

A NETWORKING INSTRUCTION AND  
RESEARCH TOOL

By

GREGG G. WONDERLY

Bachelor of Science

in Arts and Sciences

Oklahoma State University

Stillwater, Oklahoma

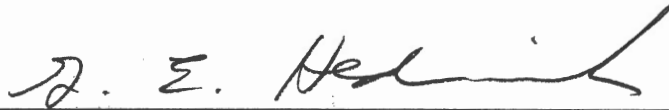
1985

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
December, 1988

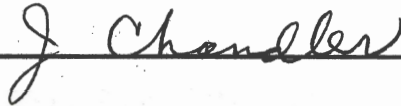
THEJD  
1988  
W8713n  
cop. 2

A NETWORKING INSTRUCTION AND  
RESEARCH TOOL

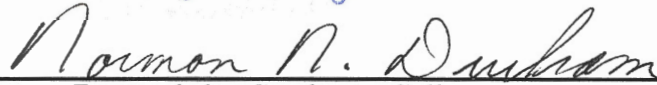
Thesis Approved:



Thesis Advisor







Dean of the Graduate College

## PREFACE

A suite of networking instructional tools is under development. This project is a joint effort between the author and the initiator, Mr. Mark Vasoll. The project promises to provide a very powerful teaching tool. All of the researchers involved in the project consulted each other concerning the fine points of the design and model. This thesis will present a portion of the project's design and implementation, so that future work can be directed to the uncompleted portions of the project.

The scope of the project is large. A complete TCP/IP implementation with user level applications ultimately is planned. The amount of time spent both on the design phase and on the partial implementation should make future work much less time consuming.

I would like to thank Mr. Vasoll for providing motivation. Most importantly, I would like to thank my family for their encouragement and my wife Tammy for waiting this long.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION . . . . .	1
Problem Statement . . . . .	2
Literature Review . . . . .	3
Thesis Overview . . . . .	5
II. AN INTRODUCTION TO TCP AND IP . . . . .	7
The Internet Protocol . . . . .	8
The Transmission Control Protocol . . . . .	8
III. REVIEWING IMPLEMENTATIONS OF TCP/IP . . . . .	10
Berkeley's 4.[23] Networking Code . . . . .	10
The User's Interface . . . . .	10
Sample Operations . . . . .	11
IV. THE PROJECT DESIGN AND STRUCTURE . . . . .	13
A High Level View . . . . .	13
Interaction with UNIX . . . . .	14
V. MODULARITY OF THE IMPLEMENTATION . . . . .	16
VI. COMPLETED WORK . . . . .	18
Interprocess Communications . . . . .	18
IP Configuration . . . . .	19
IP Operation . . . . .	21
Attach Requests . . . . .	21
Detach Requests . . . . .	22
Datagram Transmission . . . . .	22
Incoming IP Datagrams . . . . .	22
Ethernet Operation . . . . .	23
VII. FUTURE WORK . . . . .	24
The IP Module . . . . .	24
The Ethernet Module . . . . .	27
VIII. SUMMARY AND CONCLUSIONS . . . . .	28
Summary . . . . .	28
Conclusions . . . . .	29
REFERENCES . . . . .	30

Chapter	Page
APPENDICES .....	34
APPENDIX A – GLOSSARY .....	34
APPENDIX B – DIRECTORY STRUCTURE .....	35
APPENDIX C – FIGURES .....	37

## LIST OF FIGURES

Figure	Page
1. OSI Protocol Hierarchy . . . . .	38
2. Diagram of IPC . . . . .	39
3. Message passing between applications . . . . .	40
4. Sample configuration lines . . . . .	41
5. Sample alarm timer code . . . . .	41

## NOMENCLATURE

<b>ARPANET</b>	The Advanced Research Projects Agency Network.
<b>DMA</b>	Direct Memory Access.
<b>DOD</b>	The Department Of Defense.
<b>EARP</b>	The Ethernet Address Resolution Protocol.
<b>ICMP</b>	The Internet Control Message Protocol.
<b>IP</b>	The Internet Protocol.
<b>IPC</b>	InterProcess Communications.
<b>OSI</b>	Open Systems Interconnect.
<b>RFC</b>	Request For Comments.
<b>TCP</b>	The Transmission Control Protocol.
<b>TCP/IP</b>	The Transmission Control Protocol and the Internet Protocol suite.
<b>UDP</b>	The User Datagram Protocol.



## CHAPTER I

### INTRODUCTION

Networking, using high speed transmission media, is providing more flexible use of computers. The cost of high speed parallel computers currently makes it almost more cost effective to buy many small computers, and then connect them together via a high speed network, such as Ethernet. Ethernet is a high speed broadcast network capable of speeds in the one million bit per second range. The Ethernet design and specification was developed by a group of companies from the computer industry. The XEROX corporation and Digital Equipment Corporation (DEC) are two of the several companies involved in the specification of Ethernet.

The Department of Defense (DoD) funded research at the University of California, Berkeley. The research at Berkeley laid the groundwork for their TCP/IP protocol suite. TCP is an acronym for Transmission Control Protocol; IP is an acronym for Internet Protocol. These two protocols have become the backbone of a large number of high speed network applications. IP provides a very powerful packet routing mechanism. TCP provides a reliable data path by using IP for routing these packets. A third protocol, ICMP, or Internet Control Message Protocol<sup>1</sup>, provides a control mechanism which can be used by network software to communicate problems and events among the hosts on a network.

---

<sup>1</sup>ICMP is an integral part of IP, so IP really means IP and ICMP.

The DoD now utilizes these protocols in its Advanced Research Projects Agency Network (ARPANET). The ARPANET is comprised of DoD contractors who use the network to exchange information about the work they are doing. ARPANET operates over high speed leased lines for site to site communication. Within a particular organization, Ethernet is typically used to interconnect hosts. Current research of TCP/IP revolves around the 4.2 and 4.3 versions of Berkeley's UNIX, or 4.xBSD. Licensing of this software is difficult for some regions<sup>2</sup>. A new, unrestricted implementation of these protocols will make it possible for universities, limited by Berkeley's licensing, to have a networking teaching tool for laboratory use by students and faculty.

### Problem Statement

Vendor implementations of TCP/IP are proprietary, which makes it difficult for organizations or individuals to alter the behavior of the software. This limits the number of individuals who can perform research using these protocols and derivatives of them. This paper presents the introduction to a suite of programs. These programs are part of a large, incomplete project involving several individuals.

Because the programs are nonproprietary, they allow students and faculty to pursue research in the area of computer internetworking. The need for computer internetworking is self-evident. By interconnecting computers, the information and resources that are available on one computer and be shared with many others. Networking research will make it possible for better protocols to be developed for increased efficiency in computer internetworking. Students and faculty can use

---

<sup>2</sup>A 4.xBSD license binds the licensee to laws and regulations of California. Some states do not allow their institutions to sign contracts of this nature, which creates problems for universities that are governed by state law. This one fact is the original motivation for this project.

the results of this project to perform laboratory research as well as hold classroom discussions on all aspects of computer internetworking.

### Literature Review

In the DoD/ARPANET community, research often leads to the development of protocols or methodologies from which that community as a whole might benefit. The TCP/IP protocol suite is a good example. Developers often propose to make their work into a standard. It is beneficial for the developers to have the opinions of others in the community regarding their proposals. A formal document, called an RFC, or request for comment, is issued from the developer to allow others to study their proposal. The acceptance of RFC's as a standard has produced over a thousand different documents. RFC's exist that define known host names, IP addresses of ARPANET hosts, the format of electronic mail messages, as well as the hundreds of protocols in use by the DoD community today.

The RFC's numbered 791, [8] 792 [10] and 793 [11] describe the IP, ICMP and TCP protocols respectively. These documents serve as a reference as well as the standard for development of new implementations of these protocols, and are therefore required reading for those wishing to implement these protocols. Publically available publications regarding specific implementations of these protocols are few in number. Cerf and Kahn [3] specify a protocol that was at the forefront of development of internetworking protocols. Cerf was Program Manager and Principal Scientist at the Defense Advanced Research Projects Agency, DARPA from 1976 to 1982. He was responsible for the oversight of the research programs in packet switching technology [16].

The OSI protocol model is a seven layer model as shown in figure 1<sup>3</sup>. Cerf and Cain [2] discuss the DoD Internet Architecture Model (the ARPANET). Their discussion revolves around the presentation of what the DoD feels that it needs in an internetworking protocol suite. The OSI model is also discussed. It is shown how the DoD model provides certain functionality that the OSI model cannot provide due to the separation of certain functions into multiple layers where the DoD model uses only one. The problem as stated is that there are many operations that take place in internetworking, and these operation often differ drastically in data format and transmission requirements. A remote terminal login facility verses a file transfer facility is a good example. The remote terminal facility will require quick timeouts and small data packets to provide resonable response times to users requests. Whereas a file transfer facility will typically block data and take full advantage of a networks transmission speed. Cerf and Cain also present some areas that they feel need attention in the DoD model as well as the OSI model. Examples given are security, and large scale network effeciency.

Schwartz [14] discusses various aspects of computer internetworking, compares TCP with the OSI<sup>4</sup> Transport Protocol Class 4 specification. Schwartz discusses many aspects of the two protocols specifications, and provides some excellent background on TCP. Schwartz's references contain many papers on TCP/IP and current as well as previously studied problems. Postel [9] discusses the interconnection of computer networks through the use of protocols from the third and forth levels of the OSI model. TCP and IP are given as examples of protocols that provide the necessary functionality for computer interconnection.

---

<sup>3</sup>All figures are given in appendix C.

<sup>4</sup>The TCP/IP protocol suite corresponds to the level three and level four protocols in the OSI hierarchy.

One of the problems discussed by Postel is that of providing protocol translation facilities to allow one network to send and receive data from another network which uses different protocols at some levels. This discussion provides some opinions regarding whether a standard such as the OSI or DoD architecture should be universally adopted. If there was a single network architecture in use by all interconnected networks, then protocol translation schemes would not be necessary. On the otherhand, some machines and networks require different protocols based on the capability and reliability of the underlying transmission media. More research is needed in this area to discover a solution that satisfies the all the needs of the networks and computers attached.

Comer [4] discusses the use of TCP/IP in a research operating system that he has designed called XINU, an acronym standing for the phrase, XINU is not UNIX. This work has some relation to this project. However the majority of operating system issues discussed revolve around the process relations of XINU which are very different from System V UNIX<sup>5</sup>; most notable are the interprocess communications and process control mechanisms that XINU allows. Because most XINU implementations do not use hardware protection<sup>6</sup>, the XINU implementation is more oriented to standalone applications and not multiuser environments. However, some aspects of efficiency and the communication models used are of general interest.

### Thesis Overview

The above mentioned papers show a portion of the areas in internetworking that are actively being researched. This project will make it possible for many

---

<sup>5</sup>All XINU processes share one address space, whereas UNIX processes are protected from each other.

<sup>6</sup>XINU is open in the sense that it offers no hardware protection. User processes are allowed to manipulate the computer hardware in any way they want.

more individuals to participate in research in the area of computer internetworking. This paper serves as a guide to the initial project design. The paper describes many aspects of TCP/IP, and the project. Some aspects of another TCP/IP implementation are discussed. The project design and structure is reviewed, and finally, the work left to do and some ideas about that work are given.

## CHAPTER II

### AN INTRODUCTION TO TCP AND IP

The TCP and IP protocols work together to provide a reliable data transmission path. The application of these two protocols is not limited to a local area networks (LAN), because of IP's routing mechanisms. Instead, the TCP/IP protocol suite can be used to connect LAN's together. Many LAN's can, in turn, be connected via gateways to form whatever topology is desired. The gateways either can be hosts available for use by people, or they can be cheaper, dedicated machines. Some IP gateways available now learn the topology of a network by examining the headers of datagrams as they pass through the gateway. The header tells where the datagram originated. Using this piece of information, the gateway can create a table that tells it which networks which hosts are on so that subsequent datagrams to those hosts do not cause unnecessary traffic on other networks.

This strategy allows the gateway to go about its duties from the start without having to be told which host is on what network interface. A gateway generally has two or more physical networks connected to it. It can broadcast packets destined for hosts about which it does not know, to all interfaces. If the host is somewhere on one of the networks that the gateway is serving, the packet will reach it. The IP protocol does not dictate this behavior, rather the gateway merely is receiving and retransmitting IP datagrams<sup>7</sup>. The fact that TCP/IP allows for duplicate packets

---

<sup>7</sup>A datagram is a block of data or packet.

makes it possible for this to work. If a host is on two networks served by a common gateway, it is possible for the host to receive two copies of a datagram. The protocol specification allows for this, and the extra datagram is discarded.

### The Internet Protocol

The IP protocol provides the basic routing of data through a network. It does not provide sequencing, flow control (ICMP does allow some flow control to be done), or data-reliability. This task is left to higher level protocols. Because of this inherent unreliability, IP is used by higher level protocols which provide the necessary services. IP makes use of lower level protocols to access the physical network media.

Thus, through the use of a gateway, IP can cross from one network to another. If there are limitations on the size of datagram a particular network media can handle, IP's fragmentation capabilities make it possible for a packet to be broken into pieces for later reassembly. Postel [8] discusses all aspects of IP in depth.

### The Transmission Control Protocol

The TCP protocol is a highly reliable host-to-host transport in packet-switched networks. It is a connection-oriented protocol designed for use in a layered hierarchy of protocols. The protocol provides for reliable interprocess communications between pairs of processes in host computers. Very few assumptions are made about the reliability of the underlying protocols.

TCP makes use of a protocol comparable to the Internet Protocol described in [8]. This type of datagram transport mechanism satisfies the need to break



large transactions into small pieces so that unreliable networks will require fewer retransmissions. TCP assumes that the underlying protocol is IP. Postel [11] discusses all aspects of TCP in depth.

## CHAPTER III

### REVIEWING IMPLEMENTATIONS OF TCP/IP

The beginnings of TCP/IP networking are nestled in DOD funded research. A great deal of the implementation was done by California Universities. Students at the University of California at Berkeley wrote the implementation that is considered the standard today (for UNIX based systems).

#### Berkeley's 4.[23] Networking Code

Berkeley's networking code is part of the 4.[23]BSD version of the UNIX<sup>8</sup> operating system. It is an integral part of the operating system and, as such, uses CPU time. As a result, it impacts the users of the system. Since Berkeley's code is in the operating system, it knows when a process exits from the system. This allows the processes network resources to be freed, causing connections to other hosts to be terminated with proper reasons, instead of terminating with an all encompassing timeout message.

#### The User's Interface

The interface to Berkeley's TCP/IP makes use of UNIX file descriptors so that once a connection is open, elementary reads and writes can be used to perform the

---

<sup>8</sup>UNIX is a trademark of The American Telephone and Telegraph Corporation.

communications. The open file descriptor is used by the kernel to close a connection when a process exits. A connection to the network is initiated by the system call, `socket(2)`<sup>9</sup> [18]. The `socket` call creates an endpoint of communications. It returns a file descriptor that describes the type of socket that the user requested.

The services available are stream oriented and datagram oriented. An argument to the `socket` call selects which protocol is used for a particular type of service. The type of services are limited to a single stream service provided by TCP and a datagram service provided by the User Datagram Protocol. Additional services may be added in the future. The User Datagram Protocol runs on top of IP as TCP does, but does not provide an error free data path. It provides a port oriented service as TCP does. The datagram service allows other protocols to make use of the network without submitting to the overhead of the stream service.

### Sample Operations

Once the socket is open, the user can perform many different functions using the descriptor to configure the particular connection endpoint. The `setsockopt(2)` and `getsockopt(2)` system calls allow the user process to manipulate the options available at each protocol level including socket level options. There are many other system calls which provide many different functions. `Bind(2)` is used to assign a name to a socket in the domain of the service associated with the socket. A `listen(2)` call allows an application to accept requests to make use of the service the name represents. An `accept(2)` call does the actual accepting i.e. requests for the service will be answered and the process making the `accept` call will be notified

---

<sup>9</sup>The number in parenthesis is the UNIX manual section where the particular name preceding it is described.

of the request. The `listen(2)` call sets up kernel data structures to make a queue for these `[accept]` requests while they are waiting to be serviced.

A simple send and receive mechanism also is available. The `send(2)` and `recv(2)` calls may be used following a `connect(2)` call although neither is necessary. `Connect(2)` connects an application to a socket created using the `bind(2)`, `listen(2)` and `accept(2)` calls. A variation of `send(2)` allows a user to select a specific address to which to send. A variation of `recv(2)` allows the user to `recv` a message from an arbitrary address. Finally, because the socket descriptor can be used as a file descriptor, it is possible to use the `read(2)` and `write(2)` system calls. This provides data stream functionality on a socket.

The Berkeley interface is complex, but has many good points (such as the use of `read(2)` and `write(2)`). However, the actual networking code is part of the operating system. This would require the computer to be rebooted when changes are made. Since this implementation does not reside in the kernel, it is not necessary to reboot the computer or otherwise disturb the users of the computer when changes in the software are made.

This implementation can provide some of the same functionality, but some things can not be implemented due to constraints of the IPC interface provided. The biggest problem is determining that a connection should be terminated because an application has abnormally terminated. A possible solution to this problem is described in chapter 7. The library interface routines provide all the other functionality for sending and receiving data from the daemon processes.

## CHAPTER IV

### THE PROJECT DESIGN AND STRUCTURE

The structure of this TCP/IP implementation represents modular and structured programming. Each step involved in moving data through the protocol levels, involves a module that can be separated from all others. This allows different strategies to be investigated with less work involving the resolution of global dependencies.

#### A High Level View

From a high level view, the program appears as several programs. Overall, these programs work together using interprocess communication to provide the necessary layers to implement the TCP/IP suite of protocols. Figure 2 in appendix C shows the interactions of the modules with each other via the message queues. As figure 2 illustrates, each module has one point where it awaits work (a message from another process). It is intended that a module never should wait at any other point. Circular waiting is eliminated, thus preventing deadlock.

For example, it is possible for the queues between the Ethernet module and the IP module to fill. IP might be forced to wait, while attempting to write to the Ethernet queue and the Ethernet module might be forced to wait while attempting to write to the IP queue. This creates deadlock because the IP module's write

cannot succeed until the Ethernet module reads a message. Likewise, the Ethernet module could not read a message until its write to the IP module completed, so it is stuck as well. Disallowing writes to block eliminates this possibility.

The work load must be distributed amongst the modules in as equal of a manner as possible. This minimizes the bottleneck effect that a particular module has on the entire system. The package is a teaching tool, rather than a production system, but it still should be as efficient as possible within requirement and design constraints.

### Interaction with UNIX

System V UNIX's interprocess communication, or IPC, provides some interesting, although limited, features. One of the biggest drawbacks is the inability of unrelated processes (those with dissimilar uids and gids<sup>10</sup>) to communicate without allowing other processes to eavesdrop or insert data into the communication stream. Because all IPC facilities work like files, i.e. use owner/group/world permission protections, certain security related problems arise. One such problem is that the integrity of the data stream within the IPC mechanism cannot be guaranteed. To overcome this problem, the program makes use of a integer password that is verified during each transaction at both ends. When the password is not correct some, currently unspecified, action must be taken.

Other problem areas concern the ability to send messages through a message queue with `msgsnd(2)` as described in [19]. To preserve throughput, each module must not ever wait other than when reading a message to request more work. Therefore, a "don't wait to send" mechanism must be used to make sure that a

---

<sup>10</sup>uid and gid are abbreviations for user identifier and group identifier respectively.

lack of buffer space does not cause a module to block on a `msgsnd(2)` operation. This does not work correctly for the System V message passing facility when only a single process is attached to a message queue. This particular problem must be solved in each place where it can occur.

## CHAPTER V

### MODULARITY OF THE IMPLEMENTATION

Among the major design criteria for this project are modularity and portability. The concern is that the software be transportable to all systems supporting System V IPC with very little effort. At most, it only should be necessary to write an Ethernet module. By using a module based system, it will be possible to divide portability problems into tasks which are related to replacing small modules, rather than tasks of writing large programs. A model for communication between the daemons<sup>11</sup> and the higher level programs has been adopted. This model is implemented in a way that will allow its adoption into future work.

While the message passing calls are not encapsulated into functions, their calling conventions with their data arguments provide sufficient information to allow the System V message passing to be replaced by routines that might use a different procedure for message passing. An attempt has been made to minimize the number of function call levels needed to expose the total functionality of the project.

The interaction with the daemon processes is divided into five different function calls:

1. Establish a communications path and allocate protocols.
2. Receive data from the daemon.

---

<sup>11</sup>A daemon is a background process used to perform some lengthy or indefinite activity.



3. Extract data from the daemon and place it into a structure.
4. Send data to the daemon.
5. Break the communications path and deallocate protocols.

These functional interfaces provide the necessary operations to make use of the protocols. The names for the routines are unique so that program units using/serving multiple interfaces can do so without symbol name conflicts. For example, the “Send data to the daemon” routine for the IP module is called “ip\_send” while the same module for the Ethernet module is called “eth\_send”.

The header include files for using these routines are uniquely named. For the IP module, all files begin with the prefix “ip\_”, while the Ethernet module files are prefixed with “eth\_”. All macros, defined constants, and structure types in these files have similar prefixes on their names. This makes them unique while leaving items with similar purposes in different modules, with similar names (e.g. IP\_PROTO\_NUM and TCP\_PROTO\_NUM are the names of macros containing the protocol numbers for these protocols).

In summary, all modules follow a consistent naming strategy and calling mechanism. This greatly improves the readability of the code. This modularity will allow future adaptation and additions to not conflict with previously written code (providing the simple guidelines outlined above are followed).

## CHAPTER VI

### COMPLETED WORK

The work completed for this project is design related. A coherent design minimizes programming and coding. This chapter is an explanation of the overall design and operation of the project.

#### Interprocess Communications

All modules pass information to other modules using Interprocess Communications (IPC). In the current implementation, IPC is performed using the UNIX System V message passing facilities. The general protocol of this message passing involves an initial transaction between the daemon and the application (which may be another daemon, as IP is to Ethernet). Initially, the application tells the daemon that it wishes to receive data from a specific protocol in the domain of the daemon. As an example, IP tells the Ethernet daemon that it wishes to receive IP datagrams from the Ethernet. Figure 3 in appendix C describes this transaction.

It is possible for an application to service multiple protocols by informing the daemon that it wishes to do so. If a particular protocol is not available, because another application has already reserved it, the daemon conveys this back to the application. After the application sends the initial request for protocol reservation,

the daemon responds with a message telling what protocols were accepted and rejected. The application must then select the “correct” action based on this response.

For the protocols that are reserved successfully, the application may send data to the daemon to be processed. Data received from the network interfaces must be read from the daemon by the application. Currently, no asynchronous activity is supported. Instead, the message passing operations are done using the `IPC_NOWAIT` function. If there is no data available, or the daemon’s message queue can’t hold any more messages, then the application will not block which could possibly cause deadlock. A method of supporting asynchronous processing with blocking is given at the end of this paper.

When the application is finished with the protocol it has reserved, it must release the protocol by sending a detach request to the daemon. This message results in a response message from the daemon. The contents of the message tell whether or not that application has any more protocols reserved. Also, if there were problems releasing a protocol, that information also is conveyed in the response.

Any problem releasing a protocol is related directly to some application sending a delete request for a protocol with an invalid password. After this happens, that protocol is undeletable, due to security considerations. Once an ill-behaved application starts tampering with trying to take over a protocol, it should be blocked from doing this permanently. Using the above strategy assures this. It also provides information to the legitimate application by telling it there are problems.

## IP Configuration

The IP daemon must have information about the network interfaces to which it has access. This information is compiled into the daemon in the form of an

information table. This table contains function pointers to routines that attach, send, receive, and detach from a network interface. The routines that perform these functions have specific naming conventions. The configuration file that describes the network interfaces is processed by a program which converts the textual information into a structure initialization.

The daemon can use the network interface description to determine how routing should be done based on the IP networks accessible through a particular network interface. Figure 4 in appendix C shows some sample lines of configuration information.

The first line describes a specific host entity. It says that the IP address, 128.192.37.1, always will be serviced by the “eth” module. The second line describes a set of internet class C addresses<sup>12</sup> which are on the class C network whose number is 128.192.45. These addresses also are serviced by the “eth” module. The third line describes a set of class B addresses on a class B network. The hosts on this network will all be serviced by the “imp” module. The last line describes what to do with addresses that do not match any of the other addresses, the module “gate” handles these addresses.

The program author uses discretion to determine the exact action of each module. The module might contain code to look up *IP-to-Ethernet* address mappings in a file, or the module might use the Ethernet Address Resolution Protocol (EARP) to query the hosts on the network to discover a specific *IP-to-Ethernet* mapping. Support for other network interfaces can be added easily.

Each of the modules contain at least 3 functions. The required three functions are named `x_startup`, `x_shutdown` and `x_xfer`, where the initial “x” character is replaced by the module name given in the configuration line. For example, the

---

<sup>12</sup>The internet address classes are defined in [12]

“eth” module would have routines named `eth_startup`, `eth_shutdown` and `eth_xfer`. The configuration program which generates a structure definition for the IP module to use, depends on this relationship to know how to generate the structure initialization.

The IP daemon also must know its local host address which should be convenient to change. A separate file provides this information. The IP daemon only need be stopped and restarted to update its knowledge of the information in this file. The `IPCONFIG` macro defines the name of this file.

## IP Operation

The IP daemon reserves the IP protocol for its use, then waits for messages in its message queue. The type of the message determines how it is processed. Below is an overview of the processing of each type of message the daemon recognizes.

### Attach Requests

An attach request requires the daemon to verify the availability of the requested protocol(s). This involves a simple search of the data structures. When a protocol is available, its value is replaced by a invalid (the value is actually -1) protocol value in the array of requests signifying that it has been accepted. The response message sent back to the requestor has the resulting array in it. The requestor can use this information to determine which protocols were not available when there is an attach failure.

For each available protocol, the pertinent information about the requestor is placed into a protocol table entry allocated to the protocol. When all protocols have been processed, the response message is formatted and sent to the message

queue whose identifier is passed in the request message. Future references to the allocated protocol require that the password (random integer) that was passed by the requestor to be specified correctly.

### Detach Requests

A detach request is processed much in the same way as an attach request. The only difference is, if a detach request is made using the wrong password, then the protocol that was requested to be detached from will be marked undetachable. This solves an important security problem that is associated with an ill-behaved user/program trying to intercept all network traffic using a specific protocol. If the user discovers the password that is used, unauthorized traffic may be generated, but traffic cannot be received. This is a result of the fact that a user cannot change the daemon's knowledge of which queue is to receive datagrams for a specific protocol.

### Datagram Transmission

An outbound IP datagram is processed by verifying that the protocol in the datagram is currently "active"; then by verifying the password. If either of these operations fail, the datagram is ignored<sup>13</sup>. Once this dual verification has succeeded, a search is initiated for the proper network interface to use based on the configuration structure information. With the proper interface located, the `_xfer` procedure for that interface is called to process the datagram.

### Incoming IP Datagrams

The first step in processing an incoming IP datagram is verifying that the destination of the datagram is really this host. If it is not, then the datagram is

---

<sup>13</sup>IP does not guarantee that the datagram will arrive in tact.

sent back to the output side of the daemon for processing there. This happens when a packet is source routed (the exact route through the network can be specified by datagram options) through a particular machine. If the protocol is in use on the host, then it is passed through the appropriate message queue to the application.

### Ethernet Operation

The Ethernet module is distinguished from the other modules since two different processes must be performed simultaneously. Packets must be read from the Ethernet device, and outbound traffic must be read from the message queue. This requires two separate processes. These two processes must also share the protocol table information. This allows the process reading from the Ethernet device, `ethrdr`, to send packets directly to the application. The process that is reading the message queue, `ethwtr`, and directing traffic out of the host, also uses this table. `Ethwtr` fills in the table with new entries as requests are made; it also deletes entries when necessary.

The overall operation of the Ethernet module is very similiar to the operation of the IP module. The `ethwtr` process recognizes only the three basic message types in its message queue; i.e., `attach`, `detach` and `data`. The `ethrdr` process never reads a message from the Ethernet message queue. `Ethrdr` only reads packets from the Ethernet device and sends them to the proper process.

Almost all of the design and implementation has been completed. The mechanisms that are in place have been tested and proven to work. The correctness of things such as datagram checksums has not been verified due to some ambiguities in the definition in RFC 791[8].

## CHAPTER VII

### FUTURE WORK

The design of the programs essentially is complete. A few items still are outstanding. Coding remains to be done to complete the implementation of IP. A small amount of refinement is required on the Ethernet interface. Final TCP implementation has not started. There is still some software engineering work to be done. Outstanding items are surveyed below.

#### The IP Module

The IP module lacks complete support for ICMP. Code stubs specify where there should be ICMP support. These stubs should be replaced with a complete implementation of ICMP which communicates with the IP and TCP library routines. Many actions associated with ICMP messages must be performed outside of the daemon. The ICMP SOURCE QUENCH message could be handled by inserting delays between the sending of data from the library routines through the message queue to the daemon. One possible strategy would be to allow the data initially to be transmitted as fast as possible, until a SOURCE QUENCH is received. Next, an alarm driven timer would manipulate a semaphore. The process would wait on this semaphore before transmitting a packet. That way, the process would not be able to send the pack until the timer expired. The code fragment of figure 5 in



appendix C demonstrates the concept. This strategy assures that the time delay is done as an absolute delay instead of a delay just prior to sending the data to the daemon. Several additional items can be added:

- The ability to manipulate the IP options should be enhanced to allow a single `ip_set_options()` procedure to be used to establish the default options to be included in each datagram. This minimizes the amount of data manipulation required. In the future, a different strategy for `ip_send()` might be employed. The new strategy would place the options into a datagram template in static storage so only the header information and the data would need to be added to complete the datagram.
- The daemon must supply the library routines with the local host address in the response to an `IP_ATTACH` operation. This allows the library routines to compute the checksum for each datagram. Currently, an ill-defined value is placed into the IP header during assembly.
- The daemon must perform the header checksum verification on an incoming datagram prior to forwarding it to any higher level module. This assures that a particular application is not given a datagram it does not own. Currently, the datagram is forwarded blindly to the application indicated by the protocol field or it is dropped if the protocol is unknown.
- To aid in the asynchronous processing of messages both in and out of an application, some type of signal passing mechanism needs to be implemented. A semaphore can be used to notify the application that data is waiting. Other techniques may work as well.

- When implementing ICMP, it is necessary to pass ICMP messages from the daemon to the application. Currently, the library routines use the “receive the first available message” facility of the `msgrcv()` routine to receive message from the daemon. If the messages are numbered differently, then the library routine could always try to receive an ICMP message first to assign a priority to the reception of those messages.
- There are many places where timers should be used to cause timeout of the reception of datagrams in TCP segment assembly. The IP fragmentation routines, which have not been written, also require this. A set of routines which allows a single process to set many timers to interrupt processing at some time in the future has been written. The process need only supply a delay time in seconds, a routine address to be called when the timer expires, and a single data value. These routines should ease the task of maintaining these timers greatly. These routines have been tested and should work as written. Full documentation is available within these routines<sup>14</sup>.
- The daemon should make use of a timer actived routine to establish that a particular message queue is gone, and therefore the associated protocols are no longer in use. The same routine should also be called when a connect request is received. This routine can use the `msgctl(2)` system service to determine that the queue has been deleted. `Msgctl(2)` will return an error code when a queue identifier corresponding to a deleted queue is passed to it, thus making it obvious that the queue is gone.

---

<sup>14</sup>The timer routines are in the file `net/src/gen/timer.c`.

## The Ethernet Module

The future work on the Ethernet module includes minor work items, outlined below:

- The ethwtr process should verify the password on messages it receives through its message queue, then act appropriately on bad passwords. A recovery strategy similiar to the IP module's recovery strategy should be developed.
- The version of the ethwtr module for the Concurrent family of computers currently adds the local Ethernet address to the packet before sending it to the Ethernet. This can be avoided by using the mode of the driver which automatically does this. The src/h/eth\_struct.h<sup>15</sup> file should have some conditional compilation added to it to remove the Ethernet source address from the structure definition when the Concurrent version is compiled.

Most of the design already completed can be applied to future work. The original design came from considering the other protocols and applications. With the Ethernet module nearly complete, and more than half of the IP module's code completed, the basics are out of the way. The amount of design work and the amount of code that this project encompasses has limited the author to completing only a portion of the project. The work remaining primarily is completing the implementation.

---

<sup>15</sup>This is a relative path to the file, from the "net" users directory. This directory structure is described in appendix B.

## CHAPTER VIII

### SUMMARY AND CONCLUSIONS

#### Summary

The ultimate completion of this project should result in an extremely useful research tool which allows more indepth study and exploration of wide area computer networks. The TCP/IP suite of protocols should continue to be used for many years to come because of the large investment including both software and hardware currently supporting these protocols.

The software involved in this project, demonstrates the complexity of both the protocols and their implementation. This particular implementation is bulky and slow, but represents an extremely portable package that can be used on many machines in existence today; however, these protocol stills are highly complex and expensive in terms of machine resources.

It is the author's opinion that these protocols are too complicated to be implemented in the operating system of a timesharing environment. In fact, it may be very beneficial to the performance of both a network and timesharing hosts on that network to place the protocol suite on an auxiliary processor attached to the host's bus. A DMA communication path to this auxiliary processor could allow the simple tasks necessary to create ports at the TCP level.

## Conclusions

Computer networking provides the ability to access the large amounts of information accumulated on mass storage devices. Smaller computer systems may be possible, if an interface to larger machines is provided remotely and at very high speeds<sup>16</sup>. The workstation concept is fueled greatly by the possibility of eliminating the need for hundreds of megabytes of disk storage next to the workstation. By remotely locating the disk space, work space is conserved. If multiple workstations can share a single disk then costs of operating are reduced.

Remote, interactive communication of digital information also is possible using the TCP/IP protocol suite. Many protocols, including the File Transfer Protocol (FTP), the Simple Mail Transfer Protocol (SMTP) and the terminal emulation protocol (TELNET) allow most activities associated with computing to be performed remotely from other hosts, at speeds comparable to those experienced while a user is logged onto that host directly. These existing protocols and their implementation will allow future researchers to discover a better solution.

---

<sup>16</sup>Speeds approaching those of disk transfers are possible over Ethernet.

## REFERENCES

- [1] Black, Uyles, Data Communications and Distributed Networks, Prentice-Hall, Inc., 1987.
- [2] Cerf, Vinton. G., and Cain, Edward, The DoD Internet Architecture Model, *Computer Communications: Architectures, Protocols, and Standards*, IEEE Computer Society Press, 1985, Catalog no. EH0226-1.
- [3] V. G. Cerf, and Robert E. Kahn, A Protocol for Packet Network Intercommunication, IEEE Transactions on Communication, COM-22, May 1974, pp. 673-2648.
- [4] Comer, Douglas P., Operating System Design, Internetworking with XINU, Prentice-Hall, Inc., 1987.
- [5] Green, Paul E. Jr., ed., Computer Network Architectures and Protocols, Plenum Press, New York, 1982.
- [6] Groenback, I., The TCP and ISO Transport Service — A Brief Description and Comparison, NATO Technical Memorandum STC TM-726, SHAPE Technical Center, The Hague, Netherlands, Feb. 1984.
- [7] Internet Protocol, Military Standard, MIL-STD-1777, U.S. Department of Defense, May 20, 1983.

- [8] Postel, J. B., Internet Protocol, RFC 791, USC Information Sciences Institute, Marina del Ray, California.
- [9] Postel, J. B., "Internet Protocol Approaches", IEEE Transactions on Communications, vol. COM-28, no 4, April 1980, 604-611.
- [10] Postel, J. B., Internet Control Message Protocol, RFC 792, USC Information Sciences Institute, Marina del Ray, California.
- [11] Postel, J. B., Transmission Control Protocol, RFC 793, USC Information Sciences Institute, Marina del Ray, California.
- [12] Reynolds, J., Postel, J. B., Assigned Numbers, RFC 960, USC Information Sciences Institute, Marina del Ray, California.
- [13] Schwartz, Mischa, Computer-Communications Network Design and Analysis, Prentice-Hall, Englewood Cliffs, N.J., 1977.
- [14] Schwartz, Mischa, Telecommunications Networks, Protocols, Modeling and Analysis, Addison-Wesley Publishing Company.
- [15] Sunshine, Carl. A. and Dalal, Yogen K., "Connection Management in Transport Protocols", Computer Networks, vol: 2, 1978, 454-473.
- [16] Stallings, William, *Computer Communications: Architectures, Protocols, and Standards*, IEEE Computer Society Press, 1985, Catalog no. EH0226-1.
- [17] Transmission Control Protocol, Military Standard, MIL-STD-1778, U.S. Department of Defense, May 20, 1983.
- [18] ULTRIX programmers manual, binder 2A, AT&T, Digital Equipment Corporation, Regents University of California, Sun Microsystems.

- [19] XELOS programmer reference manual, AT&T, Concurrent Computer Corporation.



## APPENDICES

## APPENDIX A

### GLOSSARY

<b>Datagram</b>	A packaged piece of data, use in the Internet Protocol, which contains extra information that will aid IP in routing and verification.
<b>Data Reliability</b>	Data is reliable if the transmission path it traverses allows it to always arrive in tact, as well as sequenced.
<b>Ethernet</b>	A high speed network also known as the IEEE 802.3 network.
<b>Flow Control</b>	A mechanism that limits the data transmission rate so that the receiving host does not become overloaded with the processing of incoming datagrams.
<b>Gateway</b>	A network host that provides the ability for traffic on one network to cross over to another.
<b>Header</b>	The initial portion of a datagram which contains the information necessary to transport that datagram to its destination.
<b>Protocol</b>	A precise methodology which dictates the order in which a prescribed set of operations are carried out.
<b>Sequencing</b>	Maintaining the cronological ordering of the transmission of datagrams such that they are received in the same order as they were transmitted.

**APPENDIX B**

**DIRECTORY STRUCTURE**

<code>./Doc</code>	Contains previous reports and other associated documents.
<code>./bin</code>	Executables and shellscripts associated with project.
<code>./lib</code>	Object file archives for library routines.
<code>./etc</code>	Assorted stuff, such as hosts file.
<code>./src</code>	Root of source tree.
<code>./src/Ethernet</code>	Root of Ethernet daemons' source.
<code>./src/Ethernet/Concurrent</code>	Source for PE3230 version.
<code>./src/Ethernet/3bnet</code>	Source for AT&T 3BNET version.
<code>./src/Ethernet/lib</code>	Ethernet library routines for accessing the daemon.
<code>./src/Ip</code>	Root of IP daemon's source.
<code>./src/Ip/conf</code>	IP daemons compile time configuration directory. Contains configuration files that describe the physical network services that are available, e.g. Ethernet.
<code>./src/Ip/test</code>	IP test program to send datagrams to arbitrary addresses.
<code>./src/Ip/daemon</code>	IP daemon (ipd) source code.
<code>./src/Ip/lib</code>	Library routines for communicating with IP daemon.
<code>./src/Ip/Icmp</code>	Current implementation of ICMP for ipd.
<code>./src/Config</code>	Site configuration programs.
<code>./src/h</code>	Header include files for entire project.
<code>./src/gen</code>	General library routines.
<code>./src/Arp</code>	Partial implementation of the Ethernet Address Resolution Protocol (EARP).

**APPENDIX C**

**FIGURES**

<b>Level</b>	<b>Function</b>
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Link
1	Physical

Figure 1. OSI Protocol Hierarchy

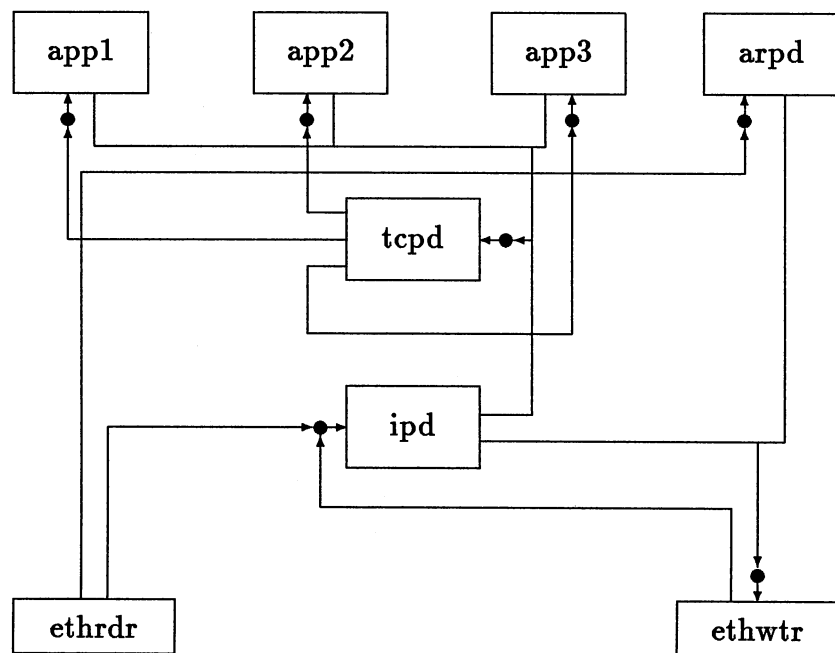


Figure 2. Diagram of IPC

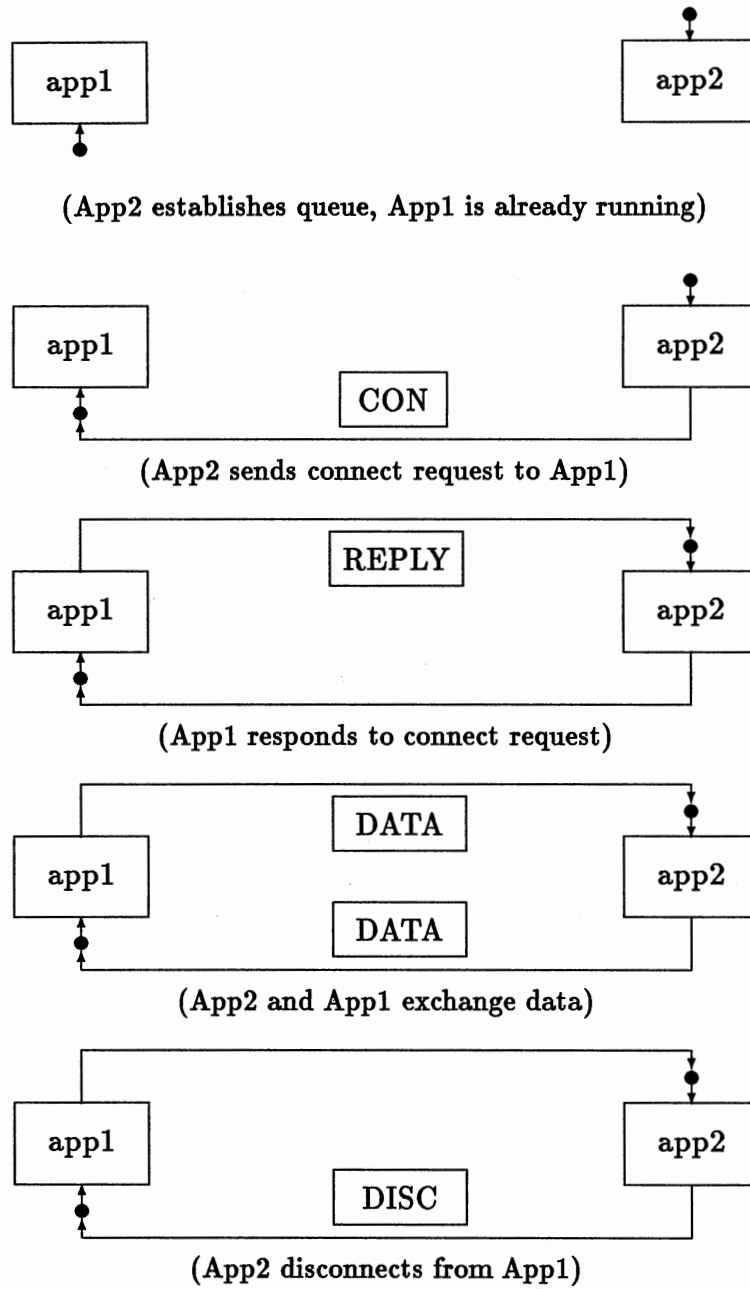


Figure 3. Message passing between applications



```

host,address=128.192.37.1,type=c,name=a.cs.okstate.edu,module=eth
network,address=128.192.45.0,type=c,name=ucc,module=eth
network,address=68.25.0.0,type=b,name=arpanet,module=imp
gateway,name=gateway,module=gate

```

Figure 4. Sample configuration lines

```

global int delay_time = 0;

read_queue()
{
    message_type_read = get_message_from_daemon (msg);
    if (message_type_read == SOURCE_QUENCH)
        delay_time += 2;
    ...
}

write_queue()
{
    if (have_set_timer) {
        sem_wait (delay);
        have_set_timer = 0;
    }

    write_queue ();

    if (delay_time > 0) {
        set_timer (delay_time, signal_semaphore);
        delay_time--;
        ++have_set_timer;
    }
}

signal_semaphore ()
{
    sem_signal (delay);
}

```

Figure 5. Sample alarm timer code

VITA

Gregg G. Wonderly

Candidate for the Degree of  
Master of Science

**Thesis:** A NETWORKING INSTRUCTION AND RESEARCH TOOL

**Major Field:** Computing and Information Sciences

**Biographical:**

**Personal Data:** Born in ElReno, Oklahoma, July 29, 1963, the son of Robert D. and Patrica A. Wonderly.

**Education:** Graduated from Putnam City West Senior High School, Oklahoma City, Oklahoma, in May, 1981; received Bachelor of Science Degree in Computing and Information Sciences from Oklahoma State University in December, 1985; completed requirements for the Master of Science degree at Oklahoma State University in December, 1988.

**Professional Experience:** VAX/VMS systems administrator, Department of Mathematics, Oklahoma State University, August, 1986, to July, 1988. Computer center diagnostician, Oklahoma State University, January 1986, August, 1986. Laserdisc researcher, TMS Inc., Stillwater, Oklahoma, June, 1985, August, 1985.