an outstanding issue which can be studied in this prototype. Also, for the first time ever an attempt is made to utilize the available parallel channels completely transparent to the user applications. In

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this developmental study, the IP module is modified to receive a single stream of data from a TCP connection and divide it into multiple streams based upon the number of parallel network interfaces available. On the receiver side, this modified IP module recombines the multiple streams of data into a single TCP stream referring to one TCP connection. The Solaris 2.0 operating systems, its TCP and IP modules are used.

5.6.1 Details of the Experimental Testbed

The parallel network prototype was built on a testbed consisting of four Sun Sparc 1 workstations, each having two Ethernet cards for two different Ethernets (shown in Figure 5.17). Two of the workstations were used as load generators on the network. The remaining two communicated with each other using parallel Ethernets. The performance was measured with respect to these two workstations. In addition, for some experiments a LAN analyzer (Sniffer) was connected to one of the Ethernets to monitor traffic and generate traffic. The results of this study are restricted to this particular setup and the number of hosts connected to the parallel networks.

For the parallel network prototype, many issues need to be taken care of at IP level. Since two network routes exist from every machine to every other machine, major program modifications were needed for the following purposes:

1. **Initialization of parallel routes:** Under Solaris 2.0 when a workstation boots up, it acknowledges the existence of multiple (two in our case) Ethernets. But when it comes to communicating with other hosts, TCP/IP usually uses the default route (Internet Route Entry, IRE) to reach the peer host. In most cases, it is the le0 interface which is used. The IP module was modified such that two parallel IREs were created and initialized when parallel networks were used. For the prototype, the code had static parallel IREs for the hosts (Horsa and Ceolwulf, see Figure

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.17). Hence, static parallel routes were established for Horsa to communicate to Ceolwulf on both the channels and vice-versa. All existing applications (like telnet, ftp and rcp) work without modification in the parallel environment. The use of parallel channels is transparent to these applications.

2. **Mismatch of Ethernet - IP address association:** The streams architecture of the TCP and IP modules was also modified. For the receive part of the IP module, the parallel incoming streams of IP datagrams were connected to the read queue of the TCP connection between the two peers. This change resulted in a significant improvement in the performance of the parallel prototype. The current implementation of the IP module can be explained with the help of an example (see Figure 5.17.). When an IP datagram for host `128.82.6.211` arrives at the `le1` interface, the `ip_rput()` service routine assumes that the datagram has arrived on the wrong interface since the IP address of `le1` is `128.82.7.211`. Although both the Internet addresses refer to the same host on the network, the existing IP module processes a datagram in such a way that it is handled through `le0` interface. With this unnecessary queueing of the datagram downwards rather than upwards, throughput of parallel networks suffered greatly (actually the throughput at the application level halved).

3. **Developing an efficient algorithm to handle load variations on the network:**

   Since the modified IP module was now entrusted with the task of dividing the incoming stream of packets from TCP connections between Horsa and Ceolwulf (see Figure 5.17), the assignment of the packets to the correct Ethernet was extremely important. If IP assigns most of the packets to a heavily loaded channel and does not effectively balance the load on the available channels, the performance of the system will suffer. This means

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

more collisions, less throughput, and higher latency for all the hosts trying to communicate over these channels. We tested several algorithms to determine which worked best under various channel conditions. The details of the algorithms are presented below.

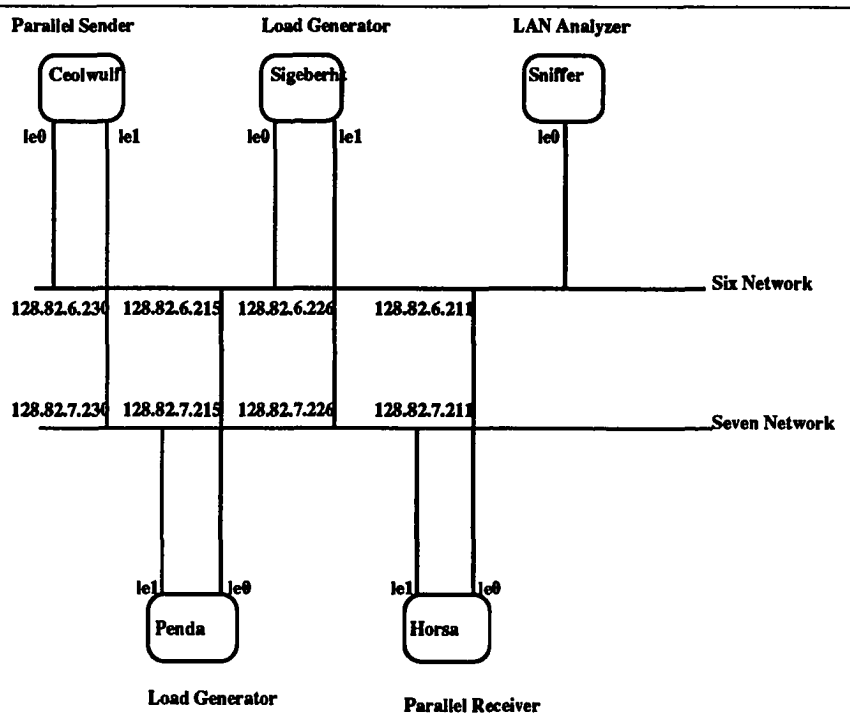

Figure 5.17: Experimental testbed for Parallel Ethernets

The modifications to the TCP module were modest. The TCP window size was increased from 4KBytes to 50KBytes so that flow control did not necessarily held up the transmit and receive capabilities of the prototype. In subsequent paragraphs, the scheduling algorithms I developed and tested on the prototype are explained. To estimate the load on the channels, collisions seen by an Ethernet interface over a period of time were used as a measure. The collisions an Ethernet interface sees when it tries to transmit were counted and not all the collisions happening on the network because counting the latter can become a big overhead to the overall Ethernet driver performance. So in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

order to determine the load of a channel, a small number of packets on each channel were transmitted and the collisions suffered were counted.

For assignment of packets to Ethernets, various scheduling policies were used. They were:

1. **Algorithm 1: Round Robin Algorithm (RRA)** Under uniform conditions on the channels, the round-robin policy of assigning packets to the available Ethernets resulted in good performance because there was not much to adapt. The RRA is :

```
1. If first time select le0; last = le0.
2. If last == le0 select le1; last = le1.
3. If last == le1 select le0; last = le0.
```

2. **Algorithm 2: Adaptive Algorithm (AA)**

   RRA performed packet assignment without considering background loads on the Ethernets and hence encountered many collisions. As a result, the latency for all the hosts increased and a percentage of the bandwidth was wasted in collisions. The following AA algorithm samples both the channels at fixed intervals given in terms of the number of packets transmitted. Each such interval is called a **Slot**. The collision count during the transmission of a **Sample** number of packets is used in deciding which channel to use for the remaining **(Slot - Sample)** packets. This sampling process is repeated after every slot packets is sent (see Figure 5.18). Both slot and sample are user controlled parameters and are set using ndd[2] utility.

   The adaptive AA algorithm is:

[2]Ndd is a maintenance command to get and set driver configuration parameters. For more details please refer to Sun Microsystems Solaris 2.0 maintenance commands manual.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

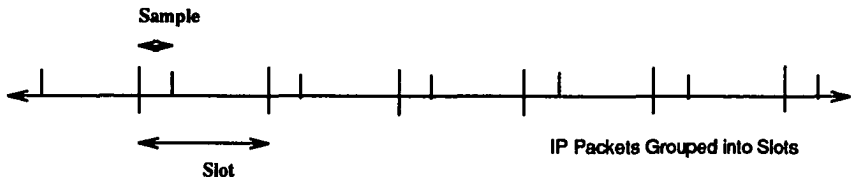

Figure 5.18: Illustration of the relationship between SLOT and Sample IP Packets for Adaptive Algorithm

```
1. Reset the Collision counts on both the channels.
2. Send sample packets using RRA and count the collisions.
3. If ( Collision count on le0 $<$ Collision count on le1)
                Send  ( Slot  - Sample ) on le0}
4. If ( Collision count on le0 == Collision count on le1)
                Send  ( Slot  - sample ) using RRA
5. If ( Collision count on le0 $>$ Collision count on le1)
                Send ( SLOT  - sample) on le1
6. Repeat the above process once SLOT packets are sent.
```

3. Algorithm 3: Improved Adaptive Algorithm (IAA)

The problem with the AA is that its selection criteria is limited to the sample packet decision period and it cannot adapt to changing traffic on both the Ethernets effectively for the (SLOT - sample) packets. The following algorithm keeps the collision history of the last one time period (slot) and uses it to assign the packets to be transmitted on each Ethernet in the next time period. This effect was achieved by estimating the load of a channel and using this information in the next time period. Freeness of a channel was computed by subtracting the ratio of number of collisions observed in the previous slot to the number of packets transmitted on it.

```
1. Freeness of le0 = 1 -  Collisions on le0/Packets sent on le0
2. Freeness of le1 = 1 -  Collisions on le1/Packets sent on le1
3. Find the number of packets to be transmitted using  1 and 2
4. Send the above calculated number of packets on le0 and le1
5. Repeat the above process for next Slot.
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The prototype was tested in different channel loadings and for all three algorithms of assigning IP packets to the networks. The results from the measurements are reported in the next section. To generate background loads on the Ethernets, (see Figure 5.17) two workstations and a LAN analyzer were used. The LAN analyzer can generate traffic on any Ethernet in a loopback form, but the two workstations are used to pass background data over both Ethernets and a single Ethernets. Background load environments of uniform (and continuous) loading of the Ethernets are created. Bursty background load environments are created when two workstations would exchange a random number of bytes after a random sleep as background data. For bursty environments, the burst transfers were approximately 1 Mbps when being tested for 10% bursty background loads. The same load pattern was used for other bursty background loads. For some experiments, the LAN analyzer is used to generate continuous background loads on one Ethernet and background load generator pair on the other. The measurements were carried for long durations ($\approx$ 400-600 sec.) and large amounts of data transfer ($\approx$ 400 Mbytes). Numerous tests were done for each experiment to take care of the variations.

5.6.2 Experiment Measurements and Results

In chapter 3, it was observed that the maximum throughput that can be achieved at the user level can be limited by the host architecture rather than the network capacities. A workstation like Sparcstation 1 can generate up to 80% of the capacity of an Ethernet LAN.

Impact of the checksum computation on the throughput is an interesting issue. Experiments were performed with and without checksum to determine its impact. With no background loads on the Ethernet, the throughput observed with and without checksum is 8.6 Mbps. Under a background load of 50% (generated by Sniffer) on Ethernet, the throughput with checksum was 3.2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Mbps and without checksum was 3.3 Mbps. Hence, for the environment we had created, we found that avoiding the checksum computation did not create a big difference in the performance of the workstations connected on a Ethernet LAN.

The other major result of these experiments was the demonstration that parallelism can result in higher throughput at the user level. One cannot say that the use of more than one Ethernets will guarantee higher throughput. The fact is that the correct load balancing on the two Ethernets should result in getting maximum utilization of the available Ethernets in the presence of other traffic. Figure 5.19 shown below illustrates this result. It is clear from the figure that the throughput at the user level improves by almost 1.8 times (3.7 Mbps for single Ethernet case as compared to parallel Ethernet case) when the background load of 50% is being pushed uniformly on the Ethernets. When the Ethernets are lightly loaded, the parallel case was able to provide only 17% improvement in the throughput as compared to the single Ethernet case. This was due to the fact that the performance of the sender (or receiver) machine was governed by its architectural limits. If we compare the performance of the round-robin algorithm to the adaptive algorithm (IAA), we observe that the blind round-robin scheduling of IP packets to the two Ethernets performs a little worse than the adaptive scheduling. Note however that the round-robin algorithm will perform very poorly if there is a large imbalance in loads on the channels.

Results of experiments with two kinds of the adaptive algorithms are also reported. The first algorithm, AA scheduling, makes the decision of the assigning all of the (slot - sample) packets to an Ethernet based on the absolute collision count observed during transmission of sample number of packets. The second algorithm, IAA scheduling, assigns packets based on the ratio of the collisions to the number of packets sent on an Ethernet. Experiments with

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

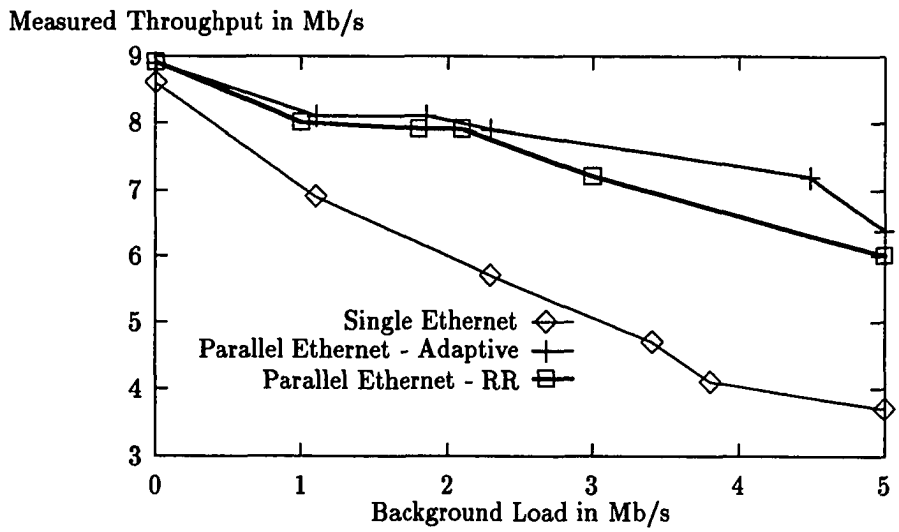

Figure 5.19: Performance Comparison of a Single Ethernet versus Parallel Ethernets

various values of slot and sample were also performed. These results are listed in Table 5.7.

B/G Load on		Parameters		Throughput	Pkts sent on	
Six-net	Seven-net	Slot	Sample	Mb/s	Six-net	Seven-Net
23%	23%	5000	50	7.6	56K	293K
23%	23%	50000	50	7.1	50K	300K
23%	23%	100	50	8.0	109K	224K

Table 5.7: Performance of AA Scheduling Under Bursty Loads on Ethernets

From Tables 5.7 and 5.8, the impact of the parameter sizes on the performance is illustrated. If scheduling is based on AA, the inability to distribute the packets on the channels effectively is evident (refer to Table 5.7) because scheduling decisions are based on short sampling periods. From second row of Table 5.7, it can be seen that lower throughput is the result of wrong assignment of 50K minus 50 packets on seven-net much of the time. Throughput

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

increases, however, to 8 Mb/s when the sampling of 50 packets is done every 100 packets, meaning that this parameter selection in AA scheduling impacts the overall performance.

For second algorithm, IAA scheduling, which makes decisions based on the history of collisions during last SLOT, a significant consistency was observed in the throughput levels. For any combination of the parameters, the throughput changed marginally. Only for 50K slot values, throughput falls a bit (Table 5.8) because the SLOT size is too big to adjust to changes in the background load quickly. The second adaptive algorithm adapts quickly to the burst changes in the background load to maximize throughput. When the RRA is compared with the IAA scheduling, the throughput obtained with round-robin was 8.0 Mb/s as compared to 7.9 Mb/s of adaptive policy. But in the round robin case, 1 collision occurred per 2 IP packets transmitted (89K collisions for 165K packets sent on Six-net and 73K collisions for 165K packets sent on Seven-Net) as compared to 1 collision per 4 IP packets transmitted (42K collisions for 138K IP packets sent on Six-net and 48K collisions for 198K IP packets sent on Seven-net). This reduction in the collisions implied a better latency for all the hosts connected to the network and less bandwidth wasted for collisions on Ethernet.

B/G Load on		Parameters		Throughput	Pkts sent on	
Six-net	Seven-net	Slot	Sample	Mb/s	Six-net	Seven-Net
23%	23%	5000	50	7.9	138K	198K
23%	23%	50000	50	7.8	151K	184K
23%	23%	100	50	7.9	168K	163K

Table 5.8: Performance of IAA Under Bursty Loads on Ethernets

For the uniform and similar background loads on the two networks, it was observed that the adaptive scheduling algorithm effectively produces the performance similar to the round robin scheduling. The round-robin scheduling

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

gave 7.8 Mb/s throughput as compared to 7.9 Mb/s for IAA scheduling. Since the loads on both Ethernets was 23% uniform and continuous, the collision counts were also not very different. For the round-robin case, there were 50K collisions for 166K packets sent on the Six-net and 43K collisions for 166K packets sent on the Seven-net. For IAA adaptive algorithm, there were 47K collisions for 160K packets sent on the Six-net and 40K collisions for 171K packets sent on the Seven-net. Hence, the IAA algorithm resolves to round robin algorithm under uniform background load conditions.

For burst loads, the round robin assignment policy results in a larger number of collisions on each network than the adaptive policy. For a burst load of 23% on both channels, the round robin and IAA adaptive policy produce 8.0 Mbps throughput. A dramatic decrease, however, in the number of collisions from round robin to adaptive policy resulted. In case of the round robin policy, the number of collisions on the Six-net were 89K for 165 packets sent (i.e. 1 collision for every 2 packets) and those on the Seven-net were 73K for 165K packets sent (i.e., 1 collision for every 2 packets). But the adaptive policy resulted in 20K collisions on the six-net for 173K packets sent (i.e., 1 collision per 9 packets) and 24K collisions on the seven-net for 158K packets sent (i.e., 1 collision per 7 packets). This significant decrease in the number of collisions implies less waste of bandwidth and lower latency for hosts connected to the Ethernets.

B/G Load	Mb/s Achieved	Collisions (le0/le1)	Coll. as % of Pkts
10%	8.1	23K/18k	14%/11%
19%	8.1	38K/32K	24%/19%
23%	7.9	47K/40K	29%/23%
50%	6.4	77K/82K	36%/60%

Table 5.9: Parallel Ethernet Performance table for uniform and continuous loading of Ethernets

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

B/G Load	Mb/s Achieved	Collisions (le0/le1)	Coll. as % of Pkts
10%	8.1	20K/16K	12%/10%
19%	8.0	21K/24K	12%/14%
23%	8.0	21K/31K	12%/19%
30%	8.1	25K/29K	15%/18%
40%	7.5	30K/32K	17%/18%

Table 5.10: Parallel Ethernet Performance table for Bursty loading of Ethernets

For general interest, the performance of IAA scheduling algorithm under different background loads on the two networks is illustrated in Tables 5.9 and 5.10. From these tables, it can be observed that there were more collisions for the uniform background loads than the bursty loads. It can be seen that the IAA (see Table 5.10) manages to keep the collisions down at the cost of a small loss in throughput when very high bursty load conditions (40%) exist on the channels.

This prototype study is the first major study to employs parallelism applied at the lower layers of a communication system. Performing this prototype study helped to verify that scheduling of packets on multiple channels is an important issue. Under unbalanced and non-uniform conditions on channels, a simple policy like round robin can result in dramatic under utilization of the channels. Thus, some form of adaptive scheduling is needed to intelligently load balance the channels. This practical prototype demonstrated the feasibility of employing parallelism in networks and using this parallelism without requiring any changes in existing applications. This demonstration of possible integration is an important step towards multilevel parallel communications. This prototype study will maximize the potential for further interest in this technique.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 6

Conclusions

In this thesis, I have analyzed the use of multi-level parallelism to provide high speed communication services to a single application at a single network node. A general framework for coarse-grain parallelism appropriate for use in a high performance network node. This framework employs parallelism at all layers of ISO OSI model has been developed. A special case of this framework was adopted to implement parallelism at upper and lower levels. The upper level includes transport and network layers while the lower level includes data and physical layers. 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) I performed detailed simulation studies of three architectural instantiations of the model. These architectures are bus-based multiprocessor workstation nodes (true representatives of current technology). Mapping of the general model into concrete architectures required selection of scheduling algorithms and assigning processes (protocol and scheduling) to physical processors. To operate near the potential speeds possible using a particular architecture, this mapping reflected the communication fabric available in the underlying hardware.

Some general conclusions about the use of multilevel parallelism for high performance networking can be drawn.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. Multilevel parallelism can deliver more than 100 Mb/s with currently available hardware platforms, such as the Sun Sparc Station 10, with existing networking protocols, such as TCP and IP, and with parallel FDDI rings.

2. Scale up is near linear in the number of channels (at least up to a few hundred Mbps). This is more significant since these performance results were obtained from the use of current hardware architectures, existing protocols and MAC layer components not designed with high speed network applications in mind. Performance scale up with increase in number of processors is limited, but this phenomenon is due to the inherent capabilities of architectures, bus bandwidth and shared memory bandwidth, rather than the framework itself.

3. Since these results are based on existing hardware without specialized software (except perhaps for some simple modifications of the FDDI drivers), they represent a low cost solution to providing multiple 100 Mb/s on current machines.

4. The proposed parallel framework is flexible in a number of ways. This architecture can incorporate any number of existing or future protocol and hardware standards. This feature is enhanced further by the use of independent parallelism at upper and lower levels.

5. The use of multiple processors providing identical services and the use of space division multiplexing will provide better reliability than monolithic approaches. Additional benefits are graceful degradation and low-cost load balancing.

6. 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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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

7. The basic architecture is able to incorporate many improvements from other work such as, reduced data movement, fast TCP, gigabit nodes, and fine-grained parallelism, again with a near linear speed-ups as these improvements become available.

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

1. The Sun Sparc-10 based parallel processing architecture is capable of delivering throughput in excess of 100 Mb/s provided that sufficiently fast memory is available.

2. Scheduling and scheduler placement has significant impact on the performance.

3. A simple scheduling policy of network scheduling fails to push expected throughput at lower round trip delay under unbalanced channel load conditions.

4. Proper assignment of application data segments to available processors is of prime importance. Incorrect assignment may result in longer round trip delay per segment.

5. The distributed network scheduling architecture outperforms the centralized network scheduling architectures in terms of higher throughput and lower round trip delay.

6. The throughput capability of a shared memory architecture is limited by the memory bandwidth. This is true for distributed memory (i.e., local

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

memory per processor architectures) as well, but the combined memory bandwidth of entire system is large enough.

7. Under balanced load conditions, a distributed memory architecture will outperform the shared memory architectures.

Some general conclusions related to properties of protocols, when used in a parallel environment, can be made.

1. The segment size of the data flowing from the application to the transport layer must be the same size as MAC frame size to reduce round-trip delay time. A small segment size causes too much overhead in protocol processing meaning performance will degrade significantly.

2. The TCP window size should be such that none of the protocol processors starves for data and no packet waits extra for available TCP send window. This optimality of window size is an important criterion.

3. As window size increases beyond the optimal size, round trip delay increases for each examined architecture.

4. Distributed mode (DRDA) of retransmissions and acknowledgments handling outperforms centralized mode (CRCA).

5. With unbalanced background traffic, the adaptive assignment of packets to channels results in higher throughput than the round-robin policy. For the DRDA case, increasing the number of protocol processors decreases the throughput when round-robin scheduling is used. Throughput increases, however, when adaptive policy is used.

6. End-to-end delay is higher in CRCA than in DRDA.

7. The resequencing delay at TCP and at the application scheduler level is higher in the centralized case than the distributed case.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

8. Efficient implementation of the TCP timeout timers mechanism in the case of multiple parallel channels is nontrivial and requires further study.

9. Errors and Packet loss on channels can significantly affect the performance of the parallel system due to many retransmissions and improper timeout calculations.

10. On large diameter networks (with large latencies) parallel channels can be used with forward error correcting techniques, such as cross channel coding, to reduce latency and to significantly reduce the need for retransmission of data.

The general conclusion is that multilevel parallelism is effective for increasing the networking capabilities of currently available hardware and is a promising approach for building true high performance network nodes. It is compatible with, and complements, much of other work toward designing such nodes.

The development of a prototype to study parallelism at lower level is the first major study of parallelism applied to networking. Study of various assignment policies used to assign packets to channels was performed. A major conclusion out of this prototype study is the verification of the claim that simple channel assignment policies cannot result in load balancing on channels. Some intelligence has to used in doing this decision making process. This prototype has demonstrated the feasibility of parallelism in networks.

6.1 Future Directions

Future works should be pursued in two areas. First, protocol processing should be mapped on to newer workstation architectures where parallel CPU's are operational. These architectures are required in order to truly determine effec-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tiveness of processing protocol operations in parallel. Questions like what operations most effectively lend themselves to parallel process should be asked. This is a many faceted question; we note that we have previously studied parallel operations based upon parallel TCP/IP processing where as La Porta[77] and others[103, 50] have implemented parallel streams within a protocol structure. Second, how does architecture, and more importantly, how do various performance requirements influence this mapping of protocol processing to workstation architectures? By performance requirements, we mean that some data must be delivered in a stream, such as video, where as other can be delivered in a batch fashion even though embedded within is video or graphic data. This might affect the ordering and priorities which we give to various processing elements in the protocol stream. Finally, memory interface, which we have found in the work to date to be extremely important, may hold the real key as to how effective parallel protocol processing really will be. Since the memory bus effectively serializes memory references, the eventual key may be how can parallel operations be most efficiently implemented while maintaining consistency in the memory.

A second area for future work must consider parallel channel utilization and control. This is similar to the load balancing problem faced in parallel computers. Various traffic types exist, e.g., voice, video and data with various message sizes. There are many operational policies related to media access control and message submittal which can be used to support message balancing. For example, in previous work on token rings [70, 69, 35], it was postulated that as many as 10 different parameters could be adjusted to support a network's availability for traffic handling. When parallel channels are used the number of combinations of parameters which are available increases significantly. In that work, it was shown that running channels with different parameters was very effective in supporting multiple traffic types without degrading overall performance over a wide load range. In [35] an algorithm was presented which

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

provides good performance for CSMA/CD access, however, it has only been tested under limited operating and disturbance conditions. More work is required in order to document this algorithm's performance over a wide range of conditions, to compare it to other algorithms, and to determine the most satisfactory methods for using the network access and message submittal policies to support integrated traffic on parallel CSMA/CD channels.

Also in the context of continuing investigations with the prototype, the prototype should be enhanced such that there is more than one background sender and receiver pair. Also, measuring the speed with which the scheduling algorithm adapts properly to the changing background loads on the network will be good metric to compare scheduler performance.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Bibliography

[1] The VME Bus Specifications, 1985.

[2] A. J. Bernstein. Analysis of Programs for Parallel Processing. *IEEE Transactions on Computers*, EC-15:757–762, October 1966.

[3] B. Bhargava, T. Mueller, and J. Reidel. Experimental Analysis of Layered Ethernet Software. *Proceedings of FJCC, Dallas, Texas*, pages 559–568, 1987.

[4] D. R. Boggs, J. C. Mogul, and C. A. Kent. Measured Capacity of an Ethernet: Myths and Reality. *Proceedings of SIGCOMM'88*, pages 222–234, August 1988.

[5] M. Booth and K. Misegades. Microtasking: A New Way to Harness Multiprocessors. *Cray Channels*, pages 24–27, Summer 1986.

[6] W. E. Burr. The FDDI Optical Data Link. *IEEE Communications Magazine*, 24:18–23, 1986.

[7] J. W. Burren. Flexible Aggregation of Bandwidth for Primary Rate ISDN. *Proceedings of SIGCOMM'89*, pages 191–196, 1989.

[8] V. G. Cerf and E. Cain. The DoD Internet Architecture Model. *Computer Networks*, 7:307–318, Oct. 1983.

[9] D. Cheriton. VMTP: A Protocol for the next Generation of Communication Systems. *Proceedings of ACM SIGCOMM'86*, pages 406–415, August 1986.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[10] D. Cheriton. VMTP: Versatile Message Transaction Protocol - Protocol Specifications. *RFC 1045, NIC*, February 1988.

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

[12] G. Chesson and L. Green. XTP-Protocol Engine VLSI for Real-Time LANS. *Proceedings of EFOC/LAN*, pages 1–4, July 1988.

[13] G. L. Chesson. The Protocol Engine Project. *UNIX Review*, pages 70–77, Sept. 1987.

[14] K. W. Cheun. Electronic Wavelength Tuning Using Acoutooptic Tunable Filters with Broad Continous Tuning Range and Narrow Channel Spacing. *OFC'89 Technical Digest*, page 155, February 1989.

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

[16] D. D. Clark. The design Philosophy of DARPA Internet Protocols. *Proceedings of ACM SIGCOMM '88*, pages 106–114, Aug. 1988.

[17] D. D. Clark, V. Jacobson, J. Romkey, and H. Salven. An Analysis of TCP Processing Overhead. *IEEE Communications Magazine*, 27:23–29, 1989.

[18] D. D. Clark, M. L. Lambert, and L. Zhang. NETBLT: A Bulk Data Transfer Protocol. *RFC 998*, 1987.

[19] A. Cockroft. *Sun Performance Tuning Overview.* Sun Microsystems Inc., 1991.

[20] D. E. Comer. *Internetworking with TCP/IP: Principles, and Architectures.* Prentice Hall, 1988.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[21] J. Davidson. *Introduction to TCP/IP.* Springer Verlag, 1988.

[22] Definicon Systems, Inc. *The Parallel Transputer System: Operation and Installation Manual, Rel. 2.0*, 1988.

[23] Digital Equipment Corporation. High Performance TCP/IP for OSF/1 Alpha AXP Workstations. *White Paper*, March 1993.

[24] D. Dykeman and W. Bux. Analysis and Tuning of the FDDI Media Access Control Protocol. *IEEE Transactions on Communications*, pages 997–1010, July 1988.

[25] A. E. Baratz et al. SNA Network of Small Systems. *IEEE Journal on selected areas in Communications*, SAC-3:416–426, May 1985.

[26] A. S. KrishnaKumar et al. Translation of Formal Protocol Specifications To VLSI Design.

[27] D. A. P. Mitchell et al. Inside the Transputer, 1990. Blackwell Scientific Publications.

[28] E. C. Cooper et al. Protocol Implementation on the Nectar Communication Processor. *Proceedings of ACM SIGCOMM '90*, pages 135–144, 1990.

[29] K. Annamalai et al. FDDI Chips: The Dawn of a New LAN. *ESD*, October 1987.

[30] R. J. Sundstrom et al. SNA Directions-A 1985 Perspective. *Proceedings of National Computing Conf.*, pages 589–603, 1985.

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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[32] J. G. Fletcher and R. W. Watson. *Computer Networks and Simulation II*, Mechanisms for Reliable Timer-Based Protocol, pages 271–290. North-Holland, 1982.

[33] M. J. Flynn. Very High Speed Computers. *Proceedings of the IEEE*, 4:1901–1909, December 1966.

[34] E. Foudriat, K. Maly, C. M. Overstreet, S. Khanna, and F. Paterra. Control Access Through Local Carrier Sensing for High Data Rate Networks and Control of Access of Synchronous Messages Through Circulating Reservation Packets, 1990. Patent awarded.

[35] E. C. Foudriat, K. J. Maly, R. Mukkamala, and C. M. Overstreet. Parallel Networking: Improving Capacity and Multi-traffic Performance Together. Technical report, Old Dominion University, Norfolk, VA 23529, February, 1993.

[36] E.C. Foudriat, K. Maly, C. M. Overstreet, S. Khanna, and F. Paterra. Carrier Sensed Multiple Access Protocol for High Data rate Ring Networks. *Computer Communications Review, Volume 21, Number 2*, pages 59–70, April 1991.

[37] E.C. Foudriat, K. Maly, C. M. Overstreet, S. Khanna, and L. Zhang. Combining Two Media Access Protocols to Support Integrated Traffic. *Proceedings of 16th Local Computer Networking Conference in Minneapolis*, October 1991.

[38] E. H. Frank and W. M. Bradley. *SBus Specification A.1.* Sun Microsystems Inc., 1990.

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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[40] A. G. Fraser and W. T. Marshall. Data Transport in a Byte-Stream Network. *IEEE Journal on selected areas in Communications*, SAC-7:1020–1033, Sept. 1989.

[41] D. Game and K. Maly. Improvements for the Next Generation of FDDI. *Proceedings of the International Phoenix Conference on Computers and Communications*, pages 710–716, March 1991.

[42] General Electric. CSF/DRIVE Control System Freeway for Drive System Applications.

[43] Z. Haas. A Communication Architecture for High-Speed Networking. *Proceedings of IEEE Infocom*, pages 433–441, 1990.

[44] Z. Haas. A Protocol Structure for High-Speed Communication over Broadband ISDN. *IEEE Network Magazine*, 1991.

[45] A. Heybey. The Network Simulator.

[46] IBM. Systems Network Architecture LU6.2 Reference, 1988.

[47] I. Jacobs. Integrated Parallel Architecture Lightwave Systems. *Proceesdings of 6th IEEE International Workshop on Microelectronics and Photonics in Communications*, June 1989.

[48] I. Jacobs and H. Leung. Survivable Optical Networks. *Proceedings of the 6th MCI Technology Symposium*, pages 73–82, October 30, 1989.

[49] I. Jacobs and H. Leung. Survivable Optical Networks. *Proceedings of the 6th MCI Technology Symposium*, pages 73–82, October 30, 1989.

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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[51] R. Jain. *The Art of Computer Systems Performance Analysis*, Analysis of Simulation Results, pages 413–436. John Wiley & Sons, Inc., 1991.

[52] M. J. Johnson. Proof that Timing Requirements of the FDDI Token Ring Protocol are Satisfied. *IEEE Transactions on Communications*, 35:620–625, June 1987.

[53] S. Joshi. High Performance Networks: A Focus on the Fiber Distributed Data Interface (FDDI) Standard. *IEEE Micro*, pages 8–14, June 1986.

[54] H. Kanakia and D. R. Cheriton. The VMP Network Adapter Board(NAB): High Performance Network Communication for Multiprocessors. *Proceedings of SIGMETRICS '88*, 1988.

[55] S. Khanna, E.C. Foudriat, C. M Overstreet, F. Paterra, and K.J. Maly. Modelling High Data Rate Communication Network Access Protocol. *Proceedings of 21st Annual Pittsburgh Conference on Modeling and Simulation*, May 1990.

[56] K. Kobayashi and I. Mito. Single Frequency and Tunable Laser Diodes. *Journal on Lightwave Technology*, pages 1623–1633, November 1988.

[57] Lawerence Livermore Laboratory. Delta-t Protocol Specifications, 1983.

[58] J. L. Larso. Multitasking on the Cray XMP-2 Multiprocessor. *IEEE Computer*, pages 62–69, July 1984.

[59] M. Le. *FDDI-II Working Paper.* National Semiconductor, June 1, 1986.

[60] K. Maly, I. Jacobs, C. M. Overstreet, R. Mukkamala, M. Zubair, E. Foudriat, J. Wiencko, and S. Midkiff. Parallel Network Communications. Technical Report 89-46, Old Dominion University, 1989.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[61] K. Maly, S. Khanna, R. Mukkamala, and C.M. Overstreet. Parallel TCP/IP for Multiprocessor Workstations. *Proceedings of HPN '92, Leige, Belgium*, pages C1–1 – C1–16, 1992.

[62] K. Maly, S. Olariu, L. Zhang, and N. Rao. A Fairer DQDB. *Proceedings of International Conference on Parallel Computing*, 1991.

[63] K. Maly, C. M. Overstreet, R. Mukkamala, M. Zubair, F. Paterra, S. Khanna, Y. S. Sekhar, and R Yerraballi. Parallelism for High Speed Networks, 6pp. *Proceedings of IEEE workshop on the architecture and implementation of high performance communication subsystems*, February 1992.

[64] K. J. 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.

[65] K. J. Maly, S. Khanna, C. M. Overstreet, R. Mukkamala, M. Zubair, Y. S. Sekhar, and E. C. Foudriat. Scalable Parallel Communications. Technical report, Old Dominion University, Norfolk, VA 23529, February 4, 1992.

[66] J. Martin. *SNA: IBM's Networking Solution.* Prentice Hall, 1987.

[67] H. E. Meleis and D. N. Serpanos. Memory Management in High-Speed Communication Subsystems, 4pp. *Proceedings of IEEE workshop on the architecture and implementation of high performance communication subsystems*, 1992.

[68] H. E. Meleis and A. H. Tantaway. The Modular Communication Machine(MCM): An architecture for High Performance Networks. Technical Report Research Report RC 14541, IBM Research Division, 1989.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[69] R. Mukkamala, E. C. Foudriat, K. J. Maly, and V. Kale. Modeling and Analysis of High-speed Parallel Token Ring Networks. *Proceedings of the 17th Annual Conference on Local Computer Networks, Minneapolis, MN, September*, 1992.

[70] R. Mukkamala, E. C. Foudriat, K. J. Maly, V. Kale, Y. S. Sekhar, and R. Yerraballi. Message Assignment Policies for High Speed Parallel Networks. *Proceedings of the 1st International Conference on Computers and Communications*, pages 77–81, 1992.

[71] A. N. Netravali, W. D. Roome, and K. Sabnani. Design and implementation of a high-speed transport protocol. *IEEE Transactions on Communications*, 38:2010–2024, 1990.

[72] D. A. Padua and M. J. Wolfe. Advanced Compiler Optimizations for Supercomputers. *Communications of the ACM*, 29:1184–1201, December 1986.

[73] A. Paranjape. A Fault-tolerant Bus Interface Unit Based on Nubus Standardized Bus Architecture. Master's thesis, Virginia Ploytechnic Institute and State University, 1988.

[74] F. C. Paterra. *Approaches to High Speed Networks* PhD thesis, Old Dominion University, Norfolk, VA 23529, 1991.

[75] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). *ACM SIGMOD Conference*, pages 106–116, June 1988.

[76] T. F. La Porta. Design, Verification, and Analysis of a High Speed Protocol Parallel Implementation Architecture, 5pp. *Proceedings of IEEE workshop on the architecture and implementation of high performance communication subsystems*, 1992.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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

[78] T. F. La Porta and M. Schwartz. A High Speed Protocol Parallel Implementation: Design and Analysis. *Proceedings of HPN '92, Leige, Belgium*, pages C3–1 – C3–16, 1992.

[79] T. La Porta and M. Schwartz. Design, Verification and Analysis of a High Speed Protocol Parallel Implementation Architecture, 6pp. *Proceedings of IEEE workshop on the architecture and implementation of high performance communication subsystems*, February 1992.

[80] J. B. Postel. Internet Protocol Approaches. *IEEE Transactions on Communications*, COM-28, April 1980.

[81] D. K. Pradhan. *Fault-Tolerant Multiprocessor and VLSI Based System Communications Architectures.* Prentice Hall, 1986.

[82] C. M. Ragdale. Narrow Band Fiber Grating Filters. *OFC'89 Technical Digest*, page 76, February 1989.

[83] K. K. Ramakrishnan. Performance Issues in Designing Network Interfaces: A Case Study. *Proceedings of HPN '92, Leige, Belgium*, pages F3–1 – F3–16, 1992.

[84] N. Rao, K. Maly, S. Olariu, L. Zhang, and D. Game. Average Waiting Time: Profiles of DQDB. *Proceedings of International Conference on Parallel Processing*, 1991.

[85] F. Ross. FDDI - a Tutorial. *IEEE Communications*, 24:10–17, May 1986.

[86] T. J. Routt. Distributed SNA: A Network Architecture Gets on Track. *Data Communications*, pages 116–134, February 1987.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[87] M. Rupprecht. *Petrinetze als Organisationsprinzip fur einen kommunikations-Controller.* Springer-Verlag, 1989.

[88] A. A. M. Saleh and J. Stone. Two Stage Fabry Perot Filters as Demultiplexors in Optical FDMA LANs. *Journal on Lightwave Technology,* pages 323–330, February 1989.

[89] D. Siewiorek. *Fault Tolerant Computing – Theory and Techniques.* Prentice Hall, 1986.

[90] P. A. Steenkiste, B. D. Zill, H. T. Kung, and S. J. Schlick. A Host Interface Architecture for High Speed Networks. *Proceedings of HPN '92, Leige, Belgium,* pages A3–1 – A3–16, 1992.

[91] Sun Microsystems Inc. *MBus Specification: White Paper,* 1992.

[92] D. L. Tennenhouse and I. M. Leslie. A Testbed for Wide Area ATM Research. *Proceedings of SIGCOMM'89,* pages 181–190, 1989.

[93] S. Thomas. FDDI Architectural, Software, and Simulation Considerations for High-Performance Networks. *FEORC Review Conf.,* April 25, 1989.

[94] D. Tolley. FDDI Hardware and Firmware Interface Considerations for High-Performance Networks. *FEORC Review Conf.,* April 25, 1989.

[95] P. Vaidyanathan and S. F. Midkiff. Performance Evaluation of Communication Protocols for Distributed Processing. *Computer Communications,* 13(5):275–282, June 1990.

[96] R. W. Watson. *Computer Networks and Simulation II,* Timer Based Mechanisms in Reliable Transport Protocol Connection Management, pages 47–56. North-Holland, 1982.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[97] R. W. Watson and S. A. Mamrak. Gaining Efficiency in Transport Services by Appropiate Design and Implementation Choices. *ACM Transactions on Computer Systems*, 5:97–120, 1987.

[98] J. A. Wiencko. Fiber Optic CSMA/CD Network Approaches. *Presented to IEEE 802.8 and 802.3 Standard Committee*, 1987.

[99] J. A. Wiencko. SAFENET Committee Presentations. *NASA Technical Briefs*, 1990.

[100] J. A. Wiencko. Cross-Channel Coding Implementation Guide, November 1991.

[101] D. Winkelstein and D. Stevenson. HiPPI Link Data Analysis System: Test Equipment for High Speed Network Analysis. *IEEE*, 1991.

[102] N. Wirth. A Note on the Program Structure for Parallel Programming. *Communications of the ACM*, 9:220–231, May 1966.

[103] M. Zitterbart. High-Speed Transport Components. *IEEE Network Magazine*, 4:54–63, 1990.

[104] M. Zitterbart. Parallel Protocol Implementations on Transputers - Experiences with OSI TP4, OSI CLNP, and XTP, 4pp. *Proceedings of IEEE workshop on the architecture and implementation of high performance communication subsystems*, February 1992.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Autobiographical Statement

I, Sanjay Khanna, was born on December 24, 1965 in New Delhi, India. I earned B.E. (Electronics and Telecommunications) in August 1987 from Delhi College of Engineering, Delhi University, India. I worked for two years as systems engineer in Bharat heavy Electricals Limited, India. In Fall of 1989, I joined Old Dominion University for M.S. (Computer Science). In Spring of 1991 I was admitted to PhD program too. I completed my M.S. in December 1991. All through my graduate studies at Old Dominion University, I was supported through research assistantships. The research reported in this thesis was funded in part by NASA, CIT, SUN and DARPA.

Address: Department of Computer Science
Old Dominion University
Norfolk, VA 23529
U.S.A.

This dissertation was typeset with LaTeX[‡] by the author.

[‡]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.