ANALYSIS OF THE HARDWARE REQUIREMENTS OF A HIGH SPEED COMPUTER INTERFACE REQUIRED TO UTILIZE FIBER DISTRIBUTED DATA INTERFACE

by

Dan Bruce Tolley

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

APPROVED:

Richard O. Claus

R. O. Claus, Chairman

J. K. Burton

I. Jacobs

C. E. Nunnally

S. F. Midkiff

November, 1990

Blacksburg, Virginia

ANALYSIS OF THE HARDWARE REQUIREMENTS OF A HIGH SPEED COMPUTER INTERFACE REQUIRED TO UTILIZE FIBER DISTRIBUTED DATA INTERFACE

by

Dan Bruce Tolley

Committee Chairman: Richard O. Claus
Electrical Engineering

(ABSTRACT)

As the use of computers in the workplace becomes more commonplace, the levels of interconnection and interoperability increases. The desire to pass large amounts of data almost instantaneously is the basis of the high speed local area network (LAN). To meet the needs of these LANs, the American National Standards Institute has developed the Fiber Distributed Data Interface (FDDI) standard. This new LAN can provides high speed fiber optic based communication between computers.

In meeting the computer/LAN interface requirements, new methods for data transfer will be required. Trade-offs between the reliability, architecture and buffer sizes must be developed. These concepts must include variables of data transfer widths, protocol processing, transfer architecture and packet length distributions.

This dissertation addresses these hardware requirements in using the high speed computer interface known as the Fiber Data Distributed Interface.

Acknowledgements

I would like to begin by thanking the members of my committee for their time and effort in making my doctorial time at Virginia Tech both enlightening and enjoyable. Most especially, to Dr. R.O. Claus, my committee chairman for his time, support and effort. I would like to also thank the members of the Fiber and Electro Optic Research Center for their help. Your friendship and support helped me through the period.

Most importantly, I would like to thank my family, Fred, Guida, Gary, Denise, Diane and Neil for their support throughout this period. I would not have made it without your encouragement. A special thanks to Fred, for reading, commenting and correcting the dissertation in its many states.

Table of Contents

Introduction

In recent years, the use of computers in all areas of the workplace has increased dramatically. As one result of the use of personal computers and workstations, the informational data base has been decentralized. To allow the sharing of data, program and resources, computers are being interconnected by computer networks. The interconnection of these computers increases the ability to transfer data, share data and to share programs. Conventional electrical network data transfer capabilities are not adequate to keep pace with the required intercomputer data transfer requirements. With present and future requirements, other data transfer media must be found. This dissertation addresses the hardware requirements required to utilize the high speed computer interface defined by the Fiber Distributed Data Interface standard.

In late 1985, the American National Standards Institute (ANSI) formed a committee of industry engineers to develop the first fiber optic intercomputer communication standard. Since that time, this committee has been developing the Fiber Distributed Data Interface (FDDI) standard, also known as the X3T9 standard. In 1990, this standard is in its final stage of revision and amounts to four sections within the standard. Three of the four sections which comprise the FDDI system are

currently approved by ANSI.

Many of the companies that helped design the FDDI standard are introducing interfaces between the fiber optic interface and the host computers. The new interface makes possible an order of magnitude increase in the data transfer rate as compared to the prevalent electrical interconnection standard, Ethernet (trademarked by Xerox Corp.). This order of magnitude increase in data rate has been achieved in the physical interconnection, but in many cases the interface to the computer has not changed significantly. This increase in the data rate also increases the amount of activity required by the computer to send, receive and transfer data. In other words, to handle data at an order of magnitude increase in data rate requires an order of magnitude increase in the computer's responses to data. Because of the order of magnitude increase in processor load required to acknowledge and process the 100 megabits per second (Mbit/s) of data available over FDDI, the host computer spends more of its resources on input/output (I/O) processing compared to what was traditionally spent on similar work over network systems such as Ethernet. To reduce the amount of host computer resources expended on I/O processing, it is desirable to use I/O processor(s) to relieve the host computer of the burden of directly interfacing with the FDDI interface. The exact method of the hardware interaction between the host computer

and the I/O processor(s) will be the main topic under examination.

Vendors are designing FDDI/VME-bus interfaces that could be used as research testbeds. The theoretical design used in the Sun Microsystems FDDI system provides a baseline that is used to develop models of hardware and software. The data available from papers, texts and operational manuals will provide the data used in the analysis. With the establishment of baseline models, a variety of hardware configurations can be modelled to evaluate performance and effects of changes to future systems.

Material presented focuses on hardware considerations and timing analyses to meet the requirements of the Star VP-Series vector processor when connected to the Fiber Distributed Data Interface. Data patterns defined by Star Technologies Inc. are used in an example of how to use the performance models to determine the system requirements. These requirements are used to evaluate hardware models as well as hardware usage design criteria. The development of the hardware requirements and assumptions of the design criteria are discussed to allow the reader an analytical example with which to analyze FDDI systems. In particular, the following items are examined.

1) Throughput comparisons model the effect of different

I/O microprocessor architectures used in the system. The comparison is made between the effects of using typical reduced instruction set computer (RISC) versus complex instruction set computer (CISC) based architectures on data movement and protocol acknowledgement at the data and instruction rates required by the FDDI standard. Specifically, it compares run-time attributes of the Intel 80386 and the Sun SPARC microprocessors.

2) Throughput comparisons model the effect of different data transfer architectures used between the FDDI interface module and the I/O processor. The two architectures compared are 1) a shared memory architecture and 2) a backplane access architecture (i.e. VME-bus).

3) Throughput comparisons model the method of segmenting and moving data. The model examine the throughput efficiency for variable length data strings and fixed length data strings based on the VP series data elements. The model also examines the memory sizes required to meet the FDDI data rate for various microprocessor clock rates, blocking parameters and data distributions.

4) Throughput comparisons of each of the above three items examined to determine their impact on the total I/O system for a defined FDDI architecture and topology. These items are used to further predict maximum throughput available for FDDI with the defined configuration.

The first chapter contains a short discussion of the requirements of computer interconnections in use today, a description of the FDDI system and a discussion on how FDDI meets the needs of current and future computer systems.

The second chapter focuses on the demands of the FDDI system and the requirements of the I/O processor to meet those needs. The demands and requirements focus on the general background of FDDI and its data throughput requirements.

The third chapter examines microprocessor architecture, data access architectures, data segmentation, and memory sizes, and the ways these items are effected by data throughput requirements. This section examines the relationship of these items in meeting system performance criteria. The measure of performance is evaluated by the probability of a message not being processed when it has been transmitted by a high speed network, also known as the blocking probability [1].

The fourth chapter discusses the specific requirements of the Star VP-Series of computers, demonstrates the use of the models developed in Chapter Three on a specific system and provides design suggestions based on information from Star Technologies, Inc.

The final chapter is the summary chapter. It discusses the results of the work and possible future work based on this research.

Chapter 1

Significance of the Problem

In recent years, the use of computers in all areas of the workplace has increased dramatically. The use of personal computers and workstations has resulted in the decentralization of information and processing. To allow sharing of these distributed computer and information resources, computer systems developed in the later part of the 1980s are often operationally interconnected. The interconnections are used to share hardware and/or software resources. These resources can be in the form of computational media, storage media, output devices (printers), data, programs, or any other items associated with computers. Due to this functional interconnectivity, system constraints demand that all parts of the system be available. Therefore, computer systems need to have rapid and fault tolerant interconnection networks. The high speed, fault tolerant computer networks are more important today than ever before.

A common interconnection architecture used to meet the shared resource requirement is the local area network (LAN). Local area networks provide both hardware and software interconnection media. The hardware interconnection, also called the physical interconnection, provides the actual

method of exchanging bits of data between machines. The physical connection defines electrical, optical or mechanical thresholds that allow data transfer. The software interconnection assists the hardware in transforming the physical bits of data into information. This information is passed to locations within the computer(s) to provide data and/or instructional information. The software and hardware combine to form an interconnection between computers and the associated information control system. The LAN provides a common hardware and software definition to allow the machines to share resources.

In general, most LANs provide a physical means of interconnecting a large number of computers for transferring data. The ability to selectively deliver messages (data streams) to selected computers is performed by the use of addressing techniques. Within the LAN, these addresses theoretically allow any machine to communicate with any other machine. When sending messages along the physical interconnection, a single sender can often address single or multiple receiving devices for a message. Currently, LANs are used in all types of environments where multiple computers or computer peripherals exist. For example, "A number of office type systems have been built which connect small computers or work stations to a disk server. Such systems typically use 1 to 10 Mbit/s [megabits/second] LANs."[2] The advantage of

the LAN is that it allows resources to be shared between many users without making each user have their own complete complement of dedicated resources. The LAN has allowed operational interconnectivity for computer systems in the 1980s to become a common practice.

Although the average computation power per seat[1] in a system has increased over the past decade, the expectation of the associated computer capability has also increased. To meet the requirements of his work, the end user, who often uses a workstation, will desire connectivity to a mainframe computer or supercomputer. The larger computer will be accessed when execution of large computational programs is required. To utilize the large computer and to justify the additional expense, such a computer will serve a number of users. An example of this type of machine is the Star Technologies VP-series vector processor. The interconnections to a vector processor will most commonly be performed using a LAN. The interconnection of such a computer to a network allows smaller computers to utilize the higher speed computation when desired. The use of a high speed, highly reliable LAN between the distributed workstation and the vector processor provides each workstation with the capability

[1] A seat is a location from which a person may access a network and perform work. This term was first used to explain that no matter where the person physically was seated to access a network, the network always appeared the same to the person.

to perform data intensive computations at a fraction of the cost of obtaining dedicated vector processors. Future plans for the vector processor, which are beyond the scope of this examination, are to use the computational power of the machine to serve either dedicated machines or networks of machines and requires major modifications of the VP-series software.

Local area networks capable of interconnecting multiple computers have existed for a considerable amount of time. Ethernet "is the most widely used local network technology [or LAN] for distributed systems."[3] It provides a maximum throughput capability of 10 Mbit/s. Although 10 Mbit/s is a non-trivial throughput capability, today's "Modern mainframe computers have nearly insatiable appetites that ... far exceed the capacity of a 10 Mbit/s network."[2] For example, the Star Technologies VP-series vector processor can utilize 80 Mbit/s continuously. For this reason, many system designers find this 10 Mbit/s data transfer rate of LANs too slow for large systems or for systems sharing large data files. To attempt to provide a higher data rate transfer interconnection scheme, the American National Standards Institute (ANSI) formed the X3T9.5 committee to produce a 100 Mbit/s fiber optic token ring architecture. The fiber optic token ring architecture defined by the X3T9.5 committee is called the Fiber Distributed Data Interface (FDDI) [4].

The use of a fiber optic interconnection was chosen by the ANSI committee because it provides high signal quality at the receiver devices. Binary digital signal quality is defined as the ability to distinguish between the two logical states of information signals. A fiber optic serial bit stream operating at 100 Mbit/s, such as used in FDDI, has very high signal quality, allowing bit error rates of 10^{-12} or less, and is considered relatively easy to achieve in theory and practice. In contrast, high quality electrical interconnections, the most common LAN medium in use today, attempting to operate at a serial transfer rate of 100 Mbit/s are extremely complex in theory and even more difficult to build. The use of electrical signals in high speed interconnections involves design of controlled impedance interconnections, matched device impedances and an understanding of crosstalk and reflections. "While it is possible to achieve this design goal [100 Mbit/s] over electrical media, fiber optic waveguides have impressive advantages for such links.... Unlike electrical media, optical fibers are immune to electrical interference, do not radiate, and reduce electrical safety concerns."[5] In other words, serial electrical communication at 100 Mbit/s for "reasonable" interprocessor interconnection length, for example, upwards of 100 meters, is much more complicated and costly than a similar speed, fiber optic interconnection. Since the use of fiber optics in high speed computer

applications is the topic under examination, similarities with electrical systems will be based on functionality, not speed or signal quality.

In addition to the speed requirements of current computer systems, there is a need for dependable LANs which will give users access to all the devices on the LAN at all times. The loss of a communication link to the network could result in users being unable to access the files or compute elements necessary to perform work. The initial solution is to replicate data and processors, but this is expensive and time consuming. While the expense may be easy for a person to brush aside as insignificant when compared to the cost of non-performance, the cost of system maintenance for multiple data bases and programs is not as easy to dismiss. Examine a simple network in Figure 1.1. It contains four items. They are a supercomputer for high speed computation, a file server with related disks, and two workstations. In a general scenario, a user could sit at either workstation, access the file server for the programs and data bases, manipulate the data on the workstation and run large tasks on the supercomputer. In Figure 1.2, each machine has its own copy of the programs and data. If a workstation were to lose its connection to the system, to the user it still appears to be attached. The user accesses his data, manipulates it, and runs the task (actually it is run on the workstation because

the program cannot access the supercomputer). Work progresses well until, as is common in today's engineering environment, the user moves to another "seat" or workstation. When the user attempts to access his work, it is lost! In the best case, he realizes that the link had been broken on the previous editing session (the previous machine was not connected to the LAN) and he quickly exits the program. By now, the network has been repaired and the file on the network has a newer revision date than the "good" data. The user now has real problems. If the system searches for the newest data, the wrong file will be used. If the system does not automatically access the newest file, the user must be able to direct the file search to the previously disconnected station. In either case, the system requires each user to have intimate knowledge of the system, or for the system manager to have intimate knowledge of every file and piece of work performed on the network. Either case is unacceptable. The problem of managing multiple copies of data bases is difficult to plan and execute. Compounding this problem with varying topology, due to unreliable links, makes the management of such a system an impossible task. To provide manageable data systems which are always available to the user requires an ultra reliable network. To meet today's needs, we need a high speed, ultra-reliable computer network.

In addition to speed and reliability, interconnections

need to be able to span considerable distances. In the 1970s, most companies had "computer rooms" which were dedicated to housing computer systems. One major advantage with this philosophy was that the distances between computers was very short. Today, with the availability of workstations and personal computers, computers are located in the workplace to promote usage, not to facilitate interconnection distances. As networks grow, the distance between nodes and the number of nodes on a single network also increase. It is easy to see that as a company grows, management may desire to keep interconnections to the network common. This allows any user to perform his or her work from any node on the network. Not only have we now introduced problems with distance and the quantity of nodes, we have now introduced electrical concerns. Assume that two nodes are located on separate floors of a building. This probably means that the computers are on different power supplies, lines and possible different transformers. Since most computer interconnections are designed for systems using a common ground and power reference, each system interconnecting to the LAN needs to be electrically isolated from the system. And, as the distance between nodes increases, the need to limit crosstalk and reflections on lines increases. These problems are not unusual, and they can all be solved. In fact, many of these problems have already been solved at many locations. However, meeting all the interconnection issues of high speed, high

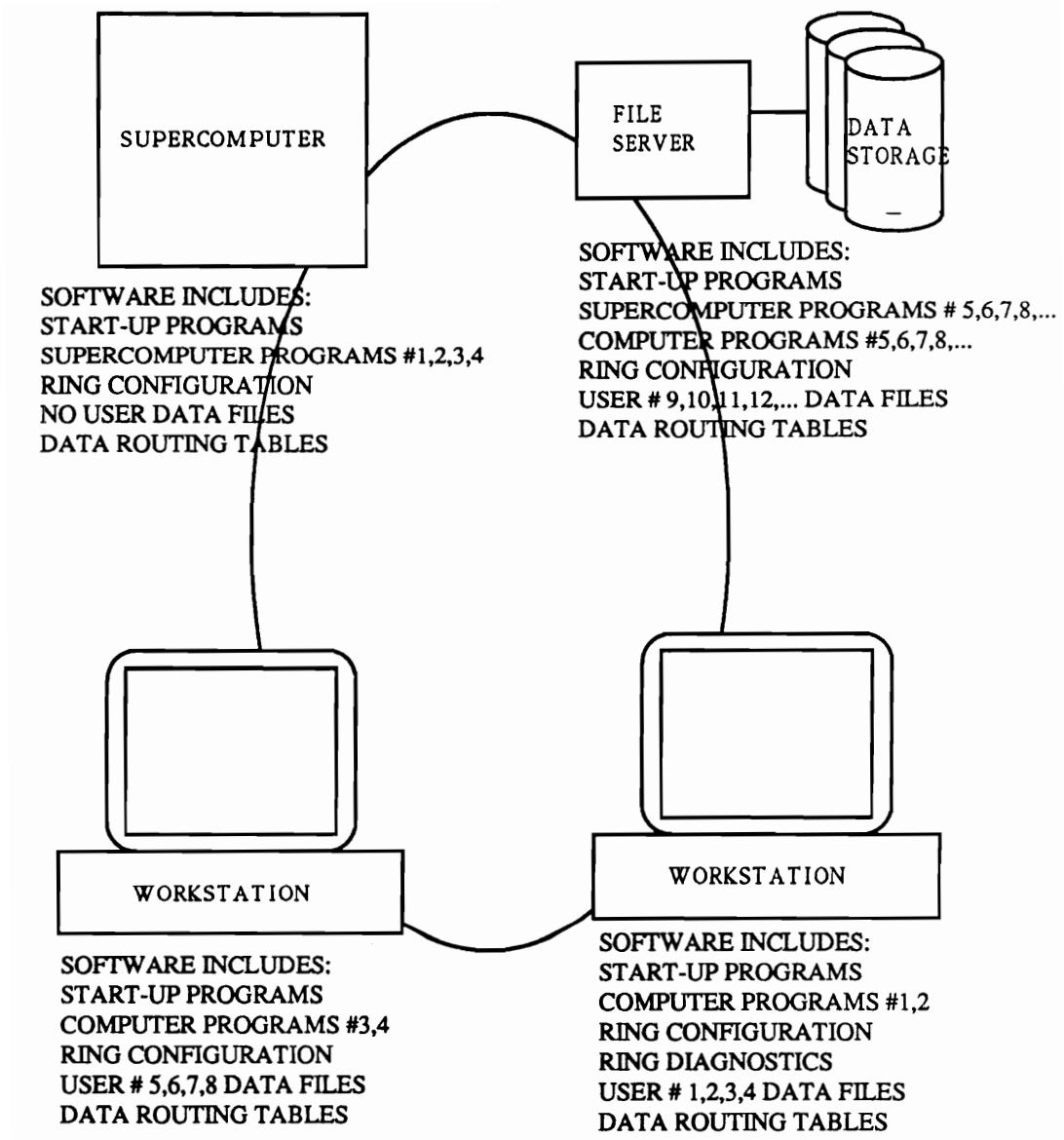


Figure 1.1: Example of the data resources of a system where data is shared over the network. Note that the application programs and data files reside only once on the ring. All data is shared between the supercomputer, file server and workstations. Sharing the data minimizes the amount of room required to store the data and decreases the amount of work required to keep data stored for accurate system operation.

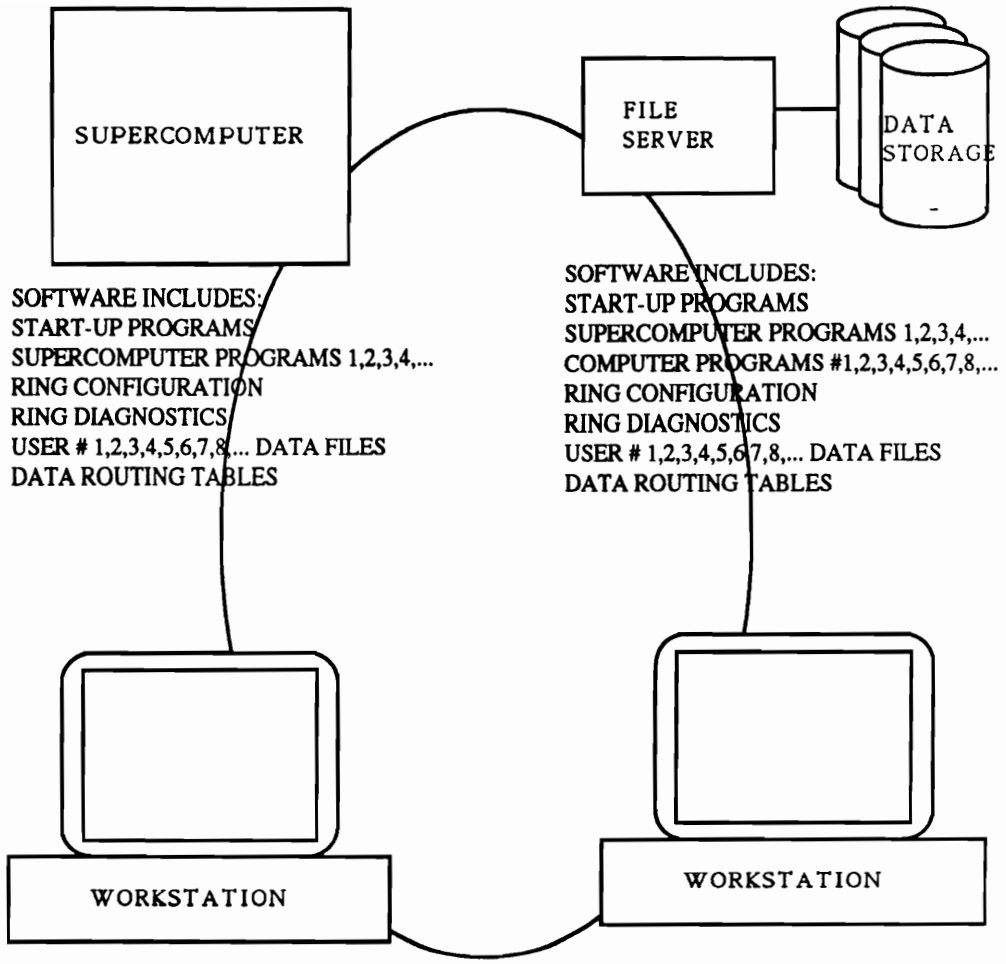


Figure 1.2: Example of the data and program distribution of a network where data is not shared. Note the application programs and data files reside on each node in the ring where they may be used. All data must be updated at each node whenever any changes take place in the data or programs. Redundant copies of data require additional storage room (as compared to single storage) and increase the amount of data base management required to keep all elements accessible and current.

reliability, electrical isolation, and long distance transmission, at once is extremely difficult to perform with electrical LANs.

The need is to provide a system that can connect large numbers of stations over considerable distances with electrical isolation between the computers or between the computer and the LAN, and to provide high speed communications with ultra-reliability. Not a simple task, but the FDDI is attempting to perform all of these tasks.

The physical interconnection required to form a high speed ultra-reliable LAN is accomplished by FDDI. To be effective, the new computer needs the physical interconnection hardware and the architecture to utilize the FDDI data rate. The utilization of the data rate requires the interfacing computers to pass a reasonable part of the 100 Mbit/s data rate available from FDDI. Current systems for networking have trouble sustaining Ethernet data rates of 10 Mbit/s. To utilize the 100 Mbit/s data rates offered by FDDI, changes in this interface must include hardware modifications and software modifications. The current systems examined are using the basic building components of Ethernet interfaces. At the beginning of the project the only company willing to discuss current file transfer rates, did so in private, and had its system operating at a continuous data rate of 6

Mbit/s[2]. This indicated that the system is achieving six percent of the maximum bandwidth of the fiber optic interface. It is clear that if this is the maximum throughput, the interface media (fiber optics) is not the limiting speed device, rather the interface and software are the limitations. This dissertation will focus on architectural considerations for a specific computer interface to better utilize the data rate of FDDI. Since the initial contacts, far superior throughput rates have been reported, but there was no data about architectures and analysis or test methods used in computing the throughput values.

1.1 Reasons Causing the Usage of Fiber Distributed Data Interfaces (FDDI) as the Interface Media

In the previous section, the need for high speed, e.g. on the order of 100 Mbit/s, ultra-reliable local area networks was discussed. This LAN would also have the desirable traits of being able to connect large numbers of stations over considerable distances and provide protection from ground loops, crosstalk, reflections and other problems associated with electrical transmission where distances are large with

[2]This company gave this information in confidence to FEORC. Note, in comparison, the generally accepted continuous data rate for Ethernet is 3 Mbit/s.

respect to the communications rate or caused computers to exist on different power distribution systems. These requirements were recognized by the ANSI X3T9.5 committee that developed FDDI. As reported in "An Overview of FDDI" by Burr and Zuqui,

> "In 1982 Floyd Ross, Jim Hamstra, and Kurt Molton, all then with Sperry Corporation, presented X3T9.5 with a detailed proposal for a 100 Mbit/s fiber optic token ring. This proposal resulted in the present FDDI standards effort. FDDI started its life as a 'back end' network for storage systems because the storage and main frame computer industries were the first to perceive a need for a high bandwidth LAN, and the advantages of fiber for this. As time has progressed, however, the application space for FDDI has grown dramatically, as other interests have joined the effort, and today it is clear that FDDI will be widely used as a backbone for other LANs, as a network for high performance workstations, as a 'real time' network for aerospace and military applications...."[6]

Because fiber optics provide extremely high bandwidth, low loss and electrical isolation, its use in advanced LANs was a predictable step in the evolution of LANs. The 100 Mbit/s data rate for FDDI allows the use of fairly simple fiber optic devices. The optical devices used by FDDI are based on the 1300nm wavelength and transmition through multimode fiber. This part of the technology has been in use for several years. The reliability of the devices is very high, and when used with the counter rotating ring architecture of FDDI, the system is very reliable. The selection of FDDI as the interface media also provides

protection from EMI and electrical noises, such as crosstalk, reflections, etc., while providing electrical isolation between units on the LAN. Before discussing the system architecture and software performances needed to allow FDDI to function correctly, a general understanding of the characteristics of FDDI is needed. The next section will focus on the attributes of Fiber Distributed Data Interface.

1.2 FDDI Discussion

The FDDI architecture is defined as a 100 Mbit/s, token ring network using an optical fiber medium. The wording of the description sounds very simple, but it involves some complex concepts. In the next several paragraphs, each of the major items will be discussed. The major items are the 100 Mbit/s, fiber optic data transfer, the ring architecture and the token passing strategy. In addition, discussions of items such as counter rotating rings and optical bypasses will be presented.

Before beginning a discussion of the system, several terms must be defined. These terms are as follows.

Data Rate: The number of bits per second of actual information transferred before any encoding scheme is used.

Baud Rate:	The maximum number of signal transitions possible. The baud rate is the actual number of transitions per second transmitted. In self-synchronizing digital systems, the baud rate is the data rate multiplied by an encoding factor.
Media Access Unit:	The Media Access Unit is the actual hardware that provides an interface between the computer in use and the local area network.
Architecture:	Organizational philosophy behind the interconnection of nodes.
Topology:	The actual location and routing of individual cables.

The 100 Mbit/s fiber optic medium used in FDDI is a fairly simple concept. The system passes 100 Mbits of data per second over a fiber optic cable. The attributes of FDDI are defined by the X3T9.5 committee in four basic documents. Each document defines actions and interactions of the different layers of the communication. The layers of FDDI are based on the International Standards Organization (ISO) Reference Model for Open Systems Interconnection (OSI) definitions. The relation between FDDI and OSI is as follows.

Physical Layer Media Dependent (PMD) - Specifies the lowest portion of the OSI physical layer. This includes power levels and characteristics of optical devices. It also covers connection receptacle footprint, connection lengths and losses, and permissible error rates.

Physical Layer Protocol (PHY) - Specifies the upper portion of the OSI physical layer including data encoding/decoding, clocking and framing. It also defines elasticity buffers, smoothing functions, and repeat filter functions.

Media Access Control (MAC) - Specifies the lower portion of the OSI data layer. It specifies access information to the network including addressing, data checking, initialization and error detection/trouble shooting functions.

Station Management (SMT) - Specifies management of the network required at each node on the network. Each node is responsible for a share of the FDDI ring control.

The 100 Mbit/s transfer uses the PHY to provide self-synchronization of the data and performs self-corrective clocking in the local media access unit by use of data encoding. To provide self-corrective clocking of the data,

the data is encoded in what is known as 4B/5B [7] encoding streams. The 4B/5B encoding expands the four bits of data into a five bit field. This requires a 125 Mbaud rate for transmission of the 100 Mbit/s data. In the expansion, the sixteen possible data groupings (four bits have 2^4 possible combinations) are reserved in 16 of the possible 32 data groupings (five bits have 2^5 possible locations). The remaining 16 characters of the five bit field are used for special control characters or for non-existent data groups. The failure of an accurate transmission can result in one of the undefined five bit fields being defined, which signals an error to the receiving device. Some of the uses of the special characters can be to provide for signalling the receiving devices as to the beginning of a message, end of a message, or idle times. The ability to use the special characters as signals within the data stream allows the system to be self corrective. For example, if a data signal is corrupted, causing the data to become improperly framed, i.e. the start and/or end of a message is incorrectly recognized, the system can be automatically reframed when the next "begin message" signal (a special character) is passed. In addition, the use of different patterns for data and control prevents a data pattern from accidentally being interpreted as a control symbol. The use of five bit fields allows the 16 bits of information and nine control bits to be encoded in a common stream without overlap of bit meaning. The five bit fields

used for data and control, excluding halt, quiet and idle, are designed to require the system to go no more than three bits between transitions. The code group shown in Table 1.2.1 is the electrical input (or output) to the corresponding transmitter (or receiver) and is not the same as the transmission code. The actual transmission output, is shown as a transition state in Table 1.2.2. The output of the transmitter, a light emitting diode (LED), transitions its current state when the input from the code group is a logical "1." The LED output does not transition when a logical "0" is input. For example, if the state of the LED is "On," i.e. transmitting light, when a logical "1" is input, the LED will turn "Off." If the LED is "On" when a logical "0" is input, the LED will remain "On." Therefore, to go no more than three bits without transition, as required by FDDI, the input to the LED, shown as the code group, must have a logical "1" available at least every fourth bit. This transition occurs with a similar regularity, at least every fourth bit, from any point in a data stream. This transition occurs across 5 bit fields or within them. The 4B/5B encoding allows both control signals and data to be sent over the same interface using different character sequences. The encoding also guarantees that the system can be self synchronizing, even at the 125Mbaud rate of FDDI.

The use of 4B/5B encoding provides reliable transitions

on a regular basis, meaning that "the FDDI code has good clock recovery properties (i.e., a comparatively narrow spectrum), and it is easy to implement. FDDI's 4 of 5 encoding is 80 percent efficient, keeping the baud rate at only 125 MHz and allowing lower overall channel bandwidths." [5] The requirement for the transition of data bits (transition of logical states) at a guaranteed maximum rate defines how long the receiver must remain synchronized to the data stream before it is resynchronized by a change in the data. In other words, the receiver approximates the timing for data periods using its own internal clock until a transition synchronizes the internal sampling clock of the FDDI Media Access Unit, MAU, to adjust to the actual clock period of the data stream. This is needed because the internal clock may not be exactly in phase with the data transition periods. By having a transition guaranteed at certain times, the receiver can compensate for the difference between the actual state change in the data and its own internal clock. This difference can be used to update the internal clock to keep it synchronized with the data clock. In FDDI this transition is guaranteed to occur every 320 nanoseconds.

Token Passing Ring

FDDI is described as a "token ring using an optical fiber medium". [4] The "token ring" defines the physical attribute

Table 1.2.1: Translational table for 4B/5B conversion.

Code Group	Symbol	Code Group	Symbol
00000	Quiet	10000	Not Used
00001	Not Used	10001	"K" (2/2 Start Delimiter)
00010	Not Used	10010	8
00011	Not Used	10011	9
00100	Halt	10100	2
00101	Not Used	10101	3
00110	Not Used	10110	A
00111	"R" (Control Ind.)	10111	B
01000	Not Used	11000	"J" (1/2 Start Delimiter)
01001	1	11001	"S" (Control Indicator)
01010	4	11010	C
01011	5	11011	D
01100	Q	11100	E
01101	"T" (End Delimiter)	11101	F
01110	6	11110	0
01111	7	11111	Idle

Note that the 4B/5B coding is used to ensure transitions in the data stream at each three bit period. This transition is based on the actual transmission code, not the code group. The actual transmission code is shown in Table 1.2.2.

Table 1.2.2: Translation table for transmitted code used in FDDI.

Transmit State	Symbol	Transmit State	Symbol
NNNNN	Quiet	TNNNN	Not Used
NNNNT	Not Used	TNNNT	"K" (2/2 Start Delimiter)
NNNTN	Not Used	TNNTN	8
NNNTT	Not Used	TNNTT	9
NNTNN	Halt	TNTNN	2
NNTNT	Not Used	TNTNT	3
NNTTN	Not Used	TNTTN	A
NNTTT	"R" (Control Ind.)	TNTTT	B
NTNNN	Not Used	TTNNN	"J" (1/2 Start Delimiter)
NTNNT	1	TTNNT	"S" (Control Indicator)
NTNTN	4	TTNTN	C
NTNTT	5	TTNTT	D
NTTNN	Q	TTTNN	E
NTTNT	"T" (End Delimiter)	TTTNT	F
NTTTN	6	TTTTN	0
NTTTT	7	TTTTT	Idle

Note that the state transmitted is based on the previous state output by the transmitter and the value of the input, i.e. group code from Table 1.2.1. T refers to a transition from the previous state and N refers to no transition from the previous state. Translation can be made by using the 5B code group output, where a "1" denotes the need for a transition and a "0" denotes a need for no transition.

and access methodology of the architecture. The architecture is made of two distinct network considerations, the physical attribute, which is the ring connection, and the access methodology, which is token passing. Examination of the physical and access methodologies will explain the reason for their selection.

Physical Connections

The three main physical computer topologies are the bus, the star and the ring, which are shown below. The bus system requires all nodes of the network to exist on a common connection. When a machine on the bus desires communication, the connection is provided sufficient power for each member of the network to obtain information. In electrical systems, which sense voltage levels, the bus connection is extremely popular. Because electrical receivers require small input powers, which are limited by the input impedance of each device, the intensity of the signal at the ends of the bus varies only slightly from one end of the bus to another. For example, if a bus has eight receivers on the line, each requiring 0.1 mW, in the "high" state, the driving device will need to supply only 0.8 mW to the line. Each item will limit the power it draws based on its input resistance. By analysis, it could be shown that with negligible line losses (0.65Ω per foot for 5 mil wide, 1 ounce copper [8]), the

position of the receivers does not greatly affect the amount of voltage sensed. Since each member of the network draws very little power, the signal remains at an acceptable level. In optical systems, information is passed by sensing the power of the light transmitted. Figure 1.2.1 demonstrates how tapping the percentage of power from the fiber will give significantly different power responses depending on the number of taps between the sending station and receiving station. In addition, if the transmitter exists in the center of the bus, some method must be used to ensure that the power is sent in both directions. In the examples below, it is shown that fiber optic busses will deliver significantly different amounts of power to the receivers based on the physical positioning of the drivers and receivers. Because optical busses take a percentage of available power at each drop-point, the systems need extremely good gain control in the electronics to convert optical "ones" and "zeros" to their electrical counterparts. For this reason, optical busses are not commonly used as a physical connection methodology in optical LANs. Star and ring configurations use physical interconnection methods that guarantee that the received power is more uniform at the receiving station, regardless which station is the transmitter.

The "two basic [fiber optic LAN] configurations are the star and the loop." [9] The star connection may use direct

connections or a switching point, called a hub, to allow all virtual connections to be made. There are two types of hub connections, active and passive. The direct connection star uses a link to each node in the network. The hub-based passive star will utilize a single output from each node and will divide the signal using splices so that the signal is divided to go to each node. In general, the hub-based passive star network will take each of the n inputs and couple the light to a group of n-1 fibers. This will give access between all machines simultaneously. Both types of passive stars allow a single node to output to all stations simultaneously in a broadcast fashion. Each node is also capable of inputting from all other nodes simultaneously. The problem with the passive star is that unless the system is capable of receiving from multiple input channels simultaneously, some information could be sent and never received. The transmission medium does not have a built in method for physically guaranteeing reception.

The active star can provide amplification of the signal before splitting it, as shown in Figure 1.2.3c, or have a single input and output line and use a selection switch which only passes a single selected signal to each output. In the first case, the active star resembles a passive star with multiple outputs to each machine, but provides signal boosting not available on the passive star. In the second case, an

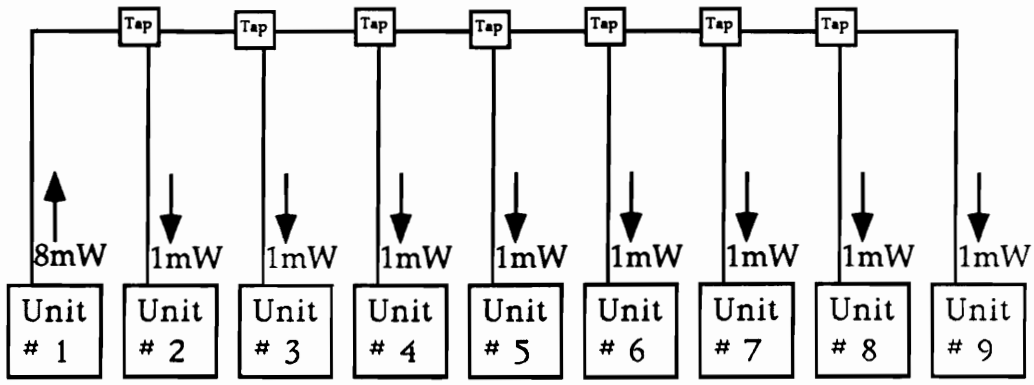


Figure 1.2.1.a: Example of the voltage levels at each point along an electrical bus given the driver is located at the end position and each unit consumes 1mW of power and no loss is assumed for the interconnection

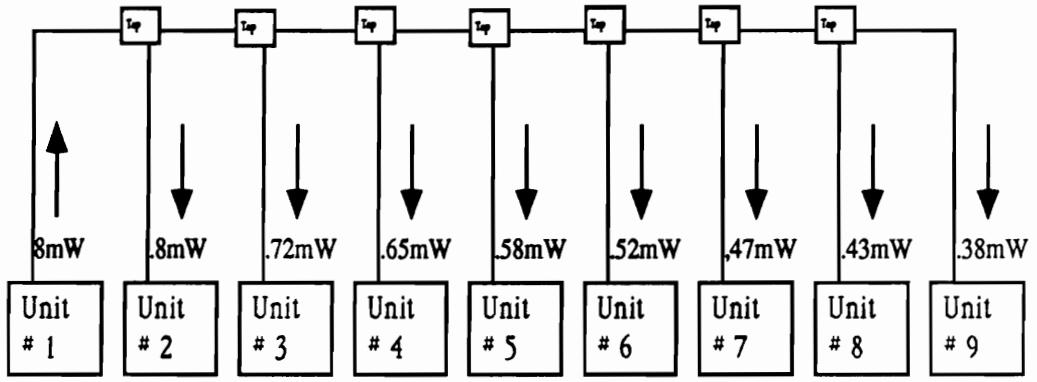


Figure 1.2.1.b: Example of the optical power levels at each point along a fiber optic bus given each tap coupler uses 10% of the power received and no loss is assumed for the interconnection. Note the great variance when the coupler takes a percentage of the power instead of a constant power as the electrical system does.

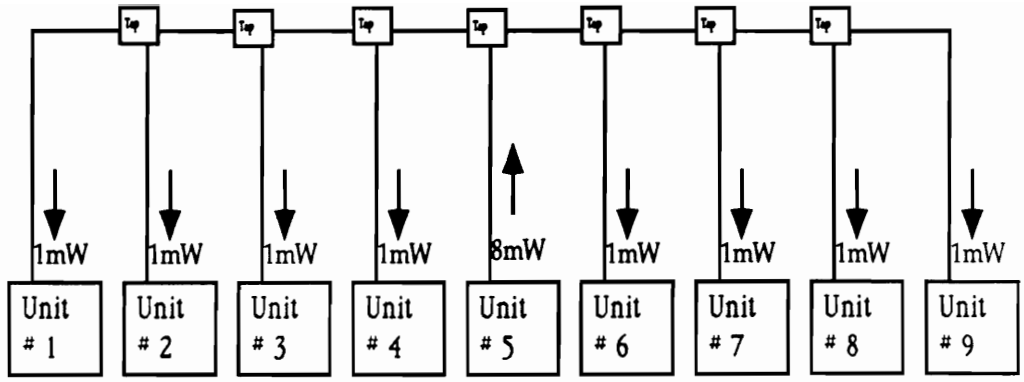


Figure 1.2.2.a: Example of the voltage levels at each point along an electrical bus when the driving location is at the center of the bus given each unit consumes 1mW of power and no loss is assumed for the interconnection.

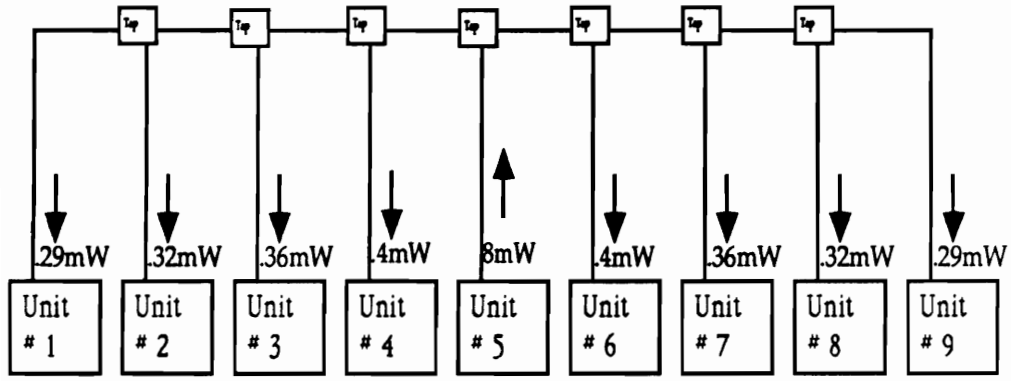


Figure 1.2.2.b: Example of the optical power levels at each point along a fiber optic bus when the driving location is at the center of the bus and the center tap can split the power and all other taps use 10% of the power received to the receiving unit and no loss is assumed for the interconnection. Note the changes between the optical bus driving the bus in the center compared to the bus driving at the end of the bus. The great power variations based on the location of the driving and sending nodes is the reason optical busses are not used.

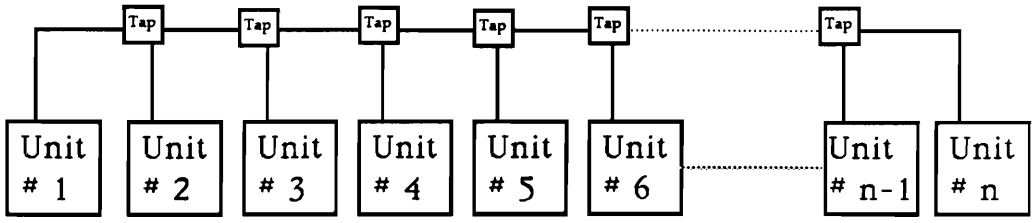


Figure 1.2.3.a: Example of a bus architecture. Note each unit recieves a portion of the power available on the bus at the connection location through a tap.

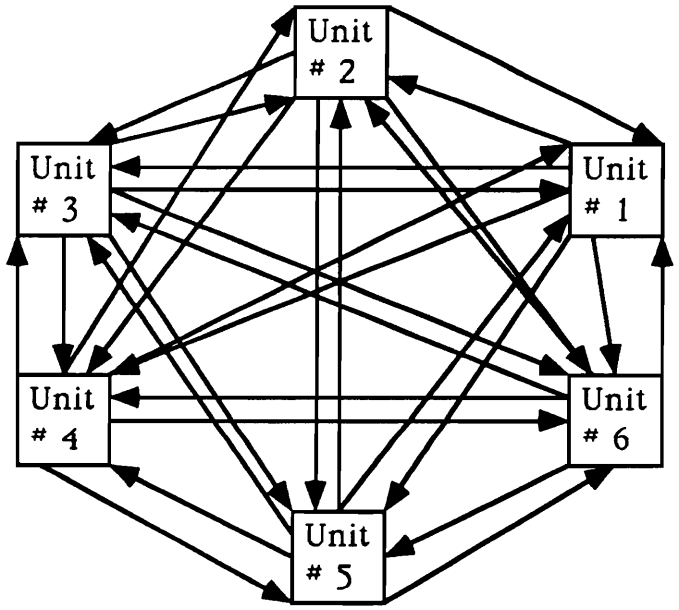


Figure 1.2.3.b: Passive star connection using no center distribution point (hub).

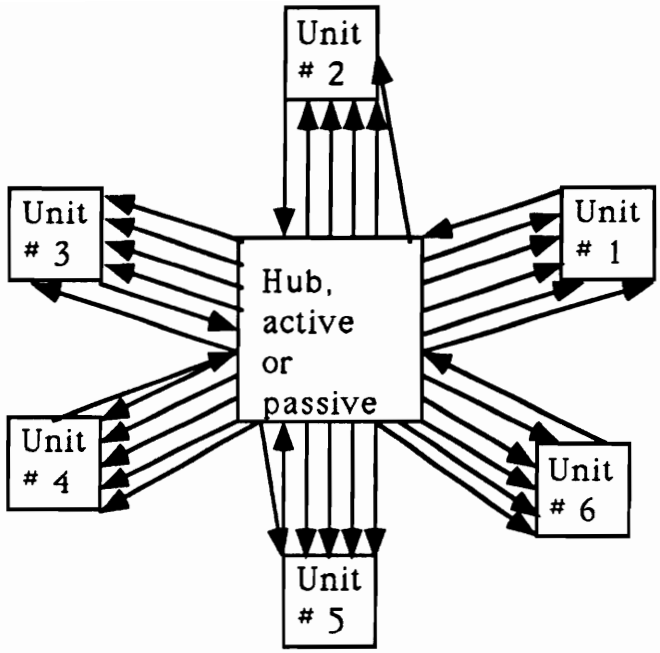


Figure 1.2.3.c: Star configuration using a hub for distribution. The hub can increase the power to the signal being distributed, making it an active hub, or the hub can just provide power splitting services, making it a passive hub.

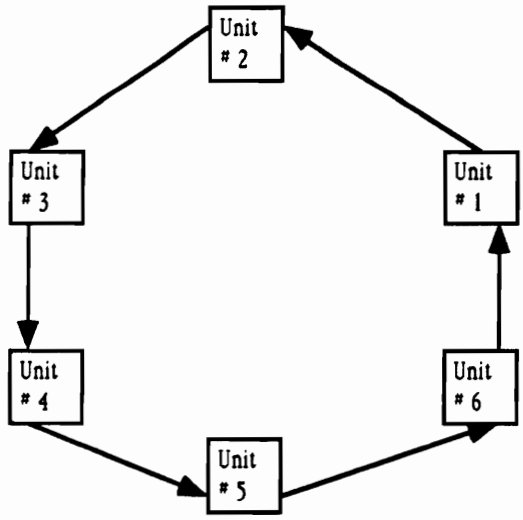


Figure 1.2.3.d: Ring configuration. At each station, the signal is regenerated to ensure data is not lost due to changes in signal intensity.

active switch receives a request from the transmitting node defining which of the computers should receive the message to be transmitted. The active switch will guarantee the connection, and may provide signal boosting and/or reconstruction on the signal. The active switched star will require some method of switching incoming and outgoing lines. This is the electrical equivalent of a n x n crossbar switch. Today, the ability to selectively route information over coupled fibers is not available.

Star configurations have two major disadvantages. One is the problem associated with message transmission contention which is the way of scheduling which messages pass if two machines wish to simultaneously access one receiving machine. If the star is active, with a single output line to each node, the star must decide which connection has priority when multiple messages are being transmitted to the same node simultaneously. If multiple outputs are used the receiving machine will have to accommodate all possible inputs simultaneously or select which inputs to use and which inputs to ignore. The second problem with the star is the single point failure possibility at the hub. If the hub has failed, no machines can communicate. The ring architecture has solved these two problems of the star network.

The ring network requires nodes of the network to be

serially connected to the communication medium. The ring network will constantly pass information in a single direction for each ring used. As a message passes through a node in FDDI, it is converted to an electrical synchronized signal and is examined for origin and destination information. If the message destination is the receiving node, the message is copied to the node's FDDI storage memory. If the message was generated by the receiving node, the message is removed from the ring. All messages on the ring are regenerated and placed back on the ring unless the node receiving the message generated the signal or unless the node receiving the message also holds the token. When an error in transmission causes the sending node to be unable to recognize its message, i.e. the source address is corrupted or there are other problems in transmission, the message will be repeated until it arrives at a node holding the transmission token. Any node holding a token, will remove all incoming data from the ring.

Figure 1.2.4 shows a single ring connection architecture. Node 1 will always pass information to node 2 providing a clockwise rotation of the information. For node 1 to communicate to node 4, the data must pass through the physical connection points, MAU, of nodes 2 and 3. Figure 1.2.5 shows a dual ring configuration. In general, dual attachment rings will often be counterrotating. Counterrotating rings use reverse transmission paths, allowing node 1 to communicate

directly to node 2 in ring 1, while on ring 2, node 2 communicates directly to node 1. Note in FDDI the main purpose of the counterrotating rings is to provide redundancy to the ring. It is possible with the use of multiple FDDI systems to have multiple networks interconnecting FDDI nodes in a variety of fashions. Figure 1.2.6 shows two dual attached counterrotating rings. Note the major directions of the rings are different.

Rings are practical physical connections for fiber optic transmission. The use of rings in a fiber optic implementation can provide a means of synchronizing all messages and retransmitting the messages with proper power to ensure reception at the next node. Rings do not have the power limitations of busses and passive stars, or the single point failure locations of stars. Rings provide a successful method of communication for fiber optic intercomputer communication.

Access Methodology

Access methodologies are the way data is allowed access to the ring. In general, data may be placed on the medium in a random or orderly fashion. The basic types of access to the ring are broadcasting, slotted ring and token ring. The broadcast methodology was prominent in early systems and

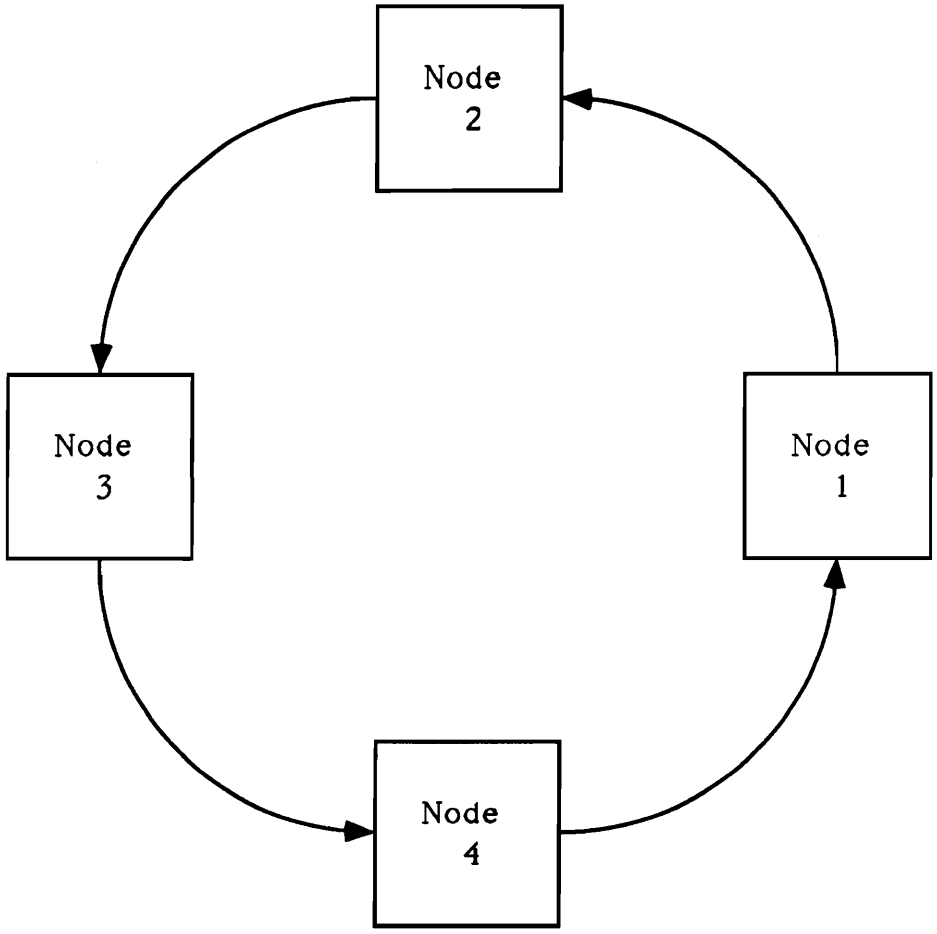


Figure 1.2.4: Shows a single ring connection architecture. The arrows show the direction of flow of information

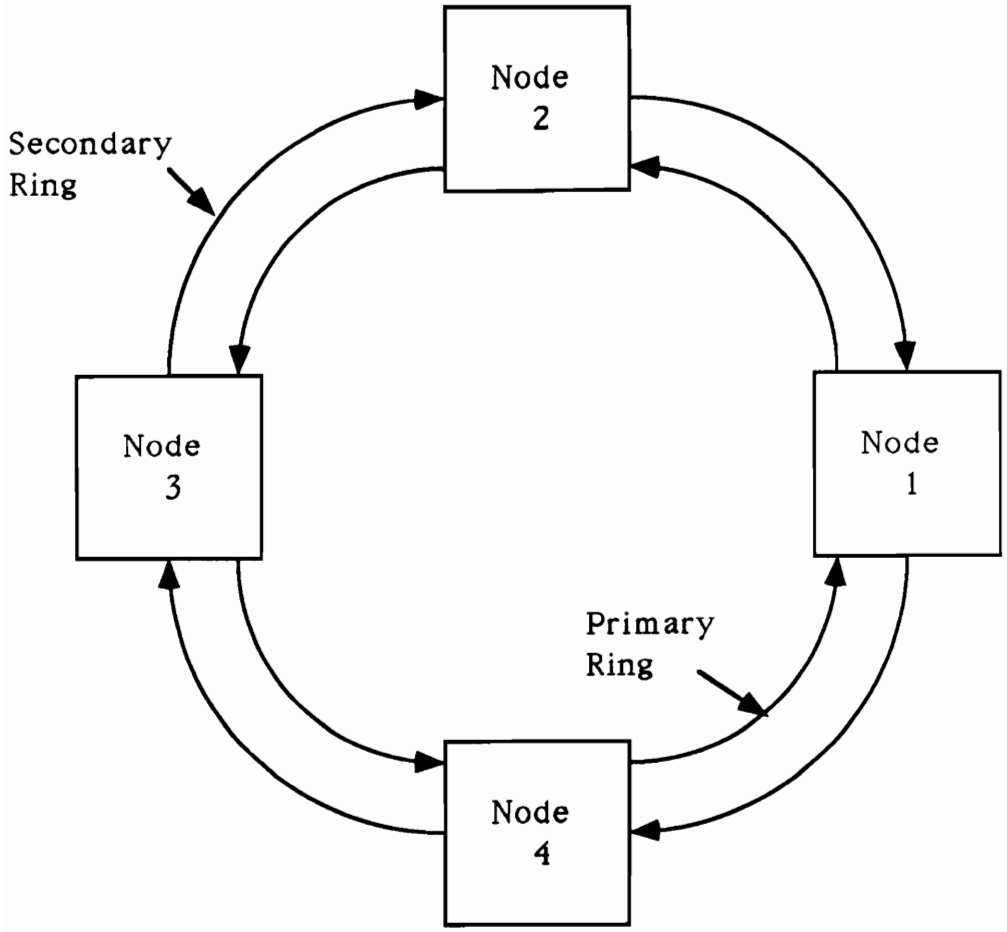


Figure 1.2.5: Shows a dual ring connection architecture capable of being used as a single counterrotating ring.

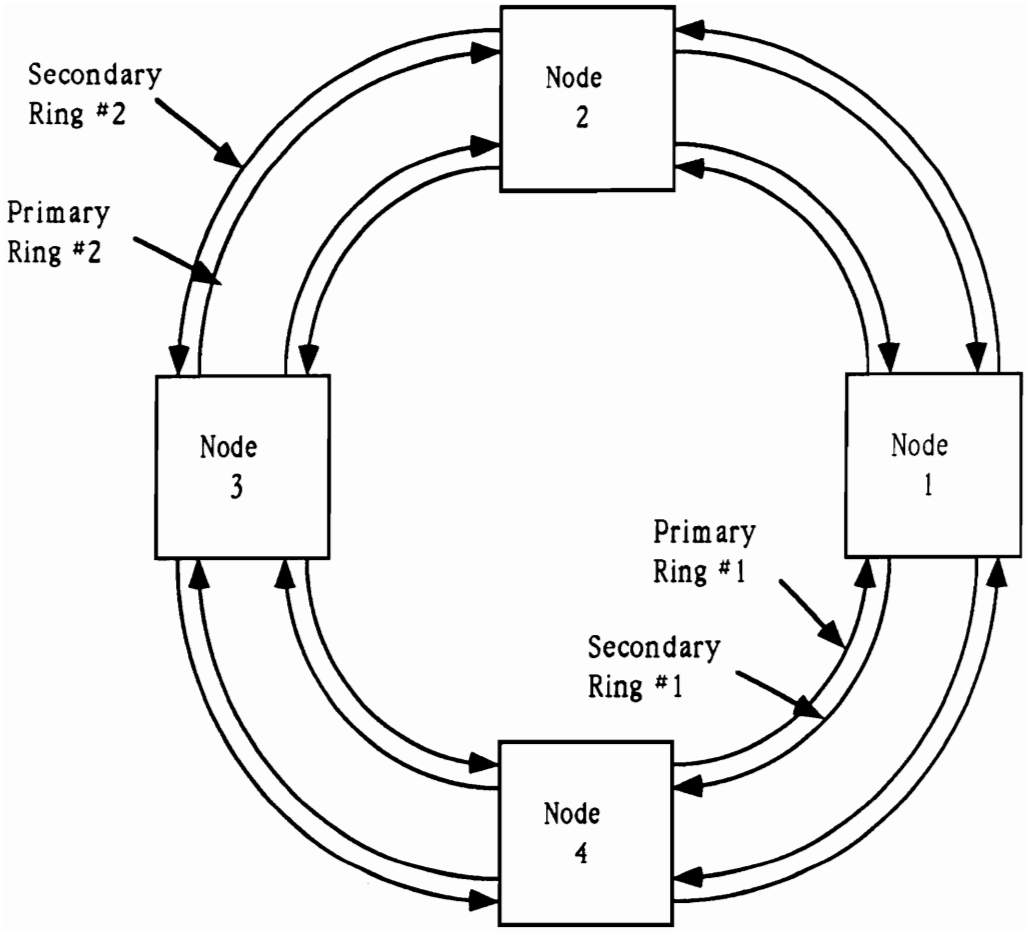


Figure 1.2.6: Shows two dual attached counterrotating rings. Notation defines each of the primary paths for ring one and two as well as the secondary rings. Note the main rings do not pass information in the same direction around the ring.

collision detection was a method of determining when two messages overlapped. Ethernet is based on such a standard access method. The IEEE standard 802.3 based networks are "described by the phrase 'carrier sensing, multiple access with collision detection' (abbreviated CSMA/CD)." [3] These 802.3 systems allow any node to begin transmission when a message is available to transmit and the medium is quiet. With each node on the structure sending messages when silent periods are detected, it is possible for multiple devices to begin transmitting with such timing that several messages coexist. This overlap of multiple messages is known as message collision. Ethernet uses three mechanisms to deal with message collision: carrier sensing, collision detection, and back-off.

Carrier sensing requires each node to monitor the medium for the existence of a currently transmitted signal. When a node is ready to transmit a message, the node determines if any messages are currently active on the medium. If the medium has a message in progress, the node with the new message waits until the medium is silent and then begins transmission. If two nodes need to transmit, they begin transmission as soon as they see the silent line. It is easy to understand how two or more nodes can begin sending information almost simultaneously. It is possible for two machines, separated by a transmission delay, t, to begin transmission at times t_1

and $t_1 + t_2$, where $t_2 < t$. Since the beginning signal can not reach the second transmitter node until after the second transmitter has begun transmitting, a collision must occur. The existence of two or more signals on the medium, a collision, requires a method of collision detection.

The transmitting stations perform collision detection. Whenever a station is transmitting a packet through its hardware output port, it also listens on its input port and the two signals are compared. If the signals at the input and output are not identical, a collision has occurred. When transmitters detect collisions, they cancel the transmission, wait for a period of time, and begin transmission after sensing the carrier being silent for a period of time. The amount of time each system waits to begin retransmission after noticing a silent medium is known as the back-off time. The value of the back-off is random for each station.

In Ethernet, "delays due to contention become noticeable when 50 percent utilization is exceeded." [3] The reason for using FDDI is the large amount of data being passed between computers. Even with the 100 Mbit/s data rate available in FDDI, the designers anticipated as much as an 80 percent utilization of the data rate. Because of the high percentage of utilization, broadcast methodologies are not consistent with FDDI.

In the slotted ring configuration, each unit on the ring has a relative time during which it may transmit messages. FDDI-II plans to allow for the utilization of slotted messages as well as token based messages. Slotted messages guarantee each node will receive access to the ring at defined times. The nodes are expected to begin transmissions at the beginning of the slot time assigned to each node. Only the node owning the slot time may transmit during its slot time. There are two major drawbacks to the slotted ring. The first is that each machine must be able to track the time in exact relation to the network. The lack of accurate timing can allow the messages to overlap, or to lose part of the available transmission time. To keep track of the time, and to allow for maximum transmission rates, system time must either be kept at the node and data always available for transmission, or the computer must track the time. If the computer tracks the time, it must account for all delays from the node interface to the computer and from the computer to the node to control the beginning and ending of transmissions. Unless the computer has interrupts which never vary in access time, the use of the computer must also add the variable of interrupt timing.

To ease the computer and interface requirements, the node interface needs to gain intelligence of the ring time. Note

that any time delay between the ring and the node needs to be accounted for in determining when to transmit or when to remain silent to prevent overlapping messages. Keeping exact time at the node or computing the exact delay to the computer are not easy propositions. Therefore, it is probably best that slotted timing clock accuracy not be a limiting factor in the first high speed fiber optic network standard.

The second drawback of the slotted ring is that each node has reserved time, even if no data needs to be transmitted. While it is reasonable to expect every node on a network to transmit or receive messages at some time, it is unreasonable to expect every node on the network to transmit on an exact schedule. Trying to anticipate the schedules would be impossible due to the distributed load of networks. Likewise, it would be difficult and time consuming to reallocate time slots as often as user and processing loads change. For these two reasons, slotted ring networks are not a good choice for the first fiber optic network.

In future fiber optic networks such as FDDI-II, slotted rings may be used. In these advanced networks, the slotted ring allows the network to transmit data during a time slot at a rate other than 125Mbaud. In the future of FDDI-II, the network will provide the "capability to support circuit switched data traffic (voice, etc.) between PABX nodes,

integrated workstations, digital trunks, etc." [4] In the final analysis, some slotted ring usage will be available in FDDI-II, but in FDDI and for the majority of FDDI-II, token ring will be the standard.

Token Ring

The basic consideration for use of token-passing for FDDI is that the "token passing provides more efficient data transmission at higher bandwidth utilization, especially as traffic on the media nears saturation (although this may depend on the number of devices on a network). Simulation studies have shown that there is only a slight increase in average frame delay (Token Rotation Time - TRT) at system utilization levels of up to 85 percent of capacity. Even at 98 percent capacity, the average delay is only approximately fifteen frame-transmission times or 22 milliseconds." [10]

The token ring is used as the FDDI access standard. The token is a unique message which exists on the ring. The token is placed on the ring immediately after the ring is initialized. (Ring initialization will be discussed later.) As the token passes around the ring, nodes needing to transmit remove the token from the ring, transmit the message which is at the top of their queue, and return the token to the ring. The token must pass all the way around the ring in FDDI

between successive transmissions by a single node. If multiple messages begin to appear on the ring an error is assumed and the ring is reinitialized. During initialization or reinitialization, all tokens are destroyed, and a single new token is issued.

The task of initialization or reinitialization is a topic worthy of discussion. Initialization or reinitialization is "a consequence of either inactivity or incorrect activity on the ring" [11]. Inactivity can quickly be noticed by the ring because the ring lacks data transitions, which are due to the system's 4B/5B encoding scheme. Incorrect activity is generally noticed because the FDDI unit has not detected the availability of a token after successive expirations of the "Token-Rotation Timer." Other conditions defining an incorrect activity can be generated by the Station Management protocol. When inactivity or incorrect activity is noticed by a node, the node is responsible for issuing a "Claim Token." Any station may issue a claim token. Since claim tokens may be generated by different devices which may be using different detection schemes, multiple claim tokens can be in existence and may have common or different issuance times. Claim tokens call for the immediate suspension of ring activity at the receiving nodes and reconfiguration of the ring takes place. The claim token process defines which process will issue the transmission token.

The claim token process is the method for determining which node will establish the token. As claim processes are generated, there may be several claim tokens in existence on the ring at one time. As claim tokens are passed around the ring, each node will examine the claim token and change the token if necessary. The process for changing the token is a bidding system where the node on the ring with the smallest Target-Token Rotation Time will eventually win the claim process. As the original number of claim tokens is generated, they pass to the node immediately downstream. The receiving node will compare the Target-Token Rotation Time, TTRT. If the receiving node has a lower TTRT than the transmitting node, the receiving node will destroy the received claim token and issue a new claim token. If the receiving nodes TTRT is larger than the transmitted token, the initial claim message is retransmitted. Arbitration takes place if the TTRT is identical for a transmitted claim token and the receiving node. This arbitration guarantees only one station will have the lowest TTRT and other attributes which allow a single node to receive its originally issued token. The node receiving its own claim token from the ring has "won" the claim process. This node initializes the ring.

Initialization of the ring will set the TTRT, which guarantees transmission access to the ring to be sufficient

for the fastest required rotation for all nodes. This is because the node winning the claim token had the shortest token rotation requirements. To initialize the ring, the node winning the claim token process sets the operating TTRT value and resets all target rotation timers (TRTs) at each node by sending an initialization token around the ring. The node winning the claim token process will then issue the nonrestricted token, a token allowing general transmission to begin. The initialization token cannot be captured by any node except for the node generating it. Note the initialization token and the first nonrestricted token will coexist until the initialization token is removed by the initializing node. The bidding for the claim token has a maximum and minimum time as described below.

$$\begin{aligned}\text{Worst Case Bid Time} &= \text{Generation of Claim Token} \\ &+ \sum_{k=1}^{N-1} \text{Transmission Delay}_k + \text{Decision Delay}_k \\ &+ \sum_{k=1}^{N} \text{Transmission Delay}_k + \text{Decision Delay}_k \\ &= \text{Generation of Claim Token} \\ &+ 2\sum_{k=1}^{N-1}(\text{Transmission Delay}_k + \text{Decision Delay}_k) \\ &+ \text{Transmission Delay}_N + \text{Decision Delay}_N\end{aligned}$$

The above equation is generated realizing that the worst case scenario is for a single node to generate the claim token, and for that node to be immediately downstream from the node requiring the smallest TTRT. For a network of N nodes, this would require (N-1) transmissions and decisions for the smallest TTRT to be entered on the ring, and one complete circulation of this signal, N transmissions and decisions.

$$\text{Best Case Bid Time} = \text{Generation of Claim Token} + \sum_{k=1}^{N} \text{Transmission Delay}_k + \text{Decision Delay}_k$$

The above equation is generated realizing that the best case scenario is for one of the nodes generating a claim token to be the station which will win the claim token process.

Reinitialization or initialization of the ring takes place any time the physical medium is disturbed. In addition, the Station Manager of a node may request a ring initialization. Theoretically, if a node is available to be installed in the network, the ring will accommodate the new node when it is installed because of automatic initialization due to the cable break required to add the new node. In the case of a failed node, reconfiguration of the ring will probably take place even if an optical bypass is used. During

the activation of the optical bypass, data will be disturbed. The detection of the data disturbance will initiate ring initialization and the ring will be reconfigured.

In cases where the data link is broken, the Claim Token process will fail. When a station determines the Claim Token process has failed, that node will issue a Beacon process. The Beacon process is used to inform stations about the break in communications and to provide diagnostic assistance to the ring restoration process through Station Management. The general idea behind the broken data link reconfiguration under Station Management is to allow wrap-back between the primary and secondary rings for complete communication. Figure 1.2.7 shows a ring with a connection failure. Note how the primary and secondary rings are joined on each side of the connection fault to provide a single ring. Although the transmission length for the ring has increased to approximately twice the original ring length, the system is still operational. Many of the concerns in this segment of the FDDI document are still not defined.

Throughout the FDDI system, the availability of the ring to perform in a dependable manner is extremely important. The availability of the token to each node on the ring in a timely fashion is also important. To guarantee timely access to the ring, the token ring uses a Target Token Rotation Time in its

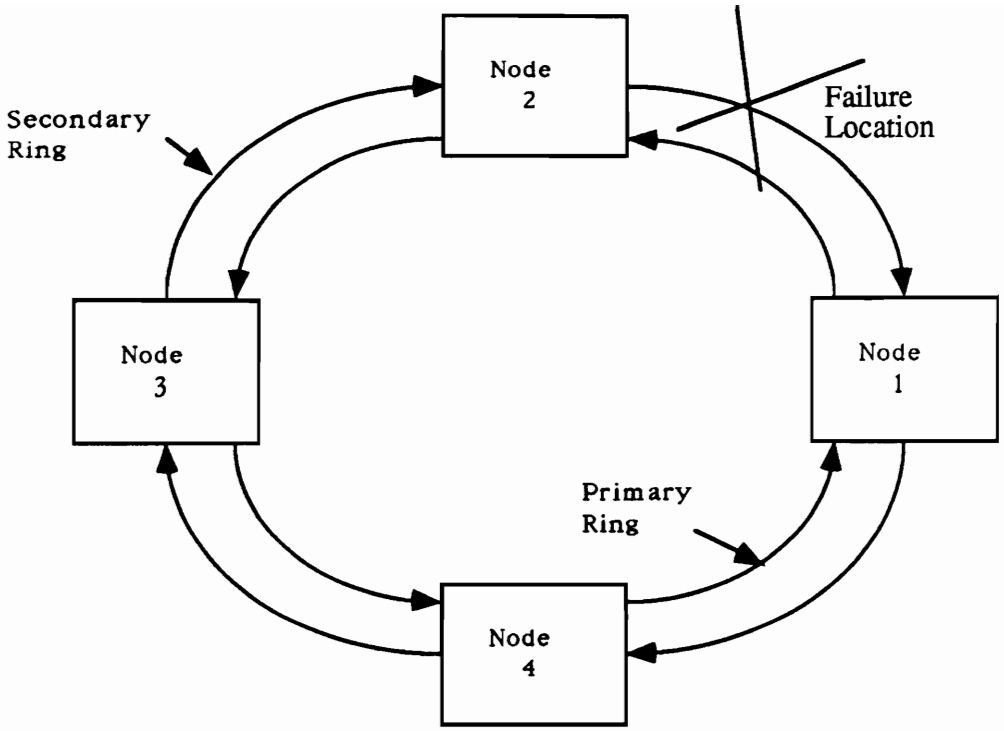


Figure 1.2.7.a: Shows a ring with a failure in the interconnection.

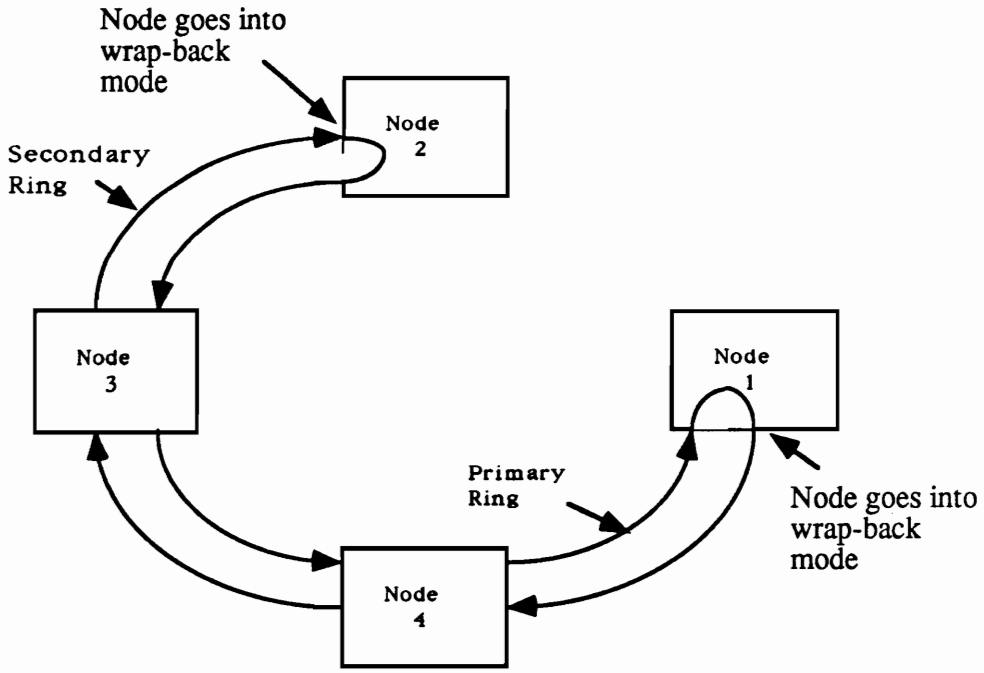


Figure 1.2.7.b: Shows the repair data path used when the nodes go into wrap-back. This allows the ring to remain in tact even though a segment may be damaged.

configuration. Its use is discussed in the next section.

Token Ring Rotation Time

One of the major problems facing future networks is the enormous volume of data which each node will want to place on the ring in a given time period. The amount of time around a properly functioning ring can be given by

Ring Rotation Time = Delay at each station, in seconds x Number of stations
+ (Amount of active fiber on a ring ÷ Velocity of light in a fiber, in seconds)
+ (Total number of bits transmitted)/(100 Mbit/s).

To attempt to meet the desired frequency of access to the ring, FDDI has incorporated a Target Token Rotation Time (TTRT). In an overview, the TTRT is designed to allow time critical messages to be passed on the ring within known periods of time. The TTRT defines the maximum time a node may hold a token when all tokens are passing time critical messages and when all nodes are operating. The individual nodes will compare the elapsed time since the previous token to the TTRT. If they are approximately equal, the node will only pass one message of time critical data. If the elapsed

time is significantly less than the TTRT, the node is allowed to pass time critical and non-time critical data. The definition of what data is defined to be time critical data is left to the discretion of the system manager. In networks used as interconnection backbones, the number and frequency of time critical messages would be expected to be very small, so the TTRT would be made very large allowing large non-time critical messages to be passed in large blocks. Networks used in real-time control applications would use the TTRT, but selection of the TTRT would need to account for the overhead associated with using small TTRTs.

It is important to note that the TTRT will define the maximum amount of data allowed to be placed on the ring by any node. From the TTRT, the node can extract the maximum amount of time it has available to transmit data and therefore the maximum amount of bytes of data which can be transmitted. Note that as rings get large and TTRTs get small, the amount of overhead will remain constant while the amount of useable data will be reduced. The reduction of useful data to overhead on the ring could produce network performance problems on the ring and at the MAU.

Bypass Switch

An optical bypass switch is an optional device in FDDI

which allows the data path to bypass the node when the node is not operational. The optical bypass routes the optical signal from the receive to the transmit cables of the node and prevents the node from operating on the signal. When the optical bypass is in place, the node cannot communicate on the FDDI network in any fashion. This option is extremely useful when machines have to be repaired and power is removed from the node. Upon the station being returned to service, a minor disruption will occur in the data and reinitialization of the ring will almost always take place.

Summary

FDDI meets the needs of future networks. As shown, FDDI provides high transfer rates, high reliability, long distance and multi-node need of high performance local area networks. FDDI has been shown to meet the requirements by using proven fiber optic devices in counterrotating token passing networks. For ultra-reliable networks FDDI can meet the needs of the system. FDDI provides support for single connect failures by the use of counterrotating rings. FDDI is immune to electro-magnetic interference (EMI) which can cause some system interruptions. FDDI uses proven optical technology (1300 nm multi-mode interconnection) with the inherent advantages of such systems over electrical systems such as the lack of ground loops, crosstalk, and reflections. This means that

FDDI can provide adequate network performance for today's computing environment. The system can meet the baseline performance of the systems, but the strategy of the system interconnection to the computer needs examination to allow the computer to operate at the rates available using the FDDI system. In later sections, we examine a current FDDI system, model components in the system, and use the model to suggest changes in architecture and software to provide a high performance system. Section 1.3 focuses on some of the increased requirements of a processor that is compatible with the high data rate made available by FDDI. The section will examine some of the attributes of Ethernet and demonstrate the increased processor requirements imposed on the host processor by FDDI.

1.3 Increased Requirements of Input/Output Processors to Alleviate Host Processor Interface Requirements to FDDI

For many years, Ethernet has been a standard network for intercomputer communication. Ethernet did not require elaborate interfaces to handle the amount of data being processed. Effectively, the computer processed the data as it was received from the network. In FDDI, many of the same transfer and protocol issues are taking place, but because of the network data rate, the increase in data and messages has

brought new requirements to the system in terms of throughput.

The increased requirements of the FDDI network interface in comparison to other networks requires the host computer to use a separate microprocessor to perform networking functions. The increased processing load consists of two major attributes. They are

1) Data Transport or Throughput
2) Networking Protocols

Data throughput is the transfer of packets between the incoming message storage area and the host computer interface. In Ethernet, this interface can appear as extended memory or an I/O port. In either system, the data is transferred from the storage location into the main computer memory for use. In FDDI, the two areas for storage and interface will be referred to as the MAU interface memory and the host computer interface memory, see Figure 3.2. Data which is only temporarily used in FDDI may exist outside of the host computer's main memory even during its use.

The MAU interface memory provides a temporary storage location for transport packets which will be passed through the FDDI chip-set and then over the fiber-optic cable. The transport packet contains the actual data and all upper layer

information required to allow intercomputer communication such as TCP/IP headers. The transport packet may be stored in the MAU interface memory in preparation for sending or upon reception.

The host computer interface memory provides an interface which stores the data for use by the host computer without any network information being available. To the host computer, it is just memory. The data in host computer interface memory may contain application and even presentation or session layer information, but contain none of the information used by the network in sending or acknowledging information transfer. The host computer interface memory in FDDI generally appears as internal memory in the Ethernet system and the Ethernet system does not use a front end processor for transmission. The expansion of the memory in the FDDI system provides flexibility on how the host computer uses incoming data and data being prepared for transport.

In FDDI, the network protocol will be the main difference between the data which existed in the interface memory and what was desired in the host computer interface memory. The network protocol will guarantee the data from one computer will be received at the proper location. In general, any network protocol which will work on Ethernet will also work on FDDI. The amount of processing per message for any given

protocol such as TCP/IP is the same whether Ethernet or FDDI is used. But because FDDI can receive so many more messages per second, the FDDI network interface must be much faster than the Ethernet equivalent.

The use of a separate MAU to control the transfer between the MAU interface and host computer interface memories can reduce the loading on the host computer. The MAU will be responsible for direct data transfer as well as network protocol and data management. The MAU will perform all those functions which are currently performed by either the host computer or an I/O processor in the current Ethernet systems. As shown above, the number of instructions or functions performed based on a message count or message length has not changed. However, because of the speed of FDDI, the amount of work to be performed at the network interface has significantly increased.

The FDDI data rate capability of 100 Mbit/s is an order of magnitude increase above the Ethernet data rate capability of 10 Mbit/s. FDDI's sustained data rate of over 80 Mbit/s is more than 25 times Ethernet's sustained data rate of 3 Mbit/s. Currently, the limitations of Ethernet appear to be the ability to quickly and efficiently perform data acquisition and network protocol. In most systems, the Ethernet interface is handled on an interrupt basis by the

host computer. The increased processing power requirements for FDDI have many implications on the host computer and the way it accepts data from network interfaces. Tables 1.3.1 and 1.3.2 show number of MIPS required by CISC-based, i.e. microprocessors with block mode capability, system to perform required data movement for FDDI and Ethernet using different transfer methodologies.

In the tables, the instruction count is based on the number of clock cycles and assumes the instruction/clock ratio is based on the "MOV register,register" command. Therefore, the instruction count is equivalent to the clock count. From the Intel 386 manual [12], we note that the 386 can perform string moves based on the amount of data moved. Instructions are available for byte (8 bits), word (16 bits) or double word (32 bits) moves. Each string move takes 4 clock cycles regardless of the instance size [12]. Given a fixed number of bytes to be moved, double word movements take one-half the number of instructions of single word movements, and one-quarter the number of instructions of byte movements. As Table 1.3.1 indicates, the bandwidth difference between FDDI and Ethernet requires the FDDI system to have 10 times as much processing power as the Ethernet system. If the processing capability were only increased in proportion to the data rate, the FDDI interface would only be able to process 30 Mbit/s. Table 1.3.2, which is based on the expected throughput

comparisons, shows FDDI requiring over 25 times the throughput of Ethernet. Both tables show FDDI requires substantial increases in processing capability for data movement.

Based on the throughput requirements for data alone, we note that the host computer would need to devote over 12.5 MIPS to FDDI just to accommodate data movement using 32-bit movements. When overhead, interrupts and other items associated with shared resources are taken into account, it is clear that a majority of the computing power in a small system could be required just for FDDI. For this reason, it appears that a dedicated MAU is required to interface to FDDI. The dedicated MAU could relieve the host computer of interface dilemmas similar to the way servers may reduce printer and file storage interface requirements. The design of such a system will be discussed in Chapter 2.

From the tables and the discussion, it is also easy to realize that the interfaces used for Ethernet systems will not meet the needs of FDDI. The amount of processing expected to take place is much greater than with Ethernet. In discussions with industry engineers, the common thought is the upper layer protocols and hardware interfaces are what limit the throughput, not the network itself. If FDDI were implemented using similar interfaces, the typical Ethernet limit of 3

Table 1.3.1: Instruction Execution Rate Associated with the Maximum Network Data Rate

Move Size	FDDI Full Speed 100 Mbit/s	Ethernet Full Speed 10 Mbit/s
8-bit	50 MIPS	5 MIPS
16-bit	25 MIPS	2.5 MIPS
32-bit	12.5 MIPS	1.25 MIPS

The number of instruction cycles required to transfer the maximum bandwidth of the given networks from a MAU memory accessed location to an internal memory location using CISC string instructions to meet different throughput and move increments.

Table 1.3.2: Instruction Execution Rate Associated with Maintaining the Expected Throughput of the Network

Move Size	FDDI Continuous 80 Mbit/s	Ethernet Continuous 3 Mbit/s
8-bit	40 MIPS	1.5 MIPS
16-bit	20 MIPS	0.750 MIPS
32-bit	10 MIPS	0.375 MIPS

The number of instruction cycles required to transfer the expected throughput from a MAU memory accessed location to an internal memory location using CISC string instructions to meet different throughput and move increments.

Mbit/s may also be the limit for FDDI. A new interface must be developed which has over 25 times the throughput of current Ethernet interfaces and probably more realistically 30 times the throughput.

The Type of Usage Will Change

Although Ethernet provides connectivity between systems, FDDI will further encourage the sharing of resources and the related system expectations will increase. While some will argue that since FDDI will replace Ethernet with more stations and limited changes in usage, this is not correct. FDDI is not an Ethernet replacement. It is an attempt to provide computer networking which promotes sharing of resources and data [13]. With the philosophy of sharing networks, the expanded bandwidth of FDDI will promote a different view of how networks function. Already, companies are expecting individual nodes to receive and transmit messages at the full speed of FDDI. These messages are typically long streams of data and thus minimize the number of messages received. Two examples are given below.

The first is the Star project, which provides the motivation for much of this research. The objective is to interface a vector processor with a network to allow large mathematical processing capability to be available over a

network. The process requires an input rate of 80 Mbit/s to fully utilize the vector processor. If the data is not available at 80 Mbit/s, the vector processor will have to delay processing cycles until data is available. This data input may be short in duration or may last for several days. In general, for the Star systems, data would be passed for a long period, and would require the FDDI MAU interfacing the VP-100 vector processor to operate at 80 Mbit/s or above. In addition, as one network application using the VP-100 finishes, another may be waiting. Applications may keep the Star FDDI MAU operating at full speed via round robin accesses.

Another proposed system uses FDDI to pass photostatic information to for document retrieval. Such a system will require high speed transfers to provide the terminal operator with information. At the end of the transfer, the data would be available to the user, but during the transfer period the operator would by unable to do work. In this type of system, high speed response for limited periods will exist at many different receiving nodes. At the sending end, the network interface will be constantly responding to requests for information, and thus transmitting at the maximum throughput rates for extended periods of time. In such a system, the sending devices will need the maximum throughput capability over a longer period of time than the receivers.

These are only two examples of how FDDI is currently progressing. In most instances where large amounts of data are needed quickly, the data will probably be available in large message blocks. In the near future, FDDI will concentrate on providing high throughput for large data messages. Because the overhead in TCP/IP is independent of message size, smaller messages will require more processing capability to obtain the 100 Mbit/s data rate of FDDI. FDDI will eventually work towards providing equal throughput for small packets.

1.4 Conclusion

In comparison to Ethernet, an FDDI network interface will perform 10 to 25 times as many data movement instructions per unit of time as an Ethernet network interface. In raw data transfer alone, FDDI will require an execution rate of over 12.5 MIPS to handle the full data rate. The system interface for FDDI needs capabilities not common in previous networks. Not only does FDDI need to be able to transfer massive amounts of data, it must be able to respond to network protocol and management of the network. The amount of work to be performed by FDDI MAUs requires a study and design different from most previously available for any other network standards.

1.4.1 Implications of the Increased Throughput

In the design of an FDDI interface, one must decide whether to pass the maximum bandwidth or just a portion of the bandwidth. In addition, the data bandwidth is only a part of the challenge in making FDDI operate correctly in all environments. If FDDI uses small data packets, the acknowledgements needed for network protocol will be a large compared to the work required to move the data.

In the basic designs discussed in this the dissertation, an attempt will be made to design for the maximum data rate. The ability to keep up with the maximum data rate will require the use of memory to store data when packets cannot be processed in the amount of time during which they arrive. As will be discussed in Section 3.2, the memory can be used to "average" or "smooth" the input arrival rate to reduce the interface processing requirements. In addition, a slight overdesign will allow for the extra processing power to meet unexpected problems. Note that since a single node cannot both receive data and place new data on the fiber simultaneously, the single node data throughput is 100 Mbit/s. If a dual counterrotating ring is used, and both fibers are used for independent data, systems must be capable of processing 200 Mbit/s.

When determining the increased requirements of FDDI, it should be noted that FDDI not only increases the data throughput, but it also increases the amount of network protocol information generated. The Ethernet systems that are replaced by FDDI will have the same network protocol requirements, but the number of transfers expected within a given period of time will increase in magnitude similar to the data rate. The exact nature of the data and overhead will be discussed in Section 2.1

Chapter 2

System Requirements to Fully Utilize FDDI

There are two major considerations in examining the maximum FDDI network system requirements. The first is data access requirements. The amount of data and the frequency with which new messages must be processed are used to define memory size and attributes of the microprocessor throughput capabilities in random instruction execution, such as those required to implement TCP/IP, versus data movement capabilities. The second component is the definition of hardware to be examined. The hardware to be examined must be defined before the network performance can be examined. This chapter will discuss the types of data and messages expected at a high speed FDDI node and the basic hardware implementation which will be used for further design analyses.

2.1 Data Access Requirements

To define the maximum system requirements for FDDI, we must first examine the realistic expectations of FDDI. Although FDDI will often be used in applications where other LANs may exist today, the increase in data rate indicates that FDDI will probably have a different traffic distribution. A

sample of the data distribution from Star Technology supports the estimation that the data distribution for an FDDI network will be different than for a typical Ethernet network. The data distribution for FDDI includes larger average packet lengths than Ethernet.

FDDI has an available data rate of 100 Mbit/s. Of this, it is expected that FDDI will be able to maintain a continuous throughput of about 80 Mbit/s [14]. The remaining bits are lost to ring overhead and network packetization. Current Ethernet packet distribution figures have often been characterized as shown in Figure 2.1.1. The two distributions shown indicate a number of packets distributed around 32 bytes in length and another group of packets distributed around 558 bytes in length. The distribution of packets located around 32 bytes in length often "contain terminal traffic, acknowledgements, etc." [15]. The distribution of packets located around 558 bytes in length are associated with file transfer traffic. These two types of packets indicate a mixture of commands and data. To use the Ethernet data distribution to predict the FDDI data mixture, we must examine how FDDI is expected to behave in comparison to Ethernet.

There are two major methods describing how FDDI and the associated data will behave. The first is as a backbone network. As a backbone network, FDDI will provide connections

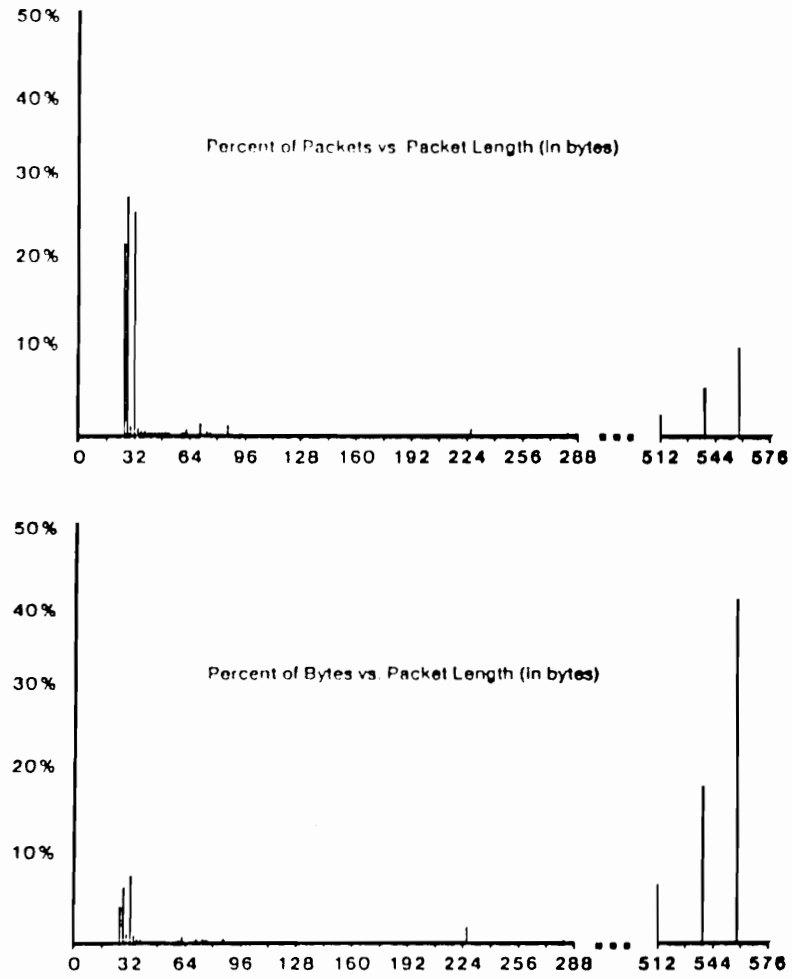


Figure 2.1.1 Packet length for Ethernet as a function of data length and frequency of transportation. ("Measured Performance of an Ethernet Local Network") [15]

between other networks such as Ethernet. An example of a backbone network is shown in Figure 2.1.2. In this type of system, each FDDI node must be able to capture data from the FDDI network and present the data to the connection network, or capture data from the connection network and prepare it for shipping on the FDDI network. When using FDDI in such a mode, the nodes only need to operate as quickly as the connecting networks. In transferring from the slower networks to FDDI, the FDDI node only needs to handle as much data as the connection networks are capable of handling. In transferring from FDDI to the slower network, the buffers must be capable of the backup due to the different bandwidths and the length of data bursts. Since the fastest network in the system is the backbone network, all other systems are slower than FDDI. Therefore, the study of the hardware requirements for a backbone node will be very similar to the node requirements of the networks being connected. These networks are significantly slower than FDDI and are not of interest in this dissertation.

The second method of describing FDDI requirements is when FDDI is used as the main network. This type of network was shown in Figure 1.1. Here, all stations must be capable of receiving and transporting all the data available over FDDI. In this situation, the ring has gained data accessing potential while the computers have not gained in their

performance attributes. Because of this, the number of commands will probably not significantly increase, but "the portion of bulk data transfer .... will probably increase. Thus one would expect to see more large packets on the network in the future than we do currently" [16].

The hardware involved in handling FDDI is the most complex when high data rates are expected. These high throughput rates are seen when FDDI operates as the main network. The network in Figure 1.1 is representative of such a system sharing resources for computational or task sharing advantages. In such systems, it is only advantageous to transfer program or task execution to a remote location when the transfer reduces the overall load to the system. Since transport protocol overhead is largely independent of packet length, the larger the block of data transferred, the lower the overhead associated with each unit of data transferred. Sending the largest blocks possible reduces the cost of transferring each piece of data to the minimum possible. This means that fewer functions must be performed on a byte-by-byte basis for large data transfers than for small data transfers to justify remote execution or services.

There are two types of load sharing which increase the system's performance. The first is when a client requests large amounts of data for use, such as a workstation

requesting a file for editing. The second is when a client requests a server to perform some functions on specialized hardware, such as computer sending a problem to an array processor or a file to a printer. It can be argued that the number of commands between nodes, such as start, stop, store, etc., transferred over either of these types of load sharing networks will be small compared to the amount of data transferred. If the process involved is small, often transferring the request to a server overshadows performing the operation on the client machine because of the overhead associated with any transfer. In FDDI, the transfer protocol overhead must be considered. Moving data or execution is practical only if the data or instruction transfers require significantly less processing or execution than performing the desired function at the client. From this discussion, we have determined that transfers only make sense if the client expends less processing time by offloading the process than by performing it locally.

Due to the overhead associated with transfer protocols and the likelihood that large data packets are transferred between machines, systems utilizing the FDDI data rates will have the majority of the information transferred in large packets. For this reason, nodes operating near the maximum transfer rates associated with FDDI will be processing primarily large block transfers. This conclusion is supported

by the data provided by Star Technologies [17] and will be discussed further in Chapter 4.

Throughout the rest of the dissertation, data transfers and their effects on hardware design will be studied. Because FDDI will tend towards larger data packets, whenever decisions must be made in theoretical analyses, the analysis will be favored towards larger packets. In most cases however, the equations and discussions are applicable to any data mixture.

From the data observed on Ethernet, the number of small packets will probably not increase, because the number of users or the speed of executing the commands will not increase greatly. The overhead increases from 26 bytes used in the study by Shoch and Hupp [15] to 56 bytes as required by the combination of FDDI and TCP/IP overhead. Therefore it is realistic to expect the average small packet which contains 6 bytes of data to contain a total of 62 bytes instead of 32 bytes as reported by Shoch and Hupp. The number of large packets is expected to increase in relationship to the increased data rates and the large packet size is expected to maximize at the 4500 byte limit of FDDI.

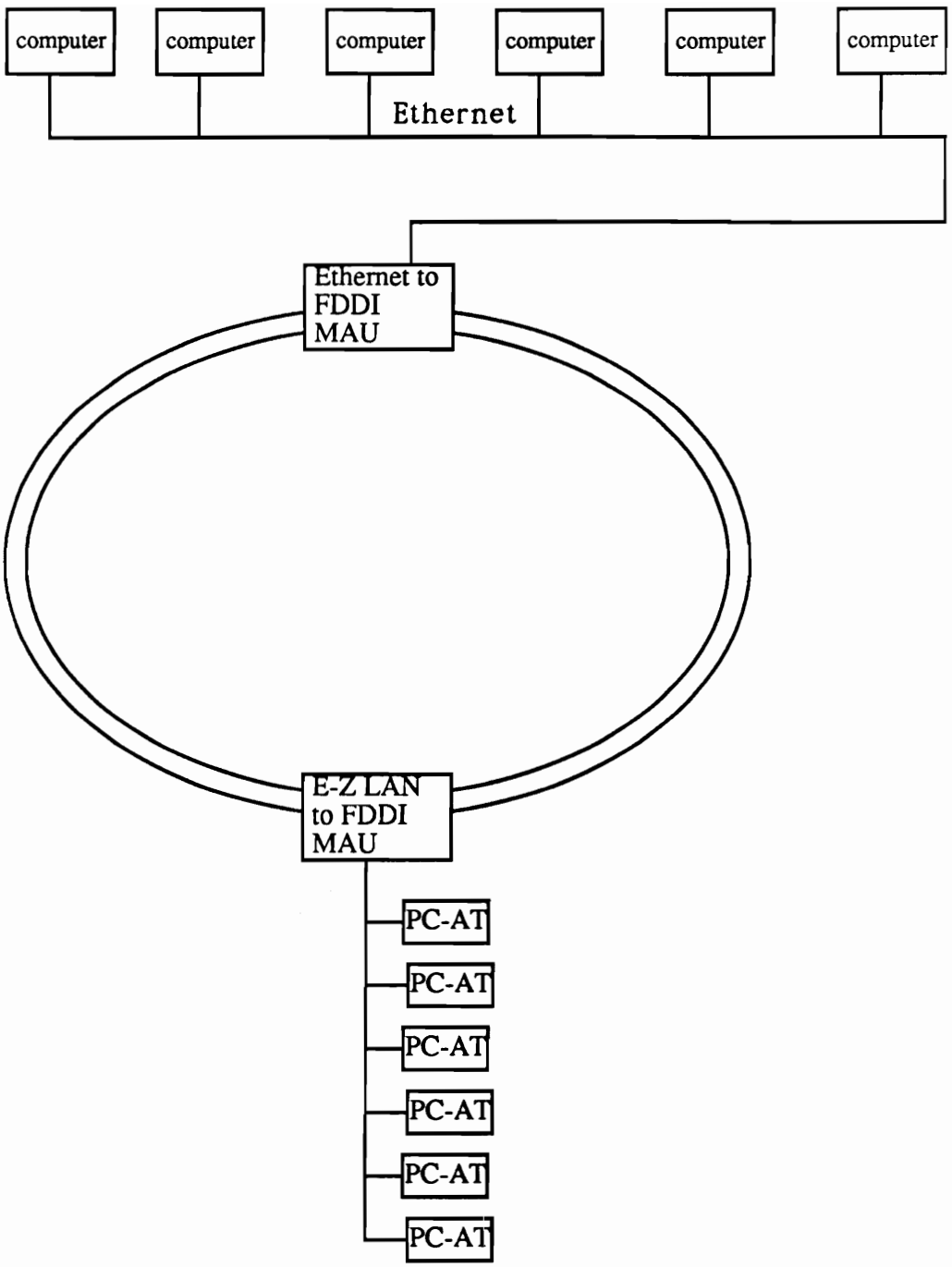


Figure 2.1.2: Example of an FDDI network used as a backbone.

2.2 Hardware Requirements to Operate at FDDI Bit Rates

To meet the needs of FDDI, including data movement, network management and network protocol issues, the interfaces to the network will need to utilize intelligent interfaces, or the host machines must be capable of meeting the interface needs of FDDI without compromising the computer's performance. As was shown in Section 1.3, the data movement alone in FDDI could require a minimum of 12.5 MIPs of processing power to handle the maximum FDDI data rate. This is a significant drain on most computers. To build the interface into a host computer would require knowledge about the FDDI network during the host computer's design. Designing the FDDI interface as an add-on interface or as an extension of the host computer's memory would allow FDDI to be autonomous from the host computer, with only a data interface required. This interface could appear as a port or as extended memory.

From the standpoint of a designer, making the FDDI media access units (MAUs) appear as a data interface to the host computer alleviates the amount of action required by the host computer and thus reduces the penalty to the host computer due to FDDI. The MAU is the hardware used to provide access to the network and provides the information required to convert computer voltage levels and data to signal intensities, network protocols, and packet sequences. There are three

major areas for optimization in the interface. They are shown in Figure 2.2.1 and are described as:

1) the media access unit memory, which is a storage location between the FDDI chip-set and the intelligent network interface.
2) the intelligent network interface, which is responsible for protocol processing , station management and data movement, and
3) the host computer interface memory, which is a data interface between the intelligent network interface and the host computer.

The hardware requirements of each of these segments of the MAU will now be discussed.

2.2.1 MAU Memory

The MAU memory provides a temporary storage location for incoming and outgoing FDDI packets. This location stores the transferred packet information for access by the FDDI chip-set and microprocessor. This unit is comprised of some type of temporary memory, and is used as a queueing area for packets about to enter the network or just removed from the network. The memory will be segmented into two physically

different physical spaces. One for outgoing packet traffic and the other for incoming packets.

The memory must be able to provide real-time access capability of 100 Mbit/s in 32-bit entities and be large enough to store packets which may need temporary storage. The memory must allow simultaneous access by both the intelligent network and the FDDI chip-set. If the memory were not dual ported, the system would need to operate twice as fast to allow reads and writes of the appropriate memories to prevent packet back-up or overrun. Although it is desirable to process packets while the next packet is entering the queue, there will be times when multiple packets will be pending in the packet queue. The memory must be segmented to allow multiple packets to exist and provide for roll-over memory or memory queueing which will indicate which memory is available and which is not.

Based on the above, realize that the FDDI MAU memory must be capable of a minimum of 100 Mbit/s transfer using 32-bit entities, multiple packet storage capability, simplified memory management and dual ported access. The trade-offs in MAU memory and host computer interface memory structure will be evaluated in Section 3.2.

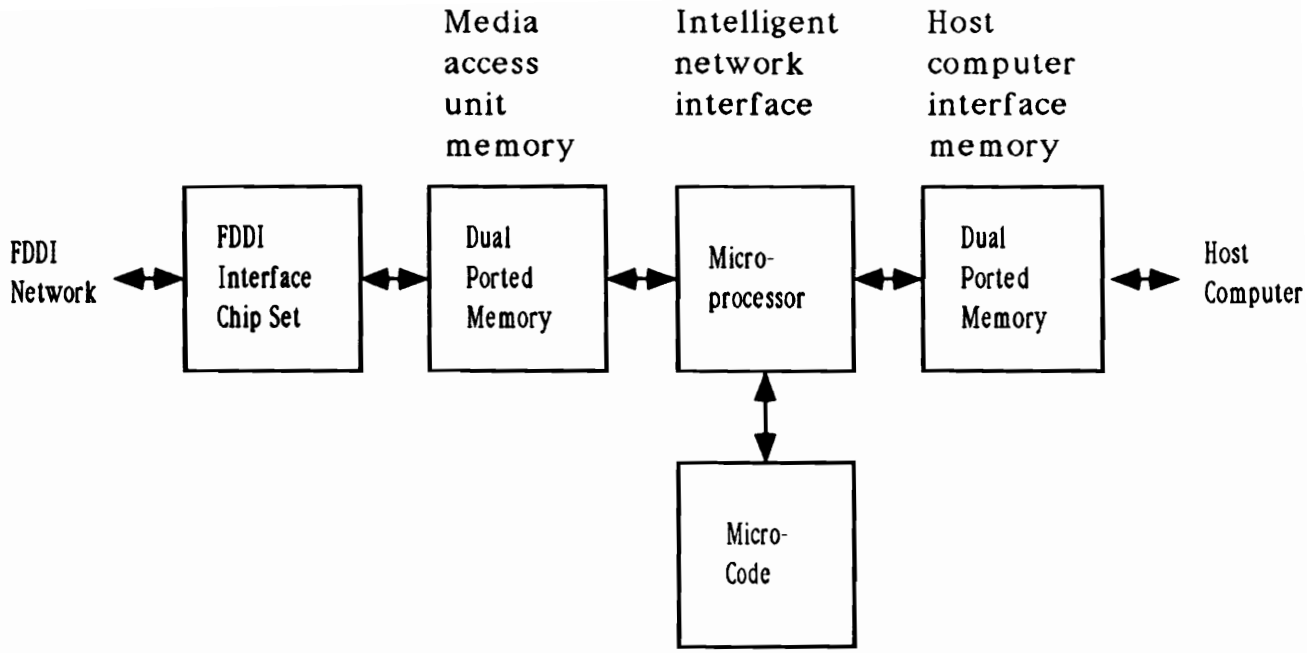


Figure 2.2.1: The major areas investigated to attempt to maximize the throughput of the FDDI interface are the media access unit memory, the intelligent network interface and the host computer interface.

2.2.2 Intelligent Network Interface

High speed networks require intelligent interfaces [18] consisting of a unit providing the processing capability for data movement, network protocol and station management. There are three major options for the design of such a unit:

1) specialized hardware interface,
2) microprocessor based protocol and station management with specialized hardware for data movement and checksum generation, and
3) microprocessor based system performing all functions, except checksum generation.

The specialized hardware interface implementation can be designed to pipeline data between the media access unit memory and the host computer interface memory. Although a specialized hardware interface will provide for efficient data movement, there are three major drawbacks to it. First, the MAU will require different hardware for each network protocol. Second, many protocols could be very difficult to implement in state machine form. Last, as protocols change in the future, new protocols would require new hardware making this option costly for network support and upgrades. For these reasons, a specialized hardware interface is not the most advantageous method of implementing an FDDI MAU.

The microprocessor based system using specialized hardware for data movement can simultaneously provide data movement and protocol acknowledgement. Figure 2.2.2a shows a block diagram of this type of system. The system would have the capability of allowing system upgrades and support in software. The common work of network protocol and station management would be performed at the microprocessor level and thus use the same hardware base for all protocols. There are however, two major drawbacks to the system. First, in circumstances where data and protocol are intricately intertwined, the coordination of memory access between the microprocessor and specialized hardware can be a larger problem than the data movement itself. Second, the additional hardware does not increase functionality since it is possible that the microprocessor has the ability to perform the required data and protocol processing. In fact, the additional hardware could decrease performance. [19] If this system were used with large data transfers, the microprocessor would perform protocol and management functions and be idle during the majority of the time that data is being moved. Furthermore, the added complexity required to implement memory control could be a source of additional design effort and development risk. In the area of advantages, it can provide a fast, flexible method of interfacing the host computer and the network.

The microprocessor-based system including microprocessor data movement provides responsiveness in the system and guarantees protocol compatibility as new network standards evolve. Figure 2.2.2b shows the block diagram of the microprocessor-based system. The design has the same abilities as option 2 in allowing upgrades and changes in network protocol and station management without affecting the hardware. Because the design uses a microprocessor for all data transfer between memories, a single bus structure is all that is needed. This design meets the system needs, provided the protocol and station management functions are small compared to the data movement. The data transfer rate using the microprocessor may be slower than a direct hardware data transfer, but provided the microprocessor-based data movement system can meet the throughput requirements, it will minimize the number of parts required thus increasing system reliability and reducing system size and cost.

The above descriptions explain why options 2 and 3, involving the use of the microprocessor for control and station management provide the best options for meeting data transfer requirements. The designs provide flexibility not available from the hardware solution. The trade-offs between the two options are found in the ease of control, parts count and increased reliability. The advantages of Complex

Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC) in each of these cases will be analyzed in Chapter 3 and evaluated in the fourth chapter of this dissertation. Note that the second and third options are identical in the protocol and station management responsibilities and only differ in the data movement method. The two systems can be compared by viewing the data transfer as a processing delay, independent of whether the data is transferred by the microprocessor, as in the third case, or by external hardware, as in the second case. The comparisons of CISC versus RISC in Section 3.1 will examine this difference.

To meet the system needs, the microprocessor will need the capacity to handle the overhead of network protocol and station management with enough extra processing capacity to support at least a 80 Mbit/s [14] transfer rate or, more probably, the complete 100 Mbit/s transfer rate. Achieving a throughput of 100 Mbit/s would guarantee that all possible data arrival rates for FDDI can be achieved. A flow diagram of the basic microprocessor based protocol and data movement system is shown in Figure 2.2.3. At the receiving end, the microprocessor sends acknowledgements for the network protocols, strips any headers from the data, and places the data in the appropriate memory space for access by the host computer. At the sending end, the microprocessor attaches

headers, generates protocol information and places the data in the interface buffer memory. The microprocessor-based system can accommodate routing changes by adjusting the routing address in the microprocessor code. This flexibility can allow new interface ports to be used simply by addressing the port over the address bus.

Figure 2.2.4 shows the interface from a board level block diagram. The FDDI chip set will perform the network interface between the MAU internal memory and the network. The FDDI chip set will be responsible for the FDDI Media Access Control (MAC), Physical Layer Protocol (PHY) and Physical Media Dependent (PMD) layers, which correspond to the Data Link and Physical layers of the ISO OSI model [20]. The MAU memory will provide a physical interconnection between the Network and Data Link layers. Network and transport layer functions are performed by the microprocessor and takes the form of TCP/IP in the discussions below. The microprocessor also implements the FDDI Station Management (SMT) protocol. By providing all these capabilities in the microprocessor-based MAU, the MAU will control the bottom four layers of the OSI model.

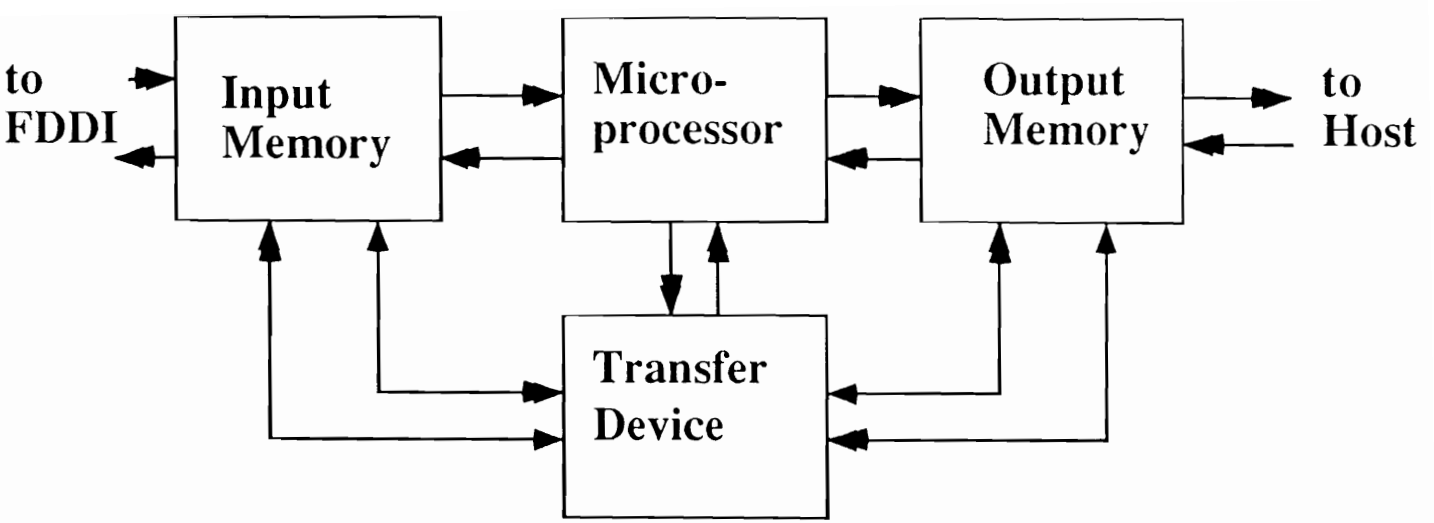


Figure 2.2.2: a) Block diagram of a microprocessor based protocol and station management with specialized hardware for data movement and checksum.
b) Block diagram of a microprocessor based system performing all functions except checksum generation.

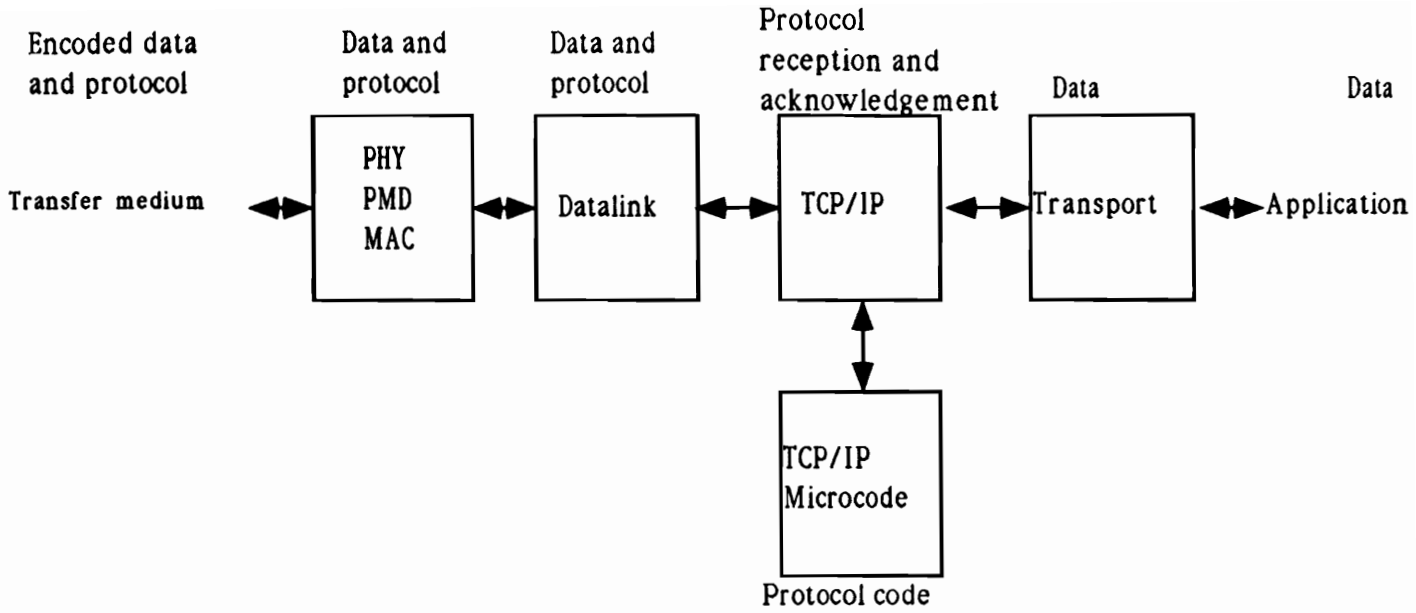


Figure 2.2.3: Flow diagram of a basic microprocessor based protocol and data movement system.

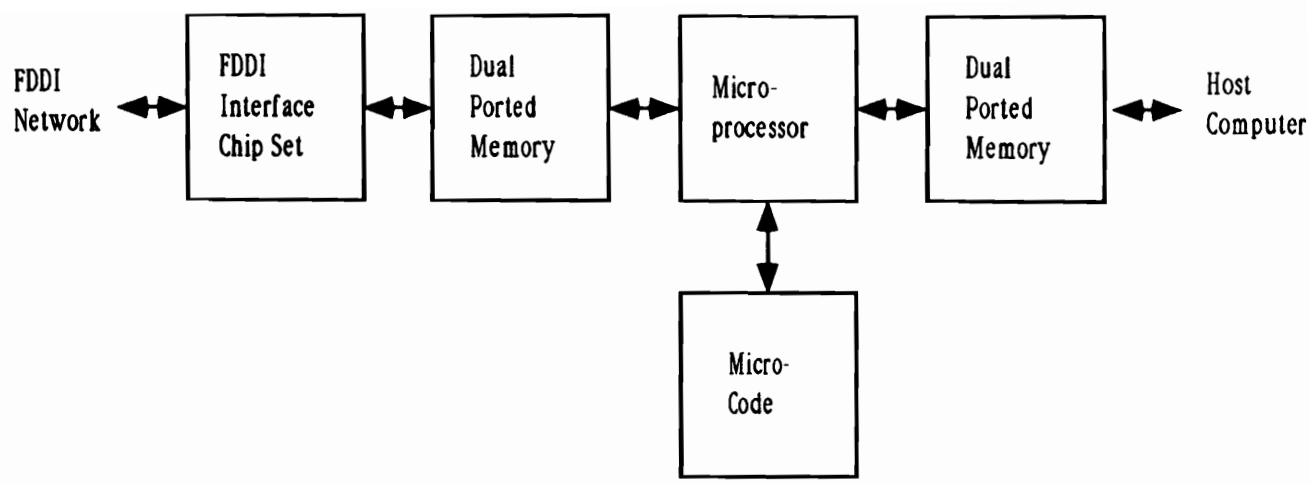


Figure 2.2.4: MAU interface as a board level block diagram.

2.2.3 Host Computer Interface

The host computer interface will provide accessibility of the data between the MAU interface and the host computer. If this design is too cumbersome, the advantages of having a microprocessor-based MAU interface will be lost. The interface will need to provide the MAU and host computer access to the data. In the design, there are two possible options of how the host computer can access the data. The first is as extended host computer memory. In this fashion, the board will appear as extended memory and be accessed accordingly. The advantages are that the interface is simple and only requires the host computer to know of the memory location and that data is available. If the data is used and rapidly discarded, this extended memory may provide the best access method. Note that because of the electrical problems associated with fast access memory, this memory needs to be located close to the main computer memory and due to delays associated with overhead processing the average internal data transfer rate needs access times in excess of the memory speeds required by FDDI. Also, if the interface is viewed as extended memory, the memory must not conflict with host memory addresses. Additionally, the method of transfer between the host computer and the interface must allow the same physical memory space to be read and written to by both the host and the MAU. Other advantages of this system are that the

extension is an add-on to basic memory and will cause little or no delay between the MAU and the host computer. When compared to the use of the interface as an extra port, the number of transfers and throughput delay are both decreased.

A second possible scenario is for the host computer interface to appear as a port accessed by well defined methodologies such as VMEbus transfers. This type of transfer moves network data from the interface to the host computer main memory by use of a standard interface such as VMEbus access protocols. This type of interface will allow easy segmentation between physical input and output memories. The memory size, control and interface access will be examined in Chapter 3 and will be evaluated against system requirements to be discussed in Chapter 4.

2.3 Conclusion

Chapter 2 has shown the requirements for an FDDI MAU if it is to meet data transfer rates of 100 Mbit/s on a continuous basis. In addition, the MAU will need to provide data protocol acknowledgements and receptions as well as station management obligations. The general data distribution will consist mostly of large data packets. The large and small data packets will require similar amounts of overhead for protocol acknowledgement and reception. Three major areas

of concern need to be examined to determine the most acceptable method of designing an FDDI interface. Each has some major questions to be resolved. They can be summarized as follows.

1) MAU memory - Must provide a minimum data transfer rate of 100 Mbit/s and be large enough to accommodate a reasonable distribution of data, especially small packets which take a longer time per byte transferred to process than long packets due to the data overhead.

2) Microprocessor-based Intelligent Network - The trade-off between CISC and RISC microprocessors need to be examined accounting for data size distributions and data throughput. Examination of the impact of the microprocessor type and processing capability on the MAU memory and the host computer interface memory will be covered in Chapters 3 and 4.

3) Host Computer Interface - This interface needs to provide a reasonable access methodology for the host computer to ensure that data is available and that the exchange of data between the MAU and the host computer is reasonable in overhead and does not slow the host computer's ability to perform processing. This interface needs to be properly sized to minimize address reservations, hardware requirements, and complexity.

These items will be analyzed in Chapter 3 and evaluated

for a specific design requirement in Chapter 4.

Chapter 3

Effects of Processors, Data Access and Data Segmentation on FDDI Attributes

In the FDDI environment, data rates are pressing the ability of the MAU to perform the required functions during a packet's arrival time. When the microprocessor is not able to keep up with the message protocol and data movement, messages begin to back-up in the data queue. As the data remains in the buffer, the buffer size needed to receive messages grows. If data can be processed as it is received, or if a majority of messages can be processed upon their reception, the host computer can receive data in a timely fashion and the total network begins to function as a very large distributed computer. Beyond FDDI, as data rates continue to grow, the trade-offs between buffer size, memory size, protocol processing and data transport becomes even more critical.

FDDI provides a unique opportunity to study the network interface system at a rate requiring an intelligent network, but not so fast that only a hardware solution is possible. Because of the data rate of FDDI as compared to the available processing rates, we examine the effect on throughput caused by different methods of data transport, data packet sizes and

microprocessor architectures. In addition, some conclusions about network protocols are discussed.

In Section 3.1, the microprocessor architectures are examined for throughput capabilities. General equations for throughput capabilities are given for different system architectures as well as different microprocessor architectures. Equations are generated to predict throughput as a function of packet size, protocol, and microprocessor-specific attributes. In addition, equations are developed to predict the microprocessor clock rates required to achieve the data rates available in the FDDI environment as a function of data message size.

In Section 3.2, memory structures and data transfer widths are examined. The argument for 32-bit data transfers is be presented. In the area of memory sizing and configuration, we examine the effects of the data-to-protocol overhead ratio on storage and segmentation. In addition, we examine the effects of transfer strategies on throughput and data access methods from the point of view of the host computer. In addition, equations are be developed to predict memory sizes required to have reliable communication based on data distributions and microprocessor rates and throughput capabilities.

In Section 3.3, the transfer structure between the MAU and the host computer is examined. The basic area under examination is whether to use a standard interface such as VMEbus instead of direct memory access for transferring data. This section focuses on hardware capability, throughput and latency associated with the use of VMEbus in comparison to direct memory access between logical divisions.

3.1 Media Access Unit (MAU) Microprocessor Analysis

The design of an FDDI MAU has three major portions, a MAU memory, a microprocessor, and a host computer interface. This section focuses on the effects of data length, method of transfer and protocol overhead on the selection of the microprocessor type to meet the needs of the FDDI data rates.

During the past several years, many benchmark tests have been performed show that Reduced Instruction Set Computer (RISC)-based architectures have faster execution times Complex Instruction Set Computer (CISC)-based architectures for general benchmark programs. In some situations which do not follow the standard benchmark mixture of instructions, operations requiring extensive movement, CISC-based architectures have some advantages over RISC-based architectures. Throughout this section, RISC is defined as

not having a string move capability and CISC does not use a reduced instruction set and does have a string move capability. For representative CISC and RISC microprocessors the models developed in this section indicate that CISC microprocessors outperform RISC microprocessors in FDDI and other high speed networks where data movement is performed by the microprocessor and the amount of data transferred is significantly higher than the amount of protocol processing overhead. For situations where the data movement is performed external to the microprocessors, the RISC microprocessor is slightly faster than the CISC processor. For the systems shown, the additional hardware required for external data movement is not justified because the CISC-based machine using internal transfers can meet the needs of the system.

In Chapter 2, a discussion presented the theory that at the higher speed of FDDI two major data flows were possible, namely

1) the data would flow around the ring faster but would not arrive at any specific computer any faster than in the low speed networks such as Ethernet, and
2) the high speed data rate devices would be large data packet intensive.

Since many machines are currently able to keep up with

slower speed networks such as Ethernet, only the higher speed, large data packet intensive networks are of primary importance. To provide proper linkage between data and throughput, this section assumes data and memory are properly matched to prevent rejected data. [21] The memory sizing and the relation between microprocessor throughput and memory size will be presented in Section 3.2.

3.1.1 Microprocessor Functional Requirements

As discussed in Section 2.2.2, the intelligent network interface will be based around the microprocessor which is performing the network protocol functions. In addition, there are two hardware systems available, either 1) the microprocessor will be responsible for actually sending the data between the MAU memory and the host interface (see Figure 2.1.1b), or 2) the microprocessor will be responsible for implementing the instructions required to start the transfers (see Figure 2.1.1a). In either of the hardware models, the basic microprocessor architectures can be either RISC or CISC. The purpose of this chapter is to develop equations to compare the processors for each of the hardware systems above, and to provide a method of comparing the different hardware systems.

When the microprocessor performs the actual transfers as

in hardware system 1, it begins the TCP/IP processing, performs the data transfers, compares the checksums and acknowledges the successful completion of the transfer or preparation of the TCP/IP header. Since FDDI has a high reliability and a bit error rate (BER) of 10^{-12} or less, it is expected that there is less than one error per 27.7 million messages. For this reason, error recovery is not considered in estimating throughput. Throughput values for the system performing data transfer internal to the microprocessor will be discussed in Section 3.1.3.

When the microprocessor does not perform the actual transfers as in hardware system 2, it begins the requirements for TCP/IP processing, initiate the transfer, compare the checksums and acknowledge the successful completion of the transfer or preparation of the TCP/IP header. In this system, the microprocessor will be idle during the transfer period. The interface between the memory devices will be based on a synchronous design because synchronous systems are easy to understand and over a long lifetime where parts may change characteristics greatly, there is little or no practical advantage to asynchronous designs [22]. Because of this synchronous design, transfer rates will be a function of the basic microprocessor clock rate. Throughput values for the system performing data transfer external to the microprocessor will be discussed in Section 3.1.4.

3.1.2 Microprocessors

In implementing the interface as defined, a major concern is the type of microprocessor to be used. To adequately compare the types of microprocessors available, the groupings of CISC and RISC are examined. CISC is often associated with the Intel 80x86 microprocessor family. For the analyses given below, statistics for the Intel 386 are used. RISC is usually associated with a microprocessor capable of limited instructions, each comprising a single clock cycle [23], or limited to very basic instructions, such as LOAD or STORE. For the comparisons below, the Cypress SPARC microprocessor was selected. The SPARC architecture is a classic example of RISC architecture and contains no major multi-cycle instructions such as BLOCK MOVE or STRING MOVE.

3.1.2.1 Cypress SPARC Microprocessor

The Cypress SPARC microprocessor [24] uses a RISC architecture with a set of simple commands that are executed quickly. The SPARC chip uses a pipeline for preparation of instructions. The pipeline has four stages: Instruction Fetch, Decode, Execute and Write. In branching, this chip

uses special hardware to evaluate conditionals and perform the branch instruction beginning in the decode stage rather than in the execute stage of the pipeline. In doing so, the branch instruction is allowed to be brought into the pipeline with only a single clock period delay at the microprocessor level. The conditionals, such as Branch on Less or Equal (BLE), execute in a single instruction time, and delay the next instruction by a single clock period while the next instruction is being sent through the pipeline. The end result is that in most cases a branch conditional can be thought of as a two cycle instruction. Attempting to fill the cycle after the conditional jump could create some difficulties in programing. The SPARC has a 32-bit data bus and a separate 24-bit address bus.

For many applications, SPARC chips can execute programs faster than a 386 when the chips are operating at similar speeds. The SPARC microprocessor will need to execute 1.3 times as many instructions as the 386, but will execute the average instruction in a single cycle, while the 386 will execute its average instruction in 2.5 clock cycles. An example of the Cypress SPARC instructions set and execution cycles is shown in Figure 3.1.1.

	Name	Operation		Cycles
Load and Store Instructions	LDSB(LDSBA*) LDSH(LDSHA*) LDUB(LDUBA*) LDUH(LDUHA*) LD(LDA*) LDD(LDDA*)	Load Signed Byte Load Signed Halfword Load Unsigned Byte Load Unsigned Halfword Load Word Load Doubleword	(from Alternate Space) (from Alternate Space) (from Alternate Space) (from Alternate Space) (from Alternate Space) (from Alternate Space)	2 2 2 2 2 3
	LDF LDDF LDFSR	Load Floating Point Load Double Floating Point Load Floating Point State Register		2 3 2
	LDC LDDC LDCSR	Load Coprocessor Load Double Coprocessor Load Coprocessor State Register		2 3 2
	STB(STBA*) STH(STHA*) ST(STA*) STD(STDA*)	Store Byte Store Halfword Store Word Store Doubleword	(into Alternate Space) (into Alternate Space) (into Alternate Space) (into Alternate Space)	3 3 3 4
	STF STDF STFSR STDFQ*	Store Floating Point Store Double Floating Point Store Floating Point State Register Store Double Floating Point Queue		3 4 3 4
	STC STDC STCSR STDCQ*	Store Coprocessor Store Double Coprocessor Store Coprocessor State Register Store Double Coprocessor Queue		3 4 3 4
	LDSTUB(LDSTUBA*) SWAP(SWAPA*)	Atomic Load/Store Unsigned Byte Swap r Register with Memory	(in Alternate Space) (in Alternate Space)	4 4
Arithmetic/Logical/Shift	ADD(ADDcc) ADDX(ADDXcc)	Add Add with Carry	(and modify icc) (and modify icc)	1 1
	TADDcc(TADDccTV)	Tagged Add and modify icc	(and Trap on overflow)	1
	SUB(SUBcc) SUBX(SUBXcc)	Subtract Subtract with Carry	(and modify icc) (and modify icc)	1 1
	TSUBcc(TSUBccTV)	Tagged Subtract and modify icc	(and Trap on overflow)	1
	MULScc	Multiply Step and modify icc		1
	AND(ANDcc) ANDN(ANDNcc) OR(ORcc) ORN(ORNcc) XOR(XORcc) XNOR(XNORcc)	And And Not Inclusive Or Inclusive Or Not Exclusive Or Exclusive Nor	(and modify icc) (and modify icc) (and modify icc) (and modify icc) (and modify icc) (and modify icc)	1 1 1 1 1 1
	SLL SRL SRA	Shift Left Logical Shift Right Logical Shift Right Arithmetic		1 1 1
	SETHI	Set High 22 Bits of r Register		1
	SAVE RESTORE	Save caller's window Restore caller's window		1 1
Control Transfer	Bicc FBicc CBccc	Branch on Integer Condition Codes Branch on Floating Point Condition Codes Branch on Coprocessor Condition Codes		1** 1** 1**
	CALL	Call		1**
	JMPL	Jump and Link		2**
	RETT	Return from Trap		2**
	Ticc	Trap on Integer Condition Codes		1 (4 if Taken)
Read/Write Control Registers	RDY RDPSR* RDWIM* RDTBR*	Read Y Register Read Processor State Register Read Window Invalid Mask Read Trap Base Register		1 1 1 1
	WRY WRPSR* WRWIM* WRTBR*	Write Y Register Write Processor State Register Write Window Invalid Mask Write Trap Base Register		1 1 1 1
	UNIMP	Unimplemented Instruction		1
	IFLUSH	Instruction Cache Flush		1
FP (CP) Ops	FPop	Floating Point Unit Operations		1 to Launch
	CPop	Coprocessor Operations		1 to Launch

* privileged instruction

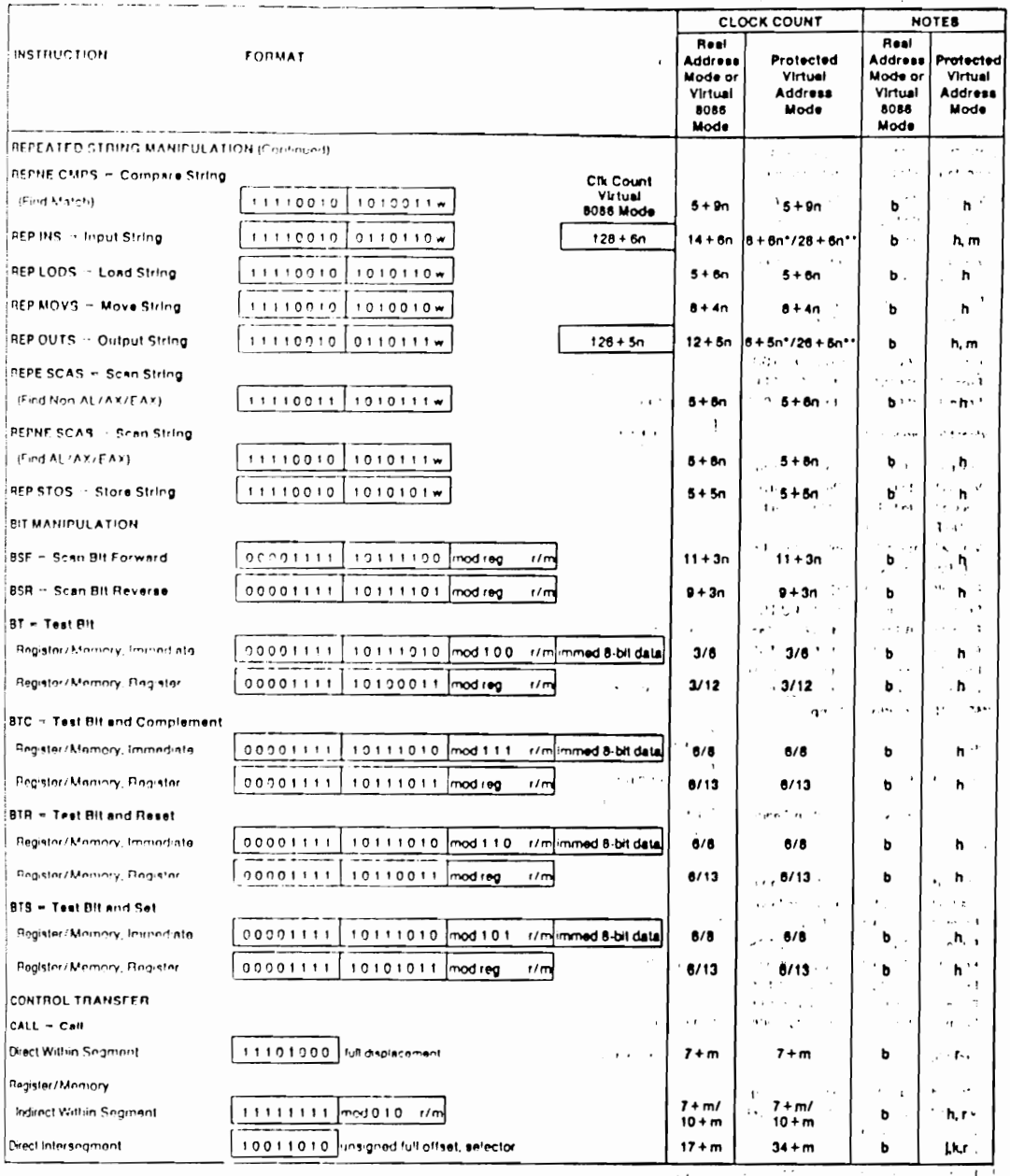
** assuming delay slot is filled with useful instruction

Figure 3.1.1: SPARC Instruction Set. Reprinted from SPARC RISC USER's GUIDE. [24, Table 6-2]

3.1.2.2 Intel 386 Microprocessor

The Intel 386 microprocessor [12] uses a CISC architecture, supporting a mix of simple and complex instructions. The 386 uses instruction pipelining, executing up to six instruction steps in parallel. Not all steps are performed for each instruction, but each step is available for each instruction. The steps available for each instruction are decode, execution, memory management and bus access. The average instruction takes 2 or 3 clock cycles to execute. An example of the 386 instruction set is shown in Figure 3.1.2. In addition to the instructions shown, the 386 maximizes data throughput by using address pipelining.

Address pipelining allows a 386 to operate with a faster basic clock rate than the SPARC given a common memory speed, since address pipelining allows the address and control information of a memory access process to begin updating the next memory address before the end of the current microprocessor cycle. The time used to read or write the data remains the same, but the delay between the microprocessor clock and the availability of a valid address is decreased compared to that of non-pipelined addressing. In effect, pipelining presents the address to the memory immediately after the next clock cycle instead of after the delay between the clock and the address generation. The removal of the



INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
REPEATED STRING MANIPULATION (Continued)					
REPNE CMPS = Compare String		Clk Count Virtual 8086 Mode			
(Find Match)	11110010 1010011w	5+9n	5+9n	b	h
REP INS = Input String	11110010 0110110w	14+6n	8+6n*/28+6n**	b	h, m
REP LODS = Load String	11110010 1010110w	5+6n	5+6n	b	h
REP MOVS = Move String	11110010 1010010w	8+4n	8+4n	b	h
REP OUTS = Output String	11110010 0110111w	12+5n	8+5n*/26+5n**	b	h, m
REPE SCAS = Scan String					
(Find Non AL/AX/EAX)	11110011 1010111w	5+6n	5+6n	b	h
REPNE SCAS = Scan String					
(Find AL/AX/EAX)	11110010 1010111w	5+6n	5+6n	b	h
REP STOS = Store String	11110010 1010101w	5+5n	5+5n	b	h
BIT MANIPULATION					
BSF = Scan Bit Forward	00001111 10111100 mod reg r/m	11+3n	11+3n	b	h
BSR = Scan Bit Reverse	00001111 10111101 mod reg r/m	9+3n	9+3n	b	h
BT = Test Bit					
Register/Memory, Immediate	00001111 10111010 mod 100 r/m immed 8-bit data	3/6	3/6	b	h
Register/Memory, Register	00001111 10100011 mod reg r/m	3/12	3/12	b	h
BTC = Test Bit and Complement					
Register/Memory, Immediate	00001111 10111010 mod 111 r/m immed 8-bit data	6/8	6/8	b	h
Register/Memory, Register	00001111 10111011 mod reg r/m	6/13	6/13	b	h
BTR = Test Bit and Reset					
Register/Memory, Immediate	00001111 10111010 mod 110 r/m immed 8-bit data	6/8	6/8	b	h
Register/Memory, Register	00001111 10110011 mod reg r/m	6/13	6/13	b	h
BTS = Test Bit and Set					
Register/Memory, Immediate	00001111 10111010 mod 101 r/m immed 8-bit data	6/8	6/8	b	h
Register/Memory, Register	00001111 10101011 mod reg r/m	6/13	6/13	b	h
CONTROL TRANSFER					
CALL = Call					
Direct Within Segment	11101000 full displacement	7+m	7+m	b	h
Register/Memory					
Indirect Within Segment	11111111 mod 010 r/m	7+m/ 10+m	7+m/ 10+m	b	h, r
Direct Intersegment	10011010 unsigned full offset, selector	17+m	34+m	b	h,r

NOTES:
† Clock count shown applies if I/O permission allows I/O to the port in virtual 8086 mode. If I/O bit map denies permission exception 13 fault occurs; refer to clock counts for INT 3 instruction.
* If CPL ≤ IOPL ** If CPL > IOPL

Figure 3.1.2: Sample of the Intel 386 instruction set and clock cycle timing from Intel Corporation, Microprocessor and Peripheral Handbook, Volume 1- Microprocessor [12].

delay allows the 386 to use a faster clock than the SPARC for a fixed memory access time.

3.1.3 Throughput Analysis for a Microprocessor-Based System Using the Microprocessor to Perform Data Movement

Throughput analysis is separated into three major areas.

1) Number of packets processed per second based on the number of bytes of data per packet. The microprocessor type and operational speed are varied in the analysis to form a complete picture of the impact of transfer philosophies on packet throughput.

2) Number of bits per second transferred based on the number of bytes of data per packet. Again, the microprocessor type and operational speeds are varied in the analysis to form a complete picture of the impact of transfer philosophies on data throughput.

3) Required microprocessor clock rate to meet the 100 Mbit/s data throughput based on the number of bytes

of data per packet. Once again, the microprocessor type is varied in the analysis to form a complete picture of the impact of transfer philosophies on required clock rates.

In the throughput analysis, three major functions of the MAU are considered. These three functions are

1) packet protocol preparation or acknowledgement,
2) station management functions, and
3) data movement.

By comparing the amount of data and the number of data packets that can be passed from the network to the host computer, we will determine which type of microprocessor most effectively meets the FDDI requirements. The focus for this study is the ability to move large amounts of data between the internal MAU memory and the host interface memory. To prevent the necessity of extra buffers, the optimum design would allow the microprocessor to meet the FDDI burst rate of 100 Mbit/s. If this burst rate cannot be met, buffers will be needed to ensure that data is not lost due to buffer overwrite or rejection. Lost packets may require retransmission of complete messages or packets. Throughout the analysis, the same microprocessor clock rate is assumed. The model, however, can be used with other clock rates.

The throughput estimate first accounts for protocol preparation and acknowledgement. Estimates for protocol preparation and acknowledgement are based on TCP/IP. The protocol reception includes any operations required to convert the TCP/IP packet to a data packet. The protocol acknowledgement includes preparation of an acknowledgement message and other items associated with TCP/IP reception. Note that sending a packet is similar in construct, requiring a message to be built and an acknowledgement to be received. TCP/IP will require approximately 300 CISC instructions to transmit or receive a message [25]. By assuming each message requires reception and acknowledgement, approximately 600 CISC instructions are required for each message, 300 for receiving the message and 300 for preparing an acknowledgement. This value of 300 CISC instructions is for a properly operating network and assumes the checksum is generated by external hardware. Note that this overhead is independent of the number of bytes in the data message. The TCP/IP header uses 40 bytes [25] of the FDDI packet.

The throughput estimate also accounts for station management (SMT) functions. SMT is responsible for items such as connection management, station initialization, station insertion, station removal, fault isolation, fault recovery, scheduling policies and statistical data collection. The

overhead required to perform station management functions is estimated to be 0.01 percent to 0.1 percent of the microprocessor work load [26]. The actual processing time required for SMT may be even smaller and the overhead is estimated per unit of time, rather than per message passed. Since this value is small and affects the throughput on a time basis, it is not included in the calculations below. If the station management cost is incorporated into the calculations, the throughput numbers may be reduced by 0.01 percent to 0.1 percent.

Data movement is the most significant factor in the throughput estimates. Data movement in the CISC microprocessor utilizes the 386 MOVS (string move) command, while the RISC microprocessor must perform a load, store, increment pointer and compare/jump sequence. In Figure 3.1.3, the 32 and 64 bit load and store sequences are estimated. Although FDDI formats data movements in byte increments, only 32-bit increments are used in the calculations. Note that all movements of data could use 32 bit instructions for the majority of the process and small increment moves, 8 or 16 bit moves, for the remainder. The small increment movements require approximately the same number of clock cycles in the RISC and CISC microprocessors.

Figure 3.1.3

A) CISC Assembly Code Instruction for String Moves

Instruction	Clock Cycles	Comments
REP MOVS	4n+7	n is the number of 32-bit entities transferred. The additional 7 clocks are the set-up cost.

B) RISC Assembly Code Instruction[3] for 32-bit String Moves

	Instruction	Clock Cycles	Comments
TOP:	LD [Base1 + offset], reg	2	Loads word from memory for (Base1 + offset) address into destination register.
	ST [Base2 + offset], reg	3	Stores word in destination register into (Base2 + offset) memory address.
	ADD offset, FOUR	1	Increment offset to point to next memory location.
	BLE offset, Repvalue, TOP	2	If the offset is less than the Repeated Value, go back to the top and send additional data. Otherwise, exit the routine.

TOTAL : 8 clocks per 32-bit entity transferred.

[3] Note all symbols such as Base1, Base2, offset, FOUR, EIGHT, nd Repvalue are values existing in registers.

Figure 3.1.3 (Continued)

C) RISC Assembly Code Instruction[4] for 64-bit String Moves

	Instruction	Clock Cycles	Comments
TOP:	LDD [Base1 + offset], reg	3	Loads double word from memory for (Base1 + offset) address into destination register.
	STD [Base2 + offset], reg	4	Stores double word in destination register into (Base2 + offset) memory address.
	ADD offset, EIGHT	1	Increment offset to point to next memory location.
	BLE offset, Repvalue, TOP	2	If the offset is less than the Repeated Value, go back to the top and send additional data. Otherwise, exit the routine.
TOTAL:	10 clocks per 64-bit entity transferred.		

Figure 3.1.3: Assembly level code with associated clock cycle accumulations required to perform string move functions for a) A CISC microprocessor, b) A RISC microprocessor using 32-bit data entities, and c) A RISC microprocessor using 64-bit data entities.

[4] Note all symbols such as Base1, Base2, offset, FOUR, EIGHT, nd Repvalue are values existing in registers.

3.1.3.1 Throughput Estimates

The throughput estimate calculates the amount of data or number of messages that can be accommodated by the FDDI MAU as a function of the number of bytes of data per message. Seven parameters are used in calculating the results. These are discussed below. With each figure representing data throughput, a table is listed giving the values used.

The significant values under consideration in the equations are as follows.

1) MCR - Microprocessor Clock Rate.

2) BPP - Bytes per Packet. For the calculations below, this value is considered to correspond to a 32-bit boundary (or 64-bit in the case of the RISC-based microprocessor using 64-bit storage). In an actual implementation, if the boundaries are not on these segments, the most efficient use of the microprocessor could be made by moving on 32-bit boundaries, and ignoring the extra bits moved.

3) IMO - Instructions for Message Overhead. This value includes TCP/IP processing and any set-up costs. The basic cost for TCP/IP is defined as 600 instructions. In the case of the 386, the set-up cost is 7 clock cycles to prepare for the string move command. In the case of the SPARC, there is

no set-up cost, because there is no string move operation.

4) ECI - Equivalent CISC Instructions. This is a multiplying factor that transforms the number of instructions estimated in CISC to the microprocessor in question. For CISC to CISC, the value is 1. For CISC to RISC, the value is 1.3 [25].

5) CPI - Clocks per Instruction. This value is the average number of clock cycles expected per instruction for the microprocessor type under examination. This value in combination with ECI gives the expected number of clock cycles to execute a typical set of instructions for a given microprocessor type knowing only the number of CISC instructions estimated for a function. For CISC, this value is 2.5, and for RISC, this value is 1 [23] [25].

6) BPX - Bytes per Transfer. This value is the number of bytes transferred for each data unit transferred. In the examples below, BPX values of 4 or 8 correspond to 32-bit and 64-bit transfers respectively. If 8 or 16-bit transfers were used, the associated transfer values would be used.

7) CPX - Clock Cycles per Transfer. This value is the number of clock cycles associated with each data unit, defined in BPX, transferred through the microprocessor. It includes the transfer and

control overhead, such as loads, stores and compares in the RISC instruction set used to transfer data and determine if the transfers are complete. For CISC, this is the clock cycles per MOVS command. MOVS is repeated to move each data element.

The number of messages transferred to the MAU per second can be calculated as the microprocessor clock rate divided by the number of clocks required for TCP/IP processing and data transfer for a given message length. The data transfer rate is a function of the number of bytes in the message multiplied by the number of clock cycles per transfer size.

$$MPS = \frac{MCR}{IMO * CPI * ECI + \frac{BPP * CPX}{BPX}},$$

where

MPS = Messages Transferred per Second

Plots of the number of messages transferred per second as a function of microprocessor type and message length are given in Figure 3.1.4. The values used in generating Figure 3.1.4 are listed in Table 3.1.1. The bytes per message (BPP) is the variable along the x-axis. The maximum transfer rate is plotted along the y-axis.

Note that the values for small messages indicate the maximum number of messages that can be processed if the microprocessor is only performing TCP and not processing data, and the microprocessor always has messages available for processing. Figure 3.1.4 shows that the maximum number of messages is processed when the number of bytes per message is small. However, in this case the data throughput is higher when the larger messages are passed.

The number of bits transferred per second can be calculated by taking the value used to determine the number of messages per second transferred and multiplying it by the number of bits per message. Expressed in terms of the values defined earlier, the equation is

$$BPS = \frac{8 * MCR * BPP}{IMO * CPI * ECI + \frac{BPP * CPX}{BPX}},$$

where

BPS = Number of Bits per Second Transferred.

Plots of the data rate as a function of microprocessor type and message length are given in Figures 3.1.5 and 3.1.6. The values used in generating Figure 3.1.5 are those used in Table 3.1.1. Bytes per message (BPP) is plotted along the x-axis, and the maximum transfer rate is plotted along the y-axis. Figure 3.1.6 is similar to Figure 3.1.5, except the clock rate is changed to 33 MHz.

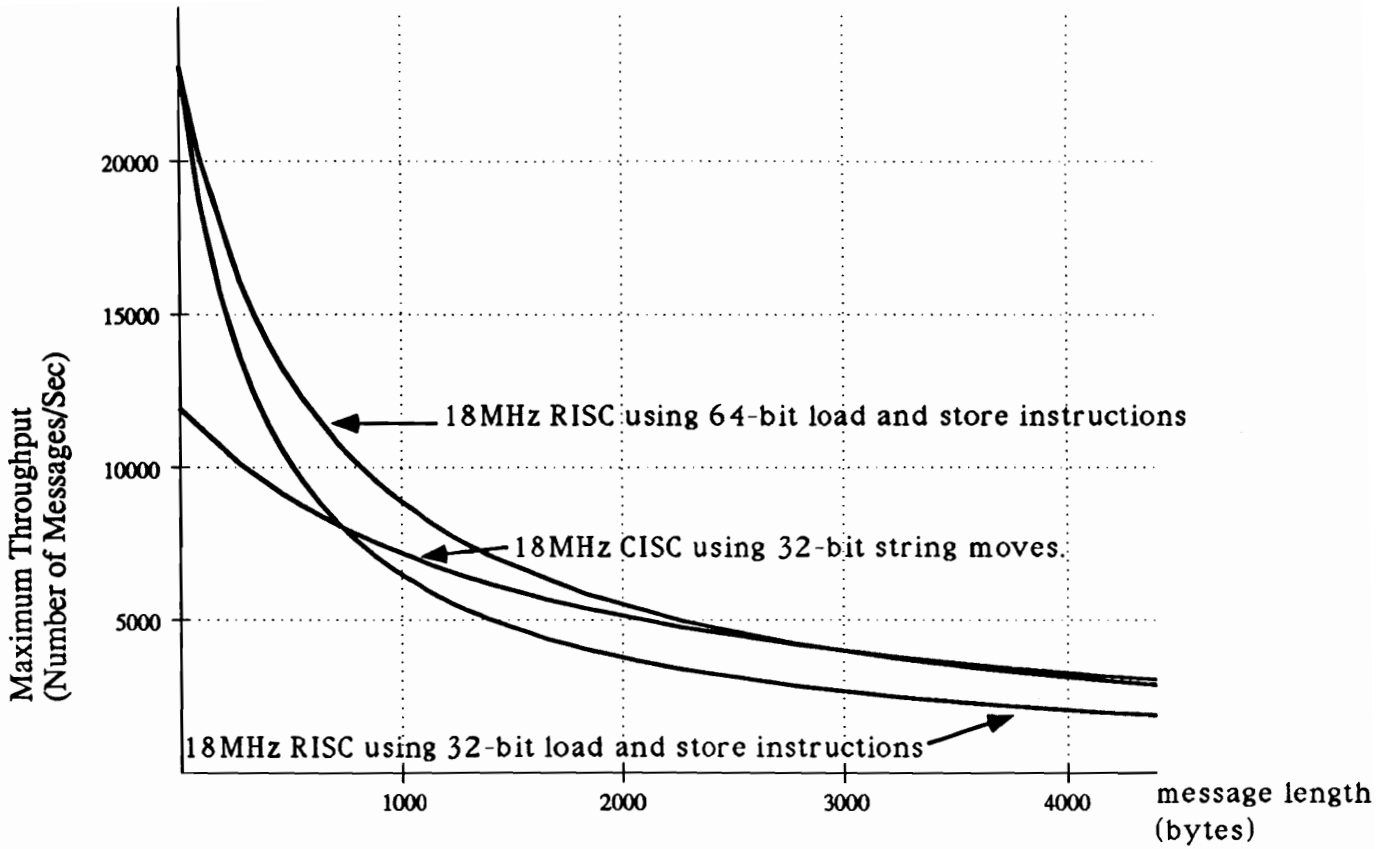


Figure 3.1.4: Number of messages processed as a function of message length (in bytes) and processor type using parameter values in Table 3.1.1.

Table 3.1.1: Values used in predicting the number of messages per second for CISC and RISC based microprocessors.

Variable	Value CISC [12] [25] 32 bit Move	Value RISC [24] [25] 64 bit Move	Value RISC
MCR	18 MHz	18 MHz	18 MHz
ECI	1	1.3	1.3
CPI	2.5	1	1
IMO	607	600	600
CPX	4	8	10
BPX	4	4	8

Figures 3.1.7 and 3.1.8 show the effect of increasing and decreasing the number of instructions associated with the message overhead. The estimate of 450 instructions per message is derived by assuming that two messages, rather than one, are acknowledged by a single TCP/IP message. The estimate of 2000 instructions assumes a large overhead for the operating system used in the MAU as well as non-optimized code. By comparing the estimates of throughput, we see that as the ratio of message overhead to data increases, the data movement advantages of CISC are reduced. Note also that as the ratio of message overhead to throughput decreases, CISC has a greater data throughput than RISC for smaller packets. By comparing the curves and equations, it can be seen that CISC outperforms RISC if the network operations such as TCP/IP and network management require a significantly smaller amount of microprocessor interaction than the data movement.

3.1.3.2 Required Clock Rates

When the data rate, protocol overhead, and message size are known, the required microprocessor clock rate to allow the MAU to provide the processing rate required to meet the needs of FDDI can be calculated. Note that for small messages, the amount of overhead can severely reduce the number of messages

required and therefore the associated number of protocol overheads which must be processed to keep up with the data rate. Figure 3.1.9 shows a comparison of the number of packets to be processed when overhead is and is not taken from the available data rate. Note that for small packets, this corresponds to a considerable reduction in the microprocessor burden. In addition, by knowing the microprocessor rate, which assumes zero wait state memory, the memory access speed can be calculated. The equation for determining the required clock rate is

$$MCR = \frac{BR}{8 * (BPP + OHB)} * (IMO * CPI * ECI + \frac{BPP * CPX}{BPX}),$$

where,

BR is the desired data rate of 100 Mbit/s, and

OHB is the number of overhead bytes in the transmitted packet.

Figure 3.1.10 shows the required microprocessor clock rate as a function of data message length and microprocessor type. The bit rate is 100 Mbit/s, the number of overhead bytes is 56, 40 for TCP/IP and 16 for FDDI. All other values, except MCR, are the same as defined in Table 3.1.1. For comparison, Figure 3.1.11 shows the required microprocessor rates for a CISC microprocessor with and without the overhead

removed from the transfer rate. Note that for small packets, this makes a large difference, while for large packets, there is virtually no difference.

Note that for large data messages, the required clock rate for the CISC microprocessor is significantly less than for the RISC microprocessor using 32-bit transfers and a little less than for the RISC-based microprocessor using 64-bit transfers. BPP is still the variable along the x-axis. MCR has become the value on the y-axis.

From examination of the required microprocessor clock rate it is clear that FDDI requires significant processing power to meet the system needs. If we add the overhead associated with most operating systems, instead of dedicating a microprocessor to just network interface functions, we can see how a large computer could lose a large portion of its processing capability to meet the interface requirements of FDDI. In comparison, the same computer could use only a small portion of its processing capability for Ethernet due to Ethernet's slower data and message rates.

3.1.4 Throughput Analysis for a Microprocessor-Based System Using the External Hardware to Perform Data Movement

When external hardware is used to transfer data, the microprocessor must provide access control to the system. In TCP/IP, or any allowing variable packet lengths, the microprocessor must determine the number of bytes to be transferred, and the starting location of the bytes to be transferred. It must also guarantee that the original copy of the data being transferred will remain in place throughout the transfer process for possible reconstruction if an error occurs. For these conditions to be met, the microprocessor must complete the input portion of the protocol before the transfer hardware can take over the transfer, and at the end of the transfer process, the microprocessor must close the transfer and take care of the transfer acknowledgement and memory management functions. Without some elaborate transfer schemes or a change to a new protocol, transfers between memories needs to be sequential to the processing associated with the protocol overhead.

In developing timing and throughput requirements for the hardware segmentation, new equations must be developed. Because of the amount of interaction expected between the protocol and the data, the processing of protocol and data

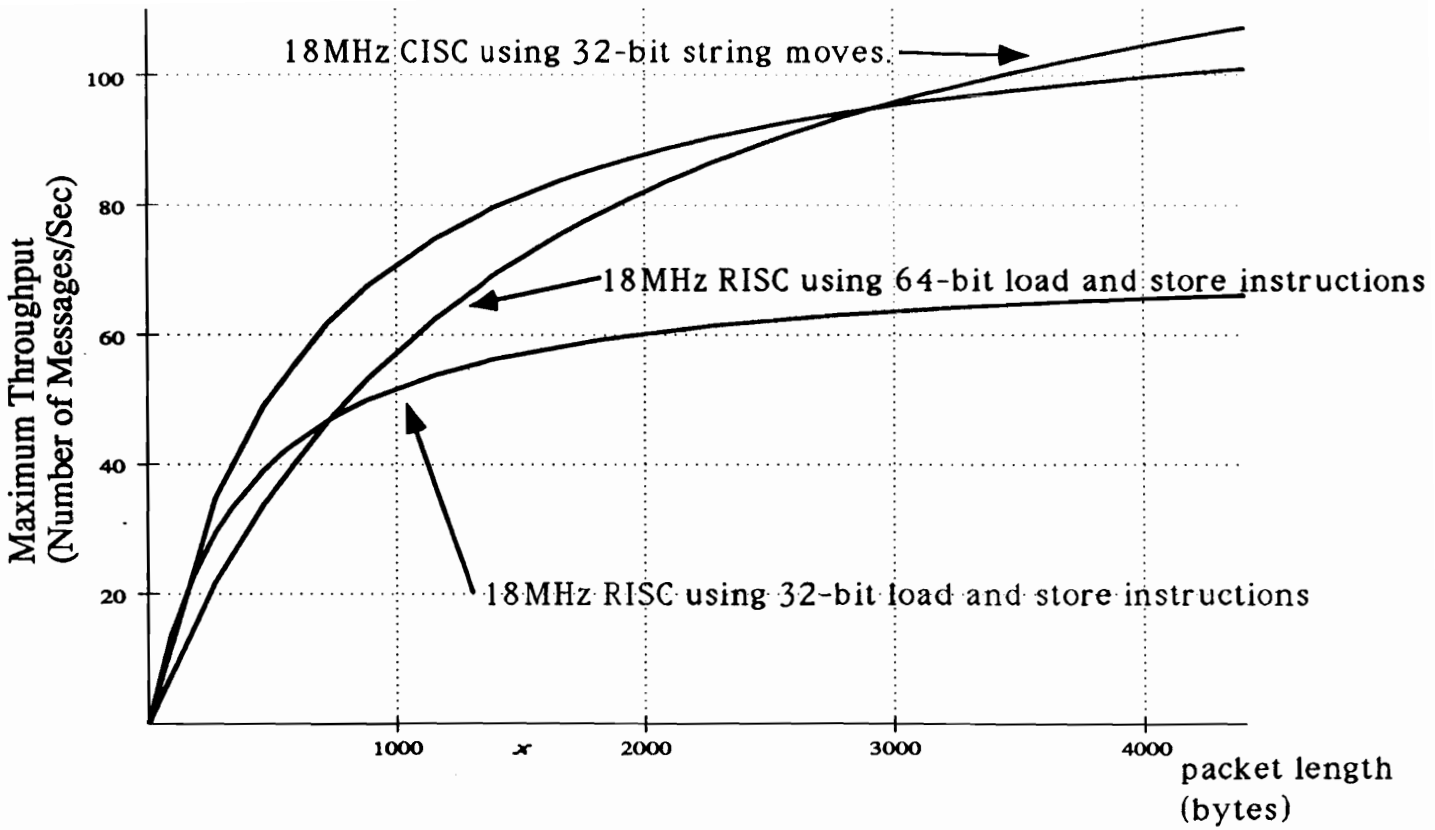


Figure 3.1.5: Number of bits processed per second as a function of the number of bytes of data per packet and microprocessor type using parameter values in Table 3.1.1.

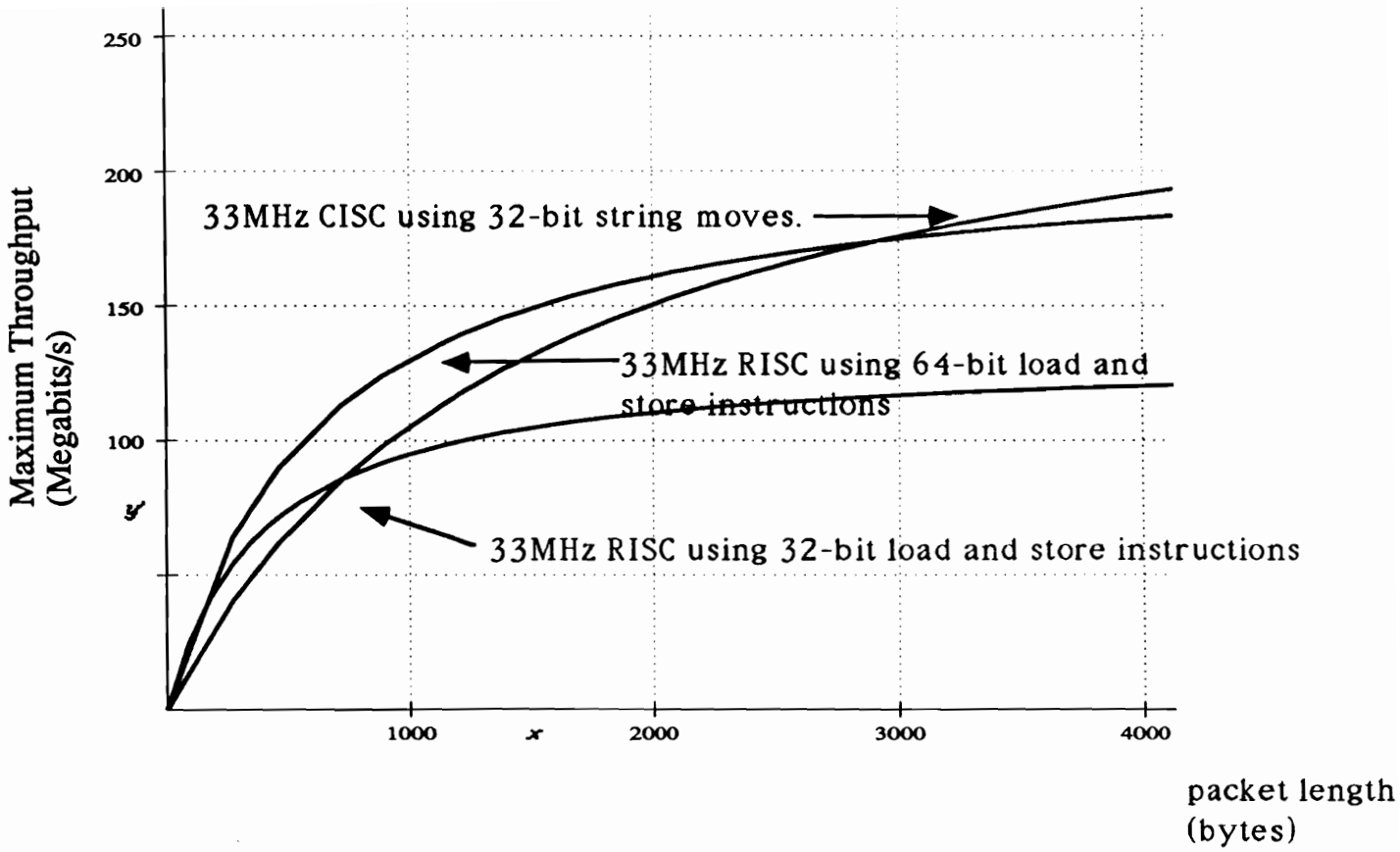


Figure 3.1.6: Number of bits processed per second as a function of the number of bytes of data per packet and microprocessor type using parameters in Table 3.1.1, except MCR = 33 MHz.

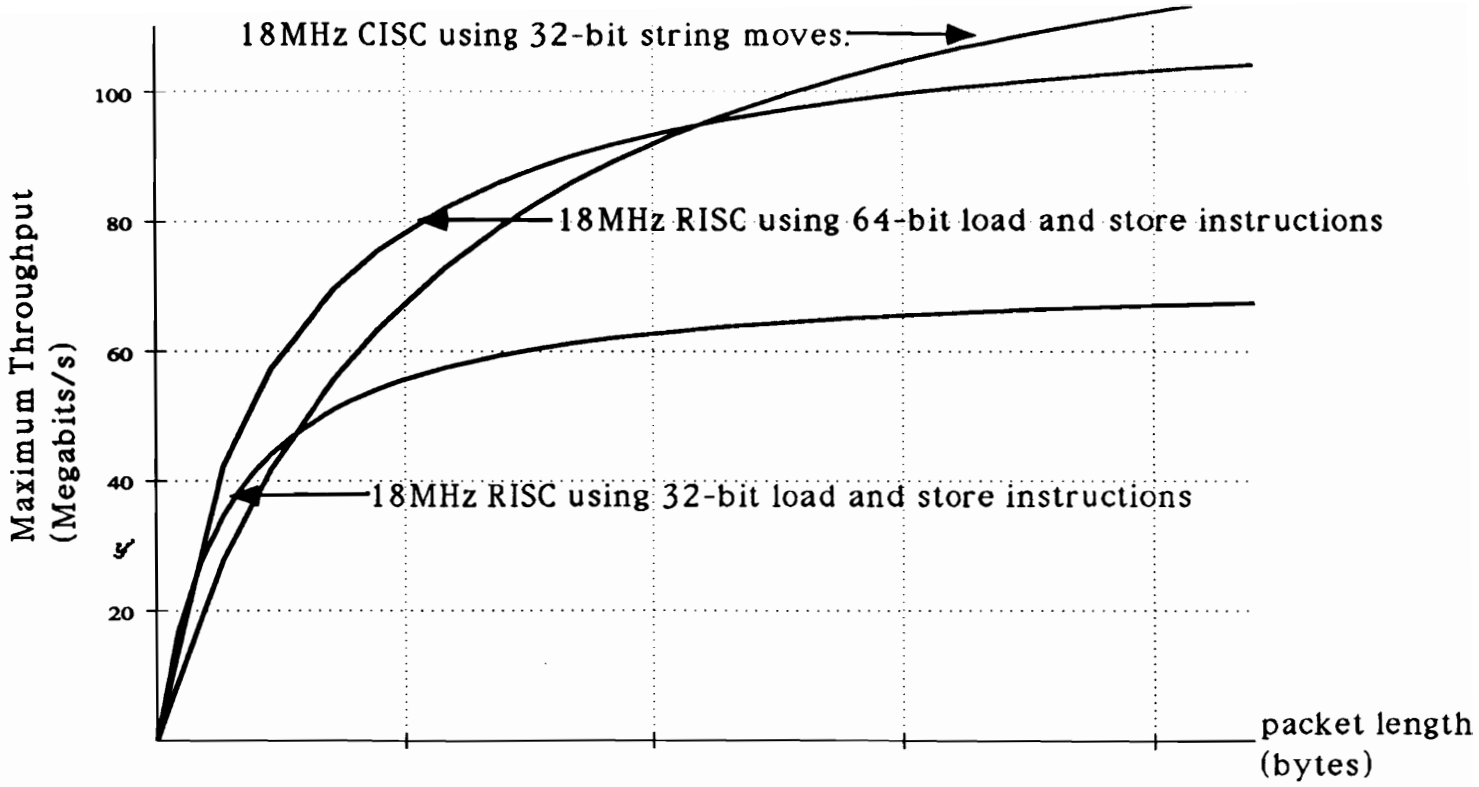


Figure 3.1.7: Number of bits processed per second as a function of the number of bytes of data per packet and microprocessor type using parameter values in Table 3.1.1, except ECI = 450 .

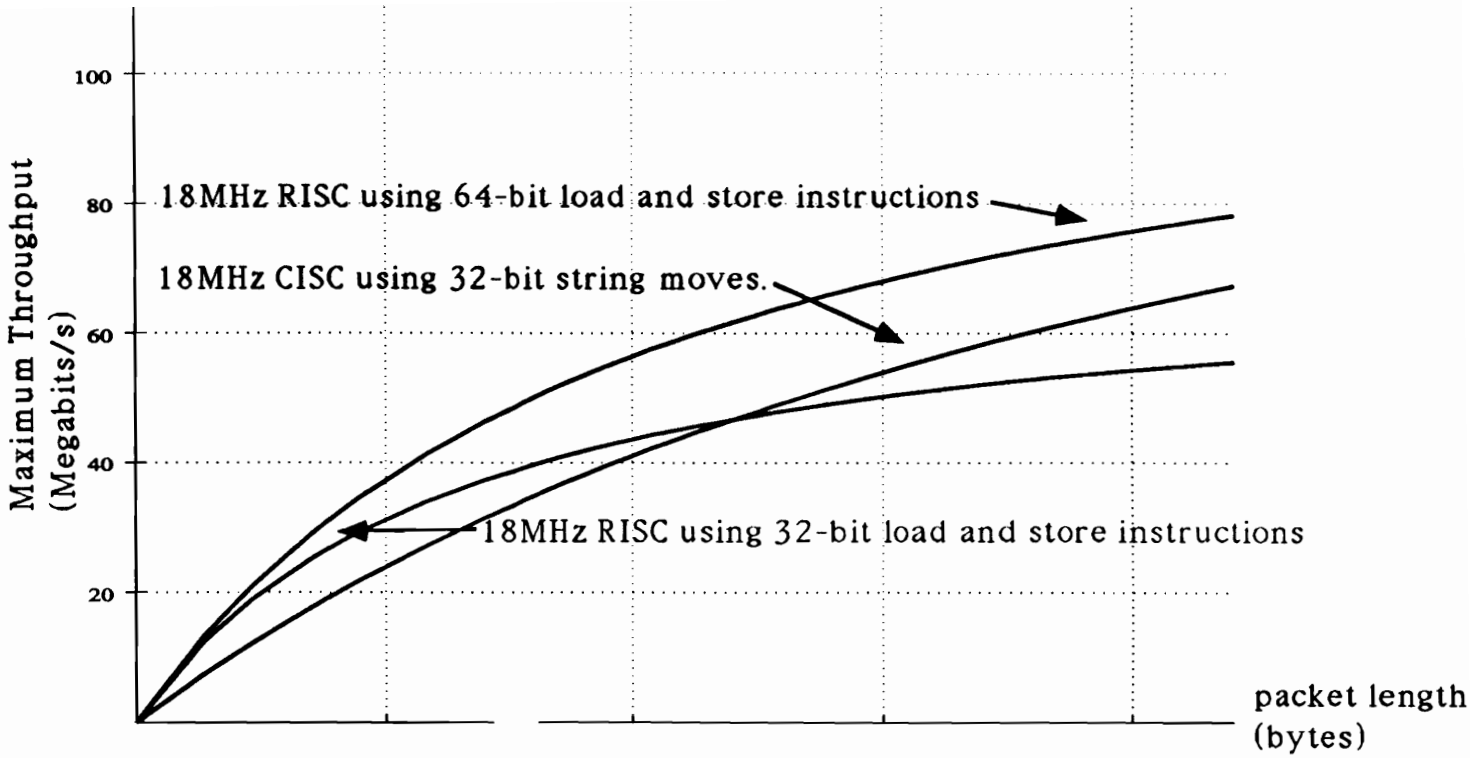


Figure 3.1.8: Number of bits processed per second as a function of the number of bytes of data per packet and microprocessor type using parameter values in Table 3.1.1, except ECI = 2000 .

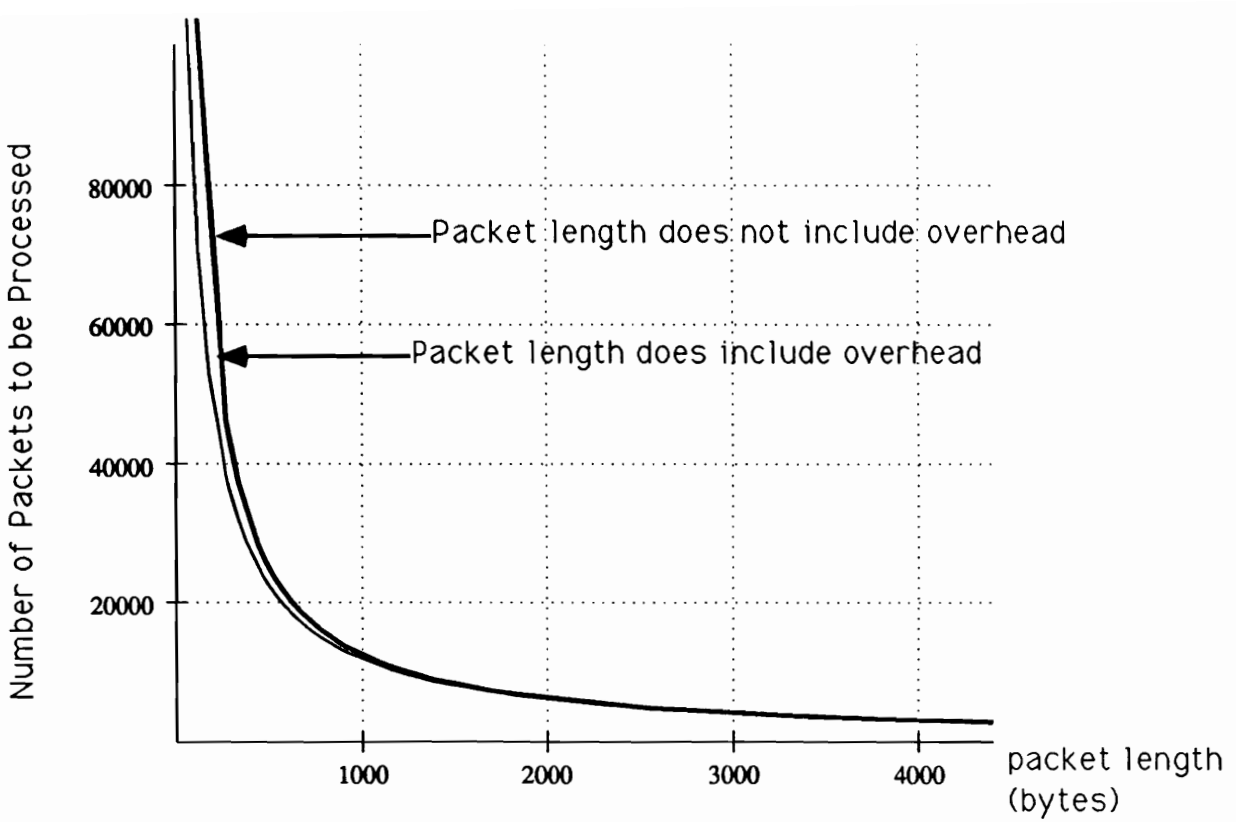


Figure 3.1.9: Graph shows the comparison of the number of packets required to be transmitted when overhead is not considered when computing the packet transfer size and when overhead is conisdered. Note the affect on the number of packets transmitted is significant when the packet length is small.

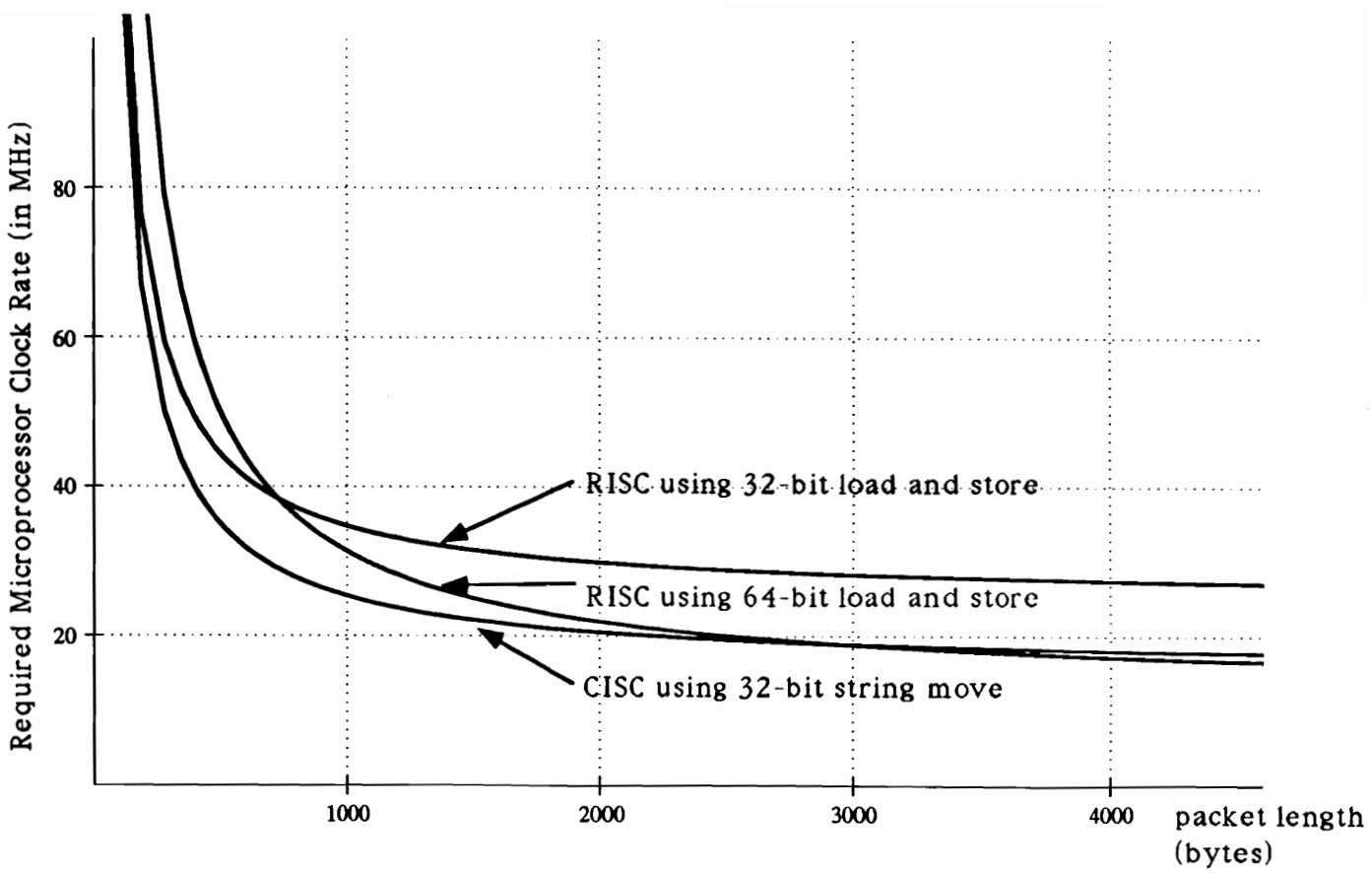


Figure 3.1.10: Required clock rate of various processors to achieve 100 Mb/s throughput. The packet length is the number of data bytes transferred per packet. Assumes 600 instructions for TCP/IP for each packet.

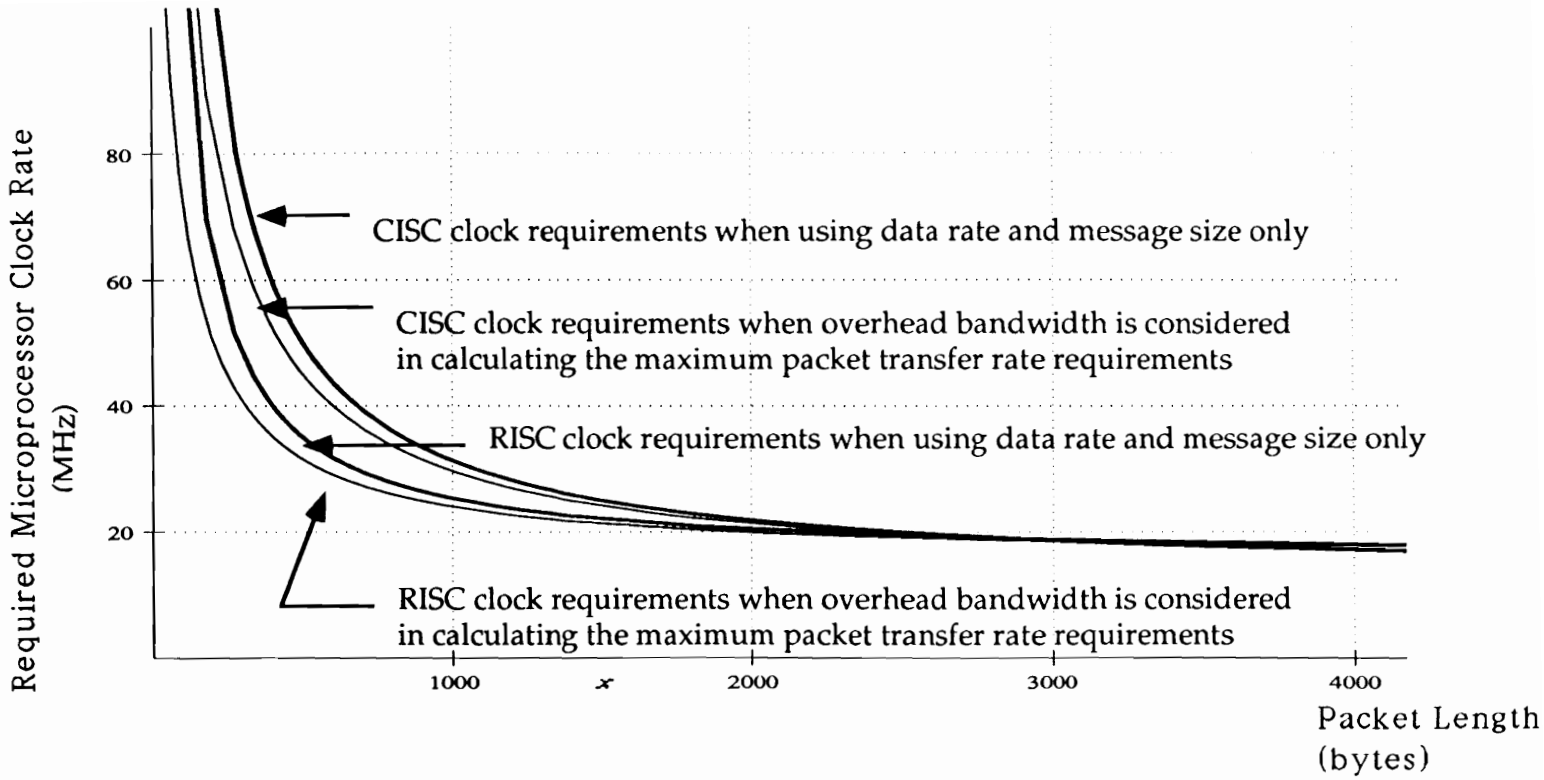


Figure 3.1.11: Comparison of the required microprocessor clock rate as a function of processor type when the overhead is subtracted from the data rate to be transferred versus the computation of processor clock rate when the overhead is not removed. Note the reduction in microprocessor clock rate for small packets.

movement will not take place simultaneously. Therefore, theequations for message and data throughput need to be redefined. The message throughput will be defined as:

Messages Transferred per Second = $\frac{1}{PPD + DTD + CTD}$ where,

PPD = Protocol Process Delay, the time required to process the protocol for a message

= $\frac{IMO * CPI * ECI}{MCR}$

DTD = Data Transfer Delay, the time required per complete data transfer for a message

= $\frac{BPP * CPX}{MCR * BPX}$,

where CPX and MCR are equal to the microprocessor clock rate because the memory transfer is synchronous to the microprocessor clock. See Section 3.1.1 for an explanation of the memory/microprocessor clock synchronization.

CTD = Control Transfer Delay, the amount of time required to load the transfer hardware registers and prepare for the transfers for each message

= $\frac{ICX * CPI * ECI}{MCR}$ where,

ICX = Instructions for control transfer. The instructions required to load hardware registers needed to implement the data transfer and to restart

the microprocessor at the end of the transfer. This is the only new term added to the transfer equation.

Therefore, the equation for the number of messages transferred per second can be reduced to:

$$MPS = \frac{MCR}{(IMO + ICX) * CPI * ECI + \frac{BPP * CPX}{BPX}}$$

The equations are plotted in Figure 3.1.12 for 32-bit transfers using the hardware transfer. The values used are those in Table 3.1.2, except ICX is a variable. If the messages are very large, on the order of 15 Kbytes, the number of messages processed would be almost identical regardless of the type of microprocessor used because the time to process the protocol for large messages becomes insignificant when compared to the time to move the data. In Figure 3.1.12, several values for ICX are used to show how the increased overhead used to control the transfer affects the throughput. In Figure 3.1.13, the transfer rates for CISC, CISC using hardware, RISC, and RISC using hardware move are plotted. The values used are those in Table 3.1.2. The difference in the number of packets transferred for the different architectures does not appear significant for the large packets, but when multiplied by the number of bits per message, the difference is significant.

The data throughput can be found by multiplying the amount of data per message by the messages per second processed. This gives the equation below.

$$BPS = \frac{8 * MCR * BPP}{(IMO + ICX) * CPI * ECI + \frac{BPP * CPX}{BPX}}$$

In Figure 3.1.14, several values for ICX are used to show how the increased overhead to control the transfer affects the maximum data rate for a common 32-bit transfer. For Figure 3.1.14, the values used for the variables are those of Table 3.1.2, except ICX varies. In Figure 3.1.15, the transfer rates for CISC, CISC using hardware movement, RISC, and RISC using a hardware movement are plotted. The values used are those in Table 3.1.2. The bytes per transfer, BPX, and the clocks per transfer, CPX, are common for RISC with the hardware transfer and both CISC architectures when 32-bit data transfers are used. These values indicate the use of synchronous transfer and utilize BPX = 4 and CPX = 4. Because of this, the difference due to protocol processing begins to show even in the large data transfer segments and the RISC architecture will begin to outperform CISC just as it does for processor-based movement of small messages. The data throughput amplifies the effect of protocol processing and control overhead on microprocessor selection.

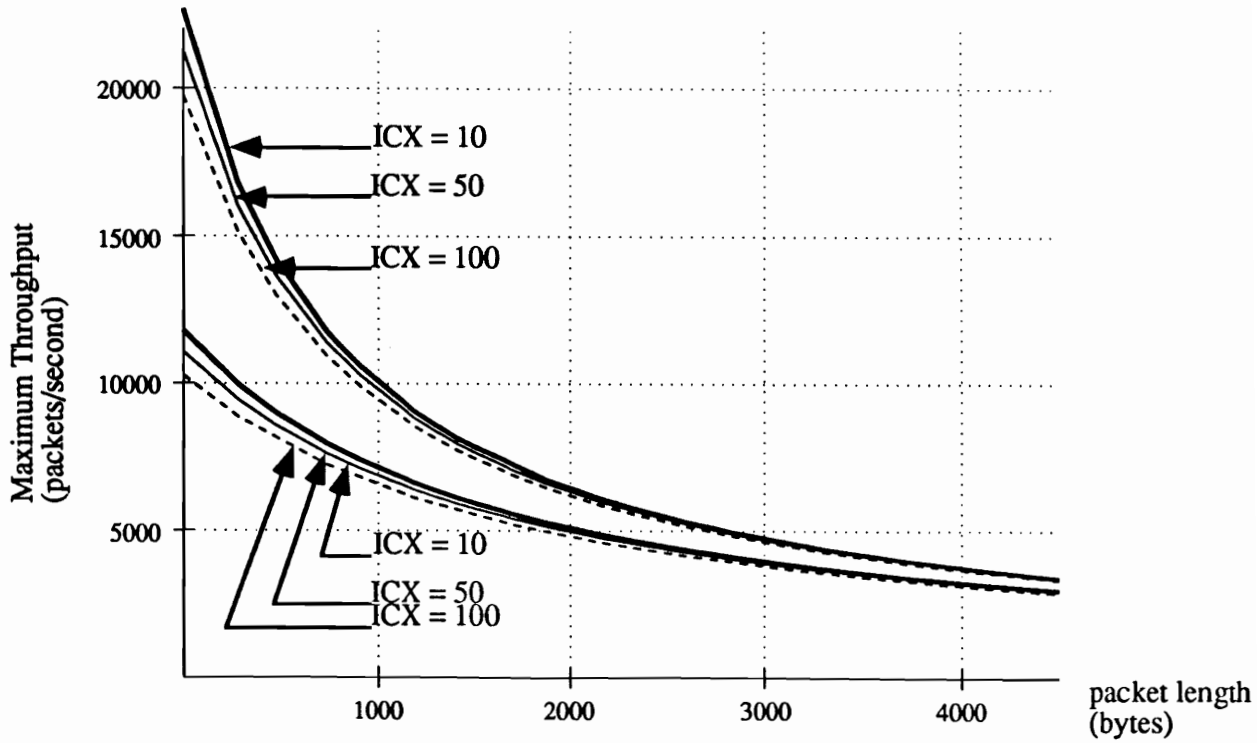


Figure 3.1.12: Number of packets transferred per second using the hardware data transfer with the microprocessor running TCP/IP protocol. See Table 3.1.2 for parameters. The chart shows the variability of throughput caused by different numbers of instructions to control external transfer (ICX) for each type of microprocessor. Note for hardware transfer, both CISC and RISC CPX values are equal.

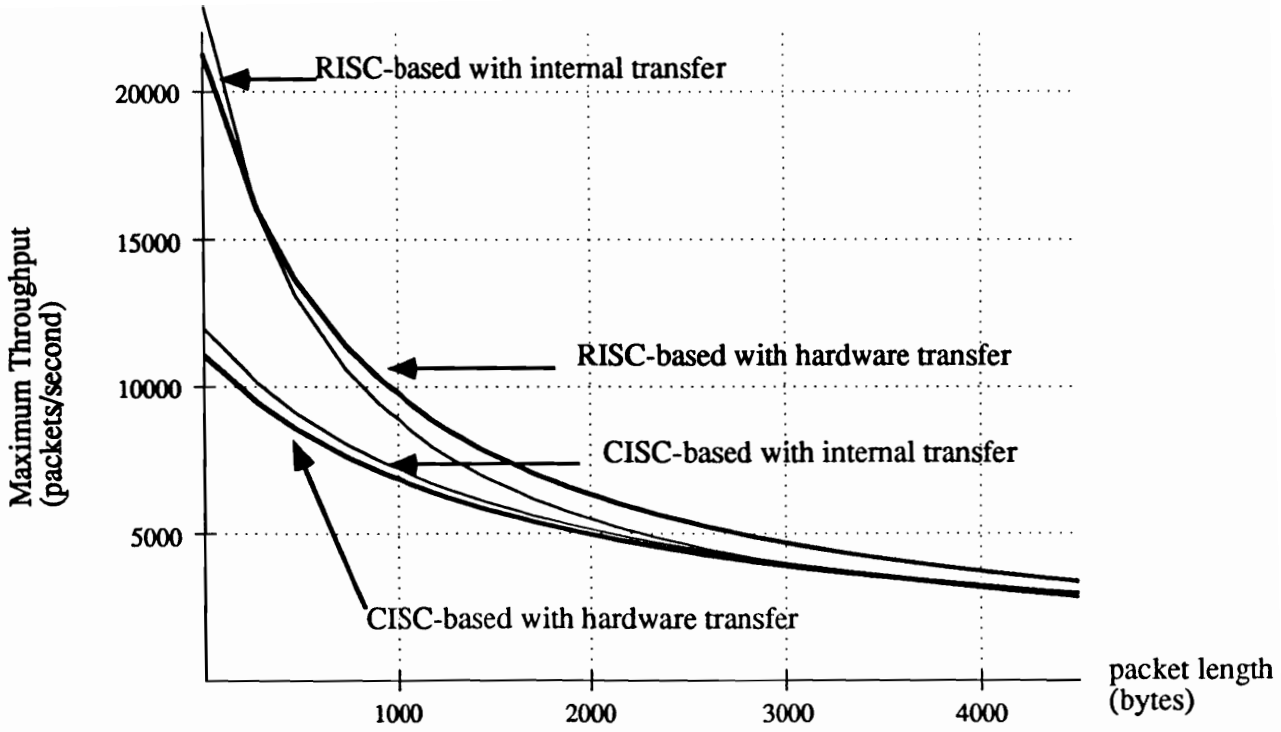


Figure 3.1.13: Number of packets transferred per second using different architectural approaches with common data transfer widths. The values used are those in Table 3.3.2.

Table 3.1.2: Values used to predict transfer rates for for CISC and RISC based microprocessors using external hardware data movement and internal microprocessor based data transfer.

Variable	CISC Hardware 32 bit Move	CISC Internal 32 bit Move	RISC Hardware 32 bit Move	RISC Internal 64 bit Move
MCR	18 MHz	18 MHz	18 MHz	18 MHz
ECI	1	1	1.3	1.3
CPI	2.5	2.5	1	1
IMO	600	607	600	600
CPX	4	4	4	10
BPX	4	4	4	8
ICX	50	0	50	0

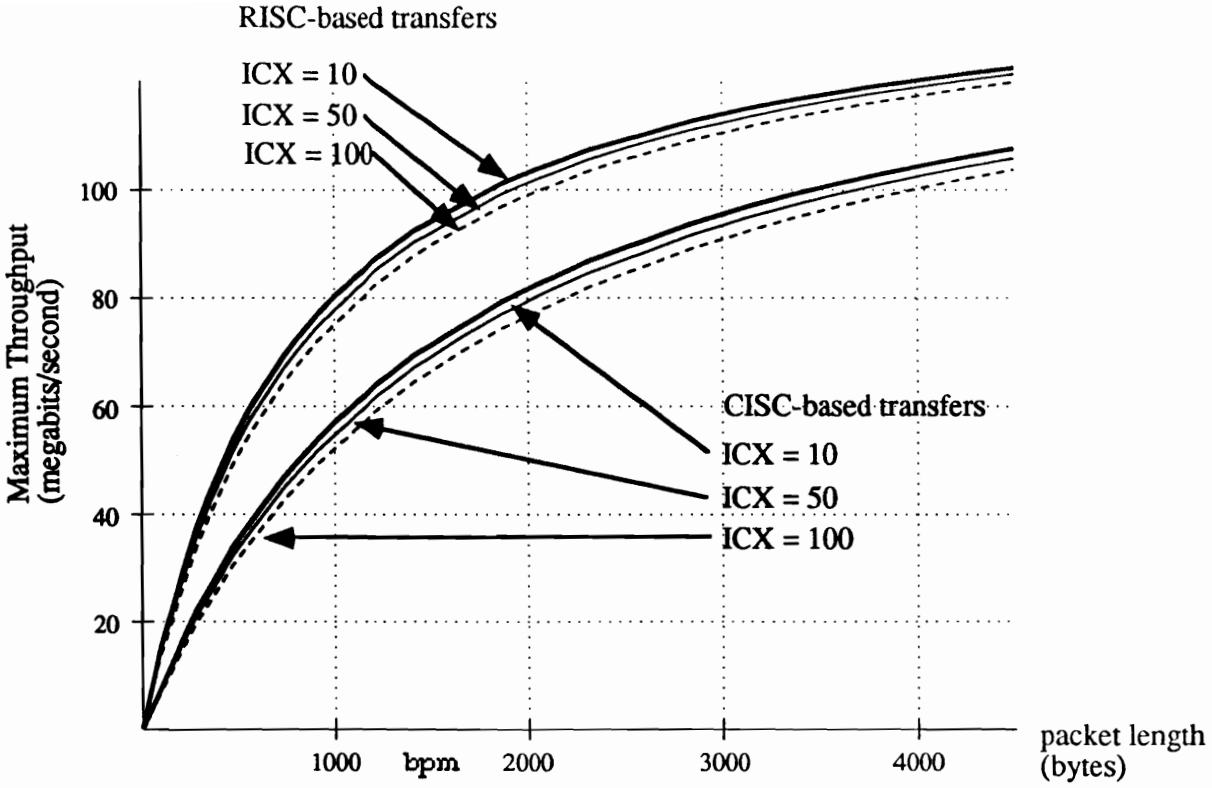


Figure 3.1.14: Number of megabits transferred per second using the hardware data transfer with the microprocessor running TCP/IP protocol. See Table 3.1.2 for parameters. The chart shows the variability of throughput caused by different numbers of instructions to control external transfer (ICX) for each type of microprocessor. Note for hardware transfer, both CISC and RISC CPX values are equal.

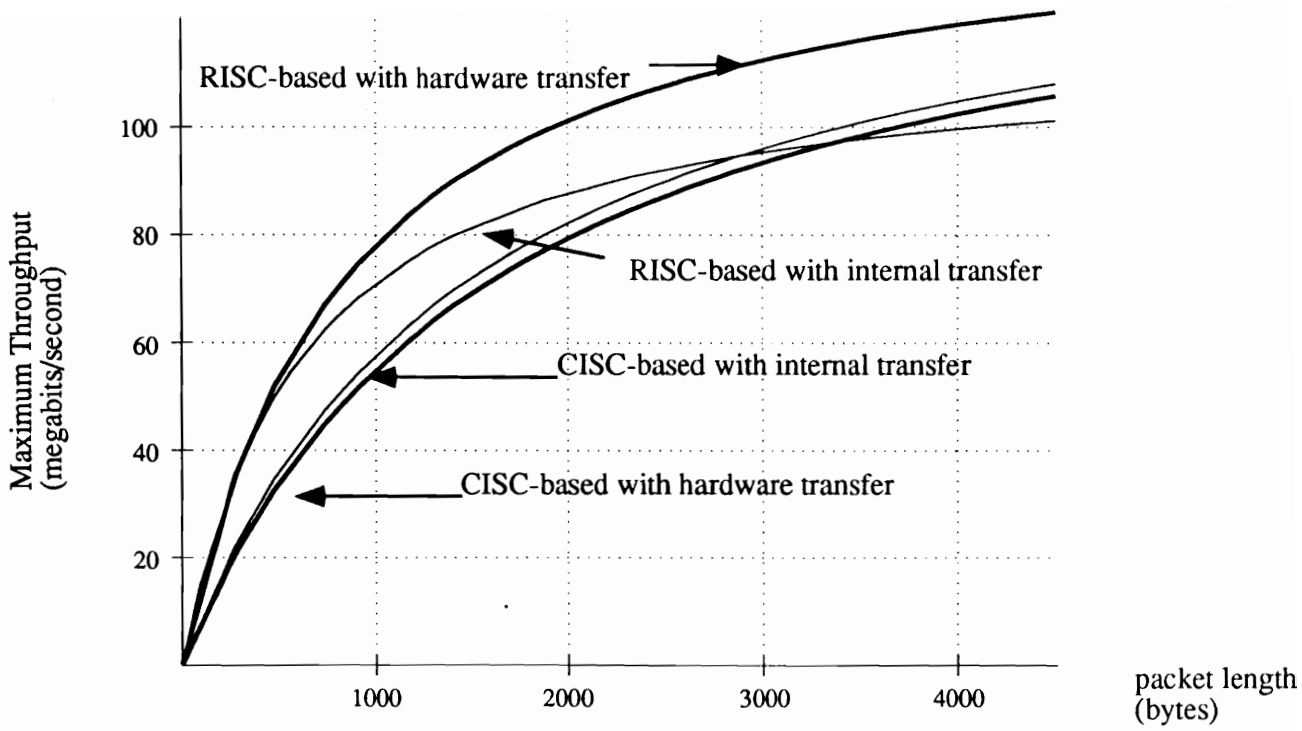


Figure 3.1.13: Number of megabits transferred per second using different architectural approaches with common data transfer widths. The values used are those in Table 3.3.2.

As can be seen in Figure 3.1.16, the bytes per transfer, BPX, variable can provide significant changes in the throughput. Figure 3.1.16 uses the values for the CISC and RISC microprocessors with hardware data transfer from Table 3.1.2 except that BPX is a variable. In Figure 3.1.16 the highest transfer rate occurs at each message length when data transfer is widest. Obviously, the increased data throughput due to the data width is only feasible for data widths which are reasonable. As the number of messages and transfer widths are maximized, the overhead plays a more important portion of the throughput and the RISC becomes even a stronger candidate. Unfortunately, large transfer widths for microprocessors which probably have a 32-bit word length are not usually reasonable or useful.

In an attempt to perform high speed transfers using TCP/IP, Cray Computer Company, which generally makes "sixty-four bit word addressable machines" [27] has found wide data transfers allow large data throughput. To maximize the performance, the transfer completes a transfer "a word [eight bytes] at a time" [27]. The maximum transmission unit used is 32 Kbytes. Cray has also rewritten much of its TCP/IP code, using as a baseline the work done by Jacobson [25] in the 4.3 BSD release. With the large data packets, wide transfers, reduced protocol and the speed associated with a Cray computer, Cray has achieved data transfer rates in excess

of 500 Mbit/s.

One of the major improvements made by Cray was to change the methods of generating the checksum. The method of improvement was to hand code in Cray Assembly Language (CAL) the section of the checksum routine that added the checksum a word at a time [27]. The way TCP/IP is defined, 16 bit transfers define boundaries for the checksum unit. Therefore all transfer units need to be multiples of 16 bits.

Because of the increased processing and transfer rates capable at the wider transfer systems, the equation for bits per second transferred makes a major simplifying assumption, which is that the data being moved always exists as a multiple of the transfer size. It is reasonable to assume that most high performance computers today that use 32-bit architectures are able to use 32 or 64-bit transfers, but it is difficult to justify the assumption for 128-bit and larger transfers.

For a given microprocessor rate, data rate, protocol overhead and cycles per transfer, the required data width can be calculated as a function of message length. The data bandwidth includes the overhead, and the overhead has been removed when calculating the required transfer rates. The equation for predicting required data width, BPX, assumes BPX is independent of CPX. The equation calculates the width of

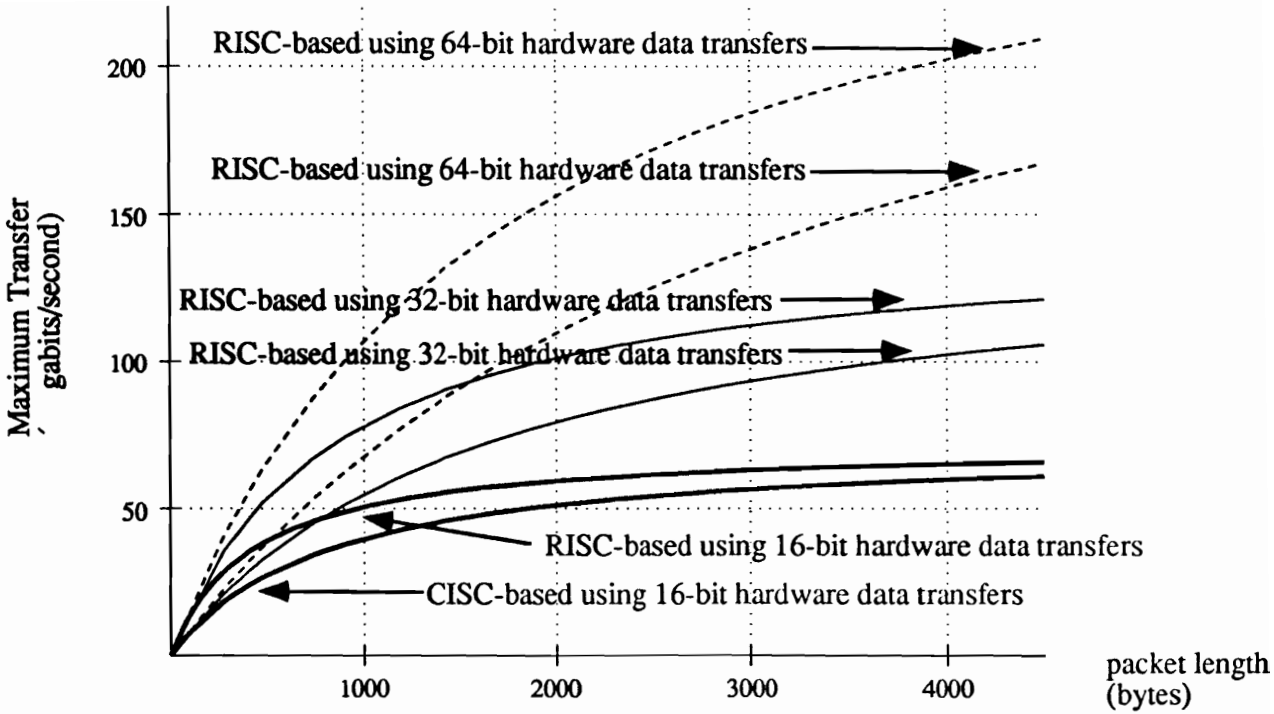


Figure 3.1.16: Number of megabits transferred per second using varying hardware data transfer widths in conjunction with the microprocessor running TCP/IP protocol. The chart shows the variability of throughput caused by different transfer widths (BPX) used for each type of machine.

transfer which allows data transfer and protocol processing requirements to take no more time than the reception time of the message from the network. Note that the values of BPX must be rounded to the next largest 16-bit increment to allow the TCP/IP check sum to operate properly. Figure 3.1.17 shows the output of this equation.

$$BPX = \frac{BR * BPP * CPX}{[8 * (BPP + OHB) * MCR] - (BR * (IMO + ICX) * CPI * ECI)}$$

For specific microprocessor speeds and data transfer widths, the minimum packet size for which the microprocessor can perform the protocol operations and transfer the data in a time less than or equal to a specific reception period is calculated in Table 3.1.2.

Since we are dealing in time, BPX is only valid for positive values. BPX/BPP must always be greater than 1, otherwise the equation estimates timing for transfers to be less than CPX. Because the minimum amount of data that can exist to be transferred in the equation must be at least equal to BPX, there are specific limits to the minimum number of bits that can be transferred by the link and still allow the microprocessor to complete processing the protocol and data transfer before the message stops arriving. This minimum message length is given by the equation,

$$BPP_{min} = \frac{BR * [CPX + (ICX + IMO) * CPI * ECI]}{8 * MCR} - OHB \quad .$$

Figure 3.1.18 shows the number of bytes per message required to make the message arrival take as long as the protocol processing and data transfer as a function of microprocessor type, architecture and clock rate. As the microprocessor clock rate increases, the number of bytes required to allow the microprocessor to process data as quickly as it is received is reduced. This is because as the clock rate increases, the protocol and other overhead processing can be done faster, while the data rate is fixed. As the clock rate increases, there is less data on the queue when the protocol processing is completed. Note that as both the clock rate and message length increase, the amount of time the microprocessor is idle increases for large data packets. This indicates that as long as the protocol overhead is significant compared to the data movement, a queue is needed to store data being received as processing progresses and the arrival sequence of data determines the amount of storage needed for the data.

Figure 3.1.19 shows the effect of data transfer width on the required data length for the same equation. As data width increases, the number of bytes required in the message to

allow the microprocessor and data movement to keep up with the message arrival decreases. In other words, if the data path is wide enough, the protocol delay becomes the main factor of concern. This points to the decision, that as long as the protocol processing time is significant compared to the data movement time, protocol processing is the limiting function. With a combination of increased microprocessor speed and wider data paths, reasonable throughput rates are achieved. However, systems using small data messages will not be able to keep pace with the data arrival rate, even with very wide data paths and extremely fast processors. Figure 3.1.19 shows the smallest data messages as a function of data transfer width for which a system can meet the arrival data rates of FDDI when the microprocessor clock rate is 18 MHz. Figure 3.1.20 shows the same message sizing information for a 33 MHz system. Note that by using the higher speed microprocessor, the delay due to protocol processing is reduced, thus reducing the advantages of larger transfer widths for the smaller messages. Because of the inability of any system to keep pace with the smallest data packets, buffering is needed to allow the system to function properly [21].

Figure 3.1.21 shows the percentage of the FDDI data rate each approach is capable of processing as a function of packet length. This chart uses the following equation to allow the true percentage of transfer rate to be calculated instead of

just the data rate of FDDI.

$$\% \text{ FDDI Throughput Met} = \frac{\text{Number of Bits/sec Transferred} * 100}{\frac{BR * BPP}{BPP + OHB}}$$

Note in Figure 3.1.21 and 3.1.22 the inability of any method regardless of the microprocessor clock rate to meet small data packet requirements and the abundance of processing power for large data packets. The data units used are those of Table 3.1.2.

The previous figures, show that fast processors (33 MHz) using internal transfers can compete with medium speed processors (18 MHz) using external hardware transfers, even with wide data paths. In Chapter 4, the equations developed in this section will be used to facilitate a design and to predict throughput, latency and buffer size. The throughput is, and will always be, a function of microprocessor type, transfer mechanism, clock rate, protocol processing, and additional computational overhead. This section has developed general equations for all of these items and has shown the ability of different microprocessor architectures to achieve the FDDI data rate throughput as a function of each of these variables.

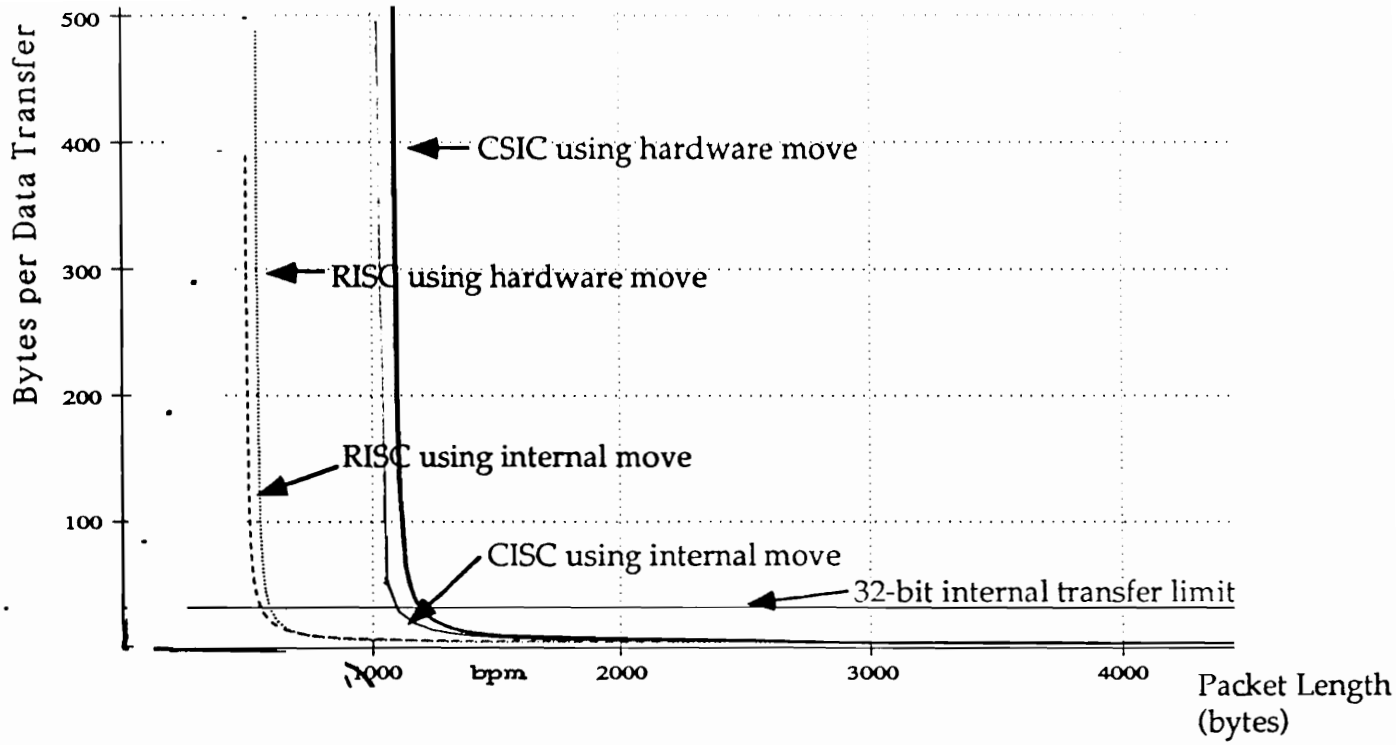


Figure 3.1.17: The transfer data width required to meet the throughput requirements of FDDI. Note that when the packet is very small, there is no way for the microprocessor to keep up with the data rate, no matter how wide the data path. Also note, in both cases, the internal transfer network processes smaller packets more quickly than the associated hardware transfers because the internal transfer does not require instructions to control external hardware (i.e. ICX = 0 for internal transfers).

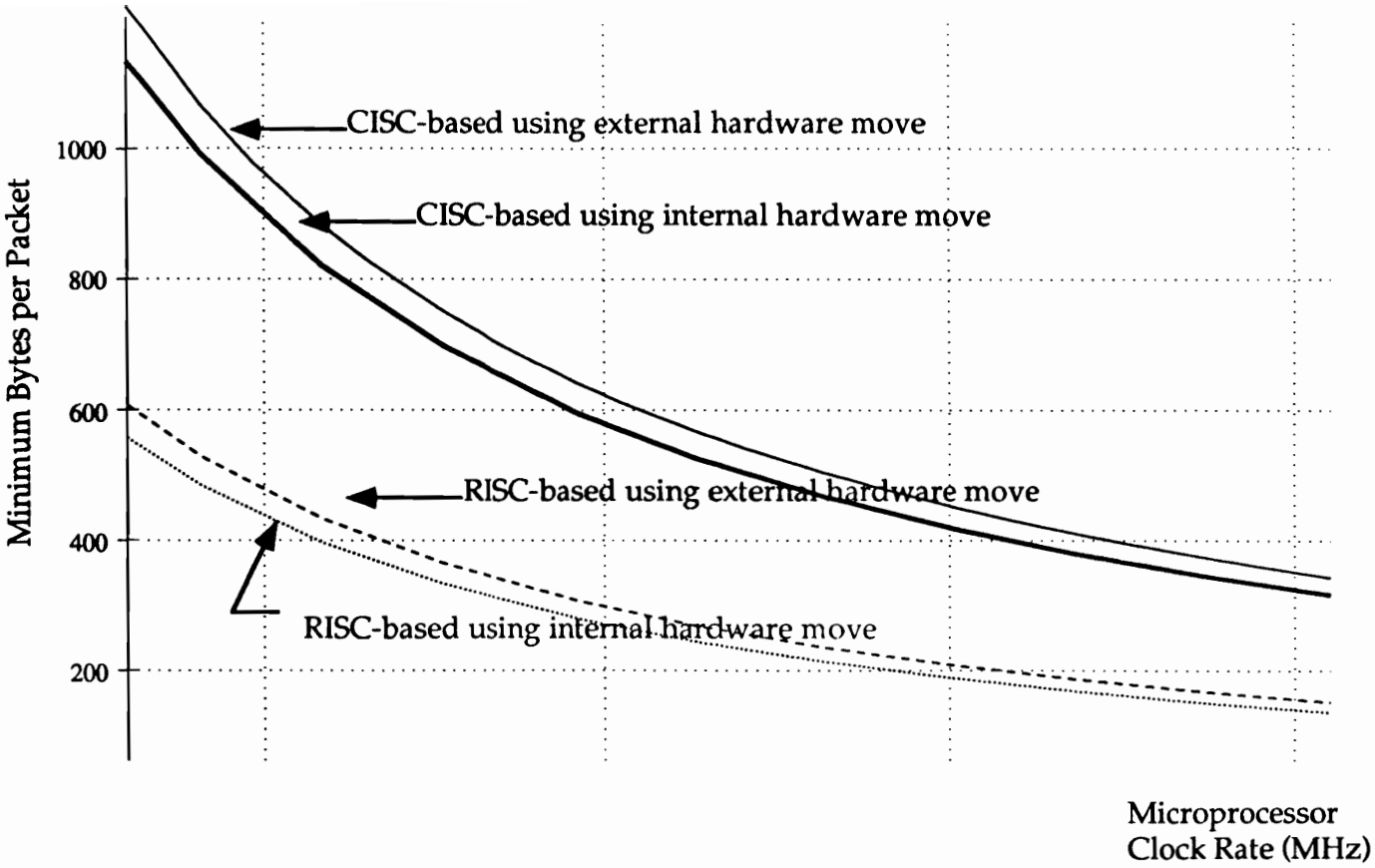


Figure 3.1.18: Minimum packet size to allow processing of a packet in the amount of time another packet of the same length is being received, as a function of clock rate. The graph is representative and all message sizes need to be rounded to 16-bit increments to allow TCP/IP to generate the correct checksum values. The values used for the generation of the graphs are those in Table 3.1.2 except MCR which is a variable and BPM/BPX =1.

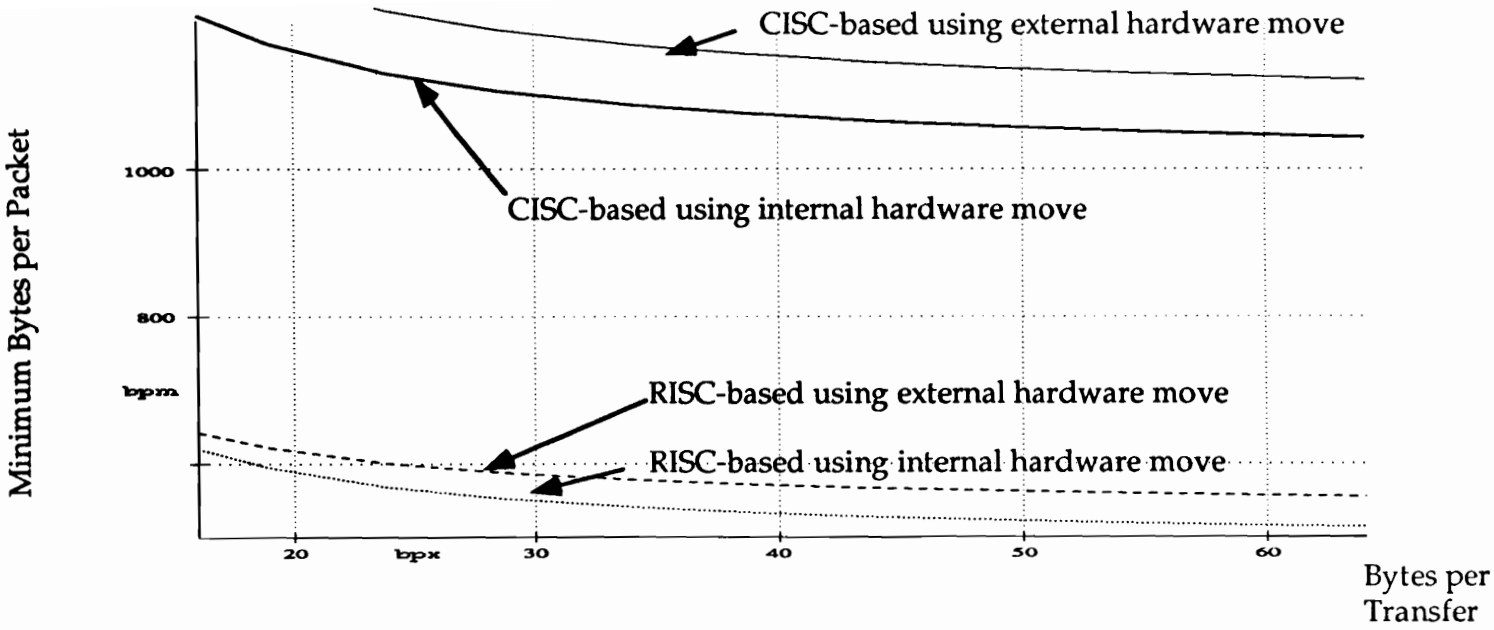


Figure 3.1 19: Minimum packet size to allow processing of a packet in the amount of time another packet of the same length is being received, as a function of the data transfer width, BPX. The graph is representative and all message sizes need to be rounded to 16-bit increments to allow TCP/IP to generate the correct checksum values. The values used for the generation of the graphs are those in Table 3.1.2 except BPX . Note the impact of the speed with which the RISC microprocessor can process the overhead.

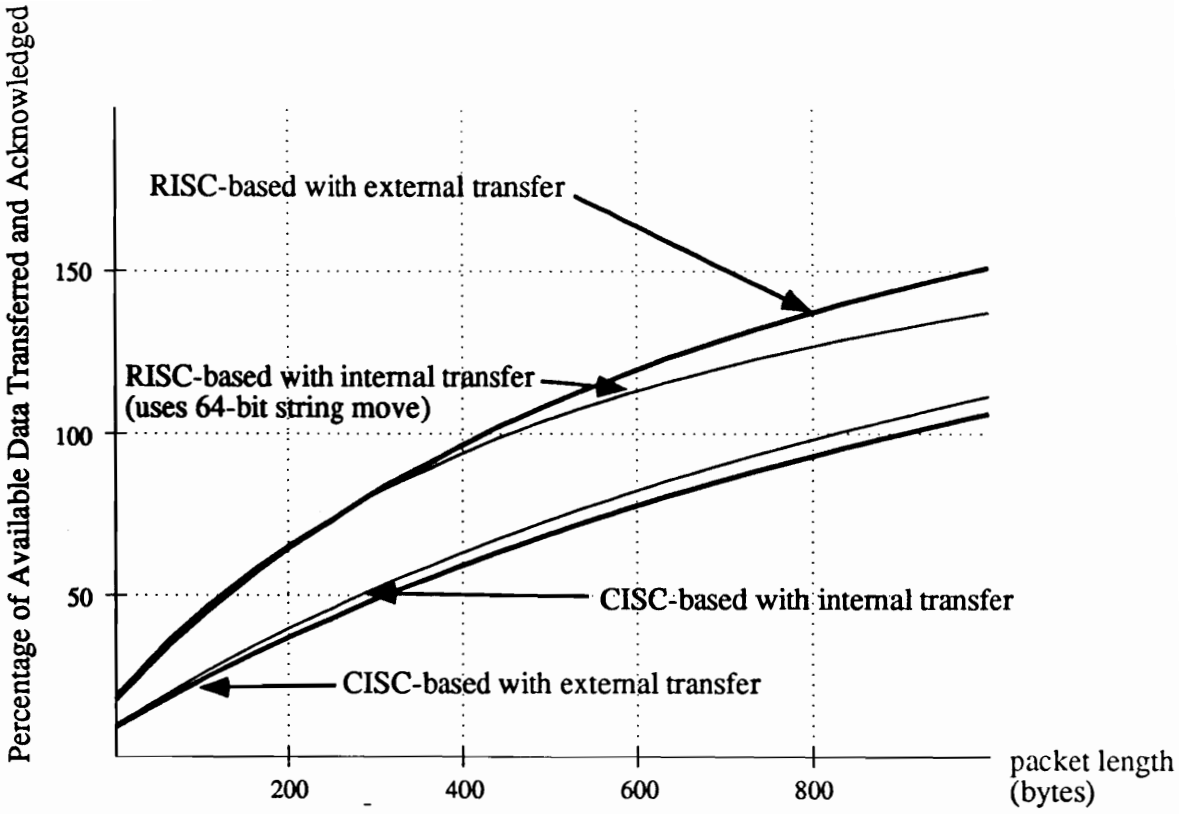


Figure 3.1.20: Percentage of data available over FDDI which can be transferred and acknowledged using different architectural approaches with common 32-bit wide data transfer. The values used are those in Table 3.1.2, except MCR = 33MHz.

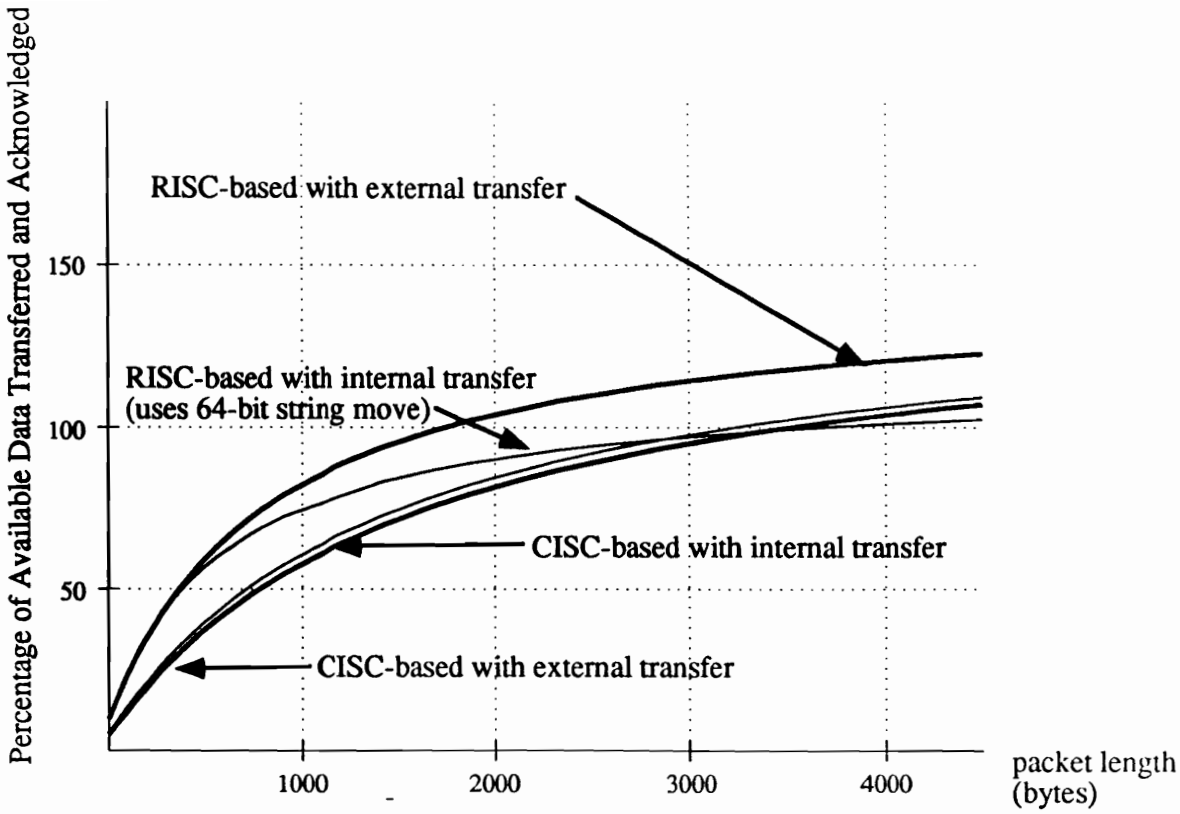


Figure 3.1.21: Percentage of data available over FDDI which can be transferred and acknowledged using different architectural approaches with common 32-bit wide data transfer. The values used are those in Table 3.1.2.

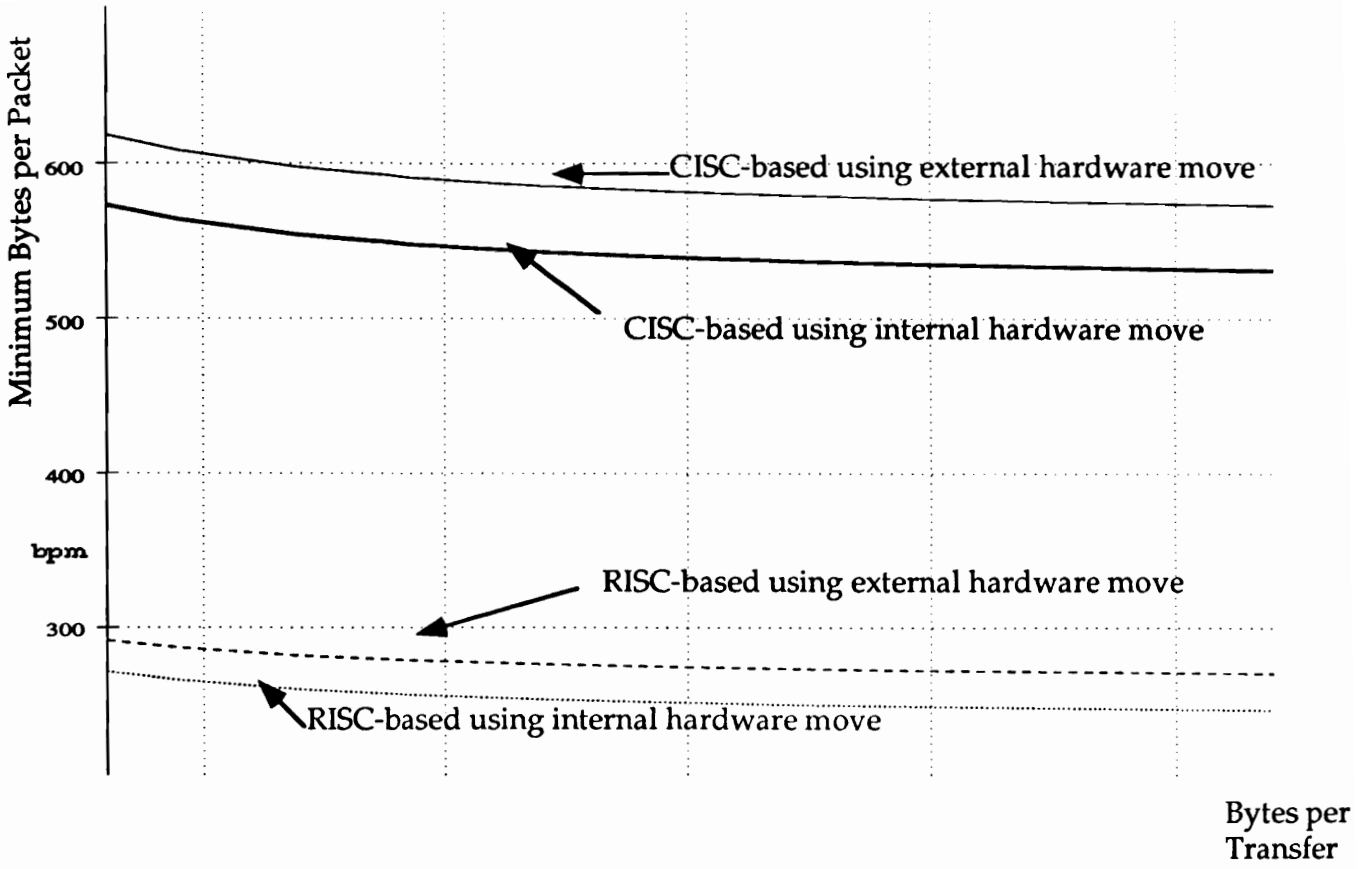


Figure 3.1.22: Minimum packet size to allow processing of a packet in the amount of time another packet of the same length is being received, as a function of the data transfer width, BPX. The graph is representative and all packet sizes need to be rounded to 16-bit increments to allow TCP/IP to generate the correct checksum values. The values used for the generation of the graphs are those in Table 3.1.2 except MCR=33 MHz and BPX is a variable.

3.1.5 Conclusions

It has been shown that the enhanced speed available in RISC over a wide range of instruction mixes does not always define RISC to be superior to CISC. For network interface applications where protocol and network management processing are small compared to data movement requirements, when the microprocessor performs the data movement, the string move capability of CISC microprocessors will outweigh the faster execution of other instructions of RISC microprocessors. In addition, this section shows CISC-based microprocessors will be able to meet the FDDI requirements provided packets are large and the microprocessor's operating system overhead is not extremely high. In high speed networks with small packets, neither RISC or CISC based systems can keep up with the data rates.

This points to a potential system bottleneck, and in fact indicates that network protocol is a significant factor in achieving high performance in high speed computer networks. From noticing the microprocessor rates discussed, and correlating these values to compute power, the necessity of offloading this intensive network management problem from the host computer to an intelligent interface is clear.

Another advantage of CISC microprocessors when data

movement is performed by the microprocessor is that for a given memory speed, the CISC microprocessor can use a faster clock rate than RISC. This is due to what is known as address pipelining, which was discussed earlier. The net effect is dramatized by the following argument. For a 16 MHz microprocessor with interleaved memory, the 386 can use 100 ns access time dynamic random access memories (DRAMs) with zero wait states when operating in pipeline addressing mode [12]. In comparison, the SPARC requires an 80 ns DRAM to accommodate a 16.4 MHz clock rate, and a 60 ns DRAM to accommodate an 18.9 MHz clock rate [28]. From extrapolation of the SPARC chip information, a 100 ns access DRAM would use a 13.9 MHz clock. Therefore, for a given DRAM with a known access time, the CISC architecture could support a 15 percent increase in the clock rate. This would make the throughput on CISC 15 percent higher than the curves shown when comparisons are made based on DRAM access time rather than equivalent clock cycles.

The method of transferring data external to the microprocessor has some significant throughput advantages, especially when used with the RISC microprocessor or when extremely wide data paths are used. When comparing a CISC-based system with special hardware for data movement using 32-bit data transfers to the CISC-based microprocessor using processor-based transfers, the processor-based data movement

architecture is faster because it performs the transfer with the microprocessor and does not have to coordinate operation of the microprocessor and the external hardware. In all cases, unless data transfers are wider than 32-bits, the normal CISC-based movement outperforms the CISC-based system with specialized hardware in throughput. When operating at the minimum data transfer sizes, the internally-based movement system will always provide better performance than the external hardware transfer system.

In comparing the RISC-based system using processor-based transfer with the RISC system using specialized hardware to perform the data movement, the RISC system with the specialized hardware has significant advantages in data throughput for almost all cases. The only case for which the RISC microprocessor with internal data transfer outperforms the external hardware movement is when the overhead for controlling the coordination of the transfer takes more time than the savings in the data transfer. This occurs at the very smallest data transfers and is a function of ICX.

With the hardware move, clocks/transfer value (CPX) associated with the RISC external hardware move is reduced to four clock cycles for any data width, while the best internal move for the RISC microprocessor using the 64-bit move takes five clock cycles per 32-bits of data. This equates to a 25

percent higher throughput with the specialized hardware. If we assume 10 instructions for the control, the specialized hardware begins to outperform the RISC internal movement alternative when more than six 32-bit transfers are required. Even if we need 100 instructions for external control, the RISC system with specialized hardware has better performance when more than 240 bytes are transferred. Therefore, if RISC will be used, it is extremely advantageous to have the specialized hardware.

When comparing message movement of the CISC-based system and the RISC-based system with specialized hardware, the RISC system has better throughput performance on all packet sizes of interest in FDDI. Note that if FDDI allowed larger packets thereby making the protocol processing insignificant when compared to the data movement, there would be no significant performance difference in the CISC approach versus the RISC with specialized hardware approach. Although the specialized RISC system has some significant advantages in throughput, the distribution of data lengths, message frequency and transfer control overhead need to be examined to determine if the use of high speed microprocessors can overcome the advantages of the specialized hardware. Using the equations generated using 32-bit wide transfers, it was determined that the throughput for large packets is equal in processing time for the CISC-based movement and RISC microprocessor with specialized

hardware if the movement control, ICX, takes 554 instructions.

From the analysis, it can be easily shown that if the data is transferred in widths significantly wider than 32-bits, a significant improvement in throughput can be seen by using specialized hardware for the data transfer. In most systems, this improvement is most significant in large data messages. However, since FDDI is defined to accommodate 8-bit data units, using data widths larger than 32-bits has some significant practical limits. Therefore, when trying to meet the 100 Mbit/s transfer rate, there is little advantage to using specialized hardware if the microprocessors are able to meet the system needs without the specialized hardware.

Throughout this analysis, it has been demonstrated that the selection microprocessor type and architecture have been significantly affected by data lengths, message frequency, protocol overhead and movement control overhead. Furthermore, the string move command is a major difference between CISC and RISC architectures and makes a major difference in the performance for this application. The best system to meet high speed network requirements will probably use microprocessors that combine the two architectures. There are some microprocessors beginning to appear on the market that have RISC instruction sets with additional capabilities

outside of RISC including string moves. These microprocessors should be given full consideration when a final system is being evaluated.

3.2 Effects of Data Segmentation on Processor Requirements

Storage of data in the high speed processing environment of an FDDI MAU is an important requirement in allowing the system to have a low BER. One basic concept required for new high-speed architectures is the "guarantee of no buffer overflow in the network" [21]. In Section 3.1 it was shown that over the range of sizes of arriving packets, the processing at the MAU can only keep up with the arrival of larger packets when packet sizes vary greatly as was discussed in Chapter 2. The packets that cannot be processed as rapidly as they are received must be bufferred in some fashion or they will be lost at the MAU and result in an error. The error in turn will require additional processing by TCP/IP and system throughput will decrease. To maintain a reasonable BER and throughput, the MAU must have sufficient buffering to support the packet arrival and processing distributions.

There are four required buffers in the MAU:

1) the incoming buffer between the microprocessor and the FDDI network, called the incoming MAU interface memory,

2) the outgoing buffer between the microprocessor and the FDDI network, called the outgoing MAU interface memory,

3) the outgoing buffer between the microprocessor and the host computer interface, called the outgoing host computer interface memory, and

4) the incoming buffer between the microprocessor and the host computer, called the incoming host computer interface memory.

The functional operation of each of these buffers is based on the types and lengths of arriving data packets, the processing requirements of the data and the amount of time the data may be stored. The amount of time the data may be stored in the outgoing MAU interface memory is a function of the time required for a TCP acknowledgement from the receiving device.

In the area of data segmentation and its effects on memory and microprocessor requirements, there are three questions.

1) Is it advantageous to send packets of a fixed number of bytes to limit the amount of processing required by the microprocessor and transfer devices?

2) How should memory be segmented to allow maximum memory utilization while minimizing the overhead of memory control?

3) How can microprocessor clock rates and memory sizes be specified based on packet length distribution estimates?

3.2.1 Fixed or Variable Data Transfer Sizes

This section focuses on the option of sending a fixed length packets of a single size, fixed length packets of several different sizes, or variable length packets. As discussed in Chapter 1, FDDI has the capability of sending from one to 4444 bytes of data per packet. By knowing that all packets are the same length, memory management would be minimized because all buffers are the exact same size and no data length options are needed. All the hardware and software control could be built for one size buffer. This option is not practical in FDDI or other computer networks because from

Chapter 2, it is known that the data packets have at least two distinctly different sizes. From Section 3.1 it was shown that the larger the packet, the easier the processing requirements are to meet. Using the largest packet would cause thousands of unused bits to be sent when small packets are transmitted. The use of the small or medium size packets would have some advantages in transfer efficiency but would dramatically increase the microprocessor requirements because of the average size of the data in comparison to the overhead. As discussed in Section 3.1, unless the overhead processing requirements are small compared to the data transport requirements, the microprocessor will not be able to process the small packets for the 100 Mbit/s data rates. The use of a single size transport packet reduces the design complexity associated with multiple length packets, but also reduces the system throughput.

The use of several different transport packet sizes reduces the number of unused bits by bounding the packet sizes to be only slightly larger than the data distribution points. The use of known length packets allows memory to be segmented in blocks and the assignment and deassignment of specific memory locations can serve as the memory management function. This type of fixed length memory reservation system takes less microprocessor overhead than dynamic allocation of completely random memory allocation and assignment [29]. By using the

absolute packet sizes, the advantage is that the RISC microprocessor using an internal move command will not need to perform counts and compares to determine if a packet is complete or not. The packet will be received, its size noted, and the control will be changed to a subroutine which transfers the exact number of bytes. This transfer would require a separate subroutine with hardcoded addresses and load and store commands for every fixed packet available. Instead of looping code, such as that in Figure 3.1.4, the code includes many steps as there are transfers taking place. For the RISC-based transfer using internal transfers, this would require two commands, LDA and STO. Because the location of each byte is known exactly in relation to the first byte stored, the microprocessor would hardcode the offset from the first location into the ASI field of the instructions [24]. This change drastically increases the number of instructions required in the control programs, but reduces by a factor of two the number of instructions executed for each double word transferred. Coding in this manner removes the ADD and BLE instructions of Figure 3.1.4. The number of clocks per transfer for the 32-bit increment move becomes 5 clocks per 4 byte transfer. This gives the 32-bit RISC-based internal move architecture the equivalent throughput of the 64-bit RISC-based internal move architecture discussed in Section 3.1. The 64-bit RISC-based internal move command string likewise improves the throughput performance by making an 8-

byte transfer in 7 clock cycles. This provides a data transfer which is approximately 14 percent faster than the RISC with external hardware move discussed in Section 3.1. While more bits are transferred per period of time, the average transfer rate for valid data may actually be lower than for the exact movement structure of the CISC-based internal transfer architecture and the RISC-based external transfer architecture discussed in Section 3.1. This change in the number of valid data bits transferred per clock cycle is because the data points under examination most likely have a normal distribution when many samples are taken. Assuming the packet distribution is bimodal, then given enough data points, the distribution tends to be normal due to the central limit theorem of probability [30].

When considering a normal distribution it is desirable to transfer a majority of the data as a block transfer. To transfer over 99 percent of all data associated with a distribution, it is necessary to transmit the mean data size plus 2.33 times the standard deviation. In Table 3.2.1 below, the average cycles per valid byte of data transferred is calculated for different means and variances.

The equation used to determine the average number of clock cycles per valid four byte data increment was evaluated by determining the number of bytes that would need to be

transferred to achieve the desired cumulative distribution multiplied by the clock cycles involved and divided by the average number of valid bytes transferred. The average number of valid bytes was the same as the mean of the bytes transferred because the truncated bytes made up an insignificant fraction of the sample. Note that the larger the percentage of packets bounded by the required packet size the more clock cycles the transfer mechanism requires per valid byte of data. In all cases examined having over 99 percent of the data distribution sent with the defined packet size, it is valid to say that when the standard deviation is nine percent or greater of the average package size, the transfer is slower than the four clocks per four byte transfer found for the internal move CISC architecture and the external move RISC architecture.

From Table 3.2.1 it is evident that many distributions of data sizes require more processing time per byte for useful data when fixed length packets are sent than when variable length packets are sent by the CISC-based system with internal transfer and the RISC-based system with external transfer. If it were determined that 64-bit increments were appropriate and the RISC internal hardware transfer system were examined, similar results would be found. The fixed length transfer system numbers calculated above are the best case since they do not lower the average number of bytes transferred to

Table 3.2.1: Calculations of the average number of clock cycles used per byte of valid data when transfer packets are of fixed length although the number of data bytes varies.

a) 99 percent of All Packets Delivered,

Average Number of Bytes	σ=5	σ=10	σ=20	σ=50	σ=100
	Average Number of Clocks Per 4 Byte Transfer[5]				
12	6.9				
50	4.3	5.1			
100	3.9	4.3	5.1		
500	3.6	3.7	3.8	4.3	5.1
1000	3.5	3.6	3.7	3.9	4.3
4444	3.5	3.5	3.5	3.6	3.7

b) 99.98 percent of All Packets Delivered,

Average Number of Bytes	σ=5	σ=10	σ=20	σ=50	σ=100
	Average Number of Clocks Per 4 Byte Transfer				
12	9.8				
50	4.7	5.9			
100	4.1	4.7	5.9		
500	3.6	3.7	4.0	4.7	5.9
1000	3.6	3.6	3.7	4.1	4.7
4444	3.5	3.5	3.5	3.6	3.8

[5] This value does not recompute the truncated normal mean since ıe number truncated represent less than one percent of the sample ›ace.

account for the truncation of the larger data sizes or decrease the average to account for smaller packets which were truncated as being too large to send with a smaller packet size. The idea of blocked memory allocations however does present an interesting concept in reducing software cost and introducing what is hoped to be a minimal hardware increase.

Figure 3.1.16 shows the change in throughput due to different data transfer widths. Table 1.3.1 also shows the relationship between the data transfer width and the number of transfer instructions required to maintain the data rates. Because FDDI has a data rate of 100 Mbit/s, the options of moving data internal to the computer in small data increments less than 32 bits wide is very costly in overhead and access time. The FDDI standard as well as the TCP protocol have provisions which support the use of 32-bit transfers. Among them are the following:

1) Although FDDI has the ability to pass data in octet increments, the data input algorithm defines the input entities to come in as 32-bit words [31],
2) TCP/IP requires the checksum to be generated using the "one's complement sum of all 16 bit words in the header and text" and to pad the last octet if needed [32], and
3) To promote use of 32-bit transfers, TCP/IP uses header padding "to ensure that the TCP header ends and data begins on a 32 bit boundary" [32, p.19].

Thirty-two-bit transfers use lower storage and access speeds than 8- and 16-bit transfers to achieve the same data rate. The throughput rates discussed in Section 3.1 show that wider transfer streams provide more throughput with a common memory speed or require lower memory speed for a given transfer throughput. To support the idea of requiring the FDDI front end processor for Star Technologies to use 32-bit data increments, realize that the additional bits needed to use 32-bit transfers would require less time than determining the number of 16-bit transfers that are needed at the end of the 32-bit transfers to get the correct number of bits transferred. Since TCP/IP requires 16-bit increments for the check-sum, the maximum number of bits needed to complete transfers on 32-bit bounds would be 16-bits. The transfer time of the 16 bits would be 16-bits/100 Mbit/s, or 160 ns. This is less than 3 clock cycles of a 16 MHz system or approximately 5 clock cycles of a 33 MHz system. Determining whether additional transfers of less than 32-bits in length are required will take longer than moving the additional bits. In addition, since the majority of microprocessors built today have word lengths of at least 32-bits, the use of 32-bit transfers as a standard is the most practical transfer size for FDDI. Using 32-bit transfers as the FDDI computer interface standard will allow transfers to be faster than sending smaller data increments and requiring the

microprocessor to determine the appropriate data transfer increments.

The 32-bit transfer requirement will increase the throughput of the system compared to 8- and 16-bit transfers. Requiring the memory to be segmented into general packet sizes, and using these preallocated memory blocks for data storage will reduce the amount of overhead required by the system [33]. The segmentation of memory is similar to the segmentation of packet sizes discussed in this section, except that the data transported is the exact data desired and the data in the memory block does not necessarily contain valid data at the end of the memory block. The memory allocation using blocked memory takes less time than the equivalent dynamic memory allocation and the memory requirement is easier to predict. The combination of 32-bit transfers with allocatable memory blocks requires small increases in the hardware requirements while decreasing the overhead of the system. The fastest and most efficient transfer possible for the FDDI interface system utilizes 32-bit data increments and reserved memory blocks when appropriate.

3.2.2 Types of Memory

In Section 3.1, it was found that when packets are small in size compared to the overhead for protocol processing, the

microprocessor simply can not process the packets as quickly as they arrive. These packets and any subsequent packets require some type of buffering so the packets can all be processed sequentially regardless of the other memory requirements placed upon the system by the network protocol. In addition, while the microprocessor is processing a packet, the memory may be receiving another packet which needs to be stored until the microprocessor is available to process the new packet. The types of data and the way they are received, processed and handled will define the requirements for memory management.

3.2.2.1 MAU Memory

In the FDDI front end processor, there are basically four types of memory. There are two basic storage locations in the MAU which interface to the network. These memory locations are shown in Figure 3.2.1 and exist at the interface between the FDDI chip set and the microprocessor. The way data arrives and is removed for these two memories is distinctly different and requires different bounds and limitations. The main purpose of the two MAU memories is to provide support for protocol processing and data storage for packets being transmitted to or received from the FDDI network. The memories are hereafter called the incoming MAU memory and the outgoing MAU memory. The incoming MAU memory needs to be able

to accept full length FDDI packets without microprocessor intervention. Because any arriving packet can be as large as the maximum size of an FDDI packet, the memory must be available to accept packets of 4500 bytes when a packet begins arriving. Because of the additional overhead that would be involved in moving packets internally, the packets must remain in the location where they first arrive until processed. Therefore, the options for segmenting the incoming MAU interface memory are dynamic allocation or allocation of maximum FDDI packet sizes for each packet. Although dynamic allocation requires more overhead, it can significantly reduce the amount of memory required for the incoming MAU interface memory. To demonstrate the decrease in memory requirements associated with dynamic allocation, consider the FDDI system which is receiving a string of 28 packets, each with 100 bytes. If dynamically allocated, they would require 2800 bytes of memory. In comparison, since their size upon arrival is unknown, the fixed length buffer allocation method would require 28 full FDDI storage locations or a total of 126 kbytes. Therefore, for the incoming memory, the use of dynamic memory allocation is advantageous even if some extra overhead is involved. With dynamic allocation the memory will appear as a FIFO, as long as packets are transmitting without interruption or error. Limits on reception size will not exist as long as the system is operating normally. When packets are not processed and released in a sequential manner,

the memory manager must fragment memory, thus preventing the memory from appearing as a FIFO. As the memory manager begins to delete sections of memory from the pool of available addresses due to fragmentation, the system may not be able to accept any new incoming packets. The inability to accept information which is incoming because no segments of memory are large enough to accept the largest potential packet is commonly referred to as memory contention[6] [34]. When memory is properly sized, memory contention problems exist only when transmission errors occur.

The outgoing MAU memory consists of packets of known length being prepared for transport onto the network. The microprocessor is required to generate the outgoing packet protocol information as well as acknowledge incoming packets. The outgoing MAU memory also retains outgoing packets in the manner required by the network protocol. The memory must be accessible by the FDDI interface, the microprocessor and any control unit used to pass data between the MAU memory and the host computer interface memory. The memory control including memory access and allocations will be the responsibility of the microprocessor. The MAU interface memory must be matched to meet the requirements of the protocols. In the case of TCP, all data must be placed in a retransmission queue to

[6] Similar to line or bus contention discussed in Data and mputer Communications.

await time-out or acknowledgement [32].

The packets remain in the queue until they are received at the destination and the acknowledgement is returned to the transmitting node and processed. The order in which the packets are deleted from the memory is a function of the receiving system's queue length, processing speed and the distance around the network. Since the data packets are not expected to be removed from memory in the order which they arrived, the dynamic allocation of memory has problems with memory contention, while the block memory allocation scheme has few problems.

The block memory allocation scheme uses several predefined pools of memory of specific sizes. For instance, the system may have blocks of 150 and 4500 byte blocks. This would provide adequate space for several data packet size distributions. When any packet shorter than 150 bytes needs to be stored, the memory manager examines the pool of 150 byte blocks and assigns the new packet to a starting address which is listed as unoccupied and changes the status of the packet to occupied. When deallocation is appropriate, the block status is changed to unoccupied. Although the system is not as frugal with memory as the dynamic memory allocation system, it requires less microprocessor intervention and can operate more quickly. By properly segmenting the data into the proper

distribution and size of blocks, the block allocated memory system can be made to be reasonably efficient. In contrast, if memory is allocated dynamically, the program needs to search for a block of memory most appropriately sized and then reserve a segment of that memory of the correct size. Dynamic allocation requires the system to track used and unused locations as well as beginning and ending locations. Dynamic memory allocation requires additional processing if data is not returned in the same sequence in which it was allocated. By using the block allocation method for the outgoing MAU interface, the memory management system minimizes the overhead for packets whose lengths are known or can be approximated when their allocation and deallocation are not sequential.

The amount of incoming MAU memory required to store arriving data is dependent on the network data rate, data length distribution, and processing rate. The required outgoing MAU memory size is mostly a function of the network time-out value and depends on the time-out to determine how many messages must be held in the queue. The required size of the outgoing host computer interface memory, discussed next, is a function of processing speed and outgoing speed when the memory acts as a buffer to some type of interface, such as with the Star Technologies requirements. In effect, the microprocessor appears to be preparing data to be sent over a private network where time-out values are not of

concern. Therefore, the outgoing host computer interface memory in the situation to be examined is like the incoming MAU memory.

3.2.2.2 Host Computer Interface Memory

The host computer interface memory has the responsibility of providing the network data to the host computer. The host computer interface makes the actual data and control packets transferred over FDDI available to the host computer, through either a direct memory access or an I/O access. In either case, the host computer interface memory is used to store all packets passing to the host through the network after the protocol information has been removed. The interface memory also provides storage for outgoing data while protocol processing is provided.

The type of host computer and the way the data is used significantly impacts the design of the host computer interface. If the host is a file storage machine, the data is read immediately and no longer used. The arrival time of the data may not be the same as the required order of storage. For this reason, even in a file storage machine, the data may not be sequentially processed. If the host is a mainframe or diskless workstation, the host modifies the data and return the data to the machine from which it was received. In this

case, the interface memory may become an integral part of the accessing machine. If the host cannot use the data directly, but is acting as a concentrator, the data is distributed over several connections to different hosts which may use the data differently. In this case, the transfer may be available over a backplane bus such as VMEbus. A final possibility is that the host machine is using the data in real-time and may perform multiple accesses on a group of data and then expect new data to be available. In any of these cases, the arrival of the data does not guarantee the sequence in which the data is processed or even the sequence in which the system releases the memory. For these reasons, both the incoming and outgoing host computer interface memory is best served by block allocation of the memory.

3.2.3 Bounding Buffer Sizes Based on Packet Length Distribution

This section examines the relationship of packet length distribution, packet arrival sequence, microprocessor throughput and buffer size. Examination of the relationship between probability theory on which packet length distribution is based and packet arrival sequence leads to a probability distribution equation which defines the worst case arrival distribution of packets for a backlog of microprocessor

processing requirements. Once this arrival sequence has been established, the microprocessor throughput requirements and buffer size can be determined. The result allows an understanding of the relation of the items of concern and makes inferences about the impact of other factors, such as overhead and block transfer rate, on the memory size requirements.

Information storage and the related memory sizing and segmentation may be one of the most complex problems associated with high speed communications systems design. The main questions are

1) What type of data must be stored?
2) How much data must be stored?
3) How long must it be stored?

These questions require the complete system to be examined. The transmission protocol plays a major role in the memory design and accounts for many of the variables which are discussed below. The method of storage and retrieval significantly impacts the feasibility and method of different memory access.

The data distribution was examined in Chapter 2. By estimating the average amount of data being received, we can

estimate the average requirements for the microprocessor and in turn the type of data storage required by the system. If we know that the distribution of packet lengths is defined by a sum of normal distributions, we can define the average data packet length as

$$\text{Mean Packet Size} = \mu = \sum_{i=1}^{n} p_i \mu_i \quad [30]$$

where,

p_i = probability of the specific normal distribution i being used,

μ_i = mean of normal distribution of i, and

n = number of distributions

Note that $\sum_{i=1}^{n} p_i = 1$

The variance is given by the equation,

$$\text{Variance of the Packet Size} = \sigma^2 = E\{(X-\mu)^2\}$$

Where, X is the random variable associated with the length of the packets. Provided X is of the discrete type, such as independent operations and the Bernoulli trials, then

$$\sigma^2 = \sum_{i=1}^{n} p_i(x_i - n)^2 \qquad \text{for} \quad p_i = P\{X = x_i\}$$

$$= \sum_{i=1}^{n} p_i \sigma^2_i$$

This simplification is possible because for independent samples, $E\{X_i\} = \mu$ [35].

By assuming that the data stream is made of a mixture of independent normal distributions, each having a known probability of occurring, p_i, we can determine an equivalent average load on the system. This average load has the mean and variance defined above and can be used in performing rough calculations about the processing power required by the system.

The equivalent normal load calculated above, is the average amount of data seen on the FDDI network. If we assume that the buffer on the system is large compared to the data, and that the amount of data in the queue is always greater than a single packet, then we can calculate the average microprocessor clock rate required to process the data by using the equation in Section 3.1.3.1.B with the variable BPP set equal to the mean packet length μ.

$$MCR = \frac{BR}{8 * (BPP + OHB)} * (IMO * CPI * ECI + \frac{BPP * CPX}{BPX}),$$

Figure 3.2.1 shows how the microprocessor clock rate changes as a function of the packet length distribution. The graph assumes two size packets, 6 bytes of data per packet and 4444 bytes of data per packet. The 6 byte distribution is from measurements of Ethernet network systems by Shoch and Hupp [15]. Note that as the percentage of 6-byte packets increases, so does the required microprocessor clock rate.

The microprocessor clock rate defined above assumes that the in-bound buffer is unlimited and, never drops any packets. However, in reality, any buffer is of limited size, and if the packets awaiting processing cause the buffer to overflow, the incoming packet will be rejected. As data arrives, the average number of small and large packets approaches a defined ratio. However, the arrival order of the data may in any given short period of time deviate significantly from the average. Memory and microprocessor requirements must be established such that the system will operate with an acceptable packet loss.

Although it would seem that the memory sizing problem would lend itself to a queueing theory approach, this is not the case. Some of the subtleties of the system, such as arrival times being based on the length of the previous packet and the queue departure being based on what was previously in the queue makes the standard queueing theory solution invalid because queueing theory uses batch arrivals and independent

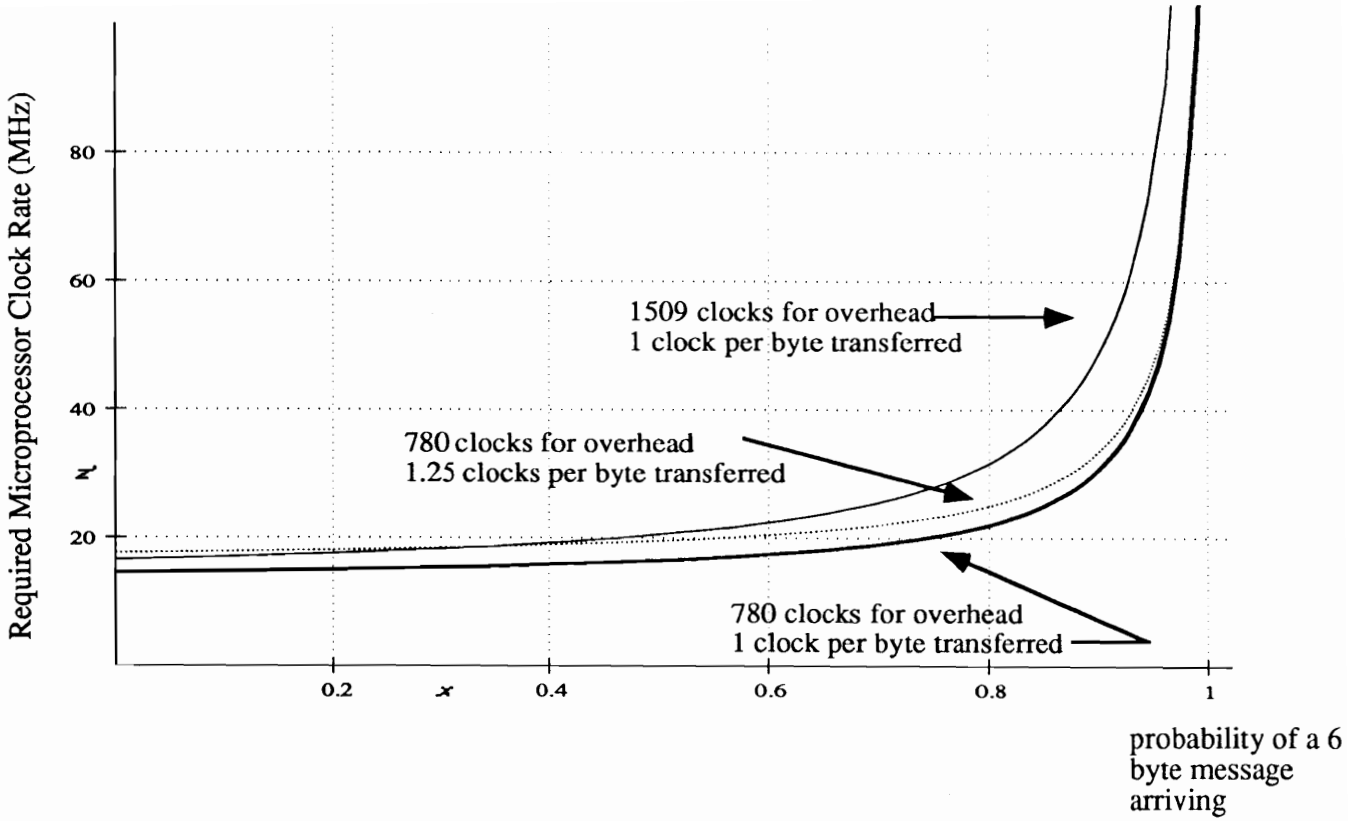


Figure 3.2.1: Required microprocessor rate to meet the average throughput requirements of the system using two message sizes, 6 and 4444 bytes of valid data. The probability of the 6 byte message occuring is along the x-axis. The required microprocessor rate is along the y-axis. This example assumes infinite buffer lengths.

service times [36].

For this reason, statistical approaches and approximations are examined and found to present a reasonable solution. For the solutions presented, several simplifying assumptions must be noted.

1) Since the bit arrival rate is constant, the same number of bits arrive during any time period regardless of the packet size. This assumes continuous arrivals.

2) For any given microprocessor clock rate, given the large packet length distributions discussed in Chapter 2 are valid, the microprocessors will process large packets faster than they arrive and small packets slower than they arrive.

3) The system does not begin processing data until a packet has been completely received.

4) As is often done for analysis of Ethernet performance, packet length distributions for FDDI will be estimated using two point mass packet sizes.

There will be two sizes of data packets, short messages, with length S1 bytes, and long message, with length S2 bytes.

The probabilities of each occurring is defined as P1 and P2. P1 and P2 are defined as the probability of these packet lengths arriving over a very long period of time, e.g. if n is the number of packets received and k is the number of small packets, then P1 = (k/n) as n goes to infinity. Since there is no expected correlation between the packet size packet arriving and the previous packet size or subsequent packet size, the arriving packet sizes are assumed to be independent. Therefore, the probability that a mixture of k small packets arriving out of n total packets arriving represents a Bernoulli Trial and can be given by

$$P(k) = \binom{n}{k} P1^k (P2)^{n-k} \text{ , where } P1 + P2 = 1.$$

To write this in the more standard form,

$$P(k) = \binom{n}{k} p^k q^{n-k} \text{ , where}$$

$$p = P1$$

$$q = 1 - p = 1 - P1 = P2.$$

In the equation above, $\binom{n}{k}$ is the combination of k objects from n trials without regard to order of which units of size S1 or S2 are in any particular position.

For the Bernoulli trials, if npq >>1, then

$$P(k) = \binom{n}{k} p^k q^{n-k} \approx \frac{1}{\sqrt{2\pi npq}} EXP[-(k-np)^2/2npq]$$

by the use of Stirling's formula [35]. As $n \to \infty$, the ratio of these two sides goes to 1.

To bound the memory sizing problem, the major questions are 1) if there is a point at which the microprocessor can process the defined distribution of packets, k/n, in the time equal to the packet arrival time and 2) if the minimum memory size can be defined to guarantee the storage of this data distribution. Given that these two questions can be defined and sufficiently solved, then reliable high speed computer communications can be achieved. By defining the ratio of k/n packages for which the microprocessor can process the packets in a time equal to their arrival time, the point at which memory expansion is no longer required is known. By limiting the memory size to hold the worst case distribution, i.e. the distribution causing the largest backlog of processing, for the k/n arrival distribution and a maximum number of samples n, the number of packets which can not be processed without packet loss is the sum of all arrivals of more than k small packets in n total packets.

Since the probability of more than k small packets arriving out of n total packets defines the point of possible failure of the system to process packets, the probability is called the blocking probability, P(k).

Since the microprocessor is sized to accommodate the k/n distribution of messages and the memory is sized to handle the buffering for the k/n ratio of n packets, then the microprocessor and memory can accommodate any arrival distribution larger than n with a blocking probability of at least the P(k) provided the additional points have a distribution less than or equal to k/n. This is the case for any given of P(k). This can be proven by taking two points in time, defined by the number of packets passed, and for equivalent values of P(k), proving that for any increase in the total number of packets, the amount of small packets increases by less than k/n. Since the increase is less than the k/n ratio the system is able to meet or exceed the original blocking probability.

The proof will be given by assuming the simplification of the Bernoulli trials with npq >>1. The proof of the above statement can be found by proving the following theorem

Theorem:

If for any normal distribution having probability P(k), there are values k,n and p defined such that k is the number of times S1 occurred in n total trials and for any given p, the probability of the S1 item occurring over the long term operation, then, for all large values of n1, where n1 = n + n_{new}, and k1 such that k1 = k + k_{new} there exists a positive

value of a such that k1 = k + n_{new}*k/n - a, causing k/n > k_{new}/n_{new}.

Proof:

For a given P(k), with p and q defined such that q = 1 - p,

$$P(k) = \frac{1}{\sqrt{2\pi npq}} EXP[-(k-np)^2/2npq]$$

and

$$P(k1) = \frac{1}{\sqrt{2\pi n1pq}} EXP[-(k1-n1p)^2/2n1pq]$$

Assume,
n1 = n(1 + x)
and
k1 = k(1 + x) - a

When P(k) = P(k1), then the increased percentage of k1/n1 is no greater than the original ratio of k/n. In fact, the increased portion of k1/n1 is (xk-a)/xn. Since the new portion of k1/n1 is less than k/n, the process will get no more packets in it's queue than were possible from the worst case of k/n provided the microprocessor can maintain the rate of k/n.

By setting P(k) = P(k1), then

$$\frac{1}{\sqrt{2\pi npq}} EXP[-(k-np)^2/2npq] = \frac{1}{\sqrt{2\pi n1pq}} EXP[-(k1-n1p)^2/2n1pq]$$

Multiplying both sides by $\sqrt{2\pi pq}$, gives

$n^{-1/2}\ EXP[-(k-np)^2/2npq] = n1^{-1/2}\ EXP[-(k1-n1p)^2/2n1pq]$

by taking the natural log of both sides, the equation further simplifies to

$-\frac{1}{2}ln(n) + [-(k-np)^2/2npq] = -\frac{1}{2}ln(n1) + [-(k1-n1p)^2/2n1pq]$

multiplying both sides by -1, gives

$\frac{1}{2}ln(n) + (k-np)^2/2npq = \frac{1}{2}ln(n1) + (k1-n1p)^2/2n1pq.$

multiplying both sides by 2pq, gives

$pqln(n) + (k-np)^2/n = pqln(n1) + (k1-n1p)^2/n1.$

multiplying both sides by n*n1, gives

$nn1pqln(n) + n1(k-np)^2 = nn1pqln(n1) + n(k1-n1p)^2.$

Now, let $n1 = n(1 + x)$, giving

$$nn(1+x)pqln(n) + n(1+x)(k-np)^2 = nn(1+x)pqln(n(1+x)) + n(k1-n(1+x)p)^2.$$

expanding,

$$nn(1+x)pqln(n) + n(1+x)(k^2 - 2knp + n^2p^2)$$

$$= nn(1+x)pqln(n(1+x)) + n(k1^2 - 2k1np - 2k1xnp + 2xn^2p^2 + n^2p^2 + x^2n^2p^2).$$

further expanding gives,

$$n^2pqln(n) + xn^2pqln(n) + nk^2 - 2kn^2p + n^3p^2 + k^2xn - 2kxn^2p + xn_3p^2$$

$$= n^2pqln(n(1+x)) + xn^2pqln(n(1+x)) + nk1^2 - 2k1n^2p - 2k1xn^2p + 2xn^3p^2 + n^3p^2 + x^2n^3p^2$$

subtracting n^3p^2 and xn^3p^2 from both sides and expanding

$k1 = k(1 + x) - a$, gives

$$n^2pqln(n) + xn^2pqln(n) + nk^2 - 2kn^2p + n^3p^2 + k^2xn - 2kxn^2p + xn_3p^2$$

$$= n^2pqln(n(1+x)) + xn^2pqln(n(1+x)) + n(k^2 + k^2x^2 + 2k^2x - 2ka - 2kxa + a^2) - 2(k + kx - a)n^2p - 2(k + kx - a)xn^2p + 2xn^3p^2 + n^3p^2 + x^2n^3p^2$$

$$= n^2pqln(n(1+x)) + xn^2pqln(n(1+x)) + nk^2 + x^2nk^2 + 2xnk^2 - 2nka - 2xnka + na^2 - 2n^2pk - 2xn^2pk + n^2pa - 2xn^2pk - 2x^2n^2pk + xn^2pa + 2xn^3p^2 + n^3p^2 + x^2n^3p^2$$

Subtracting $nk^2 - 2kn^2p + n^3p^2 + k^2xn - 2kxn^2p + xn_3p^2$ from both sides leaves,

$$n^2pqln(n) + xn^2pqln(n)$$

$$= n^2pqln(n(1+x)) + xn^2pqln(n(1+x)) + x^2nk^2 + xnk^2 - 2nka - 2xnka + na^2 - 2xn^2pk + n^2pa - 2x^2n^2pk + xn^2pa + xn^3p^2 + x^2n^3p^2$$

dividing by n and regrouping elements

$$npqln(n(1+x)) + xnpqln(n(1+x)) - npqln(n) - xnpqln(n) + x^2k^2 + xk^2 - 2x^2npk - 2xnpk + xn^2p^2 + x^2n^2p^2 = 2ka + 2xka - a^2 - npa - xnpa$$

Since $ln(n(1+x)) = ln(n) + ln(1+x)$,

$$npqln(n(1+x)) + xnpqln(n(1+x)) - npqln(n) - xnpqln(n)$$

$$= npqln(n) + xnpqln(n) + npqln(1+x) + xnpqln(1+x) - npqln(n) - xnpqln(n)$$

$$= npqln(1+x) + xnpqln(1+x),$$

which gives,

$$npqln(1+x) + xnpqln(1+x) + x^2k^2 + xk^2 - 2x^2npk - 2xnpk + xn^2p^2 + x^2n^2p^2 = 2ka + 2xka - a^2 - npa - xnpa.$$

Dividing by (1+x), leaves

$$npq\ln(1+x) + xk^2 - 2xnpk + xn^2p^2 = 2ka - a^2/(1+x) - npa.$$

Because the problem is with the smaller packets, and the equation is trying to predict a maximum number of small packets, it is known that $k/n > p$. Therefore, $k > np$, which allows the statement,

$xk^2 = xk(np + b)$ where b is the difference between k and np. Therefore,

$$\begin{aligned} xk^2 - 2xnpk + xn^2p^2 &= xk(np + b) - 2xnpk + xnp(k - b) \\ &= xknp + xkb - 2xnpk + xknp - xnpb \\ &= xkb - xnpb > 0. \end{aligned}$$

Also,

$2ka - npa > ka$

If $ka > a^2/(1+x)$, the value a must be positive and the proof holds. Since in the beginning, the value k(1+x) defined the total value of k1 including all previously known values of k, then k must always be greater than a/(1+x), because a can not decrease the initial number of small packets to lower value as the number of packets n increases. Therefore "a" is a positive number and the proof is complete.

From the above, it has been proved that for any number of samples n, for which the normal distribution is a reasonable approximation for the Bernoulli equation, namely npq >> 1, if the system can accommodate some number of small packets k, whose probability of occurring is p, meaning the system can store and process the data at a rate proportional to k/n, and the memory is capable of storing the worst case distribution, then the system can store and process any distribution of arriving messages of length n_{new} with a system capable of meeting the needs of the larger sample of packets for the known probability P(k).

3.2.4 Using the Packet Distribution for Bounding Memory Size and Microprocessor Clock Rates

From knowing any three of the following four items,

1) percentage of packets not accepted,
2) microprocessor throughput rate,
3) memory size, and
4) expected percentage of small packets,

the fourth item can be calculated. This is possible because the items are all interrelated. As was proven above, if a microprocessor can be used to process the average number of packets received in a period of time, it can also process a larger total number of packets with a similar blocking rate,

P(k), because as n gets larger the k/n ratio gets smaller while maintaining P(k). It is important to note that it is impossible to have k+1 small packets out of n+1 total packets while maintaining a constant blocking value, and since the microprocessor can process a packet of the large size in less time than its arrival time, at least one small packet has been removed from the queue during the arrival of the large packet. If a small packet is the second packet to arrive, the backlog of small packets will never be larger than k. The worst case for buffer sizing is for k small packets to arrive in a row followed by consecutive large packets arriving during the period of processing all the small packets. The reason the first large packet is present is to delay the starting time of the small packet processing as much as possible. The theory behind n-k incoming large packets assumes the n+1 packet to arrive is large and is therefore consistent with any other conditions. Placing other large packets at the beginning would not cause the microprocessor to be any further behind in processing than is caused by a single large packet, because the microprocessor can process any large packet in less time than a new one arrives.

Figures 3.2.2 and 3.2.3 demonstrate the above proof in a slightly different fashion. Given the probability of a small packet occurring, the curve generated shows the decreasing probability associated with a constant k/n ratio

as the total number of packets increases. Figure 3.2.2 uses the values given by the direct Bernoulli equation and the second graph shows the approximations given by the normal distribution. The values of k used are for cases where k/n is always greater than or equal to the probability p.

Given a desired blocking value, the percentage of possible packet combinations which results in a failure, the ratio of the small to total packets to be stored and processed, k/n, can be determined. The ratio is solved in two possible ways. The first is using the Bernoulli trials. The Bernoulli trials utilize factorial combinations and are difficult to define when the total number of packets gets large. In addition, it is difficult to determine the values of n and k explicitly since the functions are not continuous. For small values of n, the Bernoulli trial provides the correct approximation for the system. For npq >> 1, the normal distribution becomes a reasonable approximation. In addition, the use of the normal distribution provides for easy solutions to k/n ratio since the normal distribution is a continuous function.

Given that a desired blocking value is known, P(k), the ratio of k/n can be determined by successive approximations using the Bernoulli distribution or the normal distribution. Figure 3.2.4 shows a graph of the resulting k/n ratio as a

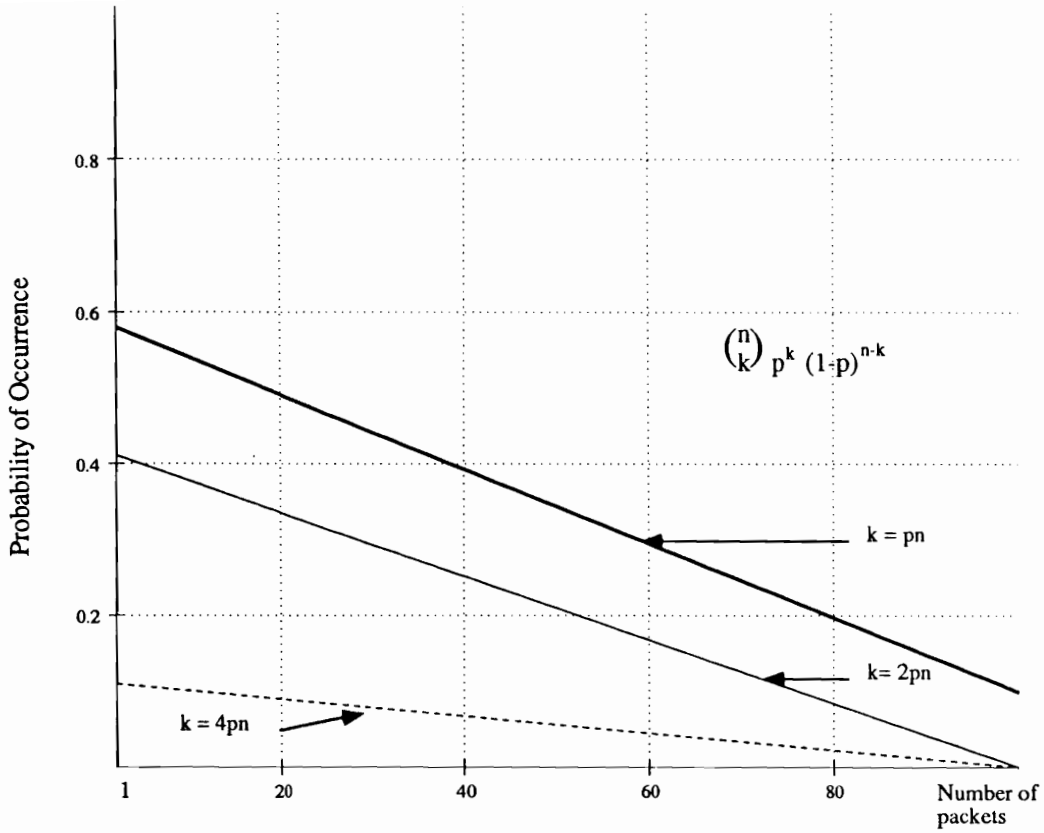


Figure 3.2.2: Probability of the ratio of k/n occuring given a long term probability of p = 0.2 as the number of occurances increases. Note that for any given k/n ratio, the probability of occurrence is a maximum at the origin. The graph was derived from the Bernoulli equation.

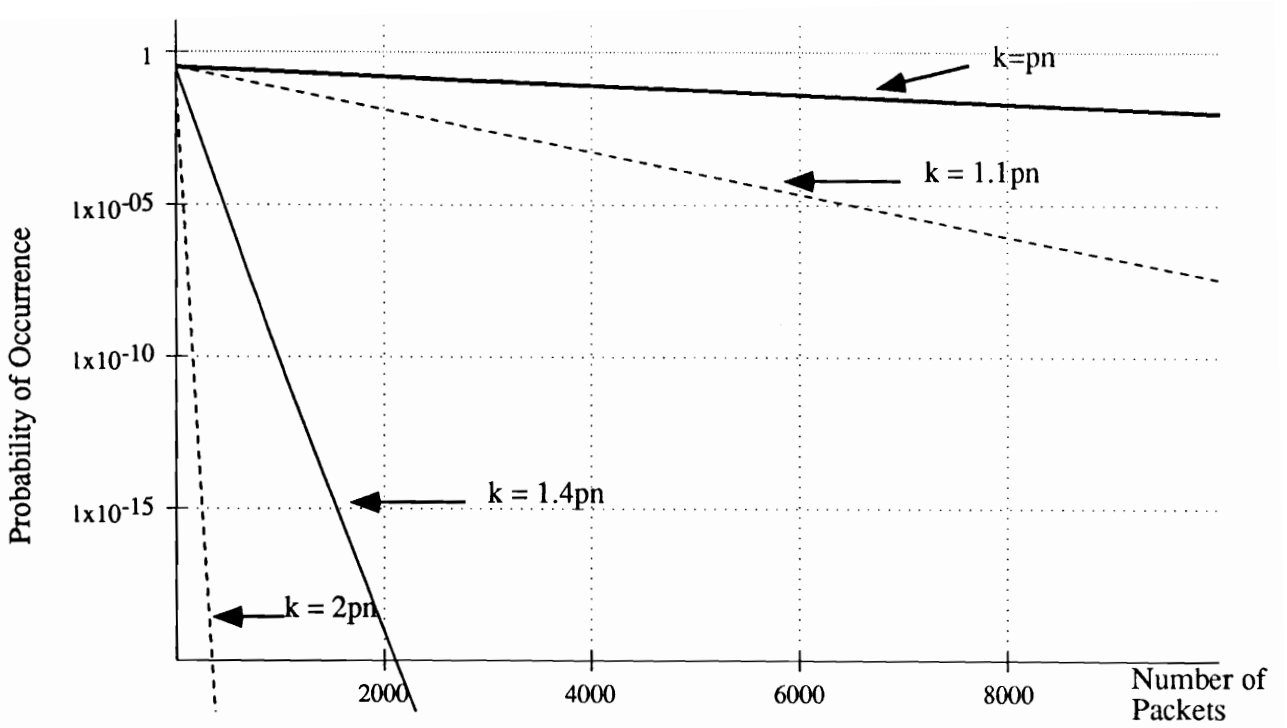


Figure 3.2.3: Probability of the ratio of k/n occurring given a long term probability of p = 0.2 as the number of occurrences increases. The graph uses the normal approximation for the arrival distribution.

function of the number of packets of interest, n, for different blocking values given that the probability of a small packet occurring, p, is 0.1. Note that as the blocking value gets smaller, the ratio of k/n which must be acceptable for any given number of packets increases. Figure 3.2.1, shows that as the value of p, or in this case k/n increases, the microprocessor rate must increase to keep up with the data distribution expected. For any given number of packets expected, the microprocessor clock rate must increase to provide a smaller blocking value.

From Figure 3.2.5, the effect on the k/n ratio for variations in the expected arrival rate, p, for the given blocking value of 10^{-8} can be seen. Note that varying p has a great effect on the expected k/n ratio especially when less than 10,000 packets are considered. This variation demonstrates the increasing influence of the expected arrival rates, p, on blocking rates. Note that for p=0.4 the minimum number of samples needed to achieve a blocking values of 10^{-8}, is considerable greater than for p=0.1.

The k/n ratio at each point of interest is the ratio of the number of small packets to total number of packets that the system must accommodate for the microprocessor to be able to process the total n packets in their arrival time. If there are k small packets of n total packets in a two point

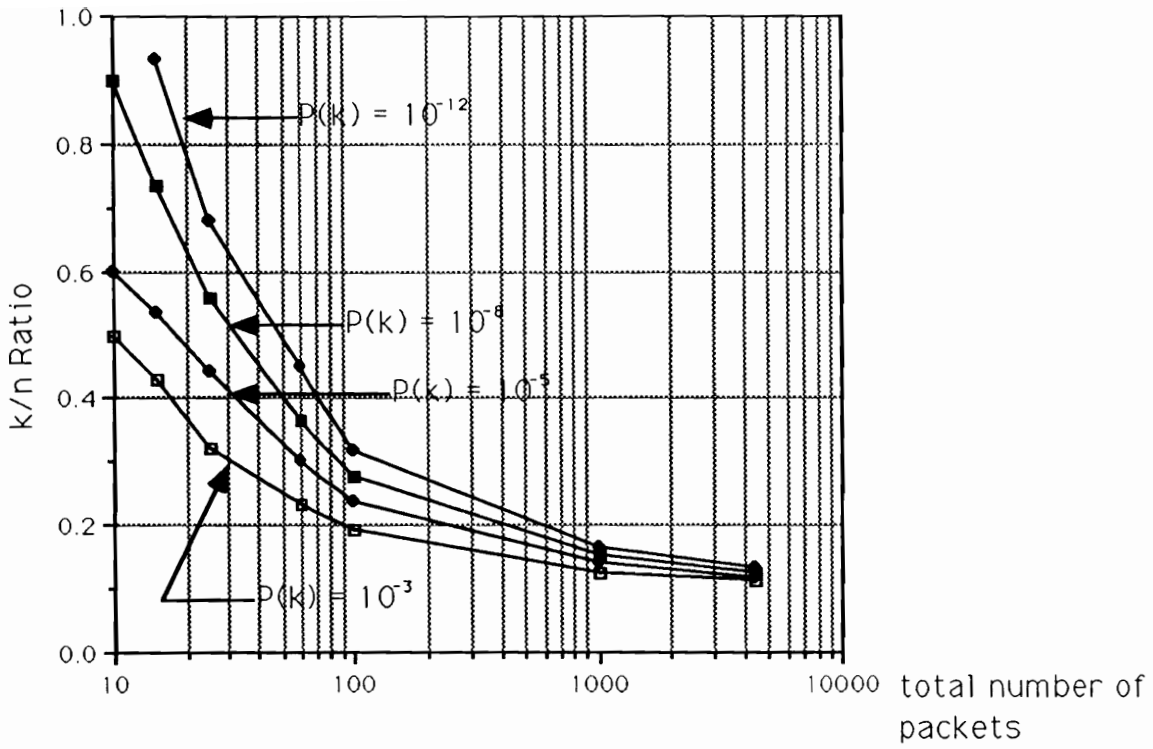


Figure 3.2.4: Graph showing the required ratio of small to total packets, k/n, given the expected distribution of p = 0.1 for several blocking values, P(k), as a function of the total number of packets expected to arrive. Calculated using the Bernoulli and Normal Distributions

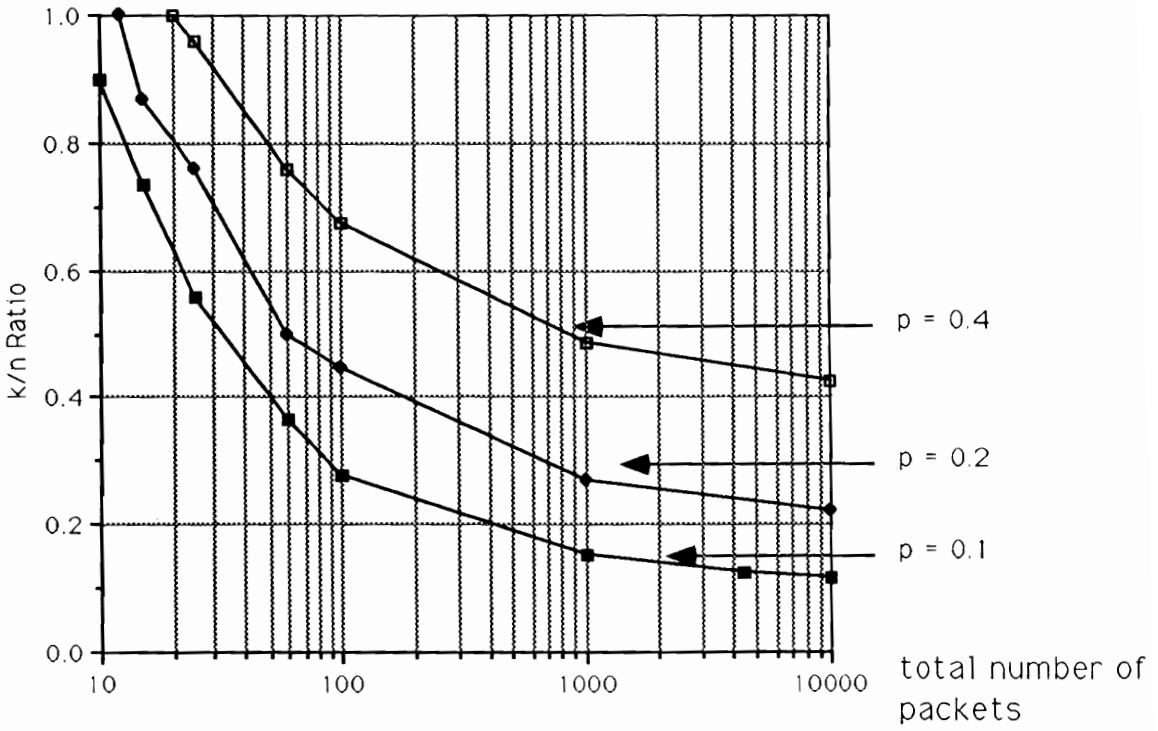


Figure 3.2.5: Graph showing the required ratio of small packets to large packets, k/n, given a desired plocking probability, P(k) = 10 for various values of p. Calculations are made using the Bernoulli and Normal Distributions for P(k).

distribution, the average packet size arriving during the period is given by k/n*S1 + (1 - k/n)*S2 where S1 is the size in bytes of a small packet and S2 is the size of a large packet. From this average packet size defining k/n, n can be found by solving for the blocking value knowing p and the desired value of P(k). The clock rate associated with being able to process the k/n ratio is given by

$$MCR = \frac{BR}{8 * (BPP + OHB)} * (IMO * CPI * ECI + \frac{BPP * CPX}{BPX}).$$

BPP is now defined as the average number of bytes arriving in the period of interest of n packets. Specifically, the equation can be rewritten as

$$MCR = \frac{BR * \left(IMO * CPI * ECI + \frac{(k/n*S1 + (1 - k/n)*S2 * CPX)}{BPX}\right)}{8 * (k/n*S1 + (1 - k/n)*S2 + OHB)},$$

This equation was used to develop Figure 3.2.6, which shows the required microprocessor clock rates as a function of the number of packets which have passed across the network for an average packet size value of p=0.1 for various blocking values. The number of clocks for overhead are assumed in the graph to be 1000 clock cycles and the bit rate is assumed to be 100 Mbit/s. The clock rates along the y-axis are the required clock rates to process the average distribution seen during n packets. Note this does not mean that after n

packets that the microprocessor will have no backlog of data to process, just that the average amount of arriving data is processed in the same amount of time required for its arrival.

In Figure 3.2.7 the blocking function is the constant and the value of p is varied for the different curves. If the number of clock cycles for overhead were made larger or smaller, the increases in clock rate would be similar to changes in throughput shown in Section 3.1.

3.2.5 Memory Sizing

The memory size between the high speed network and the processing unit defines the number and sizes of packets which may be bufferred by the system. The larger the memory, the greater the ability of the system to store data until it can be processed. Because the processing capabilities of the systems examined earlier, it has been determined that if the microprocessor is able to process an average packet of length μ, the small packets can not be processed as quickly as they arrive and large packets can be processed more quickly than they arrive. The memory size defines the ratio of small and large packets that can be stored for a network such as FDDI which requires sequential processing. Since it is desirable

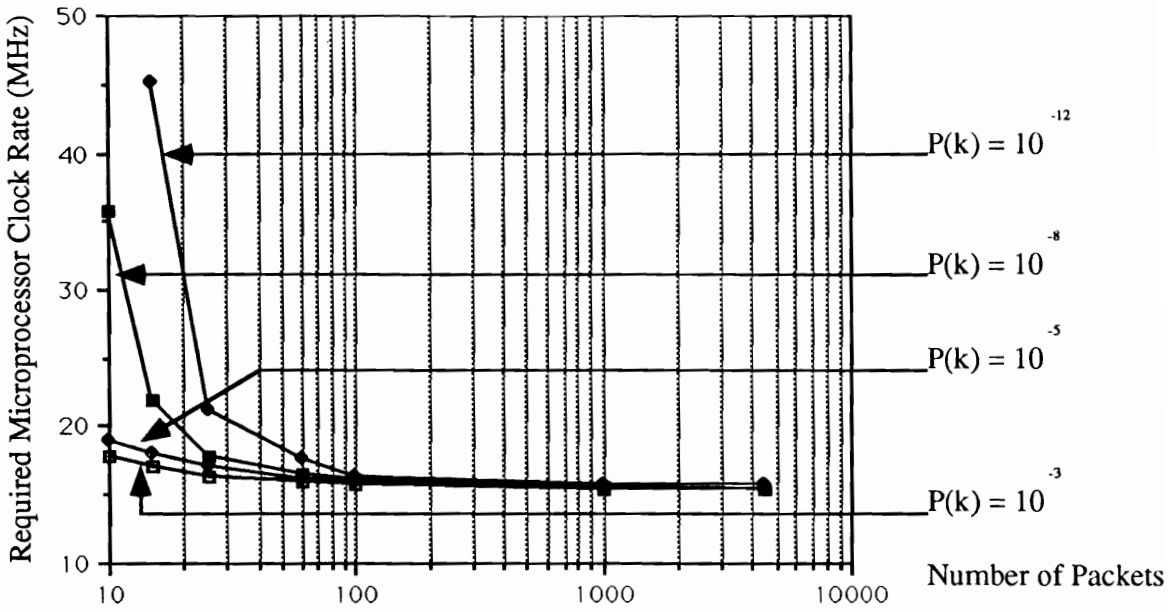


Figure 3.2.6: Shows the required microprocessor clock rate as a function of the number of packets which have passed across the network for a value of p = 0.1 for various blocking values. The number of clocks for overhead are assumed to be 1000 clock cycles with a bit rate of 100 Mbit/s.

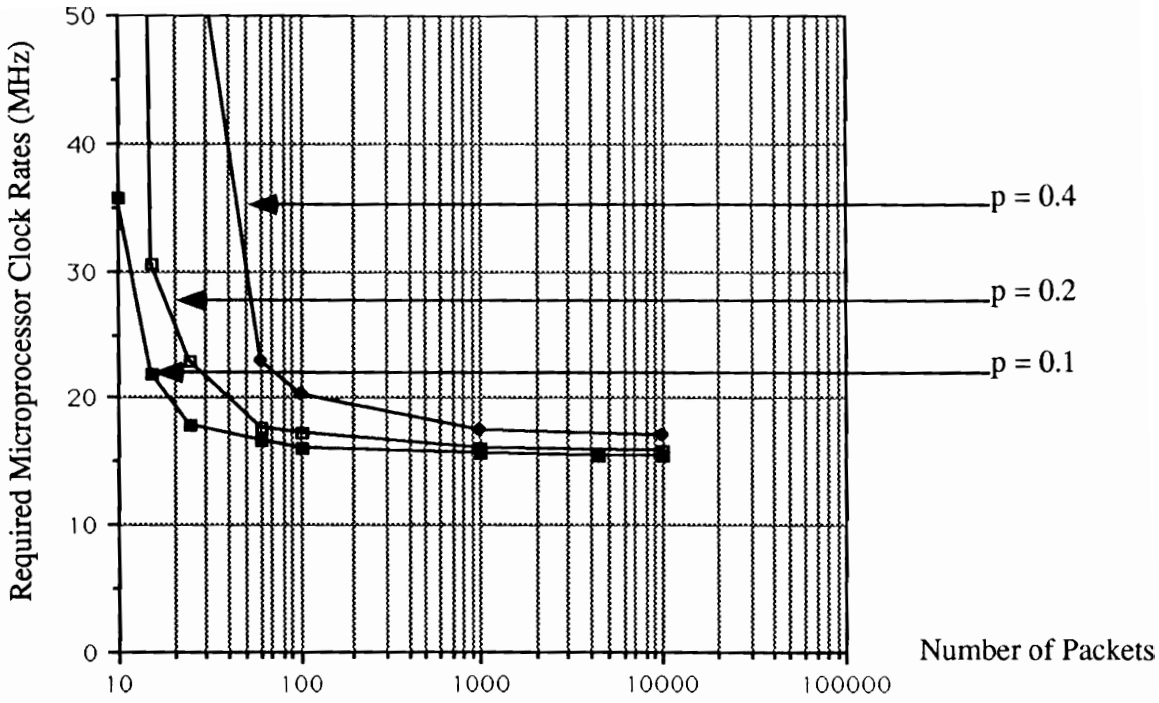


Figure 3.2.7: Shows the required microprocessor clock rate as a function of the number of packets which have passed across the network for a blocking value of 10^{-8} for various values of p. The number of clocks for overhead are assumed to be 1000 clock cycles with a bit rate of 100Mbit/s.

to have a large probability of success in processing packets, the number of small packets which may occur successively can be high when the number of packets considered is not large. This was demonstrated in Figure 3.2.4 where the k/n ratio was plotted against the total packets for a desired blocking probability. To meet the needs of a MAU with a bimodal distribution of packets, it is necessary for the microprocessor to be able to process small packets without delay or for there to be a buffer mechanism to allow packets to be stored until the microprocessor is able to process them. As was shown in Section 3.1, to process the small 6 byte packets even in the ideal case using the improved TCP/IP and no other overhead requires a microprocessor clock rate of over 100 MHz. This is an unrealistic clock rate for operation using a CISC or RISC-based machine. The next option is to allow the MAU to buffer the data. Buffers are typically sized in industrial applications through "seat-of-the-pants" estimates and testing is done to determine the adequacy of the design. When blocking values of 10^{-8} or greater are desired, the number of possible permutations of the packet distribution can take years to test. For this reason, an estimate of required memory size is desired.

The distribution of data discussed in Section 3.2.4 proves that if at any point in a data arrival distribution, when the microprocessor can meet the requirement of the worst

case ratio of small packets to large packets for the given data rate, any distribution after that observation will have a lower percentage of small for a given blocking value. At any point in a sample of n packets, there are no more than k small packets. The processing method allows the buffer to empty and fill successively and each time to accommodate and new k/n packet ratio. To determine the minimum memory size needed to allow for the worst case distribution of data, arrival distribution, and data processing rate must be examined. Because the design uses the worst case distribution, some larger k/n ratios may successfully complete due to the order of the small and large packet arrivals. The increased k/n ratio does not guarantee failure but the desired k/n ratio guarantees reception.

The worst case for data back-up begins with all small packets arriving sequentially with the microprocessor currently processing a large packet. The maximum back-up of unprocessed data exists for data arriving while the microprocessor is processing small packets. The steps to determine the maximum memory size for a memory using variable length packet storage, such as the incoming MAU interface memory, given the microprocessor attributes, blocking value and p value are:

1) Determine the k/n ratio the microprocessor is capable of

handling by applying the microprocessor attributes and determining the minimum number of bytes per packet as shown in Figure 3.1.18 and using the value of BPP_{min}, find k/n to solve the equation

$$BPP_{min} = (k/n)*S1 + (1 - k/n)*S2, \text{ giving}$$

$$k/n = \frac{S2 - BPP_{min}}{S2 - S1}$$

2) Given a desired P(k) and k/n ratio, solve for the total number of packets, n. This is done by solving the blocking equation,

$$P(k) = \frac{1}{\sqrt{2\pi npq}} EXP[-(k-np)^2/2npq] \quad \text{for n,}$$

through successive approximations. By knowing the value for n, it is possible to determine the value k, the number of small packets.

3) Knowing the number of small packets in the queue for the worst case, it is possible to compute the amount of time required to process the number of packets in the buffer. This is done by taking the number of small packets and multiplying it by the number of seconds required to process each packet and adding the delay from the large packet assumed to be processed immediately before the first small packet. This time is the delay time during which new packets can be arriving.

4) The number of bytes required to be held in the queue is the number of bytes which have arrived during the processing of the small packets plus either the original large buffer which was in the queue (when the memory defined by the processing of the small packets is less than the buffer) or the additional memory required for processing the large buffer, plus memory for one additional packet being received.

For any distribution of data of k/n less than the point k/n for which the microprocessor can process data as quickly as it arrives, the microprocessor can remove data from the queue more quickly than it arrives. At the k/n ratio, data is actually being removed faster than it is arriving because the arrival rate assumes no delay between packets while in actuality some delay is required. The raw data arrival rate of large packets is assumed to be the same as the raw data arrival rate of small packets. Since the distribution of k/n packets can be accommodated, the microprocessor can accommodate any subsequent packets. The microprocessor will clear memory at least as fast as new packets are arriving from the network. The additional memory defined for storage is used to account for a data packet which is in the process of being removed from the list but has not been cleared while a new packet is arriving. This is seen in the example for the packet size immediately following the processing of all the

small messages and the additional memory is required to store the incoming packet while the fist packet after the small packet groups begins processing. At this point in time, although buffers are being processed as fast as they are filled, they can not be released until the processing of them is complete.

If the memory size is known, the trade-off of the other items can be solved in reverse.

The results of these four steps are shown in Figures 3.2.8 and 3.2.9 which show how memory requirements change as a function of microprocessor clock rate for various blocking values and expected arrival distributions, p.

Several possible scenarios can be developed which bring up questions about the validity of the model used. These problems are:

Firstly, the worst case to be examined is that all the small packets have arrived in succession. This is a worst case, and made possible because of the truly independent method of data arrival represented by the Bernoulli trials. The possibility of computing exact distributions of data for hundreds or thousands of data packets and their distribution of arrival would take literally years due to the shear number

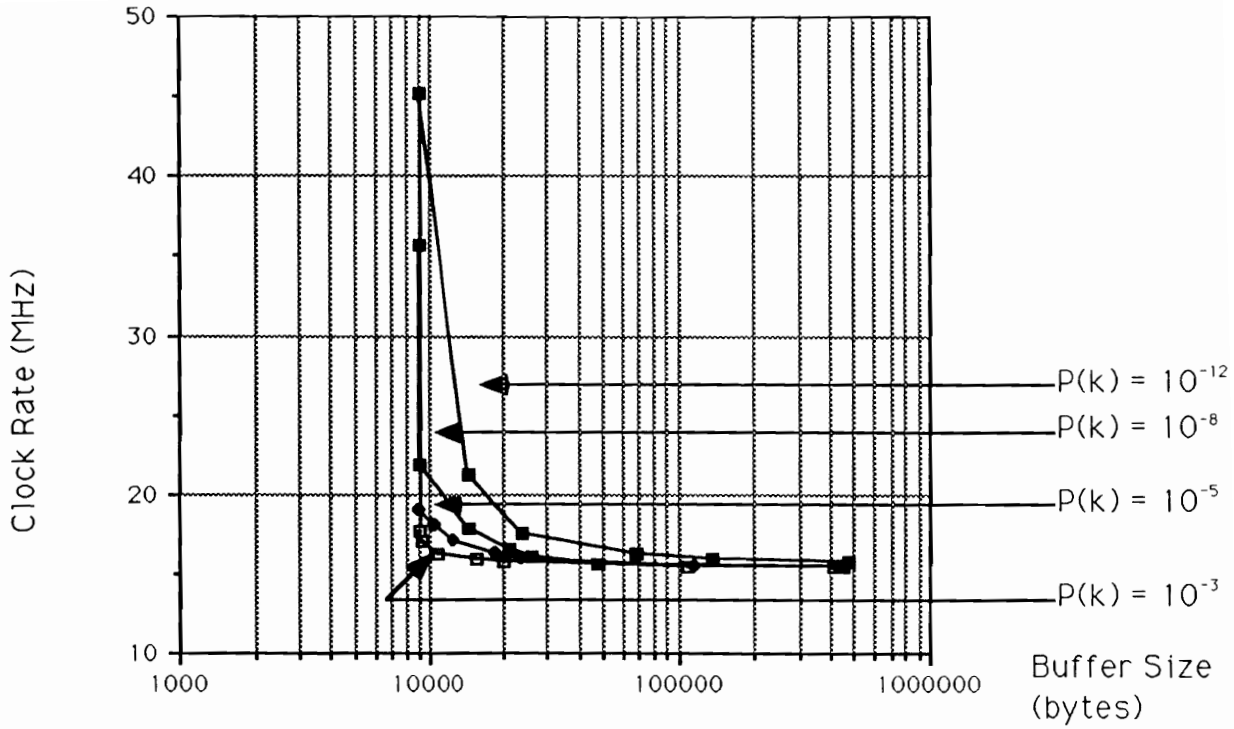


Figure 3.2.8: Plot of buffer requirements as a function of microprocessor clock rates for varying blocking probabilities for a long term probability of small messages equal to 0.1. The IPO is assumed to be 1000 clock cycles and the transfer rate is one byte per clock.

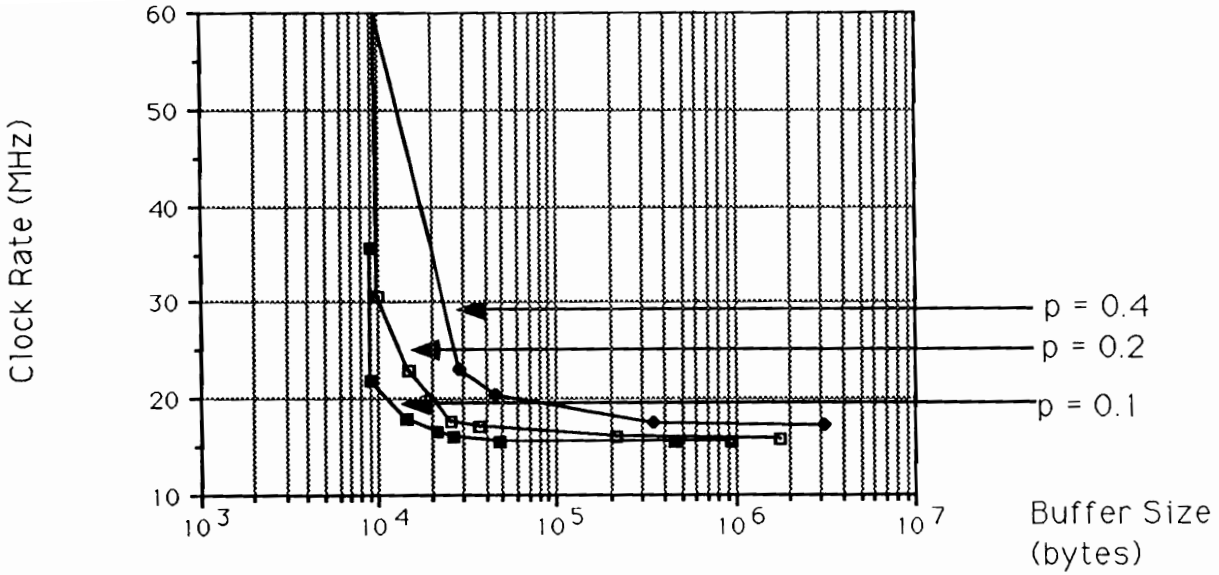


Figure 3.2.9: Plot of buffer requirements as a function of microprocessor clock rates for varying long term probability of small messages for a fixed blocking probability, $P(k) = 10^{-8}$. The IPO is assumed to be 1000 clock cycles and the transfer rate is one byte per clock.

of calculations since the possibilities would be based on the factorial value of the number of arriving packets to be examined. For design purposes, meeting the worst case will always provide proper operation for other cases.

Secondly, it is often envisioned that all large packets will arrive first, and then all small packets, followed by a distribution of packet sizes. This case is allowed for by the sizing algorithm because the microprocessor throughout the large packets is always finishing processing of the large packet before the next large packet finishes arriving. When the small packets begin to arrive, the microprocessor has one large packet being processed. The data following the reception of the last small packet is a mixture of packet sizes, but must begin with a large packet if the blocking probability is to be greater than or equal to the blocking probability for which the system is defined. Therefore the sequence of small packets is broken and the new distribution of packets can have no greater distribution of small to large packets than what can be processed without additional buffering. The worst case memory sizing is the number of bytes that can arrive during the processing of all the small packets, regardless of the packet sizes arriving. After defining the maximum number of bytes that arrive while the small packets are being processed, the buffer adds an additional 4500 bytes to guarantee that a large packet can be

held in memory as the processing begins. This defines the incoming MAU memory requirements.

In contrast, the outgoing MAU interface memory is block oriented and has different requirements. Since the outgoing memory is limited by the ability of the network to remove packets for the memory and transport it over the medium, the queue should be limited to the amount of small packets possible at any point in time. This is a function of the distribution, and the total number of packets passed. For any period under observation, the maximum number of small packets that can exist in the outgoing MAU memory is a function of the total number of packets being sent out and the number of packets arriving. Since the ratio of small packets to the total number of packets is greatest at the beginning of any sample period, the system will be able to operate properly if designed for the beginning case.

Packet acknowledgement time is a function of queue length at the receiving node, the receiving node's processing speed, the number of nodes and their separation, and the target token rotation time. These factors are difficult to calculate for a worst case performance. In addition, the outgoing MAU memory size is a function of the amount of data being available for the buffer and the network being available for the transport of the data. The simplifying assumption will

be made that the devices needing maximum FDDI data rates will have access to the bus during the burst periods for which they require the full bus bandwidth. The desire is to provide the network with data whenever the network is available. Therefore, the best way to determine the number of bytes required by the outgoing MAU interface memory is to determine the maximum number of bytes required for any timeout period and deriving the appropriate ratio k/n which meets the requirements based on the number of packets passed in the given time period of interest. This requires information about time-out values, blocking values and expected packet length probabilities, p. From these, the values of n and k are solved in an iterative fashion. Since any reasonable timeout period will consist of multiple packets, namely allowing the value npq>>1, the normal distribution for the blocking function can be used. Figures 3.2.10 and 3.2.11 show memory sizes as a function of blocking values and expected probabilities for a defined time-out of 0.1 seconds. The equations for the amount of memory space simultaneously solve the equations for the amount of packets that arrive during a time-out period and the k and n values for the blocking probability. The equation for determining the packets arriving in the time-out period is given by

Time-out = k * processing time for a small packet
+ (n-k) * processing time for a large packet.

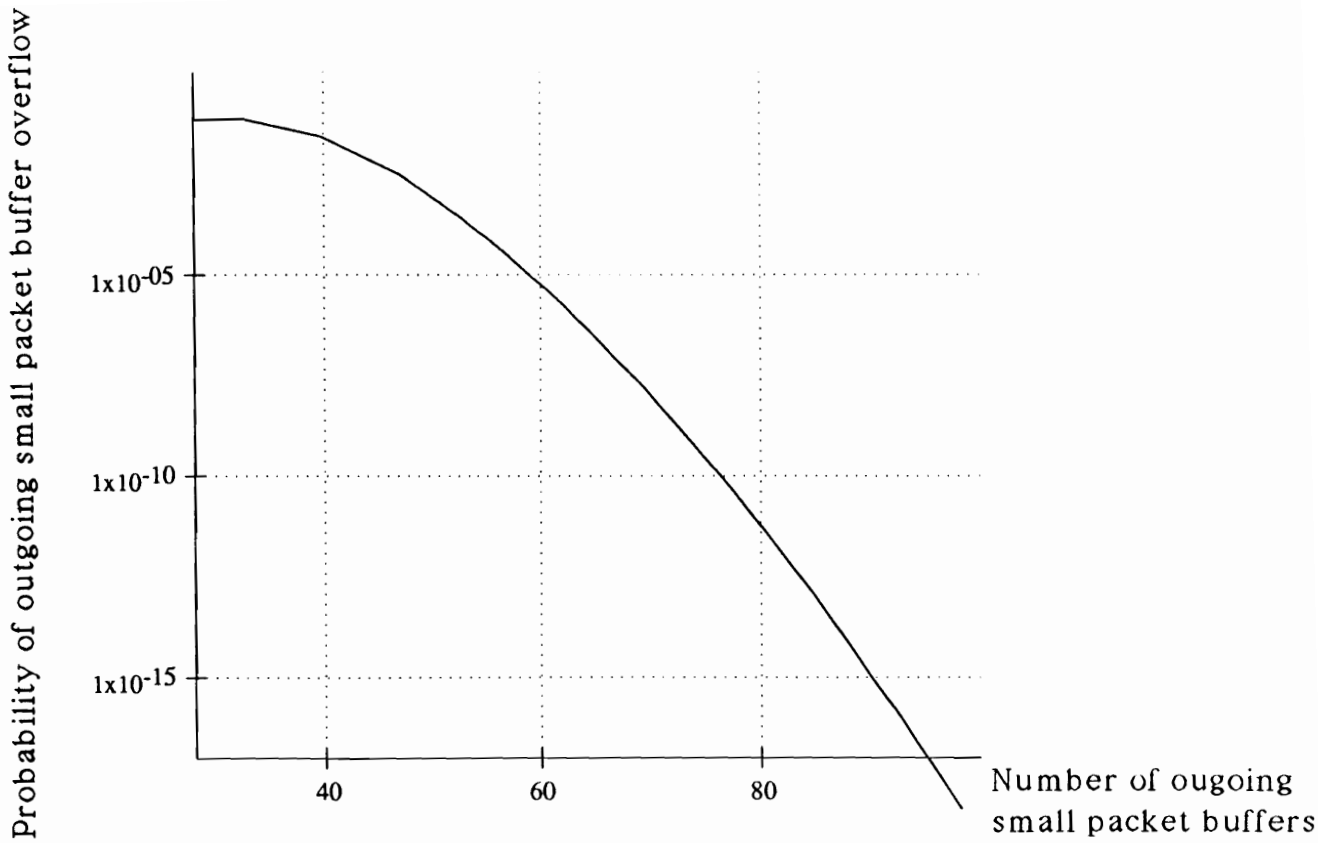


Figure 3.2.10: The number of small packet buffers preallocated versus the probability of a buffer overflow for a network protocol time-out of 0.1 seconds and a small packet transmission probability of p = 0.1.

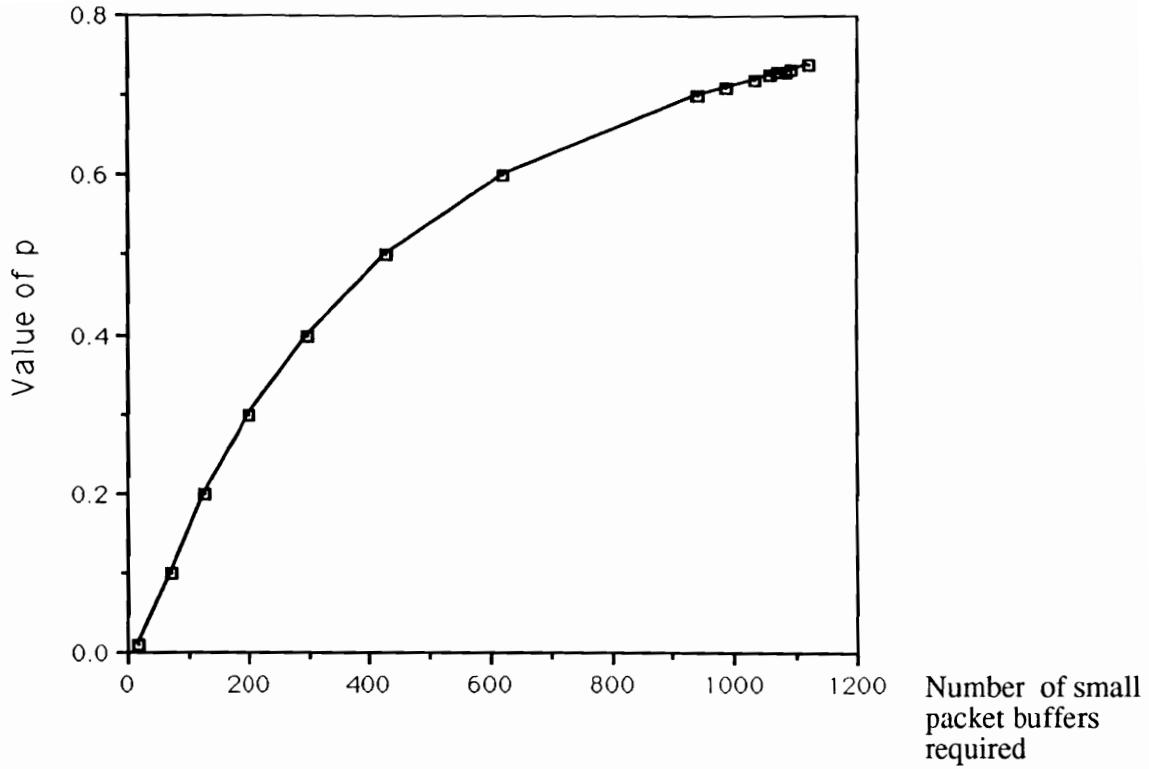


Figure 3.2.11: Number of outgoing packet buffers required to guarantee a small packet is not dropped because the network is not servicing packets as quickly as they are processed. The timeout value for the protocol is 0.1 second. The desired blocking probability is 10^{-8}.

The number of large packets is bounded by the maximum number of large packets which can be placed on the network in the timeout period.

For specific timeout values, Table 3.2.2 shows the number of 4500 byte memory units that must be reserved.

3.2.6 Comparisons to a Computer Simulation

In Appendix A, a program is presented which uses pseudo-random number generation to estimate the number of buffer overflows which occur for given long term distributions, buffer sizes, clock rates and blocking probabilities presented in this chapter. The results of the simulation are given in Figure 3.2.12 for several memory sizes and microprocessor clock rates and are compared to the estimation method presented earlier in this chapter. As expected from earlier discussions, the estimations developed in the dissertation have a higher blocking probability than the simulation predicts. The estimations present a complete method for determining worst case failures and designing systems to meet the worst case errors over any range. For the figure, the 15.7 MHz clock rate corresponded to a k/n ratio of 0.2 while the 18.9 MHz clock rate corresponded to a k/n ratio of 0.4.

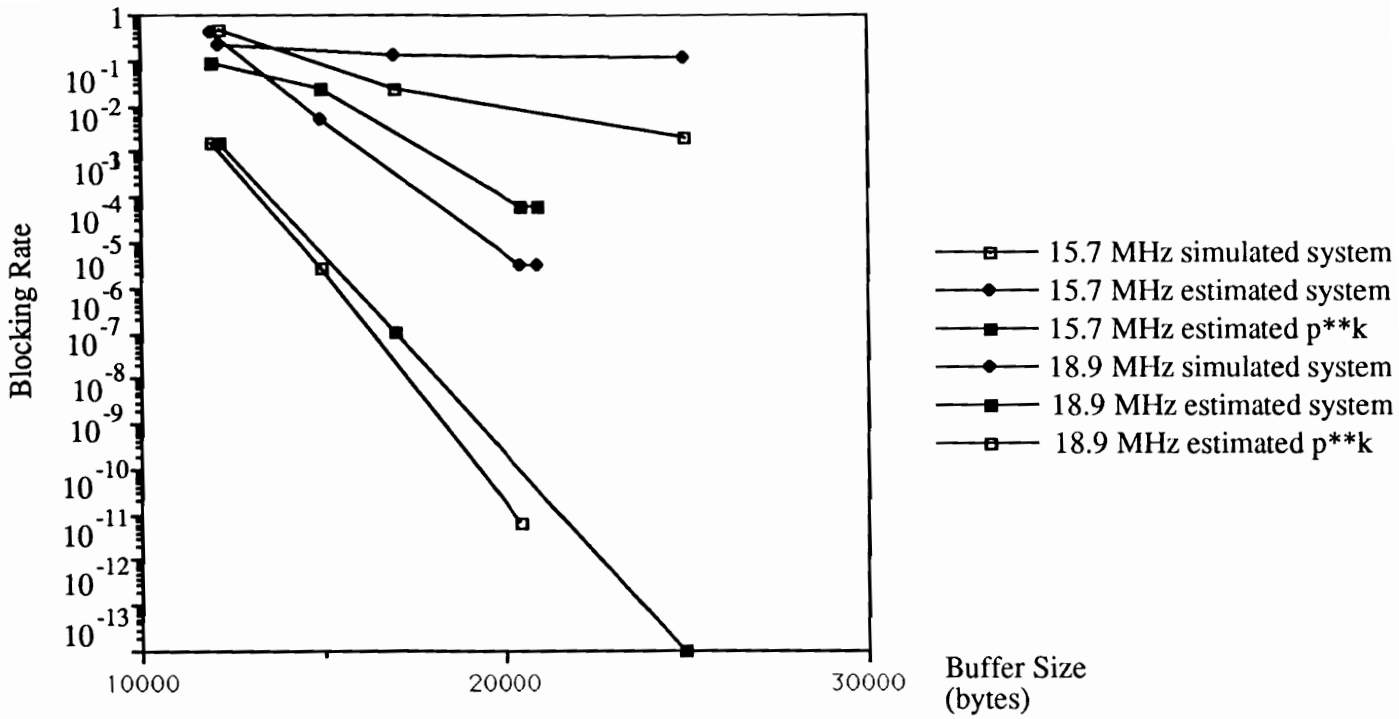


Figure 3.2.12: Plots comparing the simulated response of the MAU with the estimated blocking probability and the probability of the k messages occuring in a sequence. The estimated blocking probability provides an effective upper bound. The packet arrival was predicted using the SPARC random number generator. The simulation program is shown in Appendix A and uses IPO = 1000 clock cycles and p = 0.2.

Table 3.2.2: Required Data Storage for Large Packets of the Outgoing MAU Memory

Time-out (Seconds)	Number of 4500 Byte Packets Required	Total Memory Required (Bytes)
0.01	28	1,008,000
0.1	278	10,008,000
0.5	1,389	50,004,000
1.0	2,778	100,008,000
2.0	5,556	200,016,000
5.0	13,889	500,004,000

Required number of maximum size preallocated memory locations required for 4444 byte data packets being used to fill the FDDI ring. The number of large packets are not limited by the blocking value like the small packets, so the system must be able to accommodate all outgoing packets to be large packets.

The long term distribution was set at 0.2. For comparison, the probability of errors is estimated based on only the possibility of the k small messages occurring in sequence. The estimations using the k/n ratio provide significantly better correspondence to the simulation than the k sequential messages. The number of samples is limited and although the program can be used very effectively it is only as accurate as the random number generator. The program has approximately 180 lines of source which must be run for every iteration of the simulation. Most of the points of the graph were run as 10 million point iterations, but would need large increases when error probabilities get below 10^{-6}. The simulation does provide verification that the estimations provide a reasonable upper bound on error rates, but the simulation is not sensitive enough to provide exact blocking probabilities. In addition, if data on networks is truly grouped so that successive small packages occur with more commonality than truly random messages, then the estimation will be even closer to the real results than the random data based simulation as shown in Chapter 4.

3.2.7 Summary

Section 3.2 has shown the advantages of using 32-bit data transfers when dealing with FDDI and TCP/IP. In addition,

this section has shown how the requirement of a blocking probability can be used to define memory and microprocessor throughput rates and the relationship between memory sizing, expected packet distributions, and blocking probability. A decrease in blocking probability or an increase in the percentage of small packets leads to increase requirements in either microprocessor speed, memory sizes or both. In compliance with Section 3.1, it is realized that as always, the overhead associated with the microprocessor will affect all these variables. Chapter 4 will demonstrate how these variables can affect memory sizing and blocking values.

3.3 Extended Address Interfaces versus VMEbus Interface for FDDI/SBC Data Access

Until this point, the FDDI MAU has been assumed to have a direct interface between memories, microprocessors and I/O interfaces. At the beginning of this paper, the question of direct addressing versus standard interfaces between elements of the system was discussed as having a large affect on system latency and throughput. The equations generated in Section 3.1 assume no latency for the transfer of information. Before moving on to the specific application for the Star Technologies VP series machines, the interface strategies and their associated latency and throughput values should be

examined. The question is whether to use a direct interface or a standard interface, specifically the VMEbus interface.

The VMEbus "is the most popular 16/32-bit backplane bus. It's use of the Eurocard format, its high performance, and its versatility are some of the reasons that it appeals to a wide range of users" [37]. The VMEbus is used to provide communication between devices on VMEbus without disturbing other devices on the bus. The VMEbus is used as an interface between units which may otherwise not exist on the same board. A general view of the interface is seen below.

In analyzing the use of the types of connections in the system, the throughput of VMEbus must be considered. The VMEbus can transfer from 1 to 4 bytes in each cycle. To transfer data between points of interest, the system must use a read cycle or a write cycle. In addition, the bus has the ability to use block transfers which can read or write 1 to 256 continuous bytes of data in the 1 to 4 bytes per cycle format defined in the normal VMEbus read or write cycle. Figure 3.3.3 shows an example of a single byte read cycle. As is shown in this figure, there are many steps to setting up a VMEbus transfer cycle. The "total data transfer speed is a function of the rate at which the board(s) supplying and receiving the data drive the VMEbus data transfer acknowledge signal (DTACK*)... with a maximum VMEbus speed limit of 10

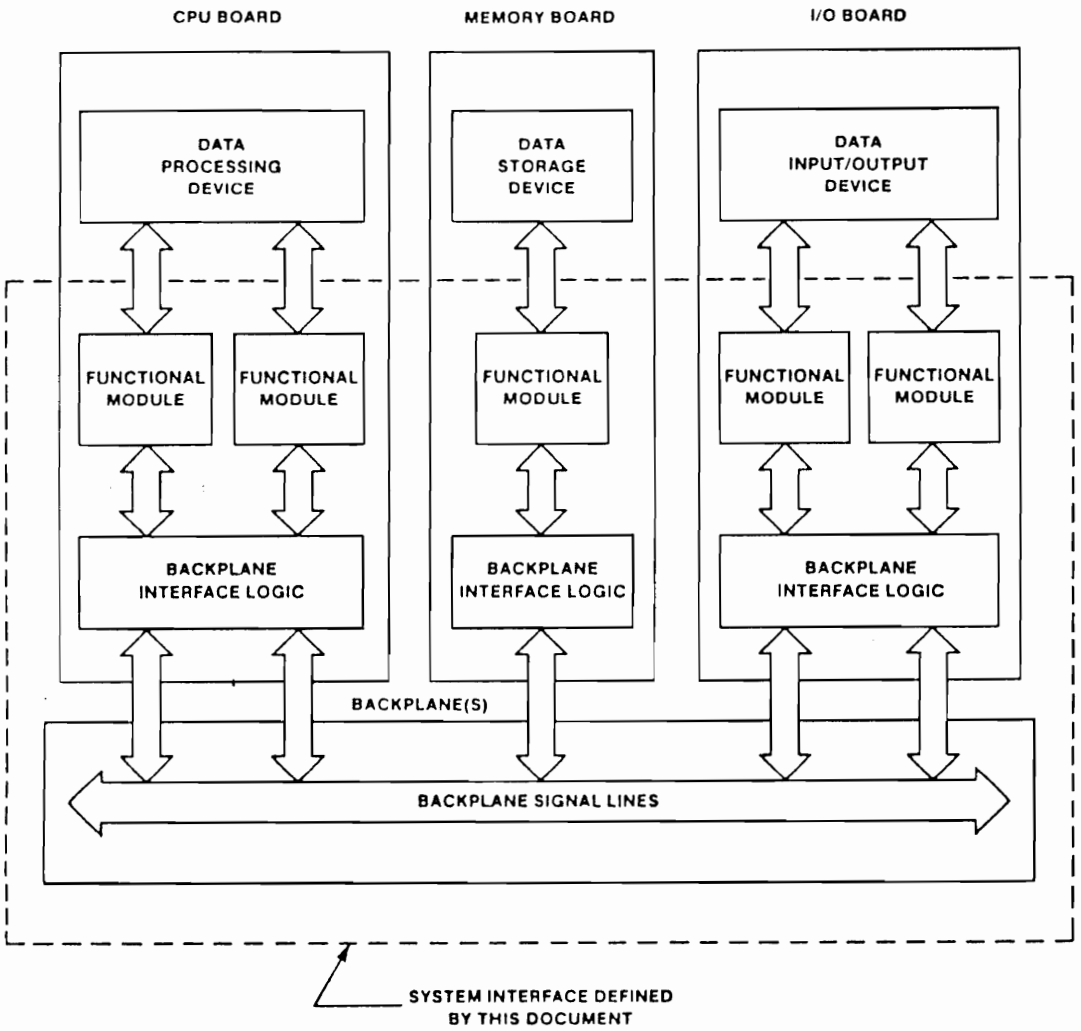


Figure 3.3.1: System Elements Defined by the VMEbus documents, from The VMEbus Specification Figure 1-1, [37].

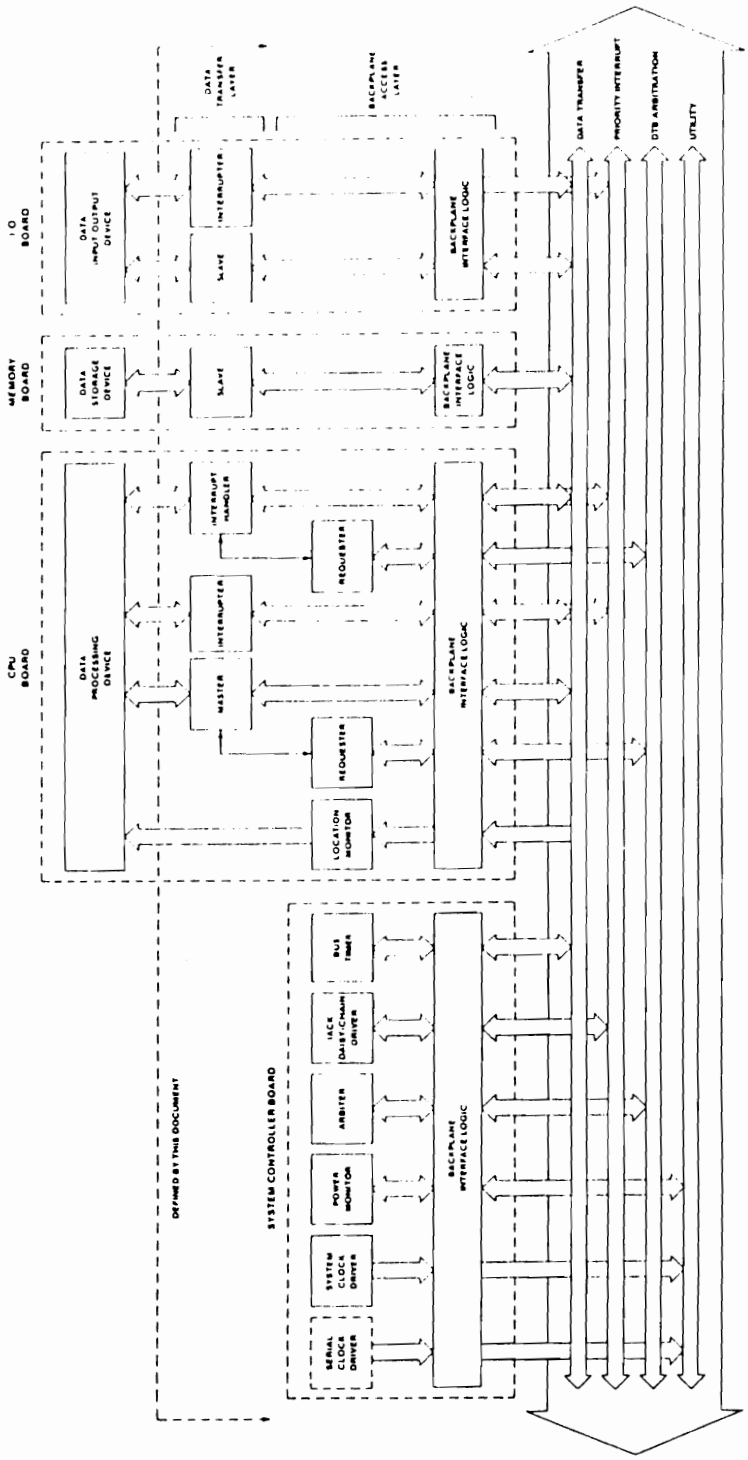


Figure 3.3.2: Functional modules and buses defined by the VMEbus specification, from The VMEbus Specification, Figure 1-2, [37].

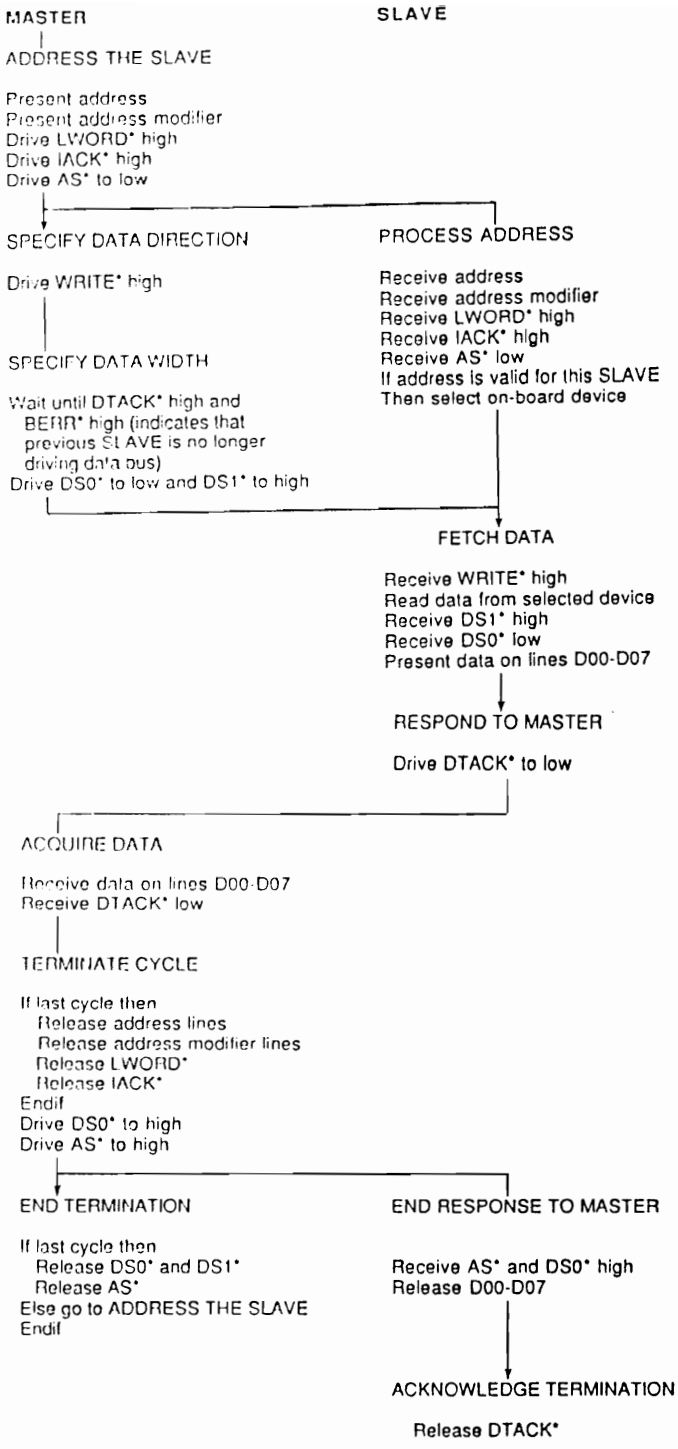


Figure 3.3.3: Example of the single byte read cycle, from The VMEbus Specification, Figure 2-8 [37].

million transfers per second" [38]. The transfer speed is dependent on the access speed of memory or the system output of the data to the interface controlling the data transfer bus. For VMEbus systems, the data transfer bus is controlled by a master controller.

The master controller is usually specified as a special card which controls all data transfers. The VMEbus is normally defined with multiple master stations, slave stations and a single bus arbitrator. The purpose of the bus arbitration system is to

1) prevent multiple master stations from simultaneously controlling the bus, and
2) scheduling requests from multiple master stations to optimize bus usage.

The VMEbus provides a means of connecting boards together over a common backplane and to arbitrate the system. A master board is a "functional module that initiates DTB [data transfer bus] cycles in order to transfer data between itself and a slave module" [37]. There is no provision in the VMEbus to allow master stations to communicate with other master stations. For this reason, a single board may contain both master and slave addresses so that it can be accessed and can access other devices. There are several busses in the VMEbus structure, but only one, called the data transfer bus, is used

to pass data and addresses. The others are used for utilities, interrupts and arbitration. The data transfer bus is the bus of main concern in analyzing the ability of VMEbus to be used in an FDDI environment.

Figure 3.3.4, shows the most extensive use of VMEbus as a transfer mechanism in the basic FDDI hardware system described in Section 3.1. In this scenario, there would be at least three separate boards vying for the bus, in which case the bandwidth of VMEbus begins to become a limiting factor in the system. VMEbus has a maximum transfer rate of 10 million transfers per second. This equates to 320 Mbit/s if every possible cycle of the bus is used for data transfer. Each of the three boards defined in Figure 3.3.4 has a data bandwidth approaching 100 Mbit/s. If more than these three boards exist in the system, the required data rate will exceed the available bus rate. If the number of VMEbus boards is limited to those shown in Figure 3.3.4, and each contains its own master and slave ports, the data rate using exclusively 32-bit transfers would require the bus utilization for data to be over 93.7 percent (300 Mbit/s / 320 Mbit/s). If the reduction of data due to stripping of the FDDI header and the TCP/IP header is taken into account at the appropriate memories, the required bandwidth for one byte data packets would be 86.67 Mbit/s and for 4444 byte data packets, the required bandwidth would be 297.16 Mbit/s. Since the

expectation is for large packets, even after removal of the appropriate FDDI and TCP/IP overheads, the VMEbus would be required to utilize 92.9 percent of the available bandwidth for data transfer leaving approximately 7 percent of the bus rate available for bus control and interface management.

Since the bus request and bus acknowledgement require 70 ns minimum between bus accesses, which are required at least every 256 bytes, the bus availability for control and interface management would be reduced by another 1.1 percent for the effective loss due to the 70 ns interruption when VMEbus is operating at its maximum transfer rate. In the best case the VMEbus will have less than 6 percent of the available connection bandwidth for control of the three VMEbus boards comprising the FDDI network interface when operating at the FDDI data rate, connected to VMEbus, and performing at the highest performance levels defined by the VMEbus specifications. This limited amount of bandwidth available for control issues is too thin a margin to guarantee the required throughput capability of the system when VMEbus is used between the block connection points of VMEbus.

Hardware Connection

An option which should be examined is using the VMEbus as a transfer mechanism for the external data transfer required in the high speed RISC system. There are boards

specifically manufactured to perform this type of duty. It would require the MAU memory and host memory boards to act as slave stations and a data transporter board, such as the Ironics IV-3272 to receive requests for transfer directly from the microprocessor. The transfer mechanism could be equipped with a checksum generator. The transporter board, however, only has a throughput bandwidth of 5 million transfers per second, because it uses 5 million for input and 5 million for output. This gives a maximum transfer rate of 158 Mbit/s after the VMEbus access grants are generated. Therefore, this transporter board could act as the external transfer mechanism discussed in Section 3.1. Note that if this were the external transfer mechanism, the transfer between the two memory boards would use such a large percentage of the VMEbus bandwidth that no other boards operating at 100 Mbit/s could be allowed into the system. If a standard bus is used, it would be best to use it for the computer interface rather than as an external transfer mechanism.

Looking at Figure 3.3.4, it appears that using VMEbus for transfer between system components will increase the amount of memory required by a system. Note that in Figure 3.3.4, a buffer is required on each side of the VMEbus interface locations. These buffers are required

1) to allow storage of data while the VMEbus may be

performing data transfer for one of the other interfaces, or

2) to allow storage of data at the interface to or from the microprocessor because the microprocessor cannot keep up with the burst speed of 320 Mbit/s of the VMEbus needed to allow all interfaces to function properly.

Due to the high speed required for all the interfaces, VMEbus will be pressed towards its maximum data transfer rate and must minimize the control required for bus arbitration. In minimizing bus control, VMEbus will use the largest transfers possible, just like the system does over the FDDI network. In doing so, the waiting time for the buffers between bus accesses will be as long as reasonable and will cause all interfaces to have some storage. The major buffering points in the direct memory transfer will not be decreased. The buffers will still require enough storage to hold the complete message and room for storage of arriving data. VMEbus adds an additional buffering area when associated with the FDDI front end processor.

In a similar argument, the VMEbus buffering system adds latency to the system when large packets are transferred. It can be argued that the latency of VMEbus can be masked by passing data across the boundaries while the system is working on data already in the buffers. When the buffers are not

full, the microprocessor is waiting for data to arrive and the additional transfer required by VMEbus and the associated latency will be seen by the system. Although the design under consideration is intended to accommodate worst case scenarios, the majority of transfers will not be worst case and therefore the VMEbus will add latency when the microprocessor and TCP/IP overhead are not limiting the throughput.

Overhead for Controlling VMEbus

Figure 3.3.5 shows a diagram of the control process for a data transfer to occur. From this process, the same information is required for the VMEbus transfer as was required for the external hardware transfer discussed in Section 3.1 including starting location, destination location and data transfer length. The one noticeable exception is that the transfer size of data elements is limited to 256 bytes. This effectively requires control of the interface to occur every 256 bytes instead of once for the complete data packet. There is a provision in the VMEbus standard to minimize this delay (see VMEbus C.1 specification Rule 2.12, Observation 2.20) but some delay, which is at least 70 ns in duration, must always exist at each 256 byte boundary. As noted by the specification the delay has a 70 ns minimum, but it can exist for a period not to exceed 2T where T is the "timeout value in microseconds" [37].

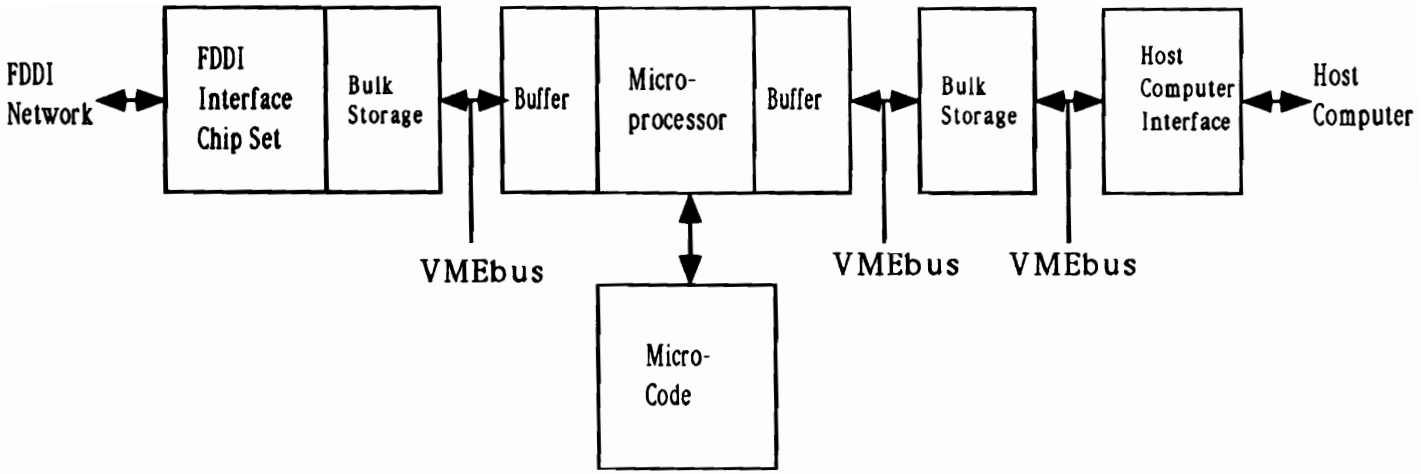


Figure 3.3.4: MAU interface diagram showing the maximum use of VMEbus in an FDDI front end processor.

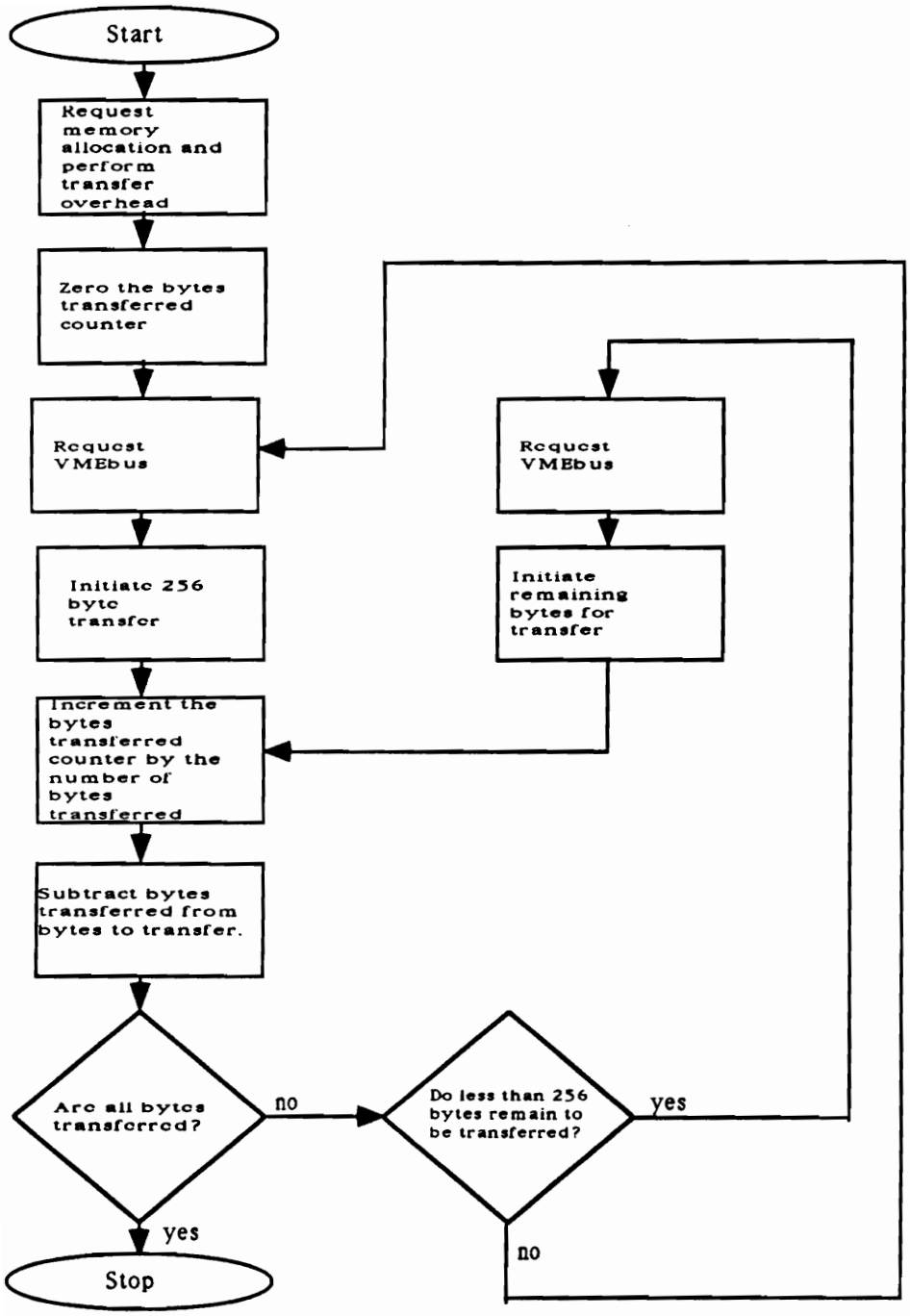


Figure 3.3.5: Diagram of the control process for extended data transfer to occur. Note that control of the VMEbus must be gained for each 256 byte data increment or portion thereof.

Because of the 256 byte boundary delay, the overhead associated with transfer when VMEbus is used increases with the packet length. If we assume that the access rate of the memories interfacing the VMEbus boards to be the limiting factors, and their associated microprocessor clock rates fully utilize the memory speed, the actual transfers of VMEbus can be as fast as those used in the external hardware transfer, but will have two types of overhead associated with the transfer of information. The first is the overhead associated with the instructions for control transfer (ICX). ICX was defined in Section 3.1 and its effects on the throughput were graphed for several values and architectures in Section 3.1. The second type of transfer overhead is that associated with exceeding the transfer limit of 256 bytes. This variable will cause the VMEbus transfer to take more time than the equivalent direct data transfer, even without considerations for conflicting transfers which will add additional delay and limit throughput. With the addition of the 256 byte transfer boundary, the equation for throughput now includes a new variable, ICB, which is the delay for control of block transfers expressed in equivalent CISC instructions. ICB is a fixed increase for each additional 256 byte segment begun. Therefore, the new equation for data throughput is

Number of Bits/sec Trans. =

$$\frac{8 * MCR * BPP}{\left[IMO + ICX + \text{Integer}\left(\frac{BPP}{256}\right) * ICB\right] * CPI * ECI + \frac{BPP * CPX}{BPX}}$$

where Integer rounds up the value of the argument following.

It is important to note that the values for bytes per transfer, BPX, for any VMEbus configuration are limited to 1, 2 or 4 bytes per transfer. Therefore, as in the earlier external transfer discussion, of Section 3.1.4, external transfers using CISC-based microprocessor cannot achieve the performance available in the CISC internal transfer unless the transfer size is greater than four bytes. It is, however, appropriate to analyze the throughput capabilities of a RISC-based system using VMEbus transfers. The throughput values for several RISC-based VMEbus transfer systems are compared to the RISC-based system with dedicated hardware transfers and the RISC-based system with internal transfers. Note the effects in Table 3.3.1 that occur when data transfer sizes are changed and as values of ICB increase. The results of a comparison of the RISC transfer systems are shown in Table 3.3.1. All values but ICB are those found in Table 3.1.2. Note that this equation assumes the availability of the bus is 100 percent.

Processor	RISC based	RISC based	RISC based	RISC based	RISC based	RISC based
Transfer Method	VME 16-bit ICB=20	VME 32-bit ICB=10	VME 32-bit ICB=50	VME 32-bit ICB=100	Internal 64-bit String mv	External 32-bit
Packet Size (bytes)	T H R O U G H P U T (Megabits/sec)					
4	0.68	0.68	0.68	0.68	0.74	0.68
100	13.8	15.2	15.2	15.2	15.9	15.2
500	38.5	53.0	51.0	48.8	51.2	53.5
1000	50.3	76.4	70.6	64.4	70.9	78.0
2000	57.3	98.1	87.3	76.7	87.8	101.2
3000	60.6	108.3	94.7	81.9	95.4	112.4
4444	62.9	116.1	100.0	85.3	101.0	121.0

Table 3.3.1: Maximum throughput estimates for VMEbus transfers assuming 100 percent bus availability with various instruction rates to allow VMEbus transfers in excess of the 256 byte boundary of VME. The parameters are those from Table 3.1.2.

As was shown in Table 3.3.1, the use of VMEbus and the associated overhead decreases the throughput of the system when compared to RISC with external transfer. This decreased throughput is in addition to the increased latency discussed earlier in this section. If the limit of 256 bytes for the VMEbus transfer causes the system to use additional instructions in processing the data, this increase in overhead, shown as ICB, can prevent the 18 MHz system from achieving the maximum throughput of 100 Mbit/s. Therefore, the use of VMEbus has the potential for seriously reducing throughput and definitely increases latency. An additional disadvantage of the VMEbus approach is the increase in the number of buffers.

Conclusion

VMEbus allows a reasonable interface for a limited number of board interfaces, one or two, if each board is a full bandwidth interface port for FDDI. Because 100 percent of the VMEbus bandwidth is not available to every bus interface unit at the instant needed, extra buffers are needed to store data awaiting transfer over the bus or being received over the bus. VMEbus used as one access link in the system, mainly between the host computer memory and the actual host, may have some advantages because of the flexibility associated with a standard such as VMEbus. There will be additional latency

associated with the use of the bus due to transfer set-up and additional transfers. This latency can often be hidden when data is arriving more quickly than it can be processed. When data is arriving and being processed without delay, i.e. when large messages are predominant, the latency of VMEbus will be noticeable and will be a function of the number of units using the bus. Use of a single VMEbus interface will not be as fast as a system without the bus, but should have minimal effects on the throughput of the FDDI system.

In general, custom interfaces will better meet the needs of FDDI interface devices and in providing high performance needed to utilize the bandwidths of high speed networks. For a single transfer element, VMEbus will provide a reasonable connection. As a standard for FDDI transfers, it will not meet the need. The customized memory interfaces shown in Section 3.1 best meet the FDDI network requirements.

Chapter 4

System Efficiency Calculations for Specific Processor, Data Access and Data Segmentation

Chapter 4 considers the specific attributes of a Star VP-Series computer and evaluates the system requirements and performance characteristics required by a proposed FDDI interface unit. The chapter then uses information from Chapter 3 to demonstrate the options that are available for the design. The chapter examines throughput and memory requirements over a limited range of options that may be available, such as varying overhead and clock rates. Final results and performance characteristics are discussed.

4.1 Data Rate Requirements of the Interface Between FDDI and the Star Technologies VP-Series Computer

The Star VP-Series computer is normally attached to a host computer as shown in Figure 4.1.1 through a specialized interface unit or through an interface card based on a standard backplane. The system in use at Virginia Polytechnic Institute and State University uses the VMEbus as a connection medium for the interface board. The board performs all accesses from the VMEbus to receive data and commands and to transmit results. The maximum burst rate of the Star VMEbus interface board is the same as the average transfer rate, 12.5

Mbyte/s [17]. In this configuration, the VMEbus interface board uses the full VMEbus bandwidth to process the 100 Mbit/s which are potentially available from the network. Of the 100 Mbit/s of data that can be transferred by the Star interface, the vector processor can accept 80 Mbit/s on a sustained rate. Any data arriving at the interface unit at a rate greater than 80 Mbit/s must be bufferred before it can be sent to the vector processor over the VMEbus interface.

4.2 Current VP-Series Communications Definition and Limits

The data requirements for a key Star-based application are used as a baseline for review of the hardware requirements for the Star FDDI interface unit. This application utilizes a small number, under 20, of small packets to set-up the computer and then passes one million 32-bit words of data for processing [17]. This equates to twenty small transfers followed by 901 large transfers, giving the probability of small transfers to be less than 0.025. Because the data distribution is deterministic, rather than random, it is possible to determine the required system transactions exactly. Were this to be done theoretically, the blocking probability would have to be made low enough for the case of

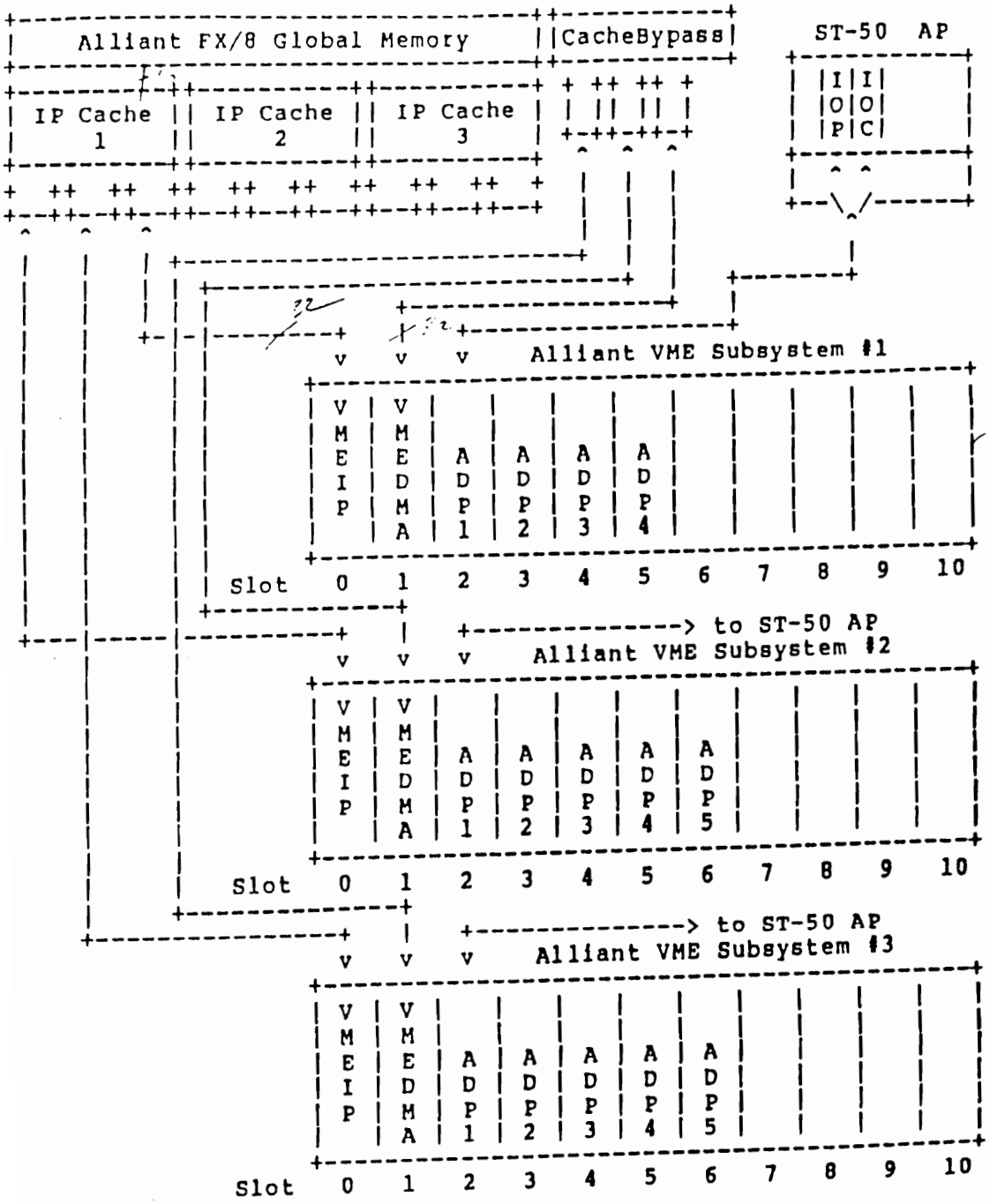


Figure 4.1.1: Diagram of the Star VP Series computer shown connected to a host computer through a VMEbus based interface, from "Star VME Interface Functional Specification", December, 15, 1987 [39].

20 of the small packets in sequence or $9 * 10^{-33}$. The direct solution in this specific case is easier to solve although the basis of the memory sizing, using the smallest packets first, is the same as was developed in the design analysis of Section 3.2. This analysis begins by determining the average data transfer size and calculating the minimum required clock rate.

The general communication between the current Star computer and the host computer uses a 16 byte header followed by the data to be transferred and a checksum. The structure is similar to the TCP/IP headers and the original objective was to provide TCP/IP as the transport and network protocols. The data structures examined show the average short transfer packet to be 12 bytes long [40]. In addition to the processing of TCP/IP for the header, a new header is needed to prepare the data for transport over the Star interface. This header will in the worst case have the same overhead cost associated with it as the reduced TCP/IP [41] discussed in Chapter 3. This gives a new value for the term IMO of 1200 instructions. Allowing for the microprocessor to keep up with the data as defined, the average packet size will be

$$\begin{aligned} \text{Avg. Packet Size} &= p*S1 + (1 - p)*S2 \\ &= 0.025 * 12 + (1 - 0.025) * 4444 \\ &= 4333 \text{ bytes} \end{aligned}$$

Note the average number of bytes for the calculations does not need to be a multiple of 32-bits, although the actual transfers are on 32-bit boundaries, for the transfer to operate as predicted.

4.3 Selection of Processor, Data Access and Data Segmentation Based on Requirements

The types of processors and data transfer rates required by the system will be discussed below. To determine the required transfer data size, first notice that the data used by Star Technologies uses 32-bit boundaries. In addition, the basis for system transfer is TCP/IP or the Star internal transfer protocol. It is known that TCP/IP requires transfers to be bounded on 16-bit values. The Star transfer assumes all transfers are made on 32-bit boundaries.

The minimum allowable clock rate to keep up with the data rate over the long term is given by the equation

$$MCR = \frac{BR}{8 * (BPP + OHB)} * (IMO * CPI * ECI + \frac{BPP * CPX}{BPX}) \quad ,$$

where the terms are defined in Section 3.1. The new value of IMO is equal to 1200 instructions. For a CISC based machine

using the parameters in Table 3.1.1, except using IMO = 1200, the required CISC-based microprocessor clock rate to utilize the complete bandwidth becomes

$$MCR = \frac{100,000,000}{8 * (4333 + 56)} * (1200 * 2.5 * 1 + \frac{4333 * 4}{4}) ,$$

$$= 20.9 \text{ MHz}$$

For a RISC-based processor using internal 64-bit based transfers, as was discussed in Section 3.1, the equivalent required clock rate is

$$MCR = \frac{100,000,000}{8 * (4333 + 56)} * (1200 * 1 * 1.3 + \frac{4333 * 5}{4}) ,$$

$$= 19.9 \text{ MHz}.$$

These values are the minimum clock rates at which the systems will operate and complete the transfers in the same amount of time that the data arrived. If the small packets did not come at the beginning of the data packet sequence, this clock rate would only guarantee that the buffer length would get no longer than the maximum caused by the delay associated with the small packets arriving sequentially.

The actual expected data transfer rate is calculated by the following equation with the terms defined in Section 3.1.

$$\text{Required Data Transfer Rate} = \frac{BPP * BR}{BPP + OHB}$$

$$= \frac{4333 * 100{,}000{,}000}{4333 + 56}$$

$$= 98.7 \text{ Mbit/s}$$

The total memory required by each of the systems defined above can be found by determining the amount of data which arrives during the processing of small packets. To do this, first determine the amount of time required by the microprocessors to acknowledge and transfer the smallest data packet with the minimum acceptable clock rate for the system having a probability of small packets of 0.025. The amount of time to process each small packet equals

$$\text{Processing time/small packet} = \frac{IMO * CPI * ECI + \frac{(BPP * CPX)}{BPX}}{MCR}$$

$$\text{for CISC} = \frac{1200 * 2.5 * 1 + \frac{(12 * 4)}{4}}{20{,}900{,}000}$$

$$= 144 \text{ µs per small packet}$$

$$\text{for RISC} = \frac{1200 * 1 * 1.3 + \frac{(12 * 5)}{4}}{19{,}900{,}000}$$

$$= 79 \text{ µs per small packet}$$

Note that the amount of time to process the small packet

using the RISC microprocessor is actually less than the CISC microprocessor because the protocol processing dominates the data transfer. The total delay for small packet processing is determined from the amount of time for each packet to be processed.

Total Small Packet Proc. Time

= Proc. time/small packet * # packets

for CISC = 144 μs pr small packet * 20 packets

= 2880 μs

for RISC = 79 μs pr small packet * 20 packets

= 1580 μs

This is the amount of time required for all of the small packets except the first packet to arrive plus the amount of time to process all of the small packets. The amount of memory required to store the packets arriving while the small packets are being processed is the worst case memory requirement as was shown in Section 3.2.4. In addition to the data arriving while the small packets are arriving and being processed, the memory must have an additional 4500 bytes available for the packet arriving after the last small packet is processed. The 4500 byte value is used because this is the largest packet that could be received by the system.

Therefore, the total memory required at the incoming MAU

memory by the system is given by the equation,

Total Memory = Total data arriving during proc. + 4500 bytes

$$\text{for CISC} = \frac{2880\ \mu s * 100{,}000{,}000 \text{bit/s}}{8 \text{bit/byte}} + 4500$$

$$= 40{,}500 \text{ bytes}$$

$$= 39.6 \text{ Kbytes}$$

$$\text{for RISC} = \frac{1580\ \mu s * 100{,}000{,}000 \text{bit/s}}{8 \text{bit/byte}} + 4500$$

$$= 24{,}250 \text{ bytes}$$

$$= 23.7 \text{ kbytes}$$

Note that in all cases where the small packets are truly "small", i.e. protocol processing dominates, the RISC-based architecture results in smaller memory requirements. In addition, the equations above indicate the smallest processing rate for which the memory requirements do not continue to grow without bound. If the microprocessor clock rate is above these rates, the amount of memory required by the systems decreases. An example of these memory reductions is shown in Table 4.2.1.

The host computer interface memory requires two size data buffer sizes, one 28 bytes long and the other 4460 bytes long

Table 4.2.1: Required Memory vs. Microprocessor Clock Rate for Fixed Overhead

for CISC microprocessor

Clock Rate (MHz)	Processing Time per Packet (µs)	Required Memory (in kbytes)
20.9	144.0	39.6
25	120.5	33.9
33	91.3	26.7
40	75.3	22.8
50	60.2	19.1
60	50.2	16.7
70	43.0	15.0

for RISC microprocessor

Clock Rate (MHz)	Processing Time per Packet (µs)	Required Memory (in kbytes)
19.9	79.0	23.7
20.9	75.4	22.8
25	63.0	19.8
33	47.7	16.1
40	39.4	14.1
50	31.5	12.1
60	26.3	10.9
70	22.5	9.9

Processing time per 12 byte data packet and the related maximum memory requirements for the Star Technologies, Inc. FDDI interface assuming packet instruction overhead of 1200 CISC equivalent instructions and other parameters shown in Table 3.1.1. Note the improved RISC-based performance due to the large overhead.

(the data lengths plus 16 bytes of overhead). The removal of packets at the host computer interface is at the rate of 80 Mbit/s. For the small packets, using the fastest processing calculated, 22.5 μs would present data to the memory at a rate of 28 bytes per 22.5 μs, or approximately 10 Mbit/s. Therefore, only two small reserved memory locations are needed in the outgoing host computer memory interface, one for the packet in transit from memory over the VMEbus and another being queued for transfer. For large packets, this is not the case. The faster a microprocessor is operating, the faster it is making the data available to the system. For a period of time, the data is being backed-up in the incoming MAU memory while the microprocessor is processing the small packets. Once the small packets have been processed, the large packets are processed at the maximum rate until the queues are empty. However, the buffer in the outgoing host computer interface is limited by the removal of data from the memory at a constant rate of 80 Mbit/s. The large packets arrive from the MAU interface at a faster rate and, therefore, the memory reaches its capacity limit when the transfer is complete. This buffer length is equal to the number of bytes arriving in the queue during the data transfer period minus the number of packets processed during that period plus one additional packet to take care of overlap of periods when one packet is beginning to fill arrive at the buffer before another has departed. The amount of time during which the 4

Mbit of data was received from the network is given by the equation

$$\text{Receive time} = \frac{\text{(Number of bit + bits of headers)}}{\text{data rate}}$$

$$= \frac{4{,}000{,}000 + [\text{Integer}(4{,}000{,}000/4444) + 1] * 56}{100{,}000{,}000}$$

$$= 0.0405 \text{ Seconds}$$

During the receive time, the host computer can receive
Host Reception = 3,240,364 bits.

For the 4 million data bits, the Star interface would handle the 4 million data bits plus the associated overhead. The total number of bits to be transferred across the Star interface is given by the equation

$$\text{Tot. Star I/F bits} = 4{,}000{,}000 + [\text{Integer}(4{,}000{,}000/4444) + 1] * 56$$

$$= 4{,}014{,}416$$

This leaves 774,052 bits to be held in large buffers. This equates to 22 large buffers plus one additional spare for overflow. The outgoing host computer interface buffer then requires about 101 Kbytes of storage regardless of which processor or clock rate is used. Similarly, the outgoing MAU memory is more dependent on the time-out value than it is on the clock rate as long as the microprocessor is capable of meeting the average output data rate of 100 Mbit/s.

4.4 Maximum System Performance Calculations

Based on earlier evaluations, the system performance for the application is achieved using the RISC-based architecture with 64-bit move commands used for all transfers. The final 32-bit move, if one exists, which is not on a 64-bit boundary would be performed using a slightly longer 32-bit transfer, on the order of one additional clock per 32-bit instance. A 33 MHz system meets all system requirements and requires the incoming MAU memory to be no more than 9.9 Kbytes in length. The worst case processing delay is 954 μs. It takes large packets 360 μs to arrive, but only 225.6 μs to be processed. Therefore, after approximately 8 large packets are processed, the system is awaiting new packets, so the normal system delay is the arrival time of a complete packet plus its processing time, or 586 μs. The outgoing host computer interface buffer size is independent of microprocessor clock rate since the host computer interface can pass small packets faster than they are processed and cannot pass large packets as fast as they are processed. Therefore, the outgoing MAU memory requires two small memory storage locations and 23 large memory storage locations. The data going from the host computer to the network was not defined by Star, but the outgoing MAU memory is a function of the time-out period for TCP and the incoming host computer interface memory is defined just like the incoming MAU memory except the data mix and

bandwidth are defined by the host computer interface.

For the system defined by Star Technologies, it has been shown that a RISC or CISC-based system performing at 19.9 MHz and 20.9 MHz respectively are the minimum microprocessor clock rates for which the system can operate. If the overhead associated with the operating system is as large as the system interface overhead, the minimum full speed operations are found to be 24.3 and 29.4 MHz respectively. It was shown that the incoming MAU memory is significantly affected by the microprocessor clock rate, but that the outgoing MAU memory is mostly affected by the memory time-out. The outgoing host computer interface memory is affected by the difference between the network data rate and the host computer interface data rate. In most cases, since the difference between small and large packet sizes is considerable, the small packets will be easily handled by the host computer interface with minimal buffering while the larger packets will require buffering to compensate for bus availability and the difference between network speeds and the host computer interface speed.

This chapter has demonstrated the application of the equations and concepts presented in Chapter 3 and how they are used in a practical design situation where the desire is to provide an FDDI network interface which handles TCP/IP and a proprietary interface protocol.

Chapter 5

Summary

In this dissertation, equations for predicting throughput, in terms of bits and packets per unit time have been developed. Equations for determining the maximum buffer sizes needed in high speed networks have been discussed. Both types of equations have shown the relation between packet size, overhead and throughput. The use of data transfer size and transfer methodology have been discussed. In addition, it has been explained that to utilize FDDI data rates with a common protocol, such as TCP/IP, requires a front-end processor to relieve the computer of the protocol processing requirements of a high speed networks.

The areas analyzes by this dissertation are discussed in Chapter 3 and the summary is presented below.

The throughput of the FDDI interface is largely a function of network protocol. Efficient implementations of TCP/IP require approximately 600 instructions for receiving and acknowledging a packet or for sending a packet and receiving its acknowledgement. The reduction in overhead associated with new TCP/IP implementations is a major factor in allowing high speed networking to be performed using microprocessor-based transfers. In addition, the throughput

equations have demonstrated the importance of minimizing the number of clock cycles required per data unit transferred in meeting the throughput requirements of high speed networks. As demonstrated for the RISC-based architecture using 32-bit internal transfers, high speed transfer rates cannot be accepted unless clock/transfer ratios are below two clocks per byte transferred. This type of high speed transfer can be accomplished by increasing the data transfer width or by decreasing the clocks per transfer cycle. In the CISC and RISC-based systems examined using high speed 32-bit transfers provided a reasonable trade-off to meet the data rates required and the transfer widths which were most appropriate.

A major accomplishment was the development of a memory sizing algorithm. The proof that the ratio of the number of small packets to the total number of packets for any given blocking probability decreases as the number of total packets increases is a key to the sizing algorithm. This equation was proven for all cases where $k/n>p$ and $npq >>1$, which is reasonable if the blocking probability is small. Since the design is for a high probability of successes, the blocking number should be as small as possible. It would be desirable to have the blocking number reasonably close to the BER of the communication medium if possible.

For any given packet distribution there are four

functions which determine or define the operation of the system. They are microprocessor speed, blocking probability, memory size and packet length distribution. All affect the ratio of small to large packets which can be successfully transferred by the system. Since small packets require a larger average number of clock cycles for processing per byte transferred than large packets, the ability to accommodate more small packets improves the performance of the system. The microprocessor must always be able to service the average data arrivals. If the microprocessor cannot meet the average arrivals of the system, then all buffer sizes will eventually overflow and the interface will fail for any blocking probability.

By using reasonable constraints on the microprocessor speed and memory size, the blocking rate can be determined for any percentage of small packets. Likewise, given any parameters, the range of acceptable values of the other numbers may be determined. These trade-offs can be used for designing a system, for predicting system performance or for analyzing system changes.

The discussion of VMEbus attributes shows why FDDI needs direct connection techniques to the host computer to ease the design requirements of the system. For the majority of VMEbus based machines encountered during the research, the Star

system using a 12.5 Mbyte/s transfer had about the average throughput seen. Using VMEbus as a common transfer scheme required almost the maximum VMEbus specified bandwidth for the data transfers. Since there has been a constant acknowledgement throughout the dissertation that other controls are needed, suggesting that transfer over VMEbus does not require external control to exist would be inconsistent. When pushing the limits of available microprocessors, even if dedicated, it is advantageous to avoid implementations whose maximum capabilities are required to meet the minimum needs of the new system. For these reasons VMEbus may be used as an interface mechanism for a limited number of data transfers, but should not be used as the required data transfer mechanism.

The dissertation has shown that current technology can meet the needs of FDDI networking systems and has developed equations to show the relationship of the items which make-up a network interface. The equations can be adjusted to account for the large variables available with design variables including, microprocessor type, microprocessor clock rate, software, hardware transfer options, and buffer sizes. The equations can be used to predict performance, verify test results or evaluate performance characteristics and how specific system changes affect performance.

5.1 Future Implications

The figures on throughput and blocking values have defined the upper bounds of throughput for the microprocessor-based architectures examined. These examples show the data transfer limit to be approximately 264 Mbit/s, for an interface using a 33 MHz clock and transferring 4 bytes every 4 clock cycles. This transfer rate assumes the protocol overhead is so small a portion of the processor load that it can be ignored. For higher performance in large packets, either the clock rate must increase or the data transfer width must increase. The increase in data transfer width would be difficult to accomplish unless the protocol and the transfer medium both support the additional width. By realizing that most if not all machines utilizing the higher bandwidth capabilities use 32-bit microprocessors, it is not unreasonable to expect that future networks will utilize 32-bit transfers.

Already on the horizon, groups are beginning to discuss gigabit, multi-gigabit and terabit networks. These networks will, like FDDI, have a specific class of machines associated with the network which will be expected to accommodate the complete system bandwidth. These systems will require several advances to occur to achieve the bandwidths suggested.

Firstly, protocol overhead must be reduced to a smaller portion of the system throughput requirements. This was successfully done by Cray Computer and was discussed in Chapter 3. The transfer rate achieved by Cray was accomplished by increasing the transfer packet size to a very large packet. Future systems will need to use large packets and also reduce the overhead associated with protocols. These changes will increase the amount of data which can be processed by each microprocessor. Multiple microprocessors may be used to further increase the throughput but the microprocessor parallelism control needs to be simplified, or the control will increase the microprocessor overhead and further decrease the performance of each node. This problem is common today in multiprocessor machines which gain in performance until the control becomes so complicated that adding compute elements does not increase the computer throughput.

Secondly, to achieve higher performance, data transfer widths need to be increased. As shown in Section 3.1, increases in the data transfer width for a common clock rate allows increases in the number of bits transferred per period of time. The use of 32-bit transfers should be a minimum. Machines needing full access to the networks with gigabit per second bandwidth and above, will not be using 8-bit

microprocessors. In fact, they will most likely be using 64-bit or wider data elements. A storage philosophy for high transfer rate networks is using wider and wider storage paths. The 128 or 256 bit wide storage system can provide high throughput with a 33 MHz clock; even using four clocks per storage cycle the system could achieve 2.1 Gbit/s capability ignoring overhead. The transfer width would also have the option discussed in Section 3.2 of being able to add bits to fill a data width faster than the microprocessors could determine the proper data transfer increments. Data widths cannot be as limited as FDDI makes them if very high speeds are desired.

Thirdly, multiple processing elements will be needed. Since protocol is a necessary part of data transfer, it must be maintained. The new TCP/IP has made significant strides in making high speed networks such as FDDI available for the user. It is doubtful that another large reduction in execution time is possible for TCP/IP. Other protocols such as XTP suggest that some relief will occur, but it is uncertain how well faster protocols will work and it assumes that every machine on the network will accept the protocol. Even if all these problems are solved, small packets will continue to exist and their processing-to-data ratio will still cause the systems problems with real-time processing of small packets. Multiple processing elements, whether

microprocessor-based or hardware based, will provide support for new systems. As mentioned above, using multiple processing units can provide relief, but the number that can be used effectively is limited.

Lastly, to improve throughput, higher speed devices can be used. The use of higher speed devices is a difficult problem to resolve. Direct hardware solutions may provide throughput improvements, provided the protocols can be implemented in hardware in an efficient manner. The use of ECL and GaAs devices can provide an improvement in throughput but at increased cost and design complexity. Before implementing devices operating at clock rates over 33 MHz, it would be wise to determine if the first three methods can provide adequate solutions.

References

1. J. Filipiak, "Structured Systems Analysis Methodology for Design of an ATM Network", IEEE Journel on Selected Areas in Communications, Vol. 7, No. 8, pp.1263 - 1273, October 1989.

2. W. Burr and F. Ross, "The Fiber Distributed Data Interface: A proposal for a Standard 100Mbit/s Fiber Optic Token Ring Network", Proc. of FOC/LAN, 1984. p254.

3. Coulouris and Dollimore, Distributed Systems, Concepts and Design, Addison Wesley Publishing Company, Wokingham, England, 1988, p. 58.

4. F. Ross, "FDDI - A Tutorial", IEEE Communications Magazine, Vol. 24, No. 5, May 1986,

5. W. E. Burr, "The FDDI Optical Data Link", IEEE Communications Magazine, Vol.24, No. 5, May 1986.

6. Burr and Zuqiu, "An Overview of FDDI", Fiber Optics Magazine, Handbook and Buyers Guide, 1989.

7. R. Jain, "Error Characteristics of Fiber Distributed Data Interface", IEEE Transactions on Communications, Volume 38, No. 7, p 1245, July, 1990.

8. W. Blood, Jr., MECL System Design Handbook, Motorola Incorporated, Phoenix, AZ, 1983. p. 34.

9. W. B. Jones, Jr., Introduction to Optical Communication Systems, Holt, Rinehart and Winston, Inc., 1988, p. 319.

10. Oliver and Moore, "FDDI - A Federal Government LAN Solution", November 1988.

11. X3T9.5, "FDDI Media Access Control (MAC-2) (Maintenance Revision), Working Draft Proposed American National Standard. Number X3T9.5/88-139 Revision 2, June 1, 1989

12. Intel Corporation, Microprocessor and Peripheral Handbook, Volume 1- Microprocessor, Santa Clara, CA, 1989, pp. 4-110 - 4-123.

13. F. E. Ross, "An Overview of FDDI: Fiber Distributed Data Interface" IEEE Journal on Selected Areas in Communications Vol. 7, No. 7, pp. 1043 - 1051, September 1989.

14. L. Green, "High Performance LAN Applications", Proc. of Wescon 1984, Session 2, Paper 3, pp. 1-5.

15. J. F. Shoch and J. A. Hupp, "Measured Performance of an Ethernet Local Network", Communications of the ACM, Vol. 12, No. 12, pp. 711-721, December 1980.

16. C. Partridge, "How Slow is One Gigabit Per Second", Computer Communication Review, Vol. 20, No. 1, January 1990.

17. E. Martello, Engineering Manager, Star Technologies, Personal Communications, July 30 - August 6, 1990.

18. I. Chlamtac and W. Franta, "Rationale, Directions, and Issues Surrounding High Speed Networks", Proceedings of the IEEE, Vol. 78, No. 1, January 1, 1990.

19. S. Nogami and S. Sumita, "Approximate Analysis of Memory Contention in a Multiprocessor System with a Single Bus", Electronics and Communications in Japan, Part 1: Communications Vol. 72, No.8 August 1989

20. L. Green "Implementation of FDDI: a 100 MBIT token ring", Proc. of Localnet, 1986, pp. 151-160.

21. H. Shimizu, M. Mera, and H. Tani, "Packet Communication Protocol for Image Services on a High-Speed Multimedia LAN", IEEE Journal on Selected Areas in Communications, Vol. 7, No. 5, pp. 782 - 788, June 1989 .

22. J. Buchanan, CMOS/TTL Digital Systems Design, McGraw-Hill Publishing Company, 1990, pp. 120-121.

23. R. Cohn, T. Gross, M. Lam and P. Tseng, "Architecture and Compiler Trade-offs for a Long Instruction Word Microprocessor", Third International Conference on Architectural Support Programming for Programming Languages and Operating Systems, Association for Computing Machinery, Boston, Mass April 3-6, 1989.

24. Cypress Semiconductor, Ross Technology Subsidiary, SPARC RISC User's Guide, San Jose, CA. 2nd Edition, February 1990.

25. D. Clark, V. Jacobson, J. Romkey and H. Salwen, "An Analysis of TCP Processing Overhead", IEEE Communications Magazine, Vol.27, No. 6, pp. 23-29, June 1989.

26. Bruce McClure, President, Synernetics Inc., Personal Communication, April 25, 1990, St. Petersburg, Florida.

27. D. A. Borman, "Implementing TCP/IP on a Cray computer", Computer Communication Review, Vol. 19, No. 2, April 1989, pg. 12.

28. Cypress Semiconductor, Ross Technology Subsidiary, Cypress RISC Seminar Notebook, San Jose, CA, 1989, p.95

29. D. Tolley, "Information Management for Control Systems", Drive Systems Division, General Electric Company, Salem, VA, 1984.

30. J. Milton and J. Arnold, Probability and Statistics in the Engineering and Computer Sciences, McGraw-Hill Publishing Co., New York, 1986, p. 205.

31. FDDI Station Management (SMT), Draft Proposed American National Standard, Rev. 5.1, September 5, 1989, Section 4.1.1.1, SMT Frame Contents.

32. Information Sciences Institute, University of Southern California, "Transmission Control Protocol, DARPA Internet Program Protocol Specification", September 1981, p. 16.

33. C. Vogt, "Buffer-Based Method for Storage Allocation in an Object-Oriented System", IEEE Transactions on Computers, Vol. 39, No. 3, pp. 375 - 383, March 1990.

34. W. Stallings, Data and Computer Communications, 2nd Edition, MacMillan Publishing, New York, 1988.

35. A. Papoulis, Probability, Random Variables, and Stochastic Processes, 2nd Edition, McGraw-Hill, New York, 1984 p. 217

36. J. Daigle, Ph.D., Personal Communication, Newcastle, VA, July, 19, 1990.

37. Motorola, The VMEbus Specification, Revision C1, Tempe, AZ, October 1985.

38. Ironics Incorporated, IV-3272 VMEbus Full Speed Data Transporter User's Manual, Ithica, New York, 1988.

39. "Star VME Interface Functional Specification", Star Technologies, Sterling, Virginia, December, 15, 1987

40. Star Technologies, Incorporated, "Array Processor Monitor (APM) External Reference Specification, 60000001", Revision I, Release 4.0, Sterling, VA, October 6, 1986.

41. Scott Thomas, Personal Communications, Blacksburg, Virginia, August 22, 1990.

```
C*************************************************************
C
C This program is used to test the projected blocking
C probabilities discussed in Dan B. Tolley's doctoral
C dissertation.  The test will use projected memory sizes
C and long-term probabilities as well as random number
C generation to show memory overloads.  Note that at the
C beginning of all memory read routines, the required
C buffer length is estimated to be 4500 bytes and is sized
C properly only after the arriving packet has
C entered the buffer.
C
C
C*************************************************************
C
C Program written by Dan B. Tolley
C
C
C*************************************************************
C
C
C DEFINE THE PARAMETERS
C
C DEFINE  SIZE       = SIZE OF PACKET UNDER EXAMINATION
C          TIME       = TIME SINCE LAST BUFFER WAS PROCESSED.

C                       RESET TO ZERO IF NO PACKETS IN BUFFER.
C         OFLOWPT    = VALUE OF MEMORY AT WHICH OVERFLOW
C                       OCCURS
C         OFLOWCT    = NUMBER OF TIMES IN SIMULATION OVERFLOW
C                       HAS OCCURRED
C         PKTCNT     = NUMBER OF PACKETS WHICH HAVE BEEN INPUT
C                       TO BUFFER
C         PKTOFL     = NUMBER OF PACKETS EXCEEDING INBUF
C                       STORAGE
C         PKTRM      = NUMBER OF PACKETS REMAINING IN BUFFER
C         BUFWAIT    = AMOUNT OF TIME REQUIRED TO PROCESS
C                       BUFFER AT THE BOTTOM OF THE QUEUE
C         PTR        = POINTS TO THE LINE INSIDE THE INBUF
C                       WHICH IS BEING ACTED UPON.
C         INBUF      = INPUT BUFFER.  COMES IN TOP, REMOVED
C                       FROM BOTTOM
C         MAXBUF     = MAXIMUM BYTES OF MEMORY AVAIABLE
C         TEMPBUF    = ESTIMATED MEMORY REQUIREMENTS SHOULD A
C                       LARGE PACKET ARRIVE NEXT.
C         ERATE      = ERROR RATE
```

```
C          APCKCNT    = INTEGER TOTAL PACKET COUNT
C            OH           = BYTES OF OVERHEAD ASSOCIATED WITH
PROTOCOL
C                         AND TRANSFER MEDIUM
C          BR         = BYTE RATE OF DATA.  NOTE BYTE RATE
C          RCTOFL     = MOST RECENT PACKET CAUSED AN OVERFLOW.
C
C          MCR        = MICROPROCESSOR CLOCK RATE
C          BPX        = BYTES PER TRANSFER
C          CPX        = CLOCKS PER TRANSFER
C           IPO         = INSTRUCTIONS FOR PACKET OVERHEAD (EQ
CLOCK
C                         CYCLES)
C
      REAL OFLOWCT,PKTCNT,PKTOFL,PKTRM,MAXBUF,TEMPBUF,ERATE
      REAL MCR,IPO,BPX,CPX
      REAL SIZE,TIME,BUFWAIT,OFLOWPT,INBUF(102),OH,BR
      INTEGER PTR,APKTCNT,RCTOFL
C       zero the input buffer values
      WRITE (*,001)
  001  FORMAT(1X,'Random  Arrival  Memory  Sizing  Test  in
Progress.')
        DO 10 PTR = 1, 10
            INBUF(PTR) = 0.0
  10    CONTINUE
        MCR = 18918625
        IPO = 1000.00
        BPX = 4.0
        CPX = 4.0
        SIZE = 0.0
        PTR = 1
        TIME = 0.0
        OFLOWPT = 0.0
        OFLOWCT = 0.0
        PKTCNT = 0.0
        PKTOFL = 0.0
        OH = 56
        BR = 12500000
        MAXBUF = 18990
        BUFSIZE = 0
        PKTRM = 0.0

* do as many times as needed,
        DO 100 APKTCNT = 1, 1000000000
        RCTOFL = 0
*     if there are packets in the buffer,
 250        IF (PKTRM .GE. 1) THEN
*         remove packets which have been processed since the
last
*           packet arrival
              SIZE = INBUF(1)
              BUFWAIT =  (IPO + SIZE * (BPX/CPX))/MCR
```

```
                IF (BUFWAIT .LE. TIME) THEN
*           adjust the position of the packets
                    BUFSIZE = BUFSIZE - SIZE
                    PKTRM = PKTRM - 1.0
                    DO 200 JOB = 2, PTR
                        INBUF(JOB - 1) = INBUF(JOB)
 200                CONTINUE
*           adjust the time clock of the removal process
                    TIME = TIME - BUFWAIT
                    PTR = PTR - 1
*             check for more packages which could have been
processed
                    GOTO 250
*          otherwise, continue with the new packets
                ELSE
                  GOTO 300
                ENDIF
*     endif
              ENDIF
 300          CONTINUE
*     if there are packets in the queue, the timer for the
*          processing of messages should be running
              IF (PKTRM .GE. 0.5) THEN
                  RCTOFL = 1
c                 write (*,999)
c999              format(1x,'set rctofl = 1')
              ENDIF
*     get random packet
              call getpktsize( SIZE )
*             write (*,101) size
*101          format (1x, 'the incoming packet = ', f12.4)
*     determine the packet size, THIS IS A TEST SIZE!
*             SIZE = 1000
*     increment the number of packets received
              PKTCNT = PKTCNT + 1
*     will adding a full size packet overflow the buffer?
              TEMPBUF = BUFSIZE + 4500
*     if so, increment the packet error counter
              IF (TEMPBUF .GT. MAXBUF) THEN
                  OFLOWCT = OFLOWCT + 1
                  RCTOFL = 1
*     else,
              ELSE
*         if there is room in the input buffer tracking system
                  IF (PTR .LE. 1024) THEN
*               add the new packet to the memory buffer
                    PTR = PTR + 1
                    PKTRM = PKTRM + 1
                    INBUF(PTR) = SIZE
*             increment the current memory size
                    BUFSIZE = BUFSIZE + SIZE
*           else, show that the memory reserved in the program
```

```
*                    was too small
                   ELSE
                     PKTOFL = PKTOFL + 1
*          endif
                   ENDIF
*     endif
              ENDIF
*     add time for the arrival of the new packet
              TIME = TIME + (SIZE + OH)/BR
*     if packet is the first in the buffer, it's arrival time
*         cannot be used for incrementing time.
              IF (RCTOFL .EQ. 0) THEN
                  TIME = 0.0
              ENDIF
*enddo
c     write (*,112)time
c112  format(1x,'time = ', f12.8)
c     write (*,113)pktrm
c113  format(1x,'packets remaining = ',f12.0)
c     write (*,114)size
c114  format(1x,'packet size = ', f12.0)
c     write (*,115)oflowct
c115  format (1x,'current overflow count = ',f12.0)
 100    CONTINUE
      WRITE (*,020)
  020 FORMAT(1X,'FINISHED WORKING WITH THE BUFFER')
*
* calculate the error to total messages processed and output
* solution
        ERATE = OFLOWCT/(PKTCNT - PKTRM - PKTOFL)
C        OPEN(6,FILE='DISS.DAT')
      WRITE (*,400)ERATE
  400 FORMAT (1X,'ERROR RATE = ', F12.11)
      WRITE (*,401)PKTCNT
  401 FORMAT (1X,'THE NUMBER OF PACKET SAMPLES = ', F15.4)
      WRITE (*,402)PKTRM
  402 FORMAT (1X,'THE NUMBER OF PACKETS REMAINING = ', F12.4)
      WRITE (*,403)OFLOWCT
  403 FORMAT (1X,'THE NUMBER OF FAILED BUFFER TRAILS = ',
F12.4)
      WRITE (*,404)PKTOFL
  404 FORMAT (1X,'THE NUMBER OF OVERFLOW FIELDS = ', F12.4)
C        CLOSE (6,STATUS='KEEP')
      WRITE (*,023)
  023 FORMAT(1X,'Program Complete.')
      STOP
      END
```

Dan B. Tolley, P.E.
Route 2, Box 391
Wytheville, Virginia 24382
(703) 637 - 6624

Education

PhD., Electrical Engineering, September 1990, Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, VA

M.S., Electrical Engineering, December 1984, Virginia Tech, Blacksburg, VA

B.S., Electrical Engineering, June 1982, Virginia Tech, Blacksburg, VA

Related Experience

Graduate Project Assistant, August 1988 - August 1990 Virginia Tech, Electrical Engineering Department, Blacksburg, VA

- Research involving interfacing an array processor to the Fiber Distributed Data Interface (FDDI) standard.
- Examination of the effects of processor type (RISC vs. CISC) on transfer efficiency.
- Analysis of transfer methodology to maximize network efficiency.
- Analysis of data packet structure used in the network to improve transfer efficiency.
- Analysis of maximized system throughput.

Senior Engineer, July 1985 - August 1988, Westinghouse Electric Corporation, Defense and Electronics Center, Baltimore, MD

- Lead engineer in the studies of high speed interprocessor communication using fiber optic and electrical mediums.
- Lead engineer in hardware development of computer systems.
- Study of electrical phenomena associated with VHSIC and ECL technologies.
- Development of guidelines to account for high speed electrical phenomena in synchronous designs.
- Developed probing mechanisms to allow minimal signal disturbance on very high density surface mounted devices (.020 inch spacing between leads).
- Responsibility for system and hardware enhancements and engineering quotations for projects.
- Interfaced with suppliers to provide device selection.

- Previewed devices for future applications.

Systems Design Engineer, July 1982 - June 1985, General Electric Company, Drive Systems Operations, Salem, VA

- Designed and tested Intel 8086 and 80286 microprocessor based information management systems for turbine and generator control systems.
- Designed and tested parallel processing control systems using Intel 8086 and 8087 microprocessors for extruding seamless stainless steel pipe in the U.S. Steel Fairfield facility.
- Designed general purpose analog and digital control systems for steel mills.
- Taught customer training courses to customers and GE instructors.

Training Courses

General Electric Edison Engineering Program, July 1982 - July 1984
General Electric Advanced Course in Engineering, September 1982 - December 1984

Papers, Patents and Disclosures

Dissertation:"Analysis of the Hardware Requirements of a High Speed Computer Interface Required to Utilize Fiber Distributed Data Interface"
Thesis:"Use of Parallel Co-Processors in High Speed Position Regulation"
"Decoupling/Termination Socket for High Speed Devices Utilizing Pin Grid Arrays"
"In-Circuit Test Fixture for Leaded Packages with Close Lead Spacing"
"On Board Test Probe Guide for Leaded Packages with Close Lead Spacing"
"Information Management for Control Systems"
"Electrical Phenomena Associated with High Speed Integrated Circuitry"

Honors

Member of Eta Kappa Nu

Licensure

Professional Engineer, Commonwealth of Virginia