AN ABSTRACT OF THE THESIS OF

Anoop R. Hegde for the degree of Master of Science in Electrical and Computer Engineering, presented on November 9, 1989

Title : An Efficient Data Transfer Protocol For Ethernet

Abstract approved : _____

Roy C. Rathja

Most general purpose protocols are found to suffer from inefficiency in an Ethernet LAN environment when used for large volume data transfer. This is especially true with the TCP/IP family which was designed for a wide area network. This research work presents a proof of this inefficiency and proposes a new protocol with simpler construction and low overheads. This Fast File Transfer Protocol has been designed to take advantage of the features that are unique to Ethernet.

Implementation of this new protocol, the Fast File Transfer Program for PC to PC communications, was found to achieve significant speedup over conventional file transfer programs. In addition to file transfer, the FFTP also implements a print server for the PC network. Design considerations for the protocol as well as the print server and some new techniques used, are discussed in detail.

An Efficient Data Transfer Protocol for Ethernet

by

Anoop R. Hegde

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed November 9, 1989

Commencement June 1990

APPROVED:

Redacted for Privacy

Professor of Electrical and Computer Engineering in charge of major

Redacted for Privacy

Head of the department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented November 9, 1989

Typed by Anoop R. Hegde for Anoop R. Hegde

ACKNOWLEDGEMENTS

It is my pleasure to acknowledge the help and guidance of Dr. Roy Rathja who is my major professor and advisor for this thesis. He could always spare some time for his students inspite of his busy schedule. He has helped me at different phases of this thesis work. I thank him for his time, consideration and for the encouragement I have recceived from him, all these days.

I would like to thank Andrew Rood, Adjunct Asst. Professor, Dept. of ECE, who initiated me into the field of networking. I have learnt more than networking from him.

I owe thanks to the faculty of the Dept. of ECE for all those enlightening courses. I would like to thank the secretarial staff of the Dept. of ECE for their prompt help whenever I needed it, Otto Gygax, David Crowe Jr. and the ECE computer support staff for their help in system and network related problems.

My thanks are due, to the Department of ECE for providing me an opportunity to pursue graduate studies, and for supporting me with teaching assistantship. This was of great help to me.

Professor Toshimi Minoura, Dept. of Computer Science, has kindly agreed to be my minor professor and I thank him for the same. My thanks are also due to Prof. James Herzog, ECE and Prof. Keith Levien, Chemical Engg. for agreeing to be in my graduate committee.

I am grateful to all my friends at Corvallis for the good time I had here, in this 'home away from home'.

TABLE OF CONTENTS

TABLE OF CONTENTS (contd.)

LIST OF FIGURES

AN EFFICIENT DATA TRANSFER PROTOCOL FOR ETHERNET

CHAPTER 1

INTRODUCTION

Local Area Networks (LANs) represent a rapidly growing facet of networking. They provide a low cost solution to the problem of strong connectivity among computers, in addition to data transfer rates as high as 10 million bits per second. LANs play a very important role in a distributed computing environment where most of the transactions involve a good amount of communication among the machines. A network, interconnecting several CAD workstations, database and file servers, is designed to have the ability to carry large amounts of data at high transfer rates. However, efficient utilization of high network capacity is possible only if the protocols involved are efficient. Quite often, the protocol turns out to be efficient only when designed with considerable attention towards the special characteristics of the underlying network. In fact, most of the general purpose protocols fail to utilize the high data rate such as the one found on Ethernet.

TCP/IP is among the most widely used family of protocols in any UNIX networking environment, often involving Ethernet. However, as Comer [COMER 87] rightly points out, the main aim of Internet architects was to come up with a transport protocol that was rugged enough to handle all possible adverse conditions

offered by a Wide Area Network (WAN) that could span the globe. However, it is an irony that the very features built into TCP, that make it so efficient in a WAN, turn out to be the causes of performance degradation in an Ethernet LAN environment, as proven in later chapters.

This research work proposes a new protocol that is tailored to Ethernet LANs to provide a speedup of more than 50% over the Internet File Transfer Program (FTP) which uses TCP. All the important features, particular to Ethernet LAN, like maximum and minimum packet sizes, extremely short round trip delays, high data rates, absence of routing, fragmentation and multiple paths between two machines, are taken into consideration during protocol design. An implementation of this protocol, the simple and elegant 'Fast File Transfer Program' (FFTP) demonstrates its effectiveness and speed. This program provides a print server in addition to PC to PC file transfer functions. An adaptive algorithm, called the 'dynamic timeout scheme', gives this program the ability to perform data transfer between two machines that differ greatly in their speed of communication, without sacrificing the efficiency. Dual buffer scheme is another novel technique used in FFTP implementation to reduce the effective delay involved in accessing the data in the secondary storage.

This thesis is organized into five chapters. Chapter 2 is an introduction to Local Area Networks and various Media Access Schemes (MAC) being used. IEEE standards 802.3, 802.4 and 802.5 (also known as CSMA/CD Ethernet, Token Bus and Token Ring respectively) are discussed. This chapter also deals with

various protocols that use Ethernet and compares them with the ISO model of networking. Chapter 3 discusses in detail, the design of the Fast File Transfer Program, underlying protocol, and some important design and implementation issues that make it the most suitable for data transfer on Ethernet. The Client-server model of distributed computing, design and implementation of the print server and the basic differences between a UNIX based print server and the one that runs on DOS, are described in Chapter 4. Chapter 5 gives a theoretical comparison of performance of the two protocols considering machine and network dependent delays and tabulates the observations. Suggestions about porting the protocol to machines running UNIX, hints for possible improvements, enhancements and installation, are also given in this chapter.

CHAPTER 2

AN OVERVIEW OF LOCAL AREA NETWORKS

2.1 An Introduction to LANs

Local Area Networks are an essential part of any computing environment. A LAN is characterized by a diameter not more than a few kilometers, high data rates exceeding 1 Mbps and short propagation delays. Unlike Wide Area Networks that communicate in a point to point fashion, LANs use the broadcast method. All the machines, connected to the network, access a common transmission medium in a way that is both fair and efficient. Control is distributed in that, one machine can talk to the other without going through a master controller. Speed of data transfer and spread of a LAN are constrained only by the limitations of the physical medium.

LANs are being used extensively to serve the needs for communication in an office building, a manufacturing floor, a wide spread campus of an educational institution, as well as a distributed computing facility. High speed LANs provide an infrastructure that can support a variety of applications simultaneously. Exchange of information and sharing of expensive resources can be achieved, in an efficient way, with a LAN supporting high speed data transfer. CSMA/CD, Token Bus and Token Ring, are among the most widely used local networking schemes. These can be differentiated by the way they access the

transmission medium. They have been standardized by the Institute of Electrical and Electronics Engineers (IEEE) as 802.3, 802.4 and 802.5 MACs respectively, and share a common 802.2 Logical Link Control layer.

2.2 MACs : Token Ring, Token Bus and CSMA/CD

Media Access Schemes (MACs) can be broadly categorized based on whether they are collision free or not. Collision occurs when a machine starts transmitting before another one is done with its transmission, thus causing interference and garbling of signals. Token ring and Token bus are collision free, whereas Ethernet allows collisions to occur, and provides a way to detect and handle it.

TOKEN RING : In this scheme, the machines are interconnected in a physical ring configuration. Each of the stations plays an active role by reading every bit on its input side and writing it to the output side. Distributed control is provided by means of a token, which is nothing but a special pattern of bits. A station can transmit only when it has the token for itself. The message passes through the nodes on the way to its destination node. Each node examines the destination address in the packet header and passes the message to the next one in the ring, copying the message into its internal buffers if it matches the address of that node. The destination alters the token after copying the message and when the packet eventually reaches the source node, the token is set free. Slotted ring, concurrent ring and bidirectional ring, are

some variations of this scheme. Because of its collision free nature, the token ring has a high degree of predictability.

TOKEN BUS : Though the topology is of that of a common bus, the system operates like a logical ring. A token is passed from a machine to the next in the logical ring. This scheme removes the restriction that the logical successor of a station must also be the physical successor, still retaining the predictability of a ring network. As it combines the features of both the ring and the bus, it has been selected for manufacturing automation applications where sensing devices, programmable controllers and robots exchange prioritized information and where time critical operations require predictable response times.

CSMA/CD : The Carrier Sense Multi Access/Collision Detect method is based on the 'Ethernet' principle developed by Xerox in 1976 [Metcalfe 76]. It is supported by DEC and Intel, and was modified later to form the IEEE 802.3 MAC. Based on a concept that is totally different from that of the above mentioned schemes, it operates on a bus. A co-axial cable or a twisted pair can be used for the bus. Being a collision detection system, it requires that the transmission time for one frame must exceed the round trip bus delay. It should also be closely tuned to its physical medium in order to reduce collisions. Possibilities of random collisions make its behavior highly unpredictable. However, the *exponential backoff* strategy used with most Ethernet implementations, tends to have a stabilizing effect and allows communication even on a heavily loaded network. The stations tend to be *1-persistent* in that a station senses the medium for any carrier activity and

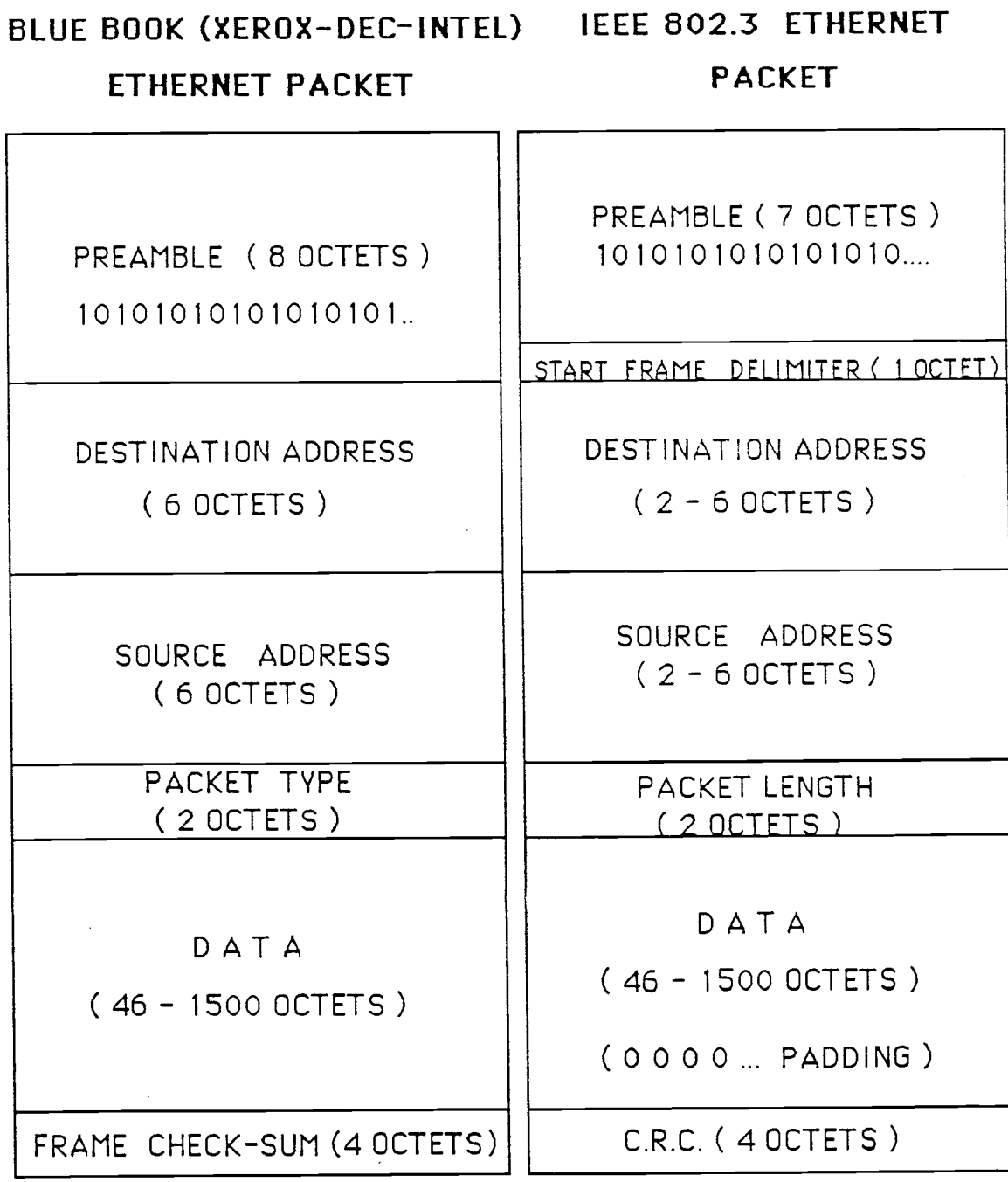
immediately starts transmitting if the medium is free. A station operates as if it has the exclusive ownership of the medium until a collision changes the situation. Adding a new station is as simple as plugging a transciever into the ether.

Ethernet has a large advantage over the other schemes in that it is simple to implement and the throughput of a station, connected to the ethernet, does not depend on the total number of stations present; it depends only on the activity of any other station when transmission is attempted. Throughput drops with increasing load, with collisions causing delays in completing an operation. Because of these characteristics, it is considered to be the ideal network for interactive computing where transmissions are short and infrequent, being caused by the activity of hundreds of users on remote terminals. Ethernet can be a 'thick wire' type or a 'thin wire' one, and operated in either baseband or broadband mode using a radio frequency carrier. Data rates can be as high as 10 Mbps, with a round trip delay as short as 45 microseconds, and it is possible to connect machines up to 500 meters apart without the need for a repeater. Maximum separation allowed between any two machines is 2500 meters.

An Ethernet frame consists of a preamble of 8 bytes of alternating zeros and ones (useful for synchronizing the receiver hardware) followed by two fields of 6 bytes (octets) of destination and source address, 2 bytes of 'packet type' field, data bytes and ends with four bytes of Frame Check Sum (FCS) as shown in figure 2.1. The IEEE 802.3 frames differ from the Bluebook Ethernet

Figure 2.1

ETHERNET PACKET FORMATS



frames, only in that the 'type' field is replaced by a field indicating the packet length.

2.3 Novell Network for Distributed Computing

Novell, Inc. contributed to the growing field of distributed computing by introducing its network operating system and servers. The *netware* includes a Novell shell, network drivers and a file server. A machine is dedicated as a file server and runs a multitasking operating system. It holds user files, account information and application software, in addition to taking care of security issues. In order to get connected to the server, the user runs the Novell shell on the workstation and logs in. Once a workstation is attached to the file server, all network operations become user-transparent and the server appears to the user as just another disk drive. All the computations are carried out at the local workstation utilizing its full power. The applications, residing on the server, are loaded into local memory before execution, by means of network transfer. All the requests by the system to transfer data into this new 'drive', are intercepted and handled by the network drivers. Thus, Novell provides network access that is transparent to the applications running on the local machine. IPX and SPX (which stand for Internetwork Packet exchange and Sequenced Packet exchange protocols) form the network and transport layers, respectively, of the Novell protocol family. These are nothing but modified Xerox Network System (XNS) protocols, known as IDP (Internet Datagram Protocol) and

SPP (Sequenced Packet Protocol). There can be several print servers to receive and process printing requests, and communication servers which enable communication with another Novell network. However, all this facility is not without some shortcomings and inconveniences, as described in Chapter 3.

2.4 TCP/IP and DOS to UNIX Communication

The TCP/IP family of protocols (Transmission Control Protocol/ Internet Protocol), widely used in Wide Area Networks like Internet, are also being used extensively for DOS to UNIX communication. IP serves as the network layer while TCP serves as the transport layer, and they support a large family of network application programs; mail, rlogin, telnet, ping, lpr, and FTP being the major ones. Implementations of NFS (Network File System) have IP/UDP (Unreliable Datagram Protocol) as their basic communication protocols. SU-PC/IP from Stanford University and the Telnet package from Clarkson University, have been the two major implementations of the Internet family of protocols in a personal computer network. However, both support only one way transaction, the PC always being the initiator. SU-PC/IP comes with its own driver, inseparably built into the rest of the programs. The driver and multitasking programs need to be memory resident before any of the applications, that use the network, can run.

Clarkson telnet uses a packet driver that confirms to NCSA (National Center for Supercomputing Applications) specifications,

and is more flexible in dealing with different network hardware, physical and data link layers.

Both of these packages support ARP (Address Resolution Protocol) and name server query, in order to get ethernet addresses from Internet addresses and for Internet name to address conversion. They allow the user to connect a personal computer to a machine running UNIX through ethernet, and to use the PC as a dumb terminal to the main computer.

2.5 The ISO Model of Networking and its Relevance to Different Protocol Implementations

The Open Systems Interconnections (OSI) model of networking, proposed by the International Standards Organization (ISO), as specified by Zimmermann [Zimmermann 80], is comprised of seven layers. The layering is done to meet the following requirements.

1. Each layer should be able to perform a well-defined function.
2. Layer boundaries should be so chosen as to minimize the data transfer across the interfaces.
3. The layers should be large enough to include similar functions together, and small enough to be practical to implement.

The seven layers and their brief functional description are given below.

PHYSICAL LAYER : This layer takes care of all physical aspects of the communication channel like voltage and current levels of the

signals, modulation and demodulation, recognition of individual bits of data, mode of transmission (simplex, half duplex or full duplex), interface of the circuits to the physical medium, connector and cable specifications, mechanical and electrical aspects of the connection, etc. The unit of information handled by this layer, is a *bit.*

DATA LINK LAYER : It uses the raw bit transmission facility, provided by the physical layer, to exchange *frames* of information between two units. It takes care of error checking by means of a frame checksum or cyclic redundancy check, and flow control using handshakes.

NETWORK LAYER : The basic unit of data, handled by this layer, is a *packet* . This layer carries out routing of the packets through several of the alternate network connections, does network error checking and reporting, and controls network congestion. However, reliable delivery is assured only up to the next machine, and not to the ultimate destination. The packet is fragmented and reassembled as required. This layer decides the division of labor between the host computers and the machines that interface the host to the network.

TRANSPORT LAYER : This is the first layer to perform host to host communication. It is the responsibility of this layer to take care of the packets lost in transmission through the network, damaged and corrupted packets, correct sequencing of the

packets that might have arrived in random order and duplicate packets. The transport layer either does reliable delivery of the message handed over to it, or reports the status that the message cannot be delivered. It also handles multiplexing and demultiplexing of the packets among multiple network connections, and end-to-end flow control.

SESSION LAYER : It is the responsibility of this layer to negotiate, setup, carry out and terminate the connection between two machines in an orderly manner. It also handles user authentication and transaction processing whenever necessary.

PRESENTATION LAYER : The major functions of this layer include machine dependent data and file format conversions, encryption and decryption of sensitive data, terminal emulation, screen handling for interactive sessions and data compression for minimizing transmission costs.

APPLICATION LAYER : Usually created by end users of the system, this layer can perform functions like providing access to the network in a way that is transparent to the user, division of a problem for distributed processing, easy-to-use user interfaces and many more.

It is an interesting fact that most of the actual implementations do not adhere to the OSI model strictly. In the Internet family, IP does most of the functions of the network

layer, except reliable packet delivery to the next machine. TCP functions as the transport layer and handles more burden due to the unreliable network layer. Application programs like FTP, Mail, Telnet, Rlogin, etc. consist of the upper three layers of the ISO model. Layers below IP are not defined, and can be system or network dependent.

Novell netware has its own driver that performs part of the data link and network layers. IPX is part of the network layer and SPX forms the transport layer. Novell shell, file server and application programs handle the rest of the layers. The distinction between several adjacent layers is not very clear. The netware drivers can be configured to use different networking hardware.

FFTP has the NCSA packet driver as the Data Link layer and almost no network layer (an elaborate network layer is not required in a LAN). Reliable packet delivery routines along with the stop and wait algorithm form the transport layer and the rest of the higher layers are combined in a set of functions.

Figure 2.2 compares and contrasts the OSI model with the above protocol implementations. The ethernet configuration in the ECE department is shown in figure 2.3.

Figure 2.2

PROTOCOL IMPLEMENTATIONS AND THEIR RELATION TO THE ISO MODEL OF NETWORKING

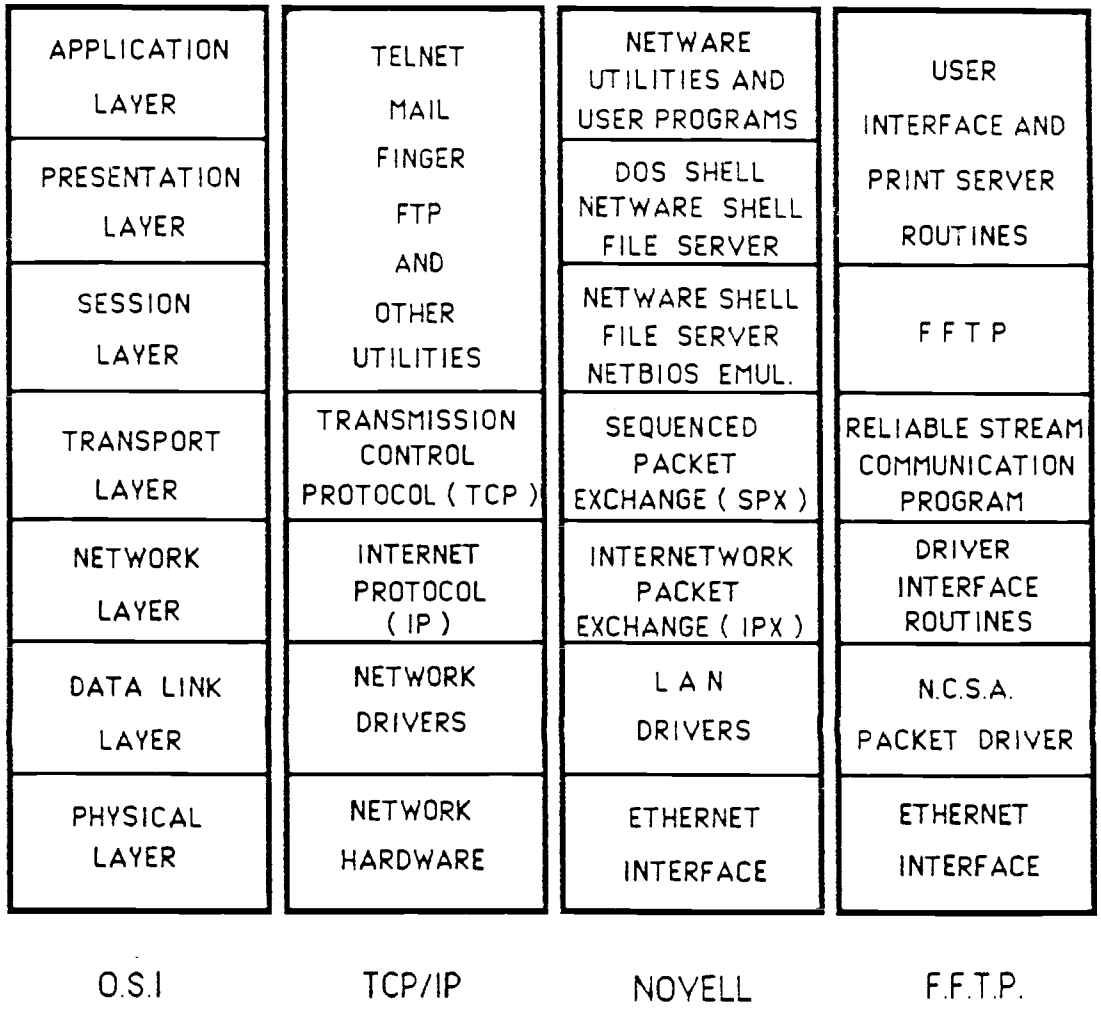
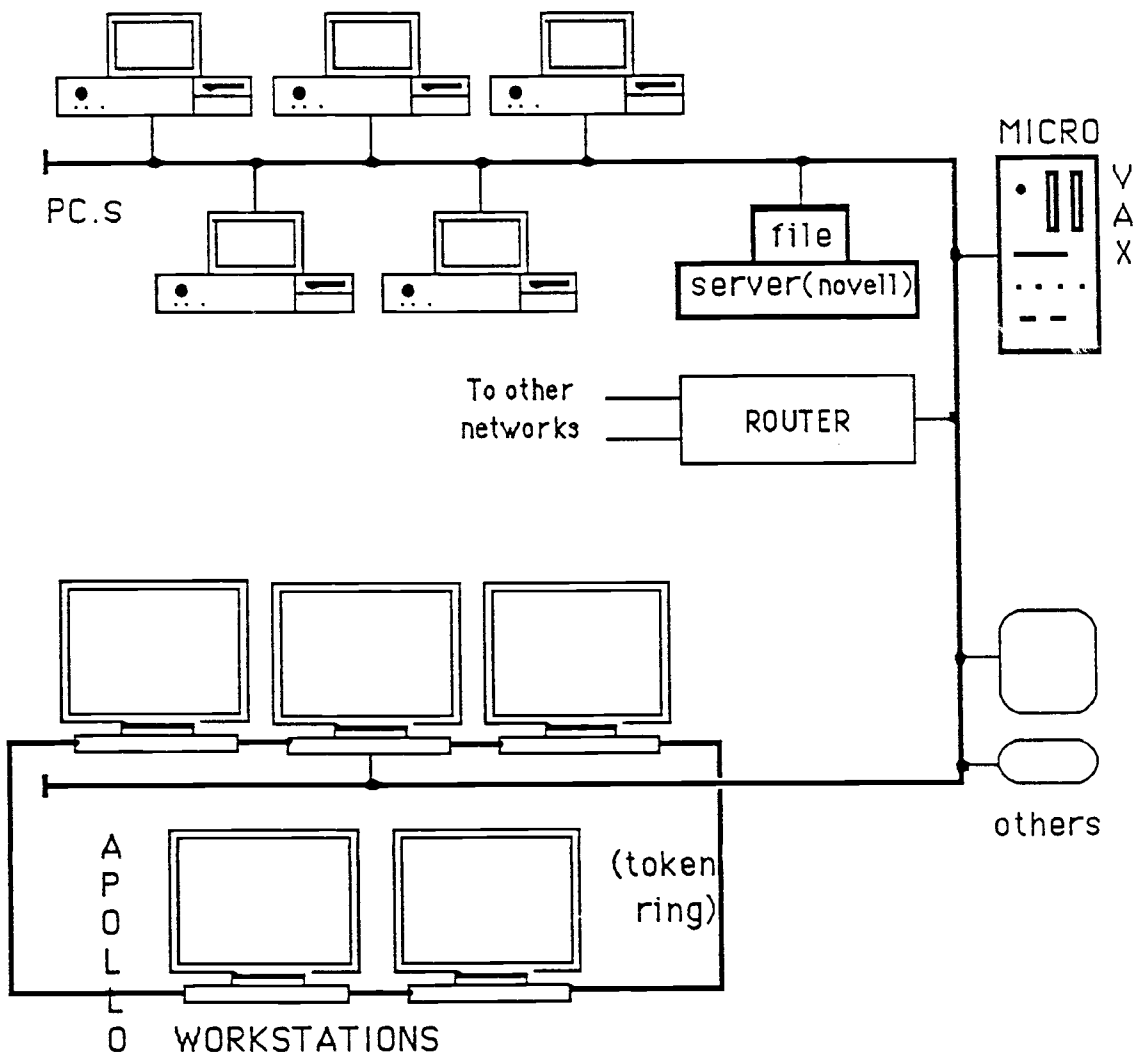


Figure 2.3

DEPARTMENT OF ECE ETHERNET CONFIGURATION



CHAPTER 3

THE FAST FILE TRANSFER PROTOCOL : DESIGN AND IMPLEMENTATION

3.1 The Motivation

Ethernet LAN supports data transfer rates of up to 10Mbps with a maximum of 1526 bytes per frame. Applications which transfer a large volume of data, like bitmap image files, document files and executable files, benefit by the presence of an efficient communication program. The two mechanisms available on our network (Novell Netware and the Internet FTP) suffer from some drawbacks in this respect.

Novell netware has created a distributed computing environment with its network shells and servers. However, there is no means for two PCs on the Novell network to communicate with each other, nor is there a way to do PC to PC file transfer without going through the file server. The only way to effect this transfer is by logging on to the file server from two PCs, copying the file from the first PC to the server drive, copying the file from the server drive into the second PC, and then deleting the file from the server storage. This procedure is both elaborate and time consuming, especially if several files are to be transferred frequently. It also puts an extra load on the file server which is not always desirable. More complications can occur during transfer of

large files especially if the user is near the limit of his quota of storage space. Another undesirable characteristic (at least in the current version) of the Netware is that it is found to interfere with other network drivers due to its interaction with the local operating system.

On the other hand, the TCP/IP family of protocols has been developed to carry out communication across the Internet, which belongs to the class of Wide Area Networks. As a result, TCP/IP includes facilities to set up and sustain connections in spite of lost, corrupted, duplicate or out of sequence packets, extremely long delays, failure of arbitrary nodes in the network, packet fragmentation, network congestion or other undesirable conditions prevalent in any WAN. It also has provisions to effectively route the data packets through a maze of networks and machines. Many of the features that make it reliable and efficient on the Internet cause excessive overheads and large delays when used in an Ethernet LAN, thereby degrading its performance. Its implementation in a highly layered fashion, large header sizes, packet sizes that are just about one third of the maximum allowed on Ethernet, and the sliding window flow control technique, as mathematically proved in Chapter 5, are among the major factors that contribute to its inefficiency. Again, the FTP software, which operates on TCP/IP, is incapable of transferring files between two PCs without going through a UNIX machine.

Most of these difficulties can be overcome by designing a protocol that takes into consideration some important aspects of LANs that are entirely different from those of WANs.

1. There is no real routing involved in an Ethernet LAN. A packet with the ethernet address of the destination, is picked up by the right machine, without any extra efforts on the part of the sender.

2. No fragmentation and re-assembly are necessary, as, the sender and the receiver are both present on the same network.

3. Sequence number fields in the header can be as small as a few bits as the delays involved are of the order of 40 to 50 microseconds.

4. The Ethernet header is sufficient to identify the source and the destination, and the error checking, performed by the DLL (Data Link Layer), is sufficient for most purposes.

5. Due to the short round-trip delays involved, there is no need for an elaborate sliding window flow control algorithm. Flow control methods, as simple as the 'stop and wait' scheme, are sufficient. In fact, this simple scheme is found superior to the sliding window, in a heavily loaded network, as it causes less number of retransmissions as shown in Chapter 5.

6. The protocol can be made to utilize the smallest possible header and the largest allowed packet size, in order to increase the data transfer efficiency.

3.2 Flow Control Methods for Transport Layer

Flow control is necessary during data transfer to prevent the faster machine from swamping the slower one with data. There

are two major schemes available for flow control, namely stop and wait and the sliding window schemes. In the stop and wait method of flow control, the sender transmits a packet and waits for an acknowledgement (ACK) before transmitting one more. Usually all the senders will have a timer associated with the packet, which is started as soon as the packet is transmitted. ACK may not arrive because either the data packet did not reach the receiver or the ACK packet was lost on the way. In either case, the timer expires and the sender re-transmits the data packet. Such a method results in wastage of bandwidth if there is a large delay involved in the network, as, the sender has to sit idle until the ACK arrives.

In the sliding window scheme, the sender and the receiver agree upon a common window size, W. Usually, this is an indication of the maximum buffer space available at the receiver. The sender transmits W packets and starts a timer for each packet. If all the packets reach the receiver, it will send a single acknowledgement indicating the next sequence number expected. Upon reception of this ACK, the sender resets all the timers and advances its window forward. On the other hand, if the receiver does not get all the packets intact, it sends ACK for the highest sequence number received, and the sender retransmits all the packets from the next sequence number, to the end of the window. The worst case is when the first packet is lost, in which case, all the packets are to be retransmitted even if the receiver got the remaining packets intact. If the ACK is lost, the sender retransmits packets as the timers expire.

Figure 3.1 shows the transactions between two machines for these two schemes. The sliding window method has great advantage over the stop and wait method if transmission delays are predominant, as it allows some processing to be done while the packets are in transit. However, it also has a disadvantage that, in order to have a large window, more memory is required. In addition, it is considerably more complex to implement. The following section discusses the limitations in detail.

3.3 Limitations of the Sliding Window Flow Control Scheme

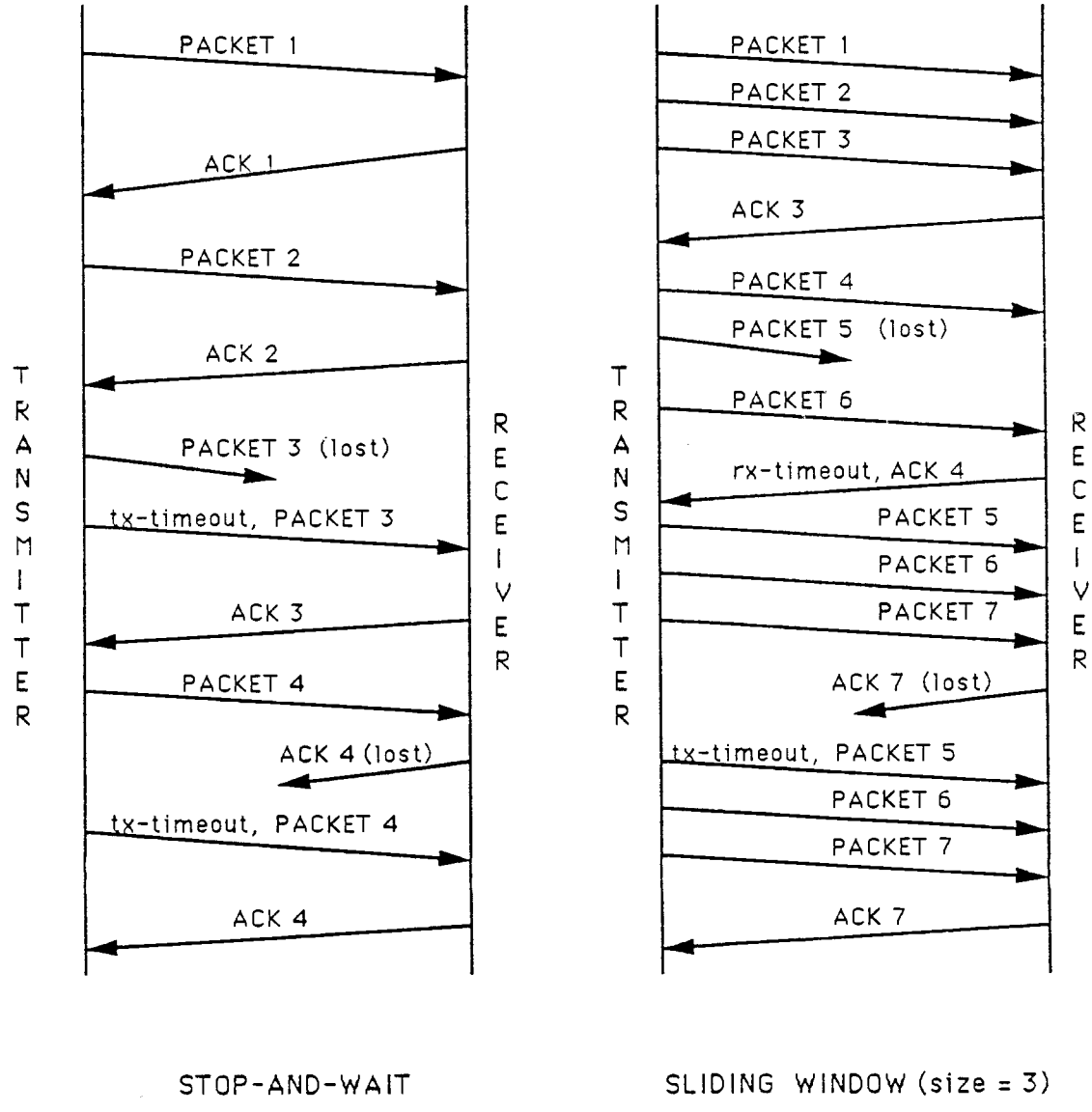
Most of the emphasis in this work is on the elimination of the sliding window scheme and on the improvements in data size to packet size ratio in the transport protocol design. Here is Carl Sunshine's [Sunshine 77] observation on the flow control methods.

> .....as long as the packet loss/damage probabilities remain low, the throughput will be flow-control-limited by a small window size. However, in networks under heavy load conditions, transmission errors and collisions are more likely, and the throughput may become retransmission limited because retransmission of pending packets have priority over the new transmissions and also the number of packets, retransmitted, is a function of the error probability as well as the window size.

Let us consider M as the ratio of arrival to consumption rate for the packets. For $M >> 1$, no amount of buffers suffice because all will be filled up in the steady state, and most arriving packets will have to be discarded. Hence a small window size, with a few

Figure 3.1

TRANSACTIONS BETWEEN TWO MACHINES USING STOP AND WAIT AND SLIDING WINDOW FLOW CONTROL SCHEMES



buffers, is preferred. The limiting case is with the machines differing greatly in speed, when window size = 1 will be reached.

However, at light loads. where M << 1, data transfer speed is limited more by the sender's production rate and not by the network speed. This is especially true where the production rate is 2 to 3 orders of magnitude smaller than the packet transmission speed, including the round trip delay. This is exactly the case with PCs attached to Ethernet. Delay in accessing the secondary storage, where all the data is initially stored, is found to be about 100 times more than the largest network delay. This aspect is also pointed out by J. Vinyes [Vinyes 86] in his article describing why a window size greater than 1 is not effective in a microcomputer network.

Another fact against the sliding window scheme is that it is more effective where acknowledgement packets can be totally eliminated by sending piggy-backed acknowledgements, as is often done if data transfer is in both directions (duplex). This is even more important where the bandwidth is a scarce resource, as with leased phone lines. Instead of pure stop and wait, if the sender can have two buffers and refill one while waiting for the acknowledgement for the other, considerable improvement in performance can be obtained. In fact, FFTP uses a variation of this scheme, where the acknowledgement timer is started after refilling the secondary buffer.

3.4 Implementation of FFTP

The Fast File Transfer Program includes network, transport, session and presentation layers in a single package. Modular design principles are followed to ensure ease of debugging and to help program development. Totally layered construction is carefully avoided in order to minimize the overheads. An important point to be kept in mind while designing any network software is that the layering of the ISO model does not imply design of six layers of software. As Comer [Comer 87] rightly points out, such an approach is bound to suffer from excessive overheads. Time critical functions like ethernet interrupt handler and the packet demultiplexer have been designed so as to minimize all possible overheads. Structured design principles are compromised wherever time is a factor of greater importance. Accordingly, practices like the use of global variables instead of parameter passing and the use of macros instead of function calls, can be found in all critical procedures.

The program was developed in Turbo C for two main reasons:

1. The C language, because of its advanced system interfaces, low level operations like bit level arithmetic and logic operations and direct memory addressing functions, has been considered to be the most ideal language for the development of operating systems, compilers, editors and other complex programs. It is also a structured language, still not as rigid as Pascal in type checking and in following

the conventions. Because of these considerations, C was the language of choice for writing this communication program.

2. Turbo C was used for its enhancements over the standard C, which give the user access to low level DOS and BIOS calls, I/O addressing capability, register addressing and interrupt handling capabilities. Also the integrated environment, with the editor-compiler-debugger combined, is highly conducive to program development. However, parts of the program, like the driver interface routines, were written in the 8086 assembly language to take care of the operations which a higher level language can not handle.

A network protocol consists of a set of rules that specify the method of session setup, data transfer, and termination (with all the handshakes involved). Basically, design of a protocol is very much similar to the design of a state machine. The arrival of a packet of a particular type acts as the event that causes state transition in this machine. All possible state transitions and abnormal conditions like session abort, timeout, user break, system and network errors are to be considered for a reliable protocol design. Elaborate timeout mechanisms are needed to prevent indefinite wait states when the remote host is not functioning properly.

Network independence is also another important aspect of protocol design. Though FFTP was developed basically for Ethernet, it can be used with all existing packet drivers that confirm to NCSA specifications, at the lowest level. This enables the program to be used with a wide range of media access

methods. For instance, a RS232 port on the PC can also serve as the communication port to the FFTP, with a driver that can handle this interface. This approach has another significant advantage in that several protocols that use the same network interface can co-exist with the FFTP software still performing normally. However, some delay parameters will have to be modified before such a porting can be efficient.

The FFTP packets use a header of 4 to 22 bytes, depending on the packet type. Sequence number, checksum and packet type are the parameters that are present in every FFTP packet header. There are six different packet types, as shown in Figure 3.2.

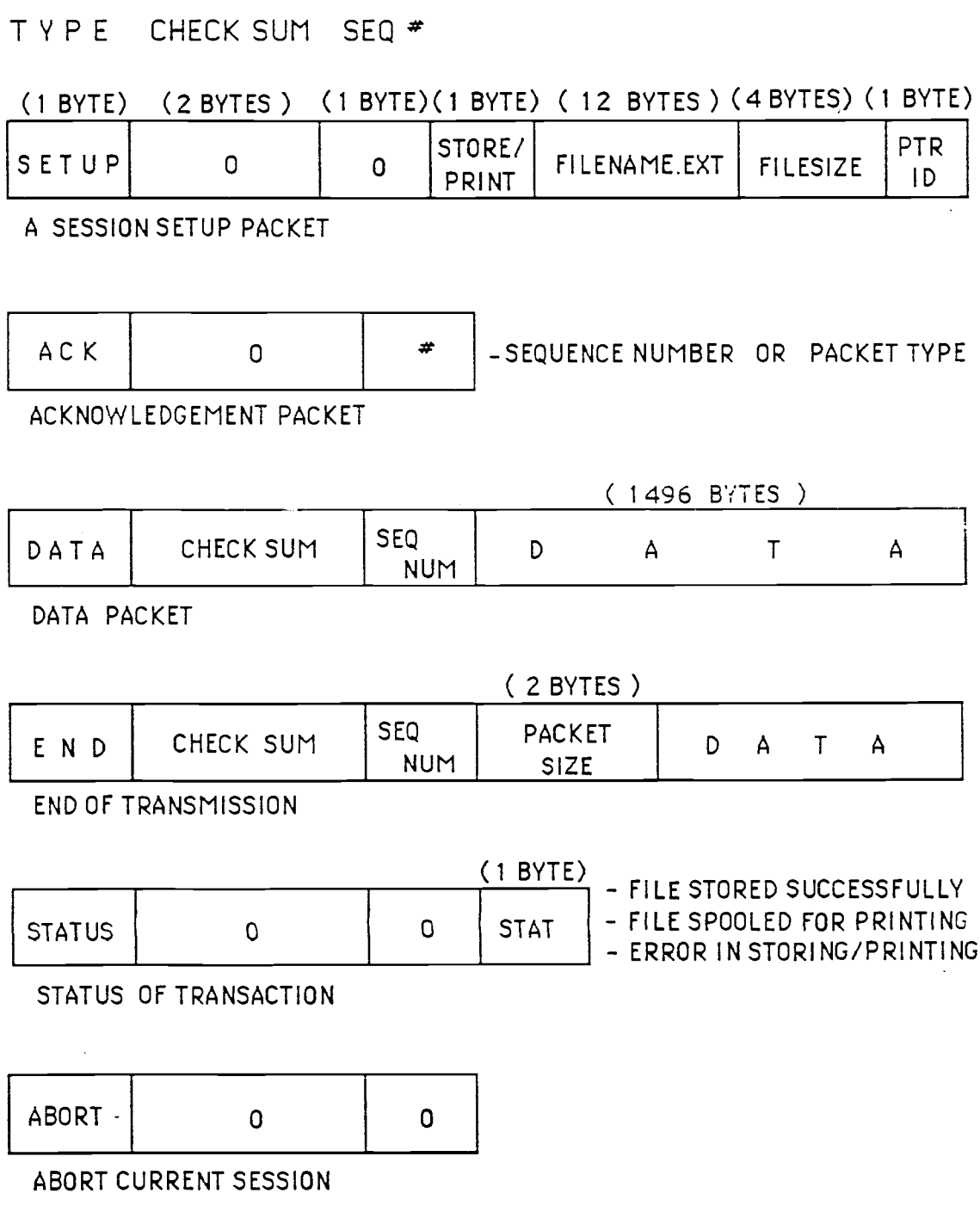
The user runs FFTP from the command line, and selects one of several options from the main menu. The program takes the user through sub menus, if required, to get all the information about the file transfer. The four major functions of the transport layer are implemented as described below.

Reliable delivery is ensured using a positive acknowledgement scheme. A packet is transmitted and a counter is started. If no acknowledgement is received by the time the timer has expired, the packet is re-transmitted and a retry count is incremented. If no acknowledgement arrives by the time the retry count exceeds its maximum limit, the session is aborted after informing the user of the fact. This approach takes care of both lost packets as well as lost acknowledgements.

When a packet is received, its sequence number is checked with that of the receiver sequence number. A packet, with non-matching sequence number, is discarded after an ACK followed by

Figure 3.2

FFTP PACKET FORMATS



an ABORT is sent. Duplicate packets are also treated in an identical way, except that no ABORT message is sent.

If checksum field of the received packet is non-zero, a 16 bit checksum is computed using the header and data portion of the FFTP packet (excluding the checksum field itself). If the checksum does not match the computed sum, the packet is assumed to be in error and is discarded without acknowledging it, relying on the sender to timeout and re-send the packet. A zero in the checksum field indicates that no checksum is desired.

Flow control is done using the stop and wait scheme with a variation, as described earlier. Additional flow control is provided by the lower layer of software (driver interface routines). If a packet is received when no buffers are free, it is discarded without notifying the higher layers.

When the driver encounters any hardware errors, it informs the higher layers by returning an error code. It is the responsibility of the higher layers to take suitable action.

The Dynamic Timeout Mechanism is a novel feature of FFTP. If a packet is acknowledged before the timer runs out, the timeout period is decremented by a number proportional to the difference between the current timer count and the maximum count. On the other hand, if timeout occurs after sending a packet, the timeout interval is incremented by a unit time and used for the retransmitted packet. This feature has interesting results, in that, it allows the program to adjust its transmission speed to that of the slower machine. Each timeout causes the next timeout to be delayed, thus slowing down the transmission.

Each case of no timeout is handled by shortening the next timeout, expecting a faster reply. Maximum and minimum values of timeout have been chosen considering the fastest system encountered. This scheme is found to have a levelling effect on the performance, which is found to remain constant under varying network load conditions. The following algorithm is used for timeout recalculation : if T is the current timeout interval and T1 is the current response time,

if T > T1

then T = T - INT((T-T1)/2) (no timeout)

else T = T + INT((T1-T)/2) (timeout occured)

Connection oriented service is used and the connection passes through three distinct phases : an establishment phase, a transfer phase and a termination phase. Connection is simplex though packets are sent and received by both sides.

3.5 Physical, Data link and Network Layers for FFTP

Western Digital Ethernet controller is used as the network interface. The WD8003 chip, used in the card, has all the the logic necessary to handle carrier sensing, collision detection, transmission and reception of data bits, address recognition, CRC (Cyclic Redundancy Check), a ring buffer for storing up to 8 octets of received data, and the interfaces to interact with a microprocessor system. The WD8003 card has a user selectable I/O base and buffer memory addresses, configurable hardware interrupt vector and software configurable ethernet address.

The NCSA packet driver forms the second layer that operates above the ethernet interface card. This driver attaches itself to a software interrupt vector and stays as a memory resident program. It intercepts the ethernet hardware interrupt and takes care of transferring the octets from the ring buffer of the driver chip into a buffer in the memory. After the whole packet is assembled, the driver upcalls a user-specified routine and hands over the packet. The driver uses the 1-persistent transmission algorithm with exponential backoff to stabilize the network operation at high load conditions. It also keeps an account of the number of bytes sent and received, the number of collisions, damaged packets and the packet type being received.

These two layers take care of receiving only the packets addressed to this unit (and all the broadcast packets), ignoring the packet type. It is the responsibility of the higher level protocol to accept or discard the packets based on their type or source address.

Development of the next higher layers was done in three phases.

1. A low level driver interface that provides the basic packet transfer mechanism, with collision and error detection, was first constructed.

2. A reliable packet delivery mechanism was built using the positive ACK and timeout-retransmission scheme. This layer is capable of informing the higher layers of the success or failure of the delivery attempt.

3. A reliable file transfer function that uses the reliable packet delivery mechanism, was next designed. It is this layer that takes care of lost, damaged, duplicate and out of sequence packets, sequences data, exerts flow control and hands over the assembled data in the form of a file to the higher layer. All the information required for further processing of the file (like store/print options, name and size of the file, etc) is also transferred along with the file. This layer handles system errors encountered in naming or storing the file, user interrupt (control C) and other catastrophic conditions. It also has a simple, menu driven user interface.

3.6 Protocol and Header Formats

The sender performs a data transfer operation as shown below.

1. A session SETUP packet, containing the file name, file size, operation to be performed on the file, and the identification of the printer on which it needs to be printed, is sent to the destination machine.

2. After receiving the ACK for the setup packet, DATA packets are sent with confirmation of delivery.

3. End of transmission is conveyed by sending an END packet.

4. The sender now expects the STATUS of the transfer to be returned, reports it to the user and exits.

... and the receiver acts accordingly.

1. The receiver processes the SETUP packet and opens a temporary file to store the data. It also preserves all the information in the setup packet and sends an ACK.

2. It receives all the DATA packets and writes them to the file, checking for error conditions. Any error causes an ABORT packet to be sent to the sender, which is expected to abort the session immediately.

3. The END packet is received and acknowledged. This triggers more processing activity. The temporary file is closed and renamed. After making an attempt to store the file or spool it for printing, as the case may be, the receiver sends back a STATUS packet to the sender. It terminates the session after receiving an ACK for the status packet. Following are some additional features built into the program.

1. After a session has been setup, if no data packet arrives within a certain amount of time, the receiver unilaterally aborts the session after sending an ABORT packet. This is done to ensure proper operation of the receiving machine in case the sender crashes after sending the setup packet.

2. All the SETUP packets result in sending an ABORT when a session is in progress. This is essential when the program is implemented on an operating system like DOS which does not support multitasking, thus making concurrent sessions impossible.

State machine representations of the FFTP sender and the receiver are given in Figures 3.3 and 3.4 respectively. The flowcharts for main routines like send_packet, send_file, receive_packet and receive_file, are given in Appendix B.

Figure 3.3

STATE DIAGRAM FOR FFTP SENDER

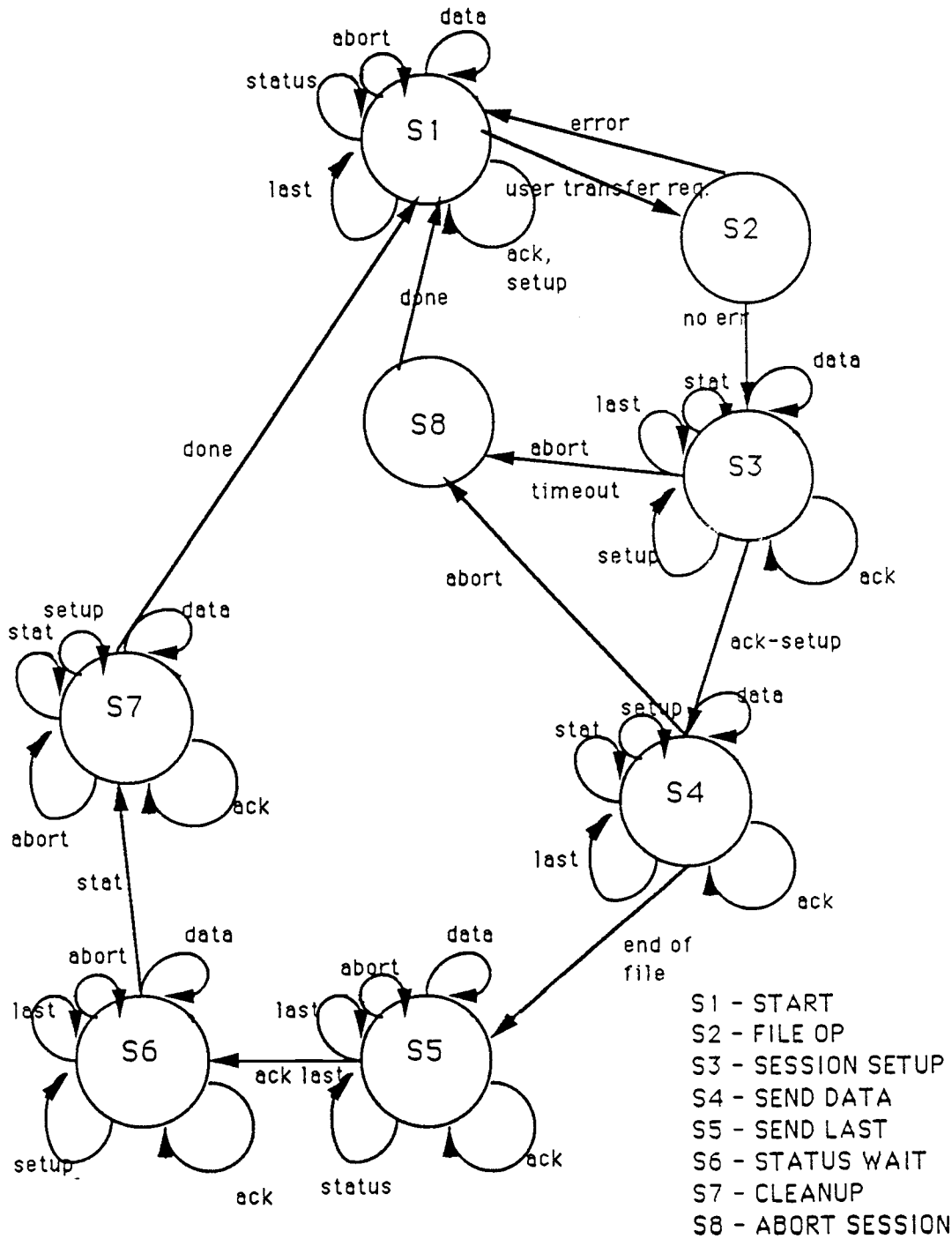
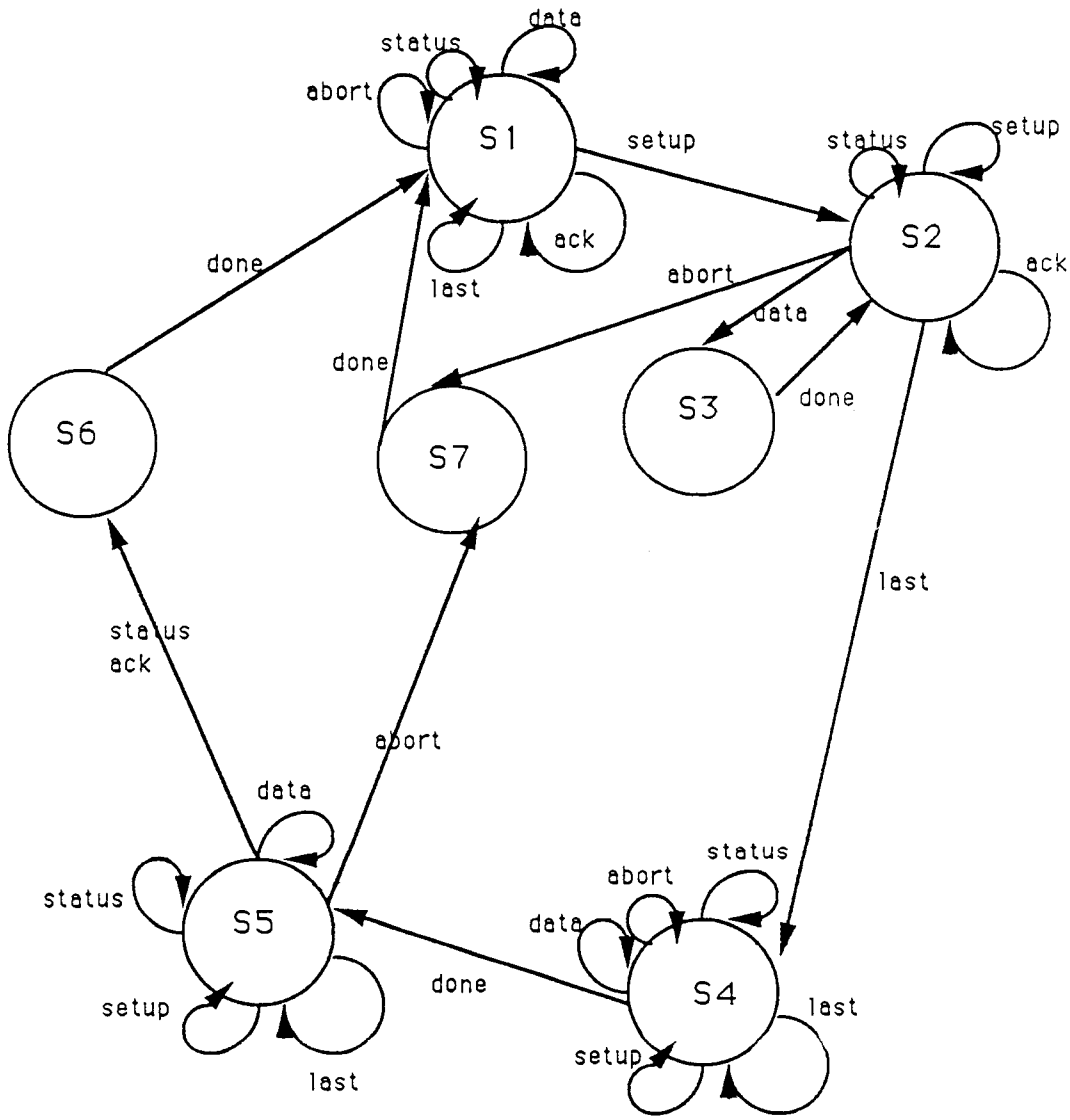


Figure 3.4

STATE DIAGRAM FOR FFTP RECEIVER



CHAPTER 4

DESIGN AND IMPLEMENTATION OF THE PRINT SERVER

4.1 The Client-Server Model of Distributed Computing

The concept of distributed computing can be applied to a wide range of machine configurations, ranging from tightly coupled multiprocessor systems to hundreds of machines connected by LANs, WANs and long haul networks. Several models of distributed computing exist, like the Hierarchical model and the CPU Cache model where a central control exists, the Pool Processor model where control is more distributed, the Data Flow model which is close to the multiprocessor system with very fine grain of computation carried out at each node, and the User-server or Client-Server model where control is totally distributed among the machines involved and the grain size is the largest. This Client-Server model is the one that is most widely implemented at this time.

A Client-server system consists of interconnected machines, each being totally autonomous. The machines that provide resources to the network, are termed as *servers* and the machines that utilize these resources, are known as *clients*. A machine that is a server, may itself be a client to some other server. Following are some major types of servers, often found in distributed systems.

1. FILE SERVER : This is a machine with large secondary storage capacity, ranging from hundreds of Megabytes to several Gigabytes of hard disk. Usually, all the file servers are accessed through explicit commands from the user. However, in a Network File System (NFS) environment, the server access may be transparent to the user.

2. BACKUP SERVER : A backup server's main goal is to provide a centralized backup storage facility to all the machines on the network. It usually has several cartridge and stream tape units that can be used by machines to do remote backup and restoration of their whole file system.

3. PRINT SERVER : This server may have several hard copy devices like plotter, dot matrix printers, high speed line printers, laser and color printers. Usually, there will be several print servers, each attached to a some of these devices.

4. DATABASE SERVER : A specialized server with large database management system and required storage space. A database can be distributed among the file servers, in which case, storage and retrieval of records will be done over the network, without the user knowing the physical location of these machines. These servers need to have enough redundancy, protection and security mechanisms to preserve the integrity of the databases.

5. COMMUNICATIONS SERVER : A communications server is usually a bridge that connects two or more similar networks, or a gateway that connects dissimilar networks. These servers allow the machines on one physical network to communicate with machines on a different network, by performing packet format and

protocol conversion. Gateways may connect LANs to long haul networks or to WANs. The communication link can be a leased phone line, satellite link, or some other long distance carrier.

6. NAME SERVER : It is a centralized unit that contains tables of names, network addresses and other information on the machines on the network and types of services offered by different machines. Internet has a domain name server system that does recursive searching for an address if it is outside its own domain.

7. COMPUTATIONAL SERVER : This is a machine equipped with vector processors, numeric co-processors etc. to provide very high speed computational capability. For instance, a super-computer on the network may be used as a computational server, and in that case, all the clients must be provided with a cross compiler that generates code that can be executed by this machine. A computationally intensive program can be sent to this server and the results obtained back.

Servers allow sharing of expensive resources in the system in a fair and efficient way. As all the machines on the network can have access to the resource, its utilization greatly increases. However, there should always be mechanisms to restrict access and security can be a major problem when everyone can have access to a resource.

4.2 Need for a Print Server

In an environment where several personal computers and workstations are connected together by a local network, the

printers attached to individual machines are the ones that are not well utilized. Attaching a printer to every workstation can be expensive especially when letter quality, laser printers and plotters are involved. Maintenance of devices scattered throughout the building, can also be a major problem. Users will have to wait for the workstation with a printer, to become free, in order to print a single file. A machine, dedicated as a print server, can be the only plausible solution under such circumstances.

Print servers are found on most of the UNIX machines attached to the network. A background process will monitor the print requests and will spool the jobs for printing. The ability to do high speed data transfer, accurate status reporting and the ability to handle multiple print requests at a time, are some of the desirable features of a print server.

4.3 Requirements for PC to PC Printing Operation

PC to PC printing is straightforward because no session setup or user authentication is required. Reliable transfer of the file from the PC to the print server is the only major operation. However, if the server PC has more than one printer attached to it, there should also be a mechanism that allows the user to specify the device required. In addition to the devices themselves, the server needs to have the drivers that allow multifont operation and such additional features.

A PC print server has been implemented utilizing the PC to PC file transfer program. The session SETUP packet contains all the

information required for printing a file, like the name of the file, file size and printer identification number. The server, after receiving the whole file, attempts to spool the file for printing and sends a STATUS packet back to the sender, informing it of the result of this attempt.

As MS DOS is not a multi-tasking operating system, the print server can handle only one print request at a time. All other requests will trigger transmission of an ABORT message to the sender, thus rejecting the transfer operation, till a file is spooled for printing.

4.4 Requirements and Mechanism of PC to UNIX Printing

A print server on the UNIX operating system is usually implemented as a *print daemon* which is a background process that is started during boot time. The process detaches itself from the controlling tty (teletypewriter or a terminal) and monitors a well known communication port. When it receives a print request, it forks off a child process and reverts back to its monitoring job. The child process accepts the print request and performs user authentication before receiving the file for printing. The result of the spooling attempt is sent back to the client, and the child terminates after ending the session. The multitasking support of the UNIX system allows the server to handle several print requests at the same time.

In the current implementation, the daemon assumes a user id (identification number) of 'puser', a user created for print serving

purposes. This scheme allows system administrators to limit access to only a few printers in the system. Another alternative is to make the print server run with the root id, at the highest possible privilege level, in which case it can have access to all the printers in the system. However, the latter scheme is not desirable because of the security risks involved. The possible error messages returned to the client are, 'no access to the printer', 'user authentication failed', 'error in storing the file', etc.

4.5 Security and Authentication Issues

When a print server is implemented on UNIX, all the security aspects of a multi-user system will have to be carefully considered. No user can logon to the system unless he/she has an account and a valid password. If the print server needs to serve only the registered users, provision should be made to obtain the user name and the password before accepting the print request. The user name is to be compared with that in the password file and the password is to be verified by calling the related system routines. This approach, though more restrictive, has advantages in that each user will be able to print only on the printers allowed for him/her and no other. The child created by the print server will assume the id of the user who is sending the file and all the related aspects of group checking etc. are handled by the system itself. The current print server is run with the id of the 'puser' and hence will have access to only the printers that are allowed for 'puser'.

CHAPTER 5

PERFORMANCE EVALUATION AND ANALYSIS OF PROTOCOLS

5.1 Analysis of Stop and Wait Flow Control Scheme

A protocol can be analyzed by taking into consideration, the size of the header it uses, the way acknowledgement is received, the network and processing delays involved in dispatching a packet, the retransmission schemes used, and time required to process acknowledgements. We choose the following notations for the parameters that are required for the analysis :

F - size of the file to be transferred, in bytes

Pf - number of data bytes sent by FFTP in a single ethernet frame

Pt - number of data bytes sent by TCP in a single ethernet frame

Hf - FFTP header size

Ht - TCP header size

p - percentage of packets lost by collision

t - round trip delay on ethernet

Tf - time required to transmit one FFTP packet

Tt - time required to transmit one TCP packet

Ta - time required to transmit an ACK packet

Of - machine overheads (processing) for one FFTP packet

Ot - machine overheads for one TCP packet

Oa - overheads associated with ACK processing,

Os - storage overheads

The total time T1 required for confirmed delivery of one FFTP packet is given by : $T1 = Of(tx) + Tf(tx) + t + Tf(rx) + Of(rx) + Oa$

which is the sum of correspponding delays at the transmitter (tx) and the receiver (rx).

The total number of data packets to be transmitted to send the file F is given by :

$$N1 = F/Pf$$

= number of acknowledgements.

The number of data packets lost due to collision = $p(N1) = p(F/Pf)$ As one ACK is sent for each data packet that is successfully received, number of ACKs = $N1(1-p)$ and out of these, $pN1(1-p)$ may be lost again. A lost data packet as well as a lost ACK packet eventually requires retransmission. These two events are mutually exclusive in that a lost data packet can not generate an acknowledgement, hence the probability of both happening for a packet is zero. Thus the total number of packets to be transmitted in the presence of collisions is given by :

$$Nf = N1 + p(N1) + pN1(1-p)$$

$$= N1(1+2p-p*p)$$

However, again p percentage of retransmitted packets can get lost, as can be their ACKs. This gives rise to an infinite series which is a geometric progression with common multiplier = 2p-p*p. Hence

$Nf = N1 + N1(2p-p*p) + N1(2p-p*p)(2p-p*p) + \ldots$ which can be simplified to $Nf = N1 (1 + (2p-p*p) + (2p-p*p)(2p-p*p) + \ldots)$

which converges to the following equation for $0 <= p < 1$:

$$Nf = (F/Pf)*(1/(1-2p+p*p))$$

Now, the total time taken to transmit a file of size F in presence of collisions is :

$Tfftp = (2Of + 2Tf + t/2)(F/Pf)*(1/(1-2p+p*p)) + Os$

$+(2Oa+t/2+2Ta)(F/Pf)*(1-p)/(1-p+p*p)$, considering the time required for data packets as well as ACK packets.

5.2 Analysis of Sliding Window Flow Control Scheme

Using the above notations again, this time considering TCP parameters, we have : $N2 = F/Pt$

Assuming that TCP transmits with a window that is W packets wide, (which may not be a very realistic assumption because the window size can vary dynamically) and one ACK is sent per every window, the number of ACK packets differs significantly from the number of data packets, given by : $N3 = N2/W = F/(Pt\,W)$. The total number of data packets lost due to collision is given by :

$Nc = pN2$ and hence only $(1-p)N2$ packets reach the destination. Totally $(1-p)N2/W$ ACKs are sent, out of which,
$N4 = p(1-p)N2/W$ ACKs are lost.

However, there is another important fact to be considered here. As one ACK is sent for W packets in the window, a lost ACK causes all the W packets to be retransmitted. Hence N4 ACKs lost, cause WN4 retransmissions. As the probability of any of the W packets in the window getting lost is the same, and a lost packet causes retransmission of that packet to the end of the window, on

an average, pN2 lost data packets cause pN2W/2 retransmissions. Thus, a total of

N2 + pN2*W/2 + p(1-p)N2/W packets will be transmitted at the first instant. Applying the same error criterion to the retransmitted data packets and ACKs, we get an infinite series which is a geometrical progression with common multiplier given by (pW/2 + p - p*p).

Therefore,

$$N_t = (F/P_t)(1+(pW/2 + p - p*p) + (pW/2 + p - p*p)(pW/2 + p - p*p) + \ldots)$$

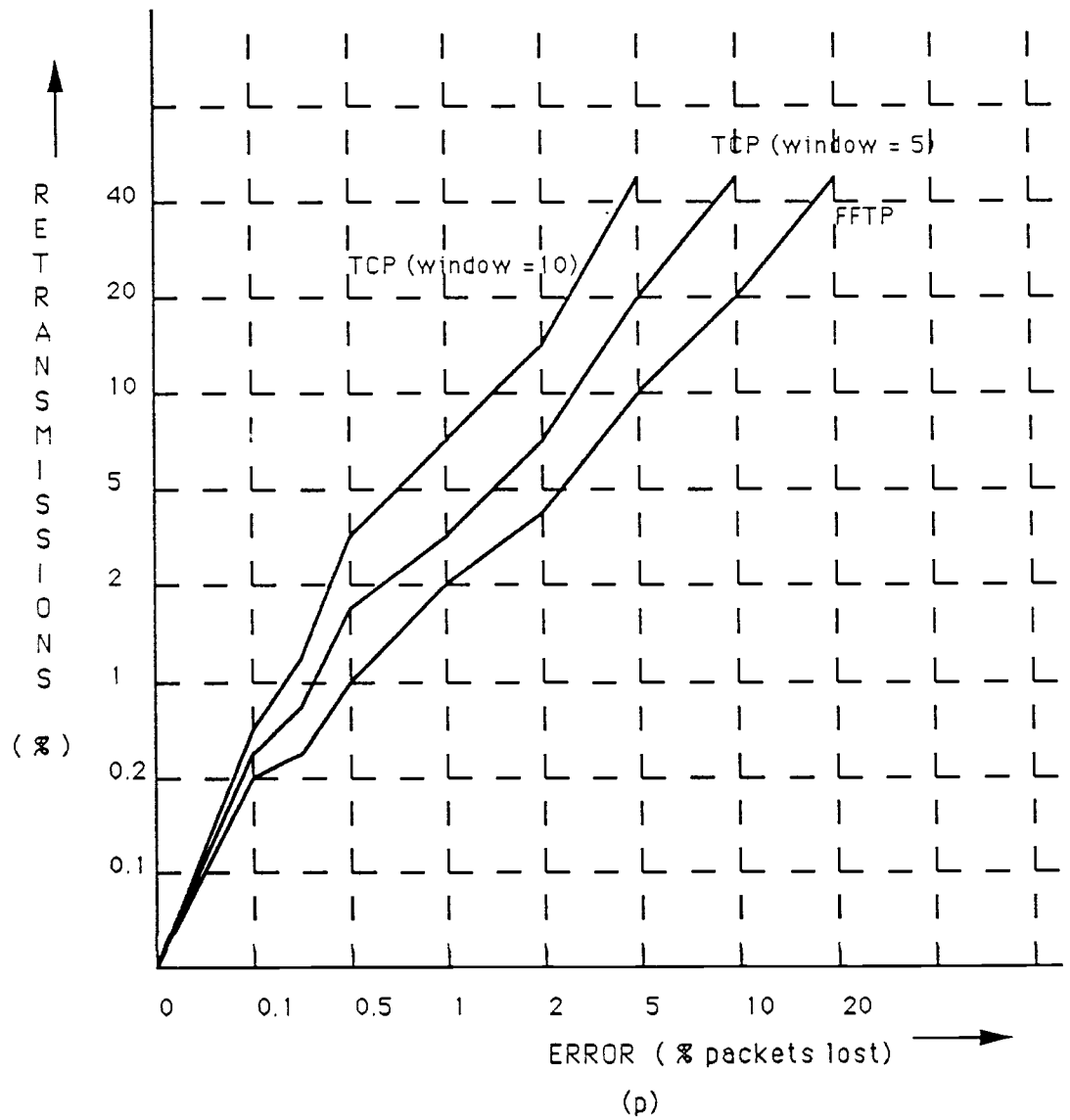
or $$N_t = (F/P_t)(1/((1 - pW/2 - p + p*p))$$

and $$T_{tcp} = (F/P_t)(2O_t + 2T_t + t/2)(1/(1 - pW/2 - p + p*p)) + (F/(WP_t)(1-p)/(1-p+p*p))(O_a + t/2 + T_a) + O_s$$

Comparing the equations for N_f and N_t, we see that for small probability p (low loads), the two protocols tend to have the same number of retransmissions. However, as the term (1-pW/2) appears in the denominator, the error probability is magnified by the window size before affecting the number of retransmissions. On the other hand, the term (1-2p+p*p), appearing in the denominator of the stop and wait protocol, does not have such a magnification factor. Figure 5.1 shows that, the number of retransmissions caused by a sliding window increase with the size of the window, and are always more than those caused by the stop and wait scheme for a particular value of error rate.

Figure 5.1

PERFORMANCE OF FTP AND FFTP WITH VARYING ERROR RATES (Calculated)



5.3 Network, Transmission, and Machine Delays

Comparison of the total time taken by each of these protocols, for transfer of F bytes of data can be realistic only if the delays caused by the machine as well as the network are taken into consideration.

An IBM PC/AT with a Intel 80286 CPU running at 8MHz clock, has the following timing restrictions.

Disk to memory transfer : 110 K Bytes/sec
Time required to transfer 1Kb : 10 mSec
Average number of clock cycles per call : 130 (approx. for saving all the registers)
Average instruction execution time : 750 nSec (6 clocks)

Corresponding network parameters are,

round trip delay on ethernet : 45 microSec
minimum packet size : 512 octets
minimum packet transmission time : 51 microSec
maximum packet transmission time : 1.25 milliSec (ms)
time required to transmit 1Kb of data : 0.833 ms.

Considering a no-collision transfer, The total time taken by FFTP, for transferring 1.5 Kb using the stop-and-wait scheme, is given by:

Ts = (10 *1.5)ms (retrieval time at tx) + (0.833*1.5)ms (t-tx) + ((0.833*1.5)ms (t-rx) + 0.05ms (round trip) + 0.05ms (ack-tx) + 0.05ms(ack-rx) which is 32.55 ms.

It will take 21.7 mSec for 1Kb to be transferred, which is equivalent to a maximum data transfer rate of 46Kbps (without considering computational overheads).

On the other hand, the total time taken by TCP, using the sliding window scheme for transferring 0.5Kb is given by :

Tw = 5ms (retrieval-tx) + 5ms (storage-rx) + 0.4ms (t-tx)
+ 0.05ms (round trip)
= 10.45 ms.

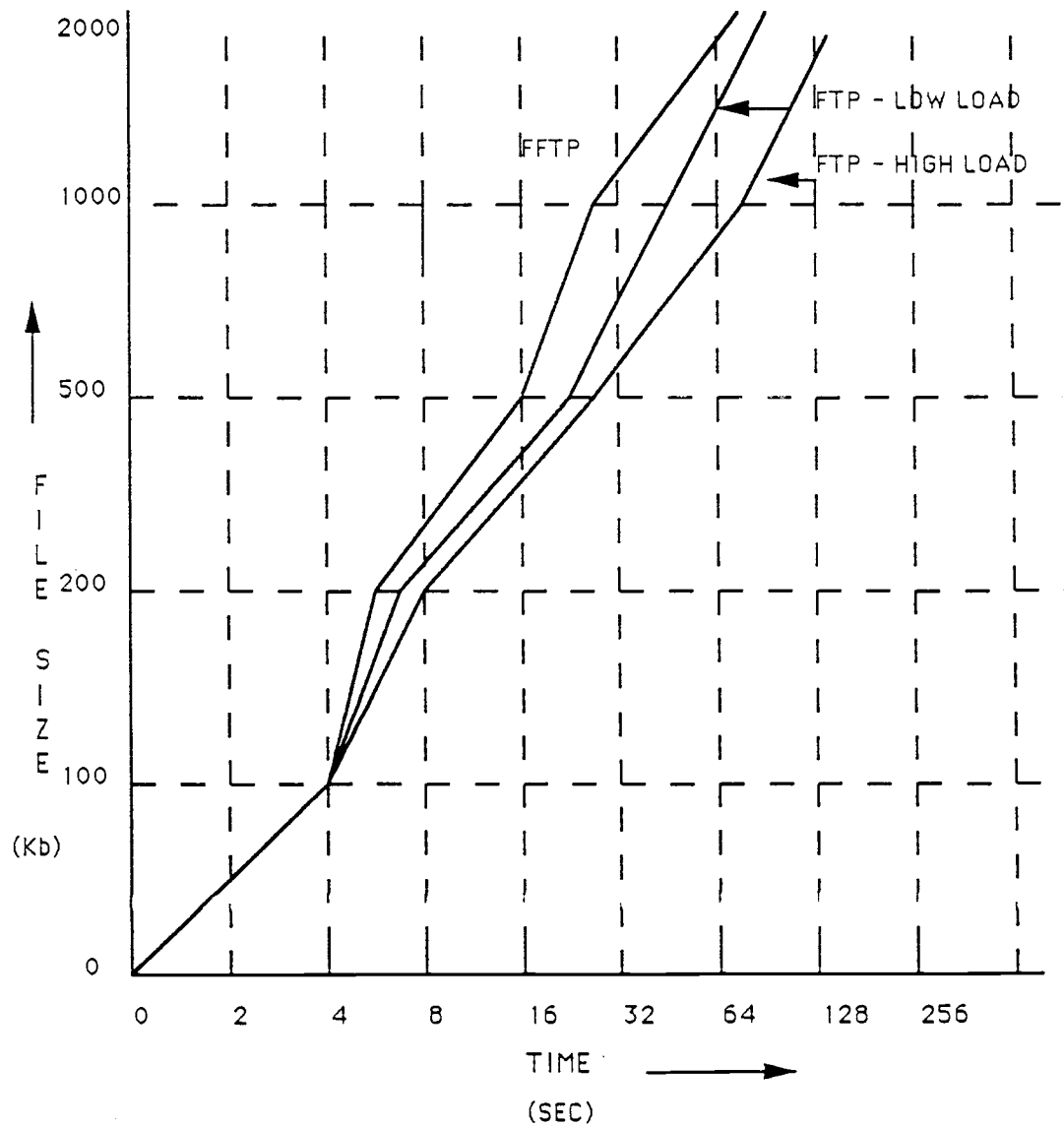
This corresponds to 20.90ms/Kb or a maximum data rate of 47.84Kbps (without considering computational overheads).

A comparison between these two indicates that TCP gains only a marginal advantage over FFTP (4% faster), in the absence of collisions. This can be attributed to the large delays involved in the storage and retrieval of data in a microcomputer environment, which nullifies the biggest advantage of the sliding window.

These results also indicate that the network delay is negligible compared to the storage and processing delays. We have seen from previous references that the sliding window is not very effective if the transmission is limited by machine performance rather than by the network delay. In fact, this result also warns us against using TCP as it is, on a local ethernet. Figure 5.2 is a graph that compares the time taken by each of these protocols to transfer files of different sizes, under varying network loads. It can be observed that TCP overheads deviate more and more from FFTP overheads as the file size increases. Timing measurements that use small file sizes, may not be accurate enough to make a good comparison. For this particular observation, the network was

Figure 5.2

PERFORMANCE OF FTP AND FFTP WITH VARYING LOAD CONDITIONS



flooded with dummy packets to simulate various loading conditions. Low load condition implies 300 packets/min, medium load - up to 1500 packets/min and high load being above 3000 packets/minute, as counted by the Netwatch program. These loads do not seem to have much effect on the rate of transfer, as can be deduced from the fact that, 3000 packets/min or 50 packets/sec amounts to just 2% of the ethernet capacity.

The difference in performance can be attributed to three major factors :

1. Smaller packet size used by IP for sending TCP segments
2. Inefficient flow control algorithm in a collision environment
3. Large header sizes that reduce the transfer efficiency.

Untill now, all the computations were done without considering the effect of header size on the data transfer efficiency and channel utilization. The following section deals with these aspects in detail.

Definite improvements in the performance of TCP have been reported by Sharon Heatley [Heatley 88] who modified the IP algorithms to use a null header with no Internet address. As Tanenbaum [Tanenbaum 81] states in his book on networking, the header size indeed has much effect on channel utilization as shown below.

Considering

A - number of bits in an ACK frame

C - channel capacity in bps

D - number of data bits per frame,

E - probability of a bit being in error,

H - number of bits in the header

F = D + H (total frame length)

I - propagation delay

P1 - probability that a data frame is lost

P2 - probability of ACK being damaged,

R - mean number of retransmissions per data frame

T - timeout interval

U - channel utilization (efficiency)

W - window size

the channel utilization turns out to be

$$U = D/(H+D)\ (1-P1)(1-P2)\ 1/(1+CT/(H+D))$$

With P1 and P2 -> 0, the utilization is given by

$$U = D/(F+CT)$$

Thus, the efficiency of transfer, in absence of errors/collisions in the channel, depends mainly on the header size relative to the frame size and the timeout interval. Since the timeout interval is based on the response time, network load and other factors, we consider header sizes for our next comparison.

TCP uses a header that has a minimum size of 24 bytes. While enclosing the TCP segments in its packet, IP places its own header which is at least 20 bytes long. An IP packet is enclosed in an ethernet frame, which adds 26 more bytes as ethernet addresses, type, checksum and synchronizing fields. That makes a

total of 70 bytes of header. Usually, IP limits the packet size to 576 bytes so that they may pass through the intermediate networks without being fragmented. This results in a data size that is 506/576 or 87.84% of the packet size. Standard TCP implementations cause extra overheads as, the maximum allowed packet size on ethernet is 1526 bytes, and only 576/1526 or a mere 38.4% of the actual capacity is utilized while transmitting one frame.

On the other hand, FFTP uses just 4 bytes of header with transport, network and all other layers added together, in addition to a packet size of 1526 bytes. This enables it to keep the header overheads to just (26 + 4)/1576 = 2.1%. Thus, FFTP gains another advantage over FTP that uses TCP/IP for communication. Figures 5.3 and 5.4 show a TCP packet and a FFTP packet respectively, enclosed in an ethernet frame before transmission.

Figure 5.3

A TCP PACKET ENCLOSED IN AN ETHERNET FRAME

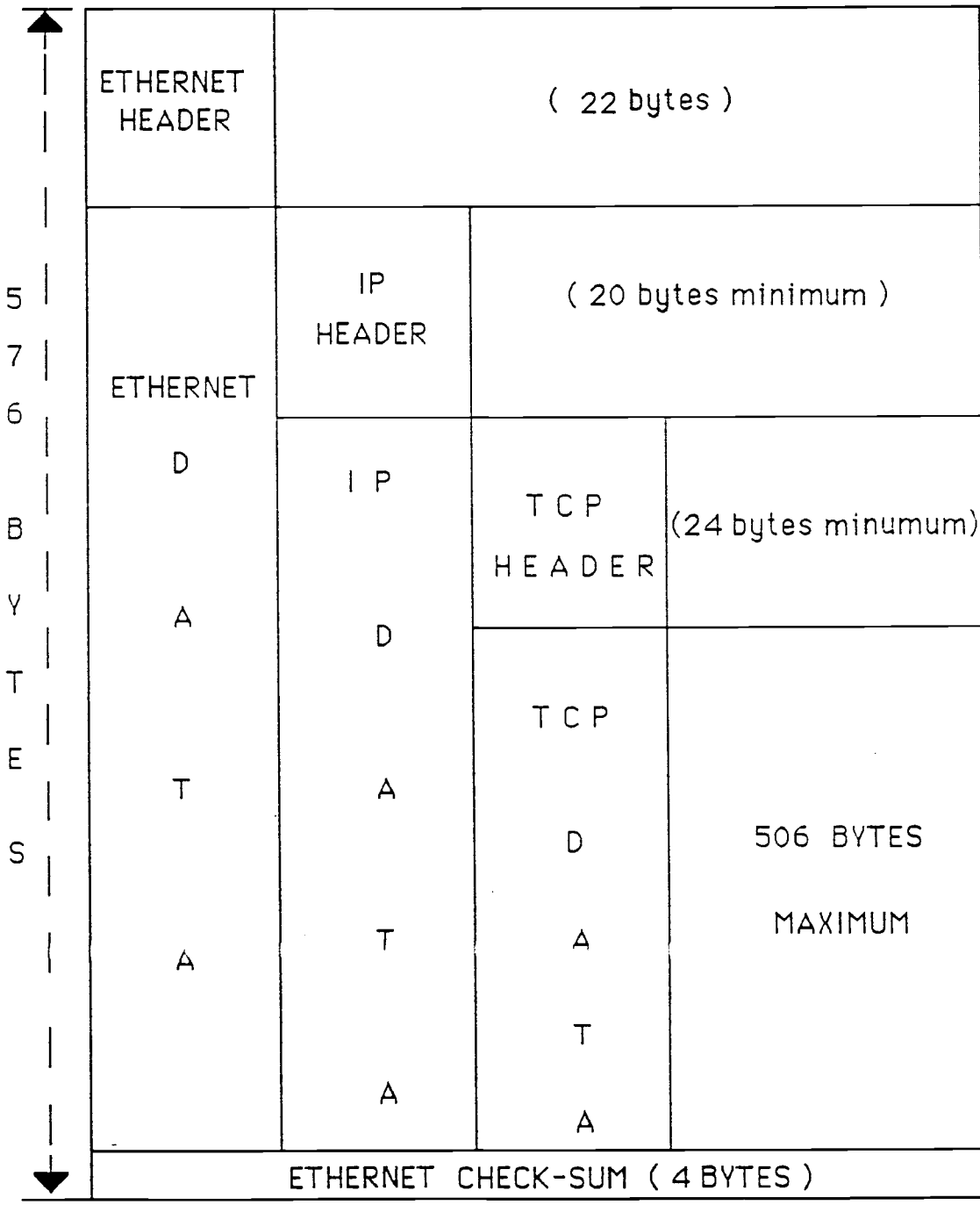
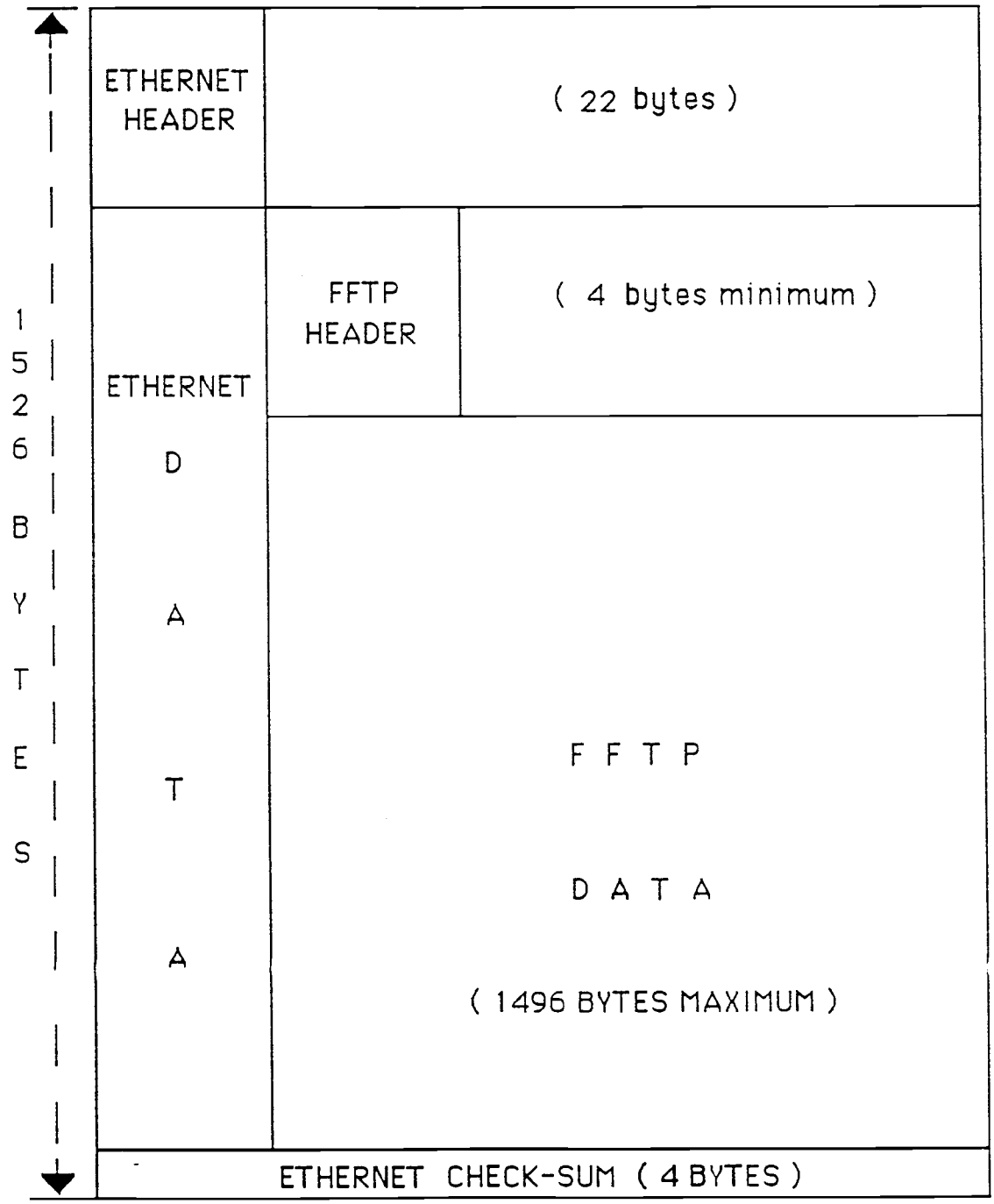


Figure 5.4

A FFTP PACKET ENCLOSED IN AN ETHERNET FRAME



CHAPTER 6

SUMMARY AND CONCLUSIONS

6.1 Observations

This work demonstrates that a general purpose protocol can be inefficient when used in a LAN environment and also an approach to design a protocol for maximum efficiency. In addition to theoretical considerations, implementation of this protocol was done by a program to effect file transfer and print serving. This program, called FFTP, was found to be faster than FTP under all testing conditions. Its performance too, was constant with varying network load and error conditions. In addition to its academic value, FFTP also has great utility value. Here are the comparisons between FFTP and FTP at a glance.

Table 6.1 : Performance of FFTP and FTP with Varying Machine Speeds

Speed of Operation	80286 (8MHz)		80386 (25MHz)	
	FFTP	FTP	FFTP	FTP
Maximum (low load)	33 Kb/s	31 Kb/s	41.6 Kb/s	33.7 Kb/s
Minimum (high load)	26 Kb/s	18 Kb/s	38.5 Kb/s	20.7 Kb/s

FFTP was observed to be at least 25% faster than the FTP.

These results also indicate the extent of improvement in protocol performance when a machine of higher speed is used. They also provide us a hint that the processor speed may not really be a limiting factor.

The implementation includes a new technique called 'dynamic timeout scheme' which helps the program to adopt itself to varying speeds of transmission. Modular program design with reduced header size, increased packet size and low overheads achieves the goal without any difficulty. Portability was also given some consideration during the design process. Hints for porting FFTP to the Unix system and installation aspects are discussed in Appendix C.

6.2 Scope for Further Enhancements

Though this is a complete work in itself, it still provides several opportunities for improvements.

1. Instead of the currently used double buffer scheme, a linked list of buffers can be used to speed up the data transfer operation. These buffers may be filled/emptied while waiting for acknowledgements or next packets.

2. A Negative Acknowledgement scheme (NAK) can be used to further reduce the number of retransmissions. In this scheme, not only will the number of packets exchanged between the machines be reduced, but also the effect of collisions on the traffic

will be reduced. NAK can be seen to be more efficient than a sliding window flow control.

3. Provision could be made to apply data compression algorithms to a file before transmitting it, and to decompress it at the receiver. There are well known algorithms that compress text files by as much as 50%. This feature may be implemented as an enhancement to the presentation layer. Though computationally intensive, this procedure serves to reduce the network traffic to a great extent. Bitmap image files readily yield themselves to such data compression.

BIBLIOGRAPHY

[Comer 87] Comer Douglas E. , "Internetworking with TCP/IP, Principles, protocols and architecture", Prentice Hall, Inc. pp 114-115, 1987

[Heatley 88] Heatley Sharon and Dan Stokesberg, "Measurement of a transport implementation over IEEE 802.3 LAN" IEEE Computer Networking Symposium 1988

[Metcalfe 76] Metcalfe R.M. and Boggs B.R., "Ethernet : Distributed packet switching for local computer networks", Communications of ACM vol 19, pp 395-404, Jul 76

[RFC 793] Request For Comments 793, "Transmission Control Protocol", DARPA Internet program : protocol specifications, Sep 1981

[Sunshine 77] Sunshine Carl A. , "Efficiency of Communication protocols for computer networks", IEEE Transaction on Communication, com 25(2), pp 287-293, Feb 77

[Tanenbaum 81] Tanenbaum Andrew S. , "Computer Networks", Prentice Hall, Inc., pp 174-177, 1981

[Vinyes 86] Vinyes J., Vazquez E. and Mills K. , "Throughput analysis of a transport protocol", IEEE Symposium on Computer Networking, 1986

[Zimmermann 80] Zimmermann H. : "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnections", IEEE Transactions on Communications vol COM-28, pp. 425-432, April 1980

APPENDICES

APPENDIX A

NCSA Packet Driver Specifications

Revision 1.08 : December-12-1988
Developed by : FTP Software, Inc. P.O.Box 150 Kendall Sq.
Boston, MA 02142, (617) 868-4878

1 Introduction and Motivation

This document describes the programming interface to PC/TCP packet drivers. Packet drivers provide a simple common programming interface that allows multiple applications to share a network interface at the data link level. The packet drivers demultiplex incoming packets among the applications by using the network media type field. Through the use of the packet driver, the actual brand or model of the network interface can be hidden from the applications.

The packet driver provides calls to initiate access to specific packet type, to end access to it, to send a packet, to get statistics on the network interface and to get information about the interface.

Protocol implementations that use the packet driver can completely coexist on a PC and can make use of one another's services, whereas multiple applications which do not use the

driver may not coexist on one machine properly. Through use of the packet driver, a user could run TCP/IP, DECnet, and a proprietary protocol implementation such as Banyan's, LifeNet's, Novell's or 3COM's without the difficulties associated with pre-empting the network interface.

Two levels of packet drivers are described in this specification. The first is the basic packet driver, which provides minimal functionality but should be simple to implement and which uses very few host resources. The basic driver provides operations to broadcast and receive packets. The second driver is the extended packet driver, which is a superset of the basic driver. The extended driver supports less commonly used functions of the network interface such as multicast, and also gathers statistics on use of the interface and makes these available to the application. All basic packet driver functions are available in the extended driver.

2 Identifying network interfaces

Network interfaces are named by a triplet of integers, <class, type, number>. The first is the class of the interface. The class tells what kind of media the interface is for: DEC/Intel/Xerox Ethernet, IEEE 802.3, IEEE 802.5/TokenRing, ProNET-10, Broadband Ethernet, Appletalk, serial line, etc.

The second number is the type of interface: this specifies a particular instance of an interface supporting a class of medium.

Interface types for Ethernet might name these interfaces: 3COM 3C501 or 3C505, Interlan NI5010, Univation etc.

The last number is the interface number. If a machine is equipped with more than one interface of a class and type, the interfaces must be numbered to distinguish between them.

The type 0xFFFF is a wildcard type which matches and interface in the specified class. It is unnecessary to wildcard interface numbers, as 0 will always correspond to the first interface of the specified class and type.

This specification has no provision for the support of multiple network interfaces. We feel that this issue is best addressed by loading several Packet Drivers, one per interface, with different interrupts (although all may be included in a single TSR software module). Applications software must check the class and type returned from a driver_info() call already, to make sure that the Packet Driver is for the correct media and packet format. This can easily be generalized by searching for another Packet Driver if the first is not of the right kind.

3 Initiating driver operations

The packet driver is invoked via a software interrupt in the range 0x60 through 0x80. This document does not specify a particular interrupt, but describes a mechanism for locating which interrupt the driver uses. The interrupt must be configurable to avoid conflicts with other pieces of software that also use software interrupts. The program which installs

the packet driver should provide some mechanism for the user to specify the interrupt.

The handler for the interrupt is assumed to start with 3 bytes of exectuable code; this can either be a 3-byte jump instruction, or a 2-byte jump followed by a NOP (do not specify "jmp short" unless you also specify an explicit NOP). This must be followed by the null-terminated ASCII text string "PKT DRVR". To find the interrupt being used by the driver, an application should scan through the handler for vectors 0x60 through 0x80 until it finds one with the text string "PKT DRVR" in it.

4 Programming interface

All functions are accessed via the software interrupt determined to be the driver's via the mechanism described earlier. On entry, register AH contains the code of the function desired.

The handle is an arbitrary integer value associated with each MAC-level demultiplexing type that has been established via the access_type call. Internally to the packet driver, it will probably be a pointer, or a table offset. The application calling the packet driver cannot depend on it assuming any particular range, or any other characteristics.

Note that some of the functions defined below are labelled as extended driver functions. As their implementation is optional, the programs wishing to use these functions should use the

driver_info() function to determine if they are available in a given packet driver.

4.1 Entry conditions

FTP Software applications which call the packet driver are coded in Microsoft C and assembly language. All necessary registers are saved by FTP's routines before the INT instruction to call the packet driver is executed. Our current receiver() functions behave as follows: DS and the flags are saved and restored. All other registers may be modified, and should be saved by the packet driver, if necessary. Processor interrupts may be enabled while in the upcall, but the upcall doesn't assume interrupts are disabled on entry. On entry, receiver() switches to a local stack. Current FTP Software receiver() routines may modify all registers except DS, so the caller must save and restore any that must be preserved across the call.

4.2 Byte ordering

Developers should note that, on many networks and protocol families, the byte-ordering of 16-bit quantities on the network is opposite to the native byte-order of the PC. (802.5 Token Ring is an exception). This means that DEC-Intel-Xerox ethertype values passed to access_type() must be swapped (passed in network order) The IEEE 802.3 length field needs similar

handling, and care should be taken with packets passed to send_pkt(), so they are in the proper order.

When a packet is received, receiver is called twice by the packet driver. The first time is to request a buffer from the application to copy the packet into. AX == 0 on this call. The application should return a pointer to the buffer in ES:DI. If the application has no buffers, it may return 0:0 in ES:DI, and the driver should throw away the packet and not perform the second call.

It is important that the packet length (CX) be valid on the AX == 0 call, so that the receiver can allocate a buffer of the proper size. This length (as well as the copy performed prior to the AX == 1 call) must include the Ethernet header and all received data, but not the trailing checksum.

On the second call, AX == 1. This call indicates that the copy has been completed, and the application may do as it wishes with the buffer. The buffer that the packet was copied into is pointed to by DS:SI.

802.3 vs. Blue Book Ethernet

One weakness of the present specification is that there is no provision for simultaneous support of 802.3 and Blue Book (the old DEC-Intel-Xerox standard) Ethernet headers via a single Packet Driver (as defined by its interrupt). The problem is that the Ethertype of Blue Book packets is in bytes 12 and 13 of the header, and in 802.3 the corresponding bytes are interpreted

as a length. In 802.3, the field which would appear to be most useful to begin the type check in is the 802.2 header, starting at byte 14. This is only a problem on Ethernet and variants (e.g. Starlan), where 802.3 headers and Blue Book headers are likely to need co-exist for many years to come.

One solution is to redefine class 1 as BlueBook Ethernet. and define a parallel class for 802.3 with 802.2 packet headers. This requires that a 2nd Packet Driver (as defined by its interrupt) implemented where it is necessary to handle both kinds of packets, although they could both be part of the same TSR module.

James B. VanBokkelen

(jbvb@ftp.com)

APPENDIX B : Flow Charts

Figure B.1

SEND_PACKET

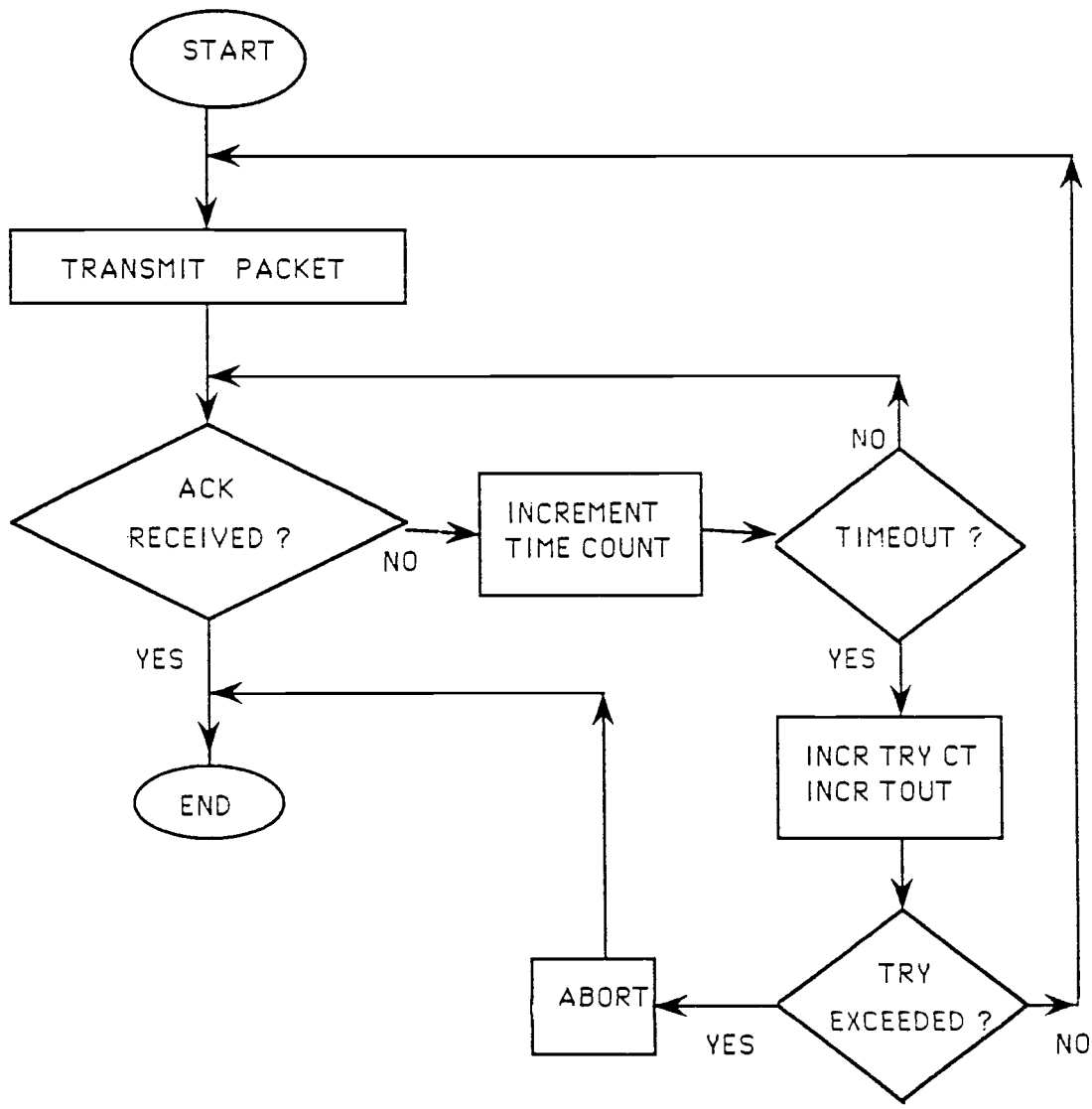


Figure B.2

SEND_FILE

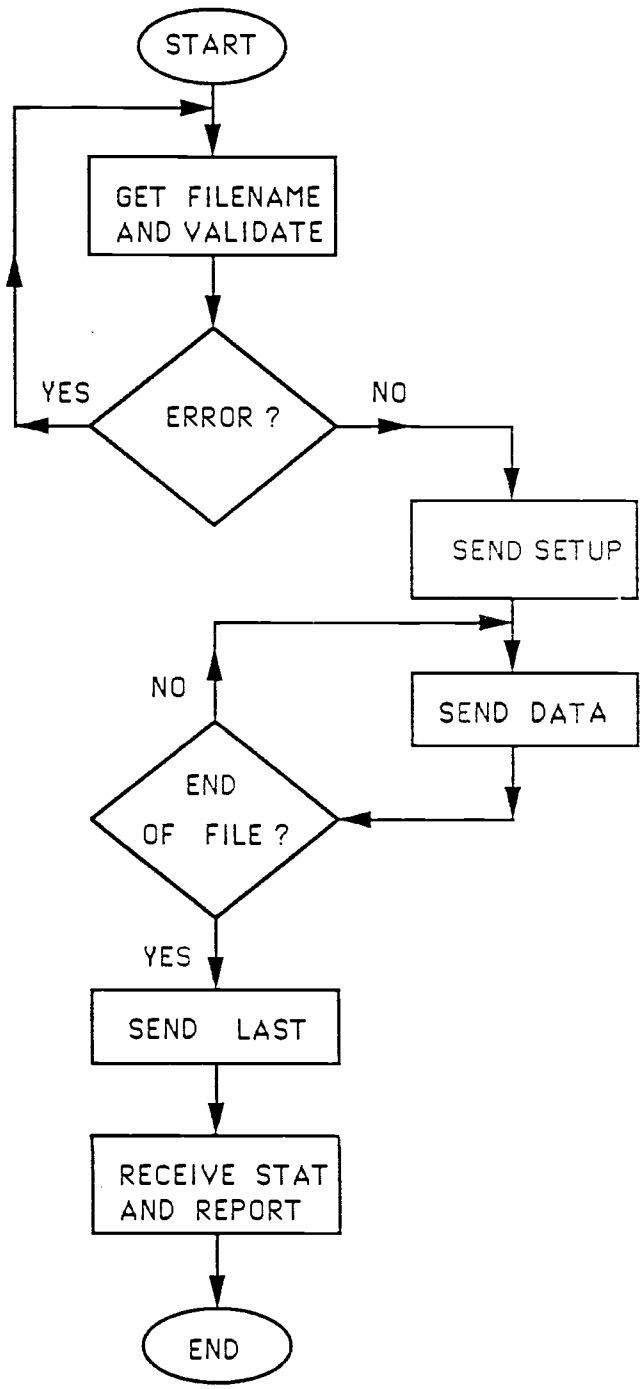


Figure B.3

RECEIVE_PACKET

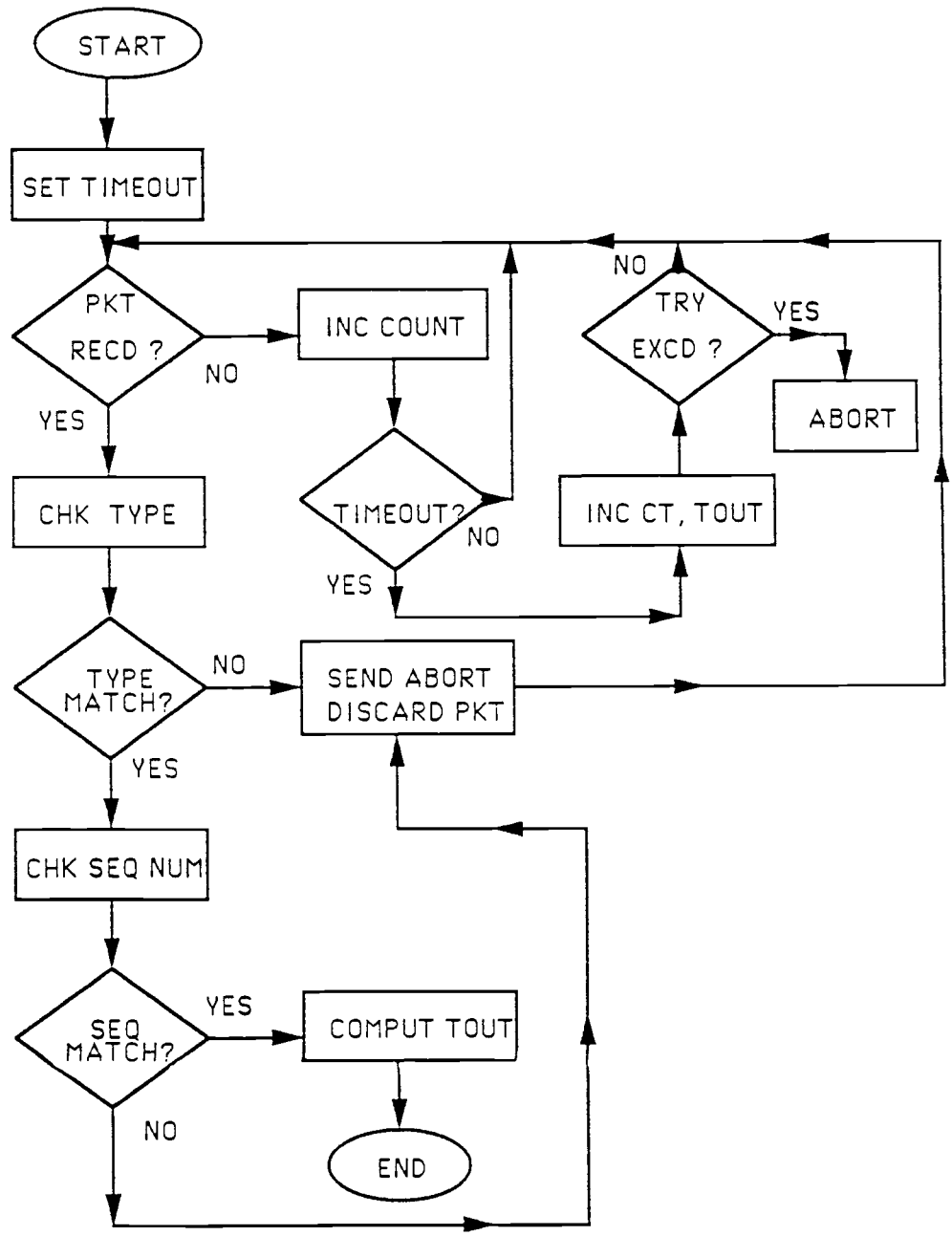
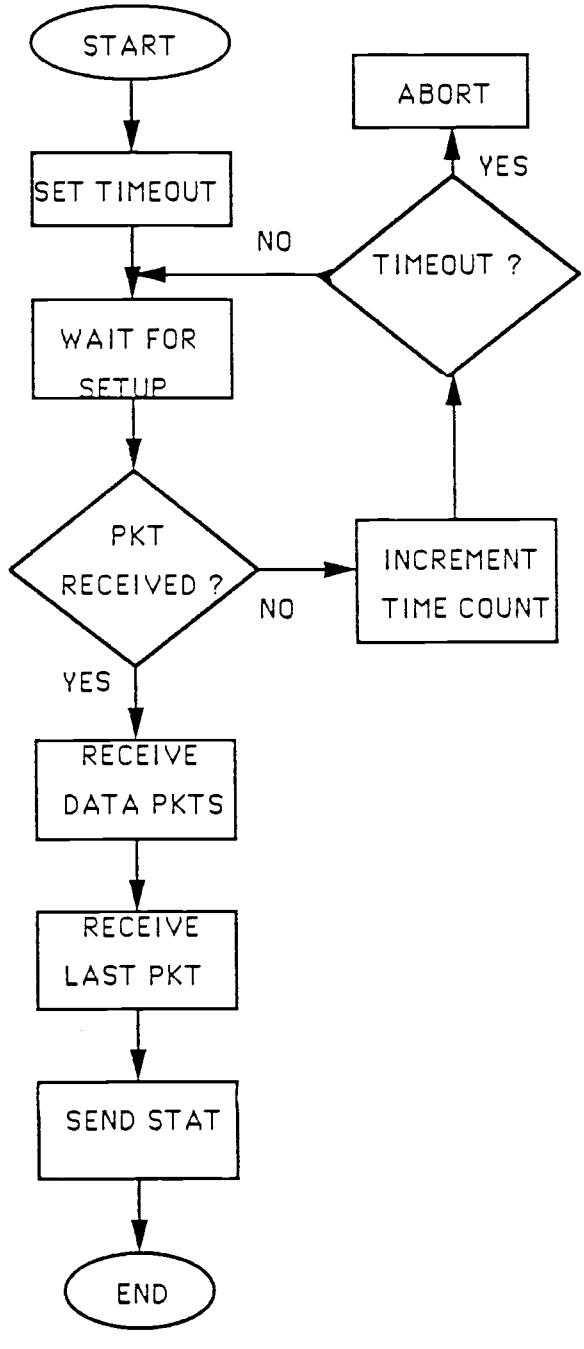


Figure B.4

RECEIVE_FILE



APPENDIX C

PORTING AND INSTALLATION OF FFTP

Hints for Porting FFTP to UNIX System

The FFTP print server is currently implemented on IBM PC/ATs. The Unix print server still uses TCP. This is mainly due to the absence of low level driver access mechanisms in UNIX. As soon as this is done, the program can be ported to UNIX without any modification. However, a layer of software is still needed to simulate the NCSA packet driver. Multiple service requests can be handled by the new print server by opening a new socket instead of a TCP socket. The following are the major requirements of a driver interface.

1. The interface should be able to provide access to ethernet packets of arbitrary 'type' field, with correct destination address.

2. The interface should be able to transmit ethernet packets handed over to it, with arbitrary destination address.

3. There should be a provision to call a user routine asynchronously when the packet with matching 'type' is received.

More requirements can be found in the NCSA Packet Driver Specifications, given in Appendix A. A name server can be

implemented as a daemon to eliminate the need for address files on each PC.

Installing FFTP on a PC

FFTP can be installed on any PC with the following accessories.

1. An ethernet interface card
2. A NCSA compatible driver for the above card

... and the following files must be present in some directory that is included in the PATH environment variable on DOS.

1. SADDRESS containing the ethernet address of the local machine in the following format :

XX a b c d e f

where XX is the machine number (1 for AT1 etc) and a..f are hexadecimal numbers representing the ethernet address. The fields must be separated by either tab or space/s

2. DADDRESS containing the number and ethernet address of all machines with which, communicaiton is desired. The file format is as shown below.

XX
y1 a1 b1 c1 d1 e1 f1
y2 a2 b2 c2 d2 e2 f2
....

where XX is the total number of machines in the file, and thee rest is similar to SADDRESS

3. The file TCPBIN.EXE if communicaiton with a UNIX machine is desired. (such a machine is currently given machine number 99)

In addition to the above, an NCSA compatible packet driver, by name WD8003E.COM, must be present in the search path if the default installation part of the FFTP program is to be used. The program tries to install the driver if it is not already installed through the AUTOEXEC.BAT file, at boot time.

Basic debugging can be carried out in case of problems by recompiling the program after setting the XDEBUG option in the header file. Currently, the program is contained in three files FFTP.C, FUNCTION.C and PDRIVER.H