AFIT/GE/ENG/91D-30

Comparison of AFITPAC versus NOS and a Packet Radio Network Design

THESIS

Thomas Alexander Iannelli
Captain, USAF

AFIT/GE/ENG/91D-30

December 1991 Master's Thesis

Comparison of AFITPAC versus NOS and a Packet Radio Network S..nulation Design

Thomas Alexander Iannelli, Captain, USAF

Air Force Institute of Technology (AU), WPAFB OH 45433-6583

AFIT/GE/ENG/91D-30

Headquarters Air Force Logistics Command
Logistics Management Support Center/Systems Support
WPAFB OH

Distribution Unlimited

This thesis encompasses two themes: 1) A comparison of the Network Operating System (NOS) software package, developed by Phil Karn and others, and AFITPAC, written by Marsh and Geier at the Air Force Institute of Technology. The comparison is based on the Air Force Logistics Command's requirements for the development of a Packet Radio Network program. The results of the comparison are a recommendation that use of the NOS package be pursued because it can meet all twelve of the requirements and complies with five of the Military Standard Protocols. 2) The design of a personal computer Packet Radio Network Simulator used to analyze the network performance of AFITPAC and NOS is proposed. The proposal concludes with a discussion of problems in development of the simulation which used MODSIM II^{TM}, C^{++}, and a 80386/25Mhz personal computer. Recommendations are made for pursuing the simulation effort on other hardware platforms and for improvements to the design.

Packet Radio, MODSIM, Simulation, NOS

61

UNCLASSIFIED UNCLASSIFIED UNCLASSIFIED UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines to meet optical scanning requirements.**

Block 1. Agency Use Only (Leave Blank)

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency. Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ..., To be published in .... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availablity Statement. Denote public availability or limitation. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR)

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- DOD - Leave blank
DOE	- DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports
NASA	- NASA - Leave blank
NTIS	- NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Acknowledgments

Words are hard to find that describe the gratitude I have for the Air Force Institute of Technology's faculty and staff. They have made my last eighteen months a difficult but enjoyable time. I would like to recognize individuals by name, Captain Mark Mehalic and Mr. Martin DeSimio. These two participated in my thesis, were my classroom instructors, and supported and counseled me through this difficult growth period of my life. I would not have what is left of my sanity without my classmates. There are a few whom I wish I had the time to thank personally: Captain Matt Foster, Captain Perry Choate, Captain Eddie Bednar, Captain Scott McGuffin, and Captain Dennis Andersh. Thanks gentlemen, for all your support. I want to acknowledge the contribution made by my best friends, Captain George Allen Barber and Captain Ellen Barber. Special thanks to a young woman, Ms. Copper Harding, who stayed up late to type in the latest modifications to this document.

Thanks, gratitude, and extra special recognition are extended to my best friend, my mentor, my partner, my wife, Linda Ayers. Without her constant support and encouragement I would literally not have completed this work.

Thank you Linda

Thomas Alexander Iannelli

AFIT/GE/ENG/91D-30

Comparison of AFITPAC versus NOS and a Packet Radio Network Design

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Thomas Alexander Iannelli, B.E.
Captain, USAF

December, 1991

Table of Contents

List of Figures

List of Tables

AFIT/GE/ENG/91D-30

Abstract

This thesis encompasses two themes: 1) A comparison of the Network Operating System (NOS) software package, developed by Phil Karn and others, and AFITPAC, written by Marsh and Geier at the Air Force Institute of Technology. The comparison is based on the Air Force Logistics Command's requirements for the development of a Packet Radio Network program. The results of the comparison are a recommendation that use of the NOS package be pursued because it can meet all twelve of the requirements and complies with five of the Military Standard Protocols. 2) The design of a personal computer Packet Radio Network Simulator used to analyze the network performance of AFITPAC and NOS is proposed. The proposal concludes with a discussion of problems in development of the simulation which used MODSIM II^{TM}, C^{++}, and a 80386/25Mhz personal computer. Recommendations are made for pursuing the simulation effort on other hardware platforms and for improvements to the design.

Comparison of AFITPAC versus NOS and a Packet Radio Network Design

I. Introduction

1.1 Background

The Air Force Logistics Command (AFLC) sponsored software development work at the Air Force Institute of Technology (AFIT) for Packet Radio Networks (PRNET) in 1988(14, 20), 1989(15) and 1990(10). A PRNET is a collection of computers using radio stations as their communications media. PRNETs are used to connect computers that are unable to be hard wired. For example, portable or mobile computers and systems located in areas where telephone lines are of poor quality, out of service, or non-existent. The AFIT work resulted in Marsh creating a software package, called AFITPAC, which provided a menu driven user interface to the PRNET(15). This enabled the user to perform tasks, such as transferring files, setting terminal node controller (TNC) parameters, and establishing connections, just by using the function keys displayed at the bottom of the screen. Then, in 1990, Geier modified the package to include an automatic/adaptive routing algorithm(10, 11). The routing algorithm builds a table of Packet Radio Units (PRUs) available on the network and the routes used to get to them(10, 11). This table is then displayed on screen to facilitate user connections to remote PRUs. Prior to this change, the user had to have knowledge of the network topology in order to establish a connection, and the user was required to enter the routing table manually into the computer.

In order for this package to work it needs to transmit and receive special packets on the network. These packets contain a control character at the beginning of the data, which marks the packet as routing information. The receiving Packet Radio Unit (PRU) can then process this information to modify its routing table. Geier's packets travel the network just as other information packets do and add overhead traffic, for route management, to the normal PRNET traffic. Geier tested his software in the limited AFIT PC laboratory environment using three PRUs. He demonstrated that the routing algorithm performed as expected on the three PRUs. However, the AFLC PRNET was designed for disaster relief efforts and wartime contingencies(10, 15). This implies a PRNET composed of a large number of portable units. If the routing algorithm's overhead traffic increases drastically under these conditions, it would interfere with vital information traffic on the PRNET, resulting in long delays in message transportation or blocked connections. A traffic flow analysis of Geier's software on the AFLC PRNET must be completed before confidently placing the package in the operational environment(10). The information gathered from such an analysis could be used in the development of software packages that are more channel efficient.

1.2 Problem Description

The challenge is to develop a method of traffic flow analysis for software packages used in a packet radio network. One possible solution would be to coordinate with local amateur radio operators to use their PRUs. After some field research, the author determined that the local amateurs have permanent radio stations in their homes. If they have portable radio stations, in general their computer systems are not portable. This type of fixed topology PRNET would not represent the operational environment desired, because the PRUs need to be portable, and therefore, makes a poor test condition. Another possibility would be to create a computer model of the PRNET that would allow for both stationary and portable PRUs.

Modeling tools are hard pressed to keep up with the dramatic changes in the nature of computer networks and their application environments. ... Modelers purchase their own tools or services, but most packages only solve a piece of the computer networking puzzle. Much of the effort goes into models that try to portray the operation of complex networks and architectures supporting a wide variety of heterogeneous equipment and software.(16)
Mitchell G. Spiegel

As Spiegel stated, most network modeling tools do not accommodate all the requirements of a particular modeler's project. The biggest limitation to network modeling tools is they can not incorporate other software packages. Therefore, the problem is to design and develop a computer model of a packet radio network with changing topology that will interface with Geier's automatic/adaptive routing algorithm package and Karn's Network Operating System(21), NOS package. NOS is examined as a possible alternative to AFITPAC because it implements many of the standard applications of Defense Data Network (DDN) hosts and its C source code is readily available.

1.3 Summary of Current Knowledge

Most of the significant work in the packet radio network simulation is sponsored by Defense Advanced Research Projects Agency (DARPA). The work includes a package called PC-NETSIM developed for DARPA in 1989, by Rockwell International Collins Defense Communications(1, 2). This discrete-event simulator provides an artificial execution environment for the actual protocols to be used in the PRNET(2). Unfortunately, PC-NETSIM, simulates a packet radio network that uses different radio system technology, specifically a direct-sequence spread spectrum system(1, 2), than the AFLC PRNET, which uses Audio-Frequency-Shift-Keying (AFSK) broadcast using Frequency Modulation (FM)(12). Furthermore, PC-NETSIM simulates a system using a Pure-Aloha channel access scheme and the AFLC PRNET uses a Carrier Sense Multiple Access (CSMA) scheme. The differences in the performance of these two modulation and channel access schemes would cause the simulation results to represent the performance of the software packages inaccurately. For a discussion of the performance differences between Pure-Aloha and CSMA see Tanenbaum(19).

1.4 Assumptions & Scope

This project is written using C^{++} and MODSIM II^{TM}[1]. These languages were chosen for the following reasons. They are portable to all of the hardware platforms currently in use in PRNETs(4). MODSIM II^{TM} also provides an interface to C^{++} which allows the original software modules of Geier's software to be used. However, MODSIM II^{TM} uses different data types than C^{++} requiring modifications to some variables in the AFITPAC software. Table 1.1 shows the correspondence of data types between the two languages. Each simulated PRU will be running its own copy

MODSIM II^{TM}	C^{++}
INTEGER	long
REAL	double
BOOLEAN	unsigned char
CHAR	unsigned char
STRING	char * to "C^{++}"
ARRAY OF CHAR	string from "C^{++}"
ANYREC or ANYOBJ	void * or int *
enumeration	unsigned char

Table 1.1. MODSIM II^{TM} data types and their corresponding C^{++} data types

of the AFITPAC software. This may cause conflicts with the video display.

Therefore, all functions causing video input/output will be removed. Problems may also arise with the AFITPAC software's use of the file system to store information, because all of the PRUs will be trying to read and write to the same file. Any of AFITPAC's functions that perform file I/O will be modified to accept an

[1]MODSIM II is a trademark and service of CACI Products Company

argument for a filename to correct this problem. Thus allowing the simulation to pass unique filenames to each copy of the software.

The researcher's computer model will simulate the portable radio units, the radio signal connectivity between PRUs, and TNC functions. For the purposes of this project, all PRUs are assumed to have equal transmitter and receiver capabilities and the portable units are off the air while relocating, to avoid the problem of connection failures due to a change in network topology.

1.5 Organization

The remaining thesis chapters are organized as follows:

- Chapter II is a comparison between AFITPAC(10), a software package written by Marsh and Geier, and NOS(21), a software package written by Phil Karn and others.
- Chapter III is a discussion of the model design and methodology.
- Chapter IV concludes with findings and recommendations.

II. Comparison of AFITPAC and NOS

2.1 Introduction

In the process of designing a PRNET simulation, other packet radio programs were examined. Looking at other software packages revealed similarities in their structures and performance. The similarities provided a better concept of what components a PRNET model needed. The name of a particular software package was repeatedly mentioned during the search for information on packet radio technology . The package is the Network Operating System which is called NOS in it latest version. Older versions of the package were called NET or KA9Q. A copy of the source code was obtained from the DDN, from host ucsd.edu, and the executable was downloaded from CompuServe©. This version of the program is called NOSVIEW, and is packaged by Ian Wade(21). NOSVIEW adds a Terminate-and-Stay-Resident (TSR) program to NOS which allows the user to view any part of the NOS documentation on-line. Although this is not the only public domain packet radio software available, it does seem to be the most comprehensive. The fact that NOS incorporates many of the different protocols and applications currently used on packet radio networks and on the DDN and its ability to act as a gateway between the two networks(14) makes it a possible alternative to AFITPAC. Therefore, a comparison of how both packages meet the AFLC software requirements follows.

2.2 How NOS Compares to AFITPAC Requirements

The list below enumerates the requirements for the AFLC packet radio network as presented by Marsh work(15).

1. The system should be easy to use with only minimal training.
2. The network routing algorithm should provide a route for transmission of messages even when there is not a direct route available between source and destination.
3. Messages will be encrypted for transmission.
4. The system will provide archival storage of messages that are transmitted or received.
5. The system should provide a transmit message queue so that multiple messages can be transmitted with little operator intervention. The queue should be able to hold at least twenty messages.
6. Queued messages that are not able to be transmitted will be flagged for the operators attention.
7. The system should provide for reception messages whether the position is attended or not.
8. The operator interface to the program should be menu driven with a data window so the operator can see what is being transmitted or received.
9. The operator interface should have help screens as necessary.
10. A converse mode should be provided to allow interactive communication between two node operators.
11. A direct connection capability between the TNC and the GILLAROO, via the packet radio system computer, should be incorporated into the system software to allow for a secure converse mode for the system.
12. A system operator handbook will be provided. It will include system wiring diagrams and a software user's guide.

Figure 2.1. AFLC Requirements for the Packet Radio Network Software

The researcher will demonstrate how the NOSVIEW implementation of NOS meets each of the AFLC requirements. For clarity it is useful to define a number of terms before beginning the comparison.

GILLAROO -	cryptographic personal computer board that encrypts data, by using a key. The data is converted into pairs of output characters. All characters are ASCII printable text, the only control characters are CTRL-Q, CTRL-S, and CTRL-Z(15).
KISS -	a protocol for computer to TNC communications. It allows the computer to control all of the AX.25 functions and the contents of the High Level Data Link Control (HDLC) frames transmitted and received(7).

RFC - "(*Request for Comments*) the name of a series of notes that contain surveys, measurements, ideas, techniques, and observations, as well as proposed and accepted Internet protocols standards. RFCs are edited but not refereed. They are available across the Internet.(8)"

PK-232 - Z-80 based TNC manufactured by Advanced Electronics Applications, Inc.

For the purposes of this comparison Captain Marsh's evaluation of AFITPAC's ability to meet each requirement will be used. All material relating to AFITPAC is directly quoted from Marsh(15) and the figures and tables referenced therein are not provided.

2.2.1 Requirements (1), (8), & (9) (1) The system should be easy to use with only minimal training. (9) The operator interface should have help screens as neccssary. (8) The operator interface to the program should be menu driven with a data window so the operator can see what is being transmitted or received.

AFITPAC - (1) The system is relatively easy to use but some knowledge of how packet radio operates is necessary to get the most advantage of the system. This information is supplied through a brief users manual contained on the AFITPAC program disk.(15) (8) The program is menu driven (figure 6) via the function keys which are displayed across the bottom of the screen. A data window covers the upper portion of the screen (lines 3 through 18, 80 columns) and is used for displaying all received/transmitted data along with any system command responses(15). (9) Help screens explaining function key choices are available (press F10) for all menus. The main menu help screen also has a second screen listing some of the PK-232 commands and how to use them. All screens can easily be changed by changing the screens listed in the HELP.C module and recompiling the program(15).

NOS - The user has two different ways in which to interact with the system; 1) a menu driven bulletin board system, see Figure 2.2, and 2) issuing commands at the **net>** prompt. Both interfaces use the computer's display screen and each session uses all 25 lines to display transmitted and received information. The bulletin board system is straight forward and easy to use with detailed

(?)help	(A)rea	(B)ye	(C)hat	(D)ownload	(E)scape	(F)inger
(G)ateway	(H)elp	(I)nfo	(J)heard	(K)ill	(L)ist	(N)etrom
(R)ead	(S)end	(T)elnet	(U)pload	(V)erbose	(W)hat	(Z)ap

Figure 2.2. The NOS Bulletin Board System Interface Screen

help available for each command. This help is supplied through the use of the **Help** command. The syntax is *H* **command**, with **command** as any of the menu choices. NOS then displays the contents of the **command.HLP** file. This option is available whether attached locally or from a remote site to the system. The **net>** command prompt offers a listing of the top level commands available by typing *help* or *?*. Slightly more detailed help is available on each command by typing **command** *?*. The **net>** command line is only available on the machine that is running NOS, and not available from remote sites.

The most detailed level of help is available by using the NOSVIEW TSR. To activate the TSR press the Right Shift key and the Spacebar simultaneously. Then select the file to be viewed. Each file in the NOSVIEW directory is named after a command in NOS and contains the entire documentation for the command as compiled by Ian Wade(21).

The individual responsible for installing the NOSVIEW package needs to be knowledgeable about packet radio network and Transmission Control Protocol (TCP)\Internet Protocol (IP) network operations. There are a number of parameters that can be set that will have an impact on the performance of the network. Help in setting up and configuring NOS will soon be available in the form of book titled, *NOSintro: The Definitive Primer for the KA9Q Network Operating System*, by Ian Wade, that will be published late in 1991(21). Currently a complete copy of the NOS manual is distributed with the package in the form of a computer file. It is also available from the DDN host ucsd.edu in various document formats including Postscript and nroff/troff.

All of the TNC functions are available in NOS, however, they are implemented in software. There are no user generated parameters as far as the actual hardware TNC is concerned. The user communicates with the software TNC using commands similar to those used in a hardware TNC, with a syntax change that commands are preceded by **ax25**. The software communicates with the hardware TNC using the Keep-It-Simple-Stupid(7) (KISS) protocol.

2.2.2 Requirement (2) The network routing algorithm should provide a route for transmission of messages even when there is not a direct route available between source and destination.

AFITPAC - The "digipeat" option of the TNC provides this function directly to the system. The operator can limit who is allowed to use the repeat via the DFROM command of the PK-232 which can be set in the START.TXT file that loads when initializing the system(15).

NOS - The NOS package provides three distinct routing algorithms enabling delivery of messages to PRUs other than the nearest neighbors[1]. They are the AX.25 digital repeating, Routing Information Protocol[2] (RIP), and Radio Shortest Path First Protocol (RSPF). NOS uses the AX.25 syntax **connect** *interface* **NODE1 via REPEATER1 REPEATER2** to establish a connection to NODE1. The *interface* argument is used to specify which hardware interface is to be used in establishing the connection. NOS allows

[1]The term nearest neighbor refers to those PRUs who are accessible without passing through any repeaters
[2]Information can be found in RFCs # 1058

for multiple connections to be active simultaneously on different interfaces. NOS also adds this route to the AX.25 routing table automatically(21).

RIP is the same protocol utilized on the DDN which allows for the applications such as Simple Mail Transfer Protocol[3] (SMTP), File Transfer Protocol (FTP), Post Office Protocol[4] (POP2), telnet, and rlogin. It automatically updates the Address Resolution Protocol (ARP) tables, and makes it possible to monitor the RIP activity by using the **RIP trace** *level* command. This will display three levels of RIP activity at the local PRU; level 1 - only RIP packets causing changes in the routing table are displayed, level 2 - all RIP packets are displayed, level 0 - disables RIP tracing.

In RSPF protocol, each PRU determines which other PRUs are its neighbors by listening for a special message sent out by other PRUs running RSPF. When a PRU hears this Router-to-Router-Hello (RRH) message it determines whether the link is bidirectional by pinging the other station. The quality of the AX.25 link is determined by comparing the number of sent and received frames on the channel from the particular node. The routing table not only contains a list of the local PRU's neighbors, but a list of all the other PRUs' neighbors as well(21). The size of this table is limited by setting the horizon, in terms of hops. For example, if the horizon was set to 2 hops, the routing table would contain only information about the local PRU's neighbors and their immediate neighbors.

2.2.3 Requirement (3) Messages will be encrypted for transmission.

AFITPAC - Files are encrypted by the GILLAROO and can either be saved on disk for later transmission or, via the secure link option, be directly sent to the distant station for immediate reception and decryption(15).

[3]Information can be found in RFCs # 788, 821, 1090
[4]Information can be found in RFCs # 918, 937, 1081, 1082, 1225

NOS - ASCII text files can be sent in NOS using Simple Mail Transfer Protocol[5] (SMTP) and both binary and text files can be sent using File Transfer Protocol (FTP).

This would allow a file encrypted by the GILLAROO and saved to disk to be transmitted using either SMTP or FTP.

NOS provides the functions uuencode and uudecode found in the Transmission Control Protocol (TCP)\Internet Protocol (IP) environment. Uuencode takes a binary file and encodes into an ASCII text files so that it may be sent by SMTP. When the file is received, it can be run through uudecode to obtain the original binary file. A common practice on the DDN is to use SMTP to transfer large binary files, i. e. executables or images. They can be encoded to ASCII, then split into a number of smaller files, and sent in a mail message. At the receiver the messages are saved to system files, the files are edited to remove all of the mail header information, and then the files are concatenated, in appropriate order, to form the original encoded file. The file is the run through uudecode to obtain the original binary file.

2.2.4 Requirement (4) The system will provide archival storage of messages that are transmitted or received.

AFITPAC - Transmitted messages are saved on computer disk at the receiving end by either the file capture option or the automatic file save function of the program (note that this does not operate in the Secure link mode)(15).

NOS - If a NOS PRU is running and the Simple Mail Transfer Protocol[6] (SMTP) then any messages sent to the users of the PRU will automatically be placed in their mailbox. The mailbox is nothing more than a designated directory for each user of the system. The next time the user connects to this PRU's bulletin board, they will be notified there is mail waiting for them. They

[5]Information can be found in RFCs # 788, 821, 1090
[6]Information can be found in RFCs # 788, 821, 1090

can read the message, delete the message, reply to the message, forward the message, or save the message to another file. If the destination NOS PRU is not running, then SMTP will send a message back to the originator informing him that the message was undeliverable and may offer a possible explanation.

NOS also provides the Post Office Protocol[7] (POP2) application intended for use in networks where hosts are normally off when not in use(21). POP2 allows mail to be collected at a mailserver which is always on so, when a client PRU powers on, it will request mail from the mailserver. The client receives all of the mail destined for it and then distributes it to its users' mailboxes. This type of mail delivery avoids the problem of SMTP not being able to deliver mail to PRUs that are down.

2.2.5 Requirements (5), (6), & (7) (5) The system should provide a transmit message queue so that multiple messages can be transmitted with little operator intervention. The queue should be able to hold at least twenty messages. (6) Queued messages that are not able to be transmitted will be flagged for the operators attention. (7) The system should provide for reception messages whether the position is attended or not.

AFITPAC - (5) - A transmit Queue of 20 files is presently in the system. A larger queue can be created by changing the TQ_MAX in the TXFILE.C module and recompiling the entire program(15). (6) - Files that can not be transmitted will cause the system to stop the transfer and alert the operator by sounding the beep at the computer. The operator can then continue sending the remaining file in the queue simply by pressing the start TX key (F3) again(15). (7) If a station is unattended, files will automatically capture via the automatic file save function built into the transmit and receive modules of the program. The receive module examines every character received

[7]Information can be found in RFCs # 918, 937, 1081, 1082, 1225

and if, it is one of the special function control codes (CTRL-E, CTRL-J, or CTRL-Q), other actions are initiated such as auto-file save or the remote "station heard" command(15).

NOS - If the Simple Mail Transfer Protocol[8] (SMTP) feature of NOS is used, then the only limitation to the number of files that a can be queued for transmission is the amount of free disk space. SMTP uses the directory **N:\NET\SPOOL\MQUEUE** to store messages waiting for transmission. The advantage to this method is if a PRU goes down during transmission, the contents of the message and of the outbound queue are not lost. When the PRU comes back up, it can resume transmission of all unsent messages without user intervention.

If the File Transfer Protocol (FTP) feature of NOS is used; there is effectively no limitation on the number of files that can be queued for transmission. Queuing can be accomplished by using the FTP commands. **mput** and **mget**. However, the method is vulnerable to PRUs going down, meaning that if transmission is aborted, the user must reissue the request.

2.2.6 Requirement (10) A converse mode should be provided to allow interactive communication between two node operators.

AFITPAC - Operators can converse directly with each other just by connecting with another station and then typing on screen. When the operator hits the return key all of the typed data will be sent out as a packet(15).

NOS - NOS provides two methods for conversations between stations; 1) through the normal procedure of establishing a AX.25 connection between the two stations 2) through the Chat command available in the bulletin board system. In both cases, whatever is typed on the screen is transmitted to the other PRU when a carriage return is entered.

[8]Information can be found in RFCs # 788, 821, 1090

The Chat function is similar to the PHONE application available on VAX/VMS systems and the TALK application on Unix systems. Except Chat mode is a direct line to the system operator whose bulletin board the user is logged into. When a user selects Chat, the system notifies the operator of the request. If he is available he can converse with the user requesting to Chat.

2.2.7 Requirement (11) A direct connection capability between the TNC and the GILLAROO, via the packet radio system computer, should be incorporated into the system software to allow for a secure converse mode for the system.

AFITPAC - For packet computers with 2 serial ports and equipment set up as in configuration 1, the operators can converse securely. The operator sets up a connection and both ends go into the secure link mode. All data typed in at the TEMPEST computer is then encrypted by the GILLAROO, passed through the packet computer to the PK-232. When the buffer is full (128 bytes of user typed data) or when the return on the packet computer keyboard is pressed, a packet will be sent to the other station where it will be decrypted and displayed on the TEMPEST computer screen.

NOS The function of requirement (11) is not native to the NOS package. However, further examination and modification of the Point-to-Point Protocol[9] (PPP) could provide this capability. PPP allows for network connections to be made over serial lines and this could be exploited to create the required service.

2.2.8 Requirement (12) A system operator handbook will be provided. It will include system wiring diagrams and a software user's guide.

AFITPAC - The adequacy of the documentation will be determined by the thesis sponsor upon completion of this thesis(15).

[9]Information can be found in RFCs # 1134, 1171, 1172, 1220

NOS - The NOS documentation is a living document, which is continually updated as new features are added. Since the source code for all functions is provided, and the software is free to the public, a myriad of authors have written code and documentation. Ian Wade's book(21) is the first published attempt at formalizing the documentation. All of the documentation is available from a user's manual computer file, distributed with the software.

2.3 Commentary on Comparison

Table 2.1 provides a summary of how NOS compares to the AFLC requirements. There are three requirements, (3), (8), and (11), that NOS does not meet that warrant further discussion. Since the C source code is available, modifications could be made to the NOS package to bring it more in line with the AFLC requirements. In regards to meeting requirement (3), there are two ways: 1) save the GILLAROO encryption devices output to a disk file and transmit the file using either File Transfer Protocol (FTP) or Simple Mail Transfer Protocol[10] (SMTP) 2) create a software module that would encrypt the message files and add it to the NOS package. This second solution would also allow NOS to meet requirement (11), the keyboard input could be piped through the encryption module before transmission. As for requirement (8), the command line prompt could be replaced with a menu system. Using today's C programming tools for window environments an interface could be created to take advantage of NOS's multiple session capability and provide the menu driven interface.

There are other factors that should also be considered in this comparison. First, AFITPAC is limited to the area that can be covered by the packet radio network that it supports. AFITPAC nodes that become isolated due to problems with radio transmission have no other recourse than to hope that broadcasting conditions improve. NOS nodes have the option of using DDN to "wormhole" the transmission

[10]Information can be found in RFC's # 788, 821, 1090

Req. #	Meets Req.	Does Not Meet Req.	Comments
(1) (8) (9)	X X X	 X 	The bulletin board system is relatively easy to use. It is menu driven using single letter commands. Help is available for each command typing **Help** command. The command prompt interface is slightly more difficult to use. However, experienced DDN users will find it a familiar interface. Three levels of help are available: 1. Typing *help* or *?* lists the commands accessible. 2. Typing the command followed by a *?* lists the parameters acceptable for the command. 3. Using NOSVIEW to read the detailed documentation for each command.
(2)	X		Three routing methods are provided: 1. RIP 2. RSPF 3. AX.25 digital repeating
(3)		X	NOS does not provide any data encryption services.
(4) (5) (6) (7)	X X X		NOS provides SMTP services which do not require operator intervention to transmit and receive messages. SMTP uses disk directories for its transmit and receive queues, allowing for queue sizes as large as the disk space will support. If a message is unable to be delivered SMTP returns a message to the originator indicating the problem. This message is treated just like any other piece of mail and will exist until the originator deletes it.
(10)	X		Converse mode is provide by two means: 1. AX.25 connections 2. Chat command on the bulletin board system.
(11)		X	NOS does not provide a transparent link from another serial that would allow the TEMPEST machine with the GILLAROO card to use the PRNET.
(12)	X		Comparing the AFITPAC documentation with the NOS documentation, NOS meets the users requirements.

Table 2.1. Summary of how NOS compares to AFLC Requirements

of AX.25 packets encapsulated in Internet Protocol (IP) packets(21). The protocol used for this is declared in RFC1226. This would allow groups of NOS packet radio networks to be linked via DDN cabling. The procedure in NOS for making this type of connection is **connect** *destination* **via** *local_axip_call remote_ax25_call*, which is the same as an AX.25 connection made via digipeaters. Wormholing AX.25 over DDN or similar networks dramatically increases the range of the NOS packet radio networks.

Second, both packages supply complete C source code listings. Since AFITPAC is a much smaller package, there is less code to understand before making modifications. NOS has many modules written by scores of authors. The version of the source code obtain for this project has approximately 270 program files, header files, and libraries. These factors cause a problem for anyone who needs to make modifications to the package as Lebano found(14).

> The design and implementation of the DES [Data Encryption Standard] Enhanced KA9Q [NOS] Internet package was an evolutionary process. this research project was undertaken with a minimal understanding of the TCP/IP protocols and no knowledge of packet radios and the C programming language. In addition, the KA9Q [NOS] package, although very well structured, did not have any true design documentation other than a user's manual and had limited documentation in the source code. This presented a learning curve which had to be overcome before design modifications could be achieved.(14)

Third, AFITPAC is the property of the United States Air Force, allowing for the unfettered distribution and modification of the software. NOS is currently free for distribution to the public for non-commercial use. However, all portions of the package are copyrighted by the authors. This means that the Air Force's use and modification of the NOS software is best determined by the lawyers.

2.4 Summary

If there are no legal barriers to using the Network Operating System, NOS, then this should be the package of choice. It satisfies the AFLC requirements and in those areas that it falls short, it was shown how NOS could be modified to meet them. NOS provides the services spelled out in five Department of Defense (DOD) Military Standard Protocols: 1) MIL-STD-1777 Internet Protocol (IP), 2) MIL-STD-1778 Transmission Control Protocol (TCP), 3) MIL-STD-1780 File Transfer Protocol (FTP), 4) MIL-STD-1781 Simple Mail Transfer Protocol (SMTP), and 5) MIL-STD-1782 TELNET Protocol(17). These standards are used throughout all DOD organizations. Therefore, the use of NOS could be widely accepted because of its familiarity to computer network users DOD wide.

Finally, NOS provides many applications to its users, allows them to have multiple concurrent sessions, and has many services that run unattended. An active NOS node could easily generate a tremendous amount of network traffic. An AFITPAC node however, only allows for one session to be running at a time. While a node is transferring a file the operator cannot connect to another node. Also AFITPAC has only one service that runs unattended, which is Geier's routing algorithm. The file capture routine of AFITPAC can receive files while the station is unattended, but someone must initiate the transmission and once the transmission is complete the function terminates. Both Geier's routing algorithm and the NOS unattended services can be influenced by the operators, but they run continuously while the node is active. Consequently, a comparison has to be made between NOS and AFITPAC to determine which yields better channel usage for the amount of services provided. Is the fact that NOS provides all of these services outweighed by a saturated packet radio network because of them? Does AFITPAC poorly use the available channel capacity or provide such limited services that it does not meet user's needs? The answer to these questions lies in the task about to be undertaken: the design of a simulation to analyze the performance of packet radio network programs.

III. Discussion of Model Design and Methodology

3.1 Introduction

This chapter begins with a discussion of modularity and how it relates to both computer communications networks and simulations. Dr. Bernard Zeigler's concepts of modules with ports which communicate via messages are introduced(22). The messages are generated by both external and internal events and the components are responsible for processing the messages. Then by examining the physical and data link layers in the Open System Interconnection (OSI) network model, as they pertain to a PRNET, a design for a packet radio network simulator is derived. The development of simulation modules to mimic the layers above the data link layer are not necessary because these are the functions being analyzed.

3.2 Model Design Techniques

Any discussion that involves computer communications networks and simulations begins with the topic of modularity. Modularity is defined as "the ability of a system to be expanded to [be] changed with minimal difficulty.(6)" For the purposes of this discussion, modularity takes this definition one step further saying that it is achieved by using discrete components or modules. Zeigler discusses some of the benefits and characteristics of a simulation designed using discrete components(22). There are two significant points made in the article. First, a component possesses input and output ports through which it interacts with its environment. In the discrete-event case, the events determine what values will appear on these ports. The external events cause activity on the input ports of these components, which then respond to this activity. During the response the component may change its state causing internal events, which may cause activity on the output ports. Second, a benefit of modular construction is that components can be independently tested.

The ability to do such testing at each stage of development improves the reliability and efficiency of the final simulation model.(22)

The ideal of modularity motivated the International Standards Organization (ISO) in 1984 to adopt its seven layer model for computer communications networks(18).

> The communications functions are partitioned into a vertical set of layers. Each layer performs a related subset of the functions required to communicate with another system. It relies on the next lower layer to perform more primitive functions and to conceal the details of those functions. It provides services to the next higher layer. Ideally, the layers should be defined so that changes in one layer do not require changes in the other layers(18).
> – *Stallings*

This is the reference model used to define standards for linking heterogenous computer systems.

Each layer in this communications model can be thought of as a discrete subcomponent. All that each subcomponent needs to know are: the services provided by the other subcomponents, and which port should send messages to obtain such services. For the purpose of this discussion, we are mainly interested in the first four layers of the model in Table 3.1(18), namely physical, data link, network, and transport. These will be used to explain the OSI terminology. The OSI model calls the ports of each layer service access points (SAPs)(18, 19). The transport layer sends a service request to the network layer across their common interface at the SAP. The service request is in the form of an Interface Data Unit (IDU). The IDU consists of a Service Data Unit (SDU) and Interface Control Information (ICI)(18, 19). The network layer examines the contents of the ICI to determine the exact type of services requested. The network layer will determine the length of the SDU from the transport layer, and fragment it, as necessary. Then each fragment receives its own network layer header to form a Protocol Data Unit (PDU). The network layer makes a service request to the data link layer via a SAP. The network layer will pass

Layer Name	Description
1. Physical	Concerns the transmission of unstructured bit stream over physical medium; deals with the mechanical, electrical, functional, and procedural characteristics to access the medium.
2. Data Link	Provides for the reliable transfer of information across the physical link; sends blocks of data (frames) with the necessary synchronization, error control, and flow control.
3. Network	Provides upper layers with independence from the data transmission and switching technologies used to connect systems; responsible for establishing, maintaining, and terminating connections.
4. Transport	Provides reliable, transparent transfer of data between end points; provides end-to-end error recovery and flow control.
5. Session	Provides the control structure for communication between applications; establishes, manages, and terminates connections (sessions) between cooperating applications.
6. Presentation	Provide independence to the application processes from differences in data representation (syntax).
7. Application	Provides access to the OSI environment for users and also provides distributed information services.

Table 3.1. The OSI Layers

its PDU as an SDU to the data link layer along with some ICI. See Figure 3.1 for clarification.

Note that on the receiving end of the communications network, each layer receives a PDU from its peer layer at the transmitting end. The network layer would have received all of the PDUs from the transmitting network layer and recombined them, as necessary, into a PDU for its transport layer. The receiver's transport layer would be signaled by the receiver's network layer that data was available for it at the SAP. The transport layer would then process the PDUs and pass a PDU up to the session layer in a similar fashion.

The transport layer's service request is viewed as an external event by the network layer. This event places data (SDU) on an input port (SAP). The network layer processes the SDU and if the ICI causes internal events, the network layer makes a service request of the data link layer. The similarity between the OSI layers passing of SDUs and Zeigler's discussion of passing messages between discrete components motivates the modular design for the Packet Radio Network simulation. The most difficult task in the design process was determining which components to model and the level of abstraction needed. The problems encountered during the design process and the attempted programming will be discussed in the Chapter IV. The design solution chosen is the next topic of discussion.

3.3 Overview of the Design

As discussed previously, a packet radio network consists of a group of Packet Radio Units (PRUs). Each PRU consists of a computer, a Terminal Node Controller (TNC), and a transceiver. The computer may be of any brand name and running any operating system as long as it has some way to communicate with the TNC. The TNC may be from various vendors and can support different protocols. The broadcast radio may be from any vendor and support various frequencies. The diversity of products makes it necessary to construct a simulation model based on

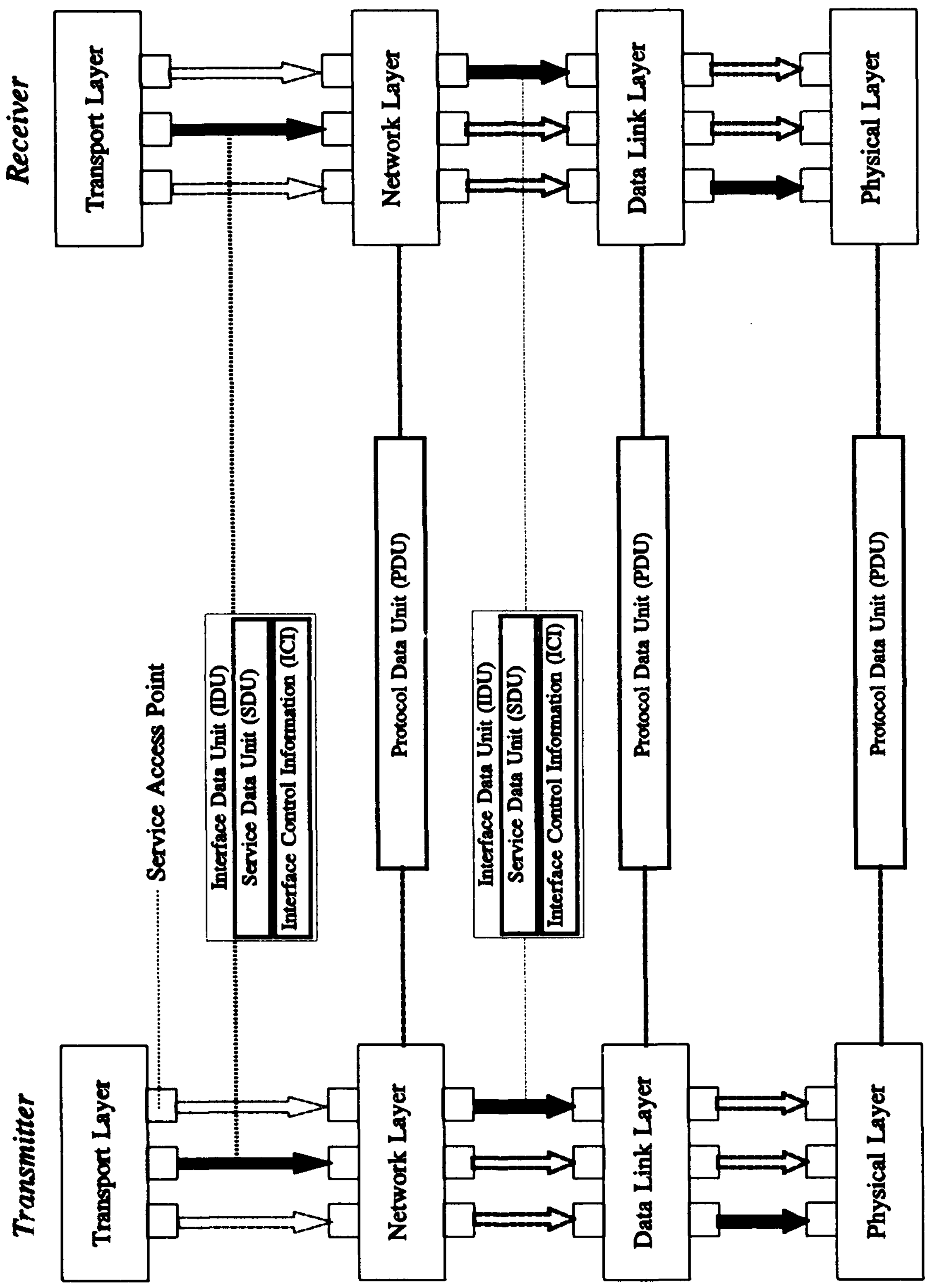

Figure 3.1. Figure Illustrating PDU, SDU, and ICI Passing.

behavioral implementations of the components. For the purpose of this thesis, the collection of all of these components will be called a Node.

The simulation design consists of multiple Nodes passing messages to each other through another component called a Channel. The Channel will take the messages from the Nodes and distribute them to the other Nodes in the simulated network. As the simulation progresses, data about the message traffic will be collected by a Monitor. The data collected will be analyzed upon completion of the simulation.

3.4 The Channel

The logical place to start is the bottom at OSI layer one, the physical layer. In the specific case of a packet radio network, the physical layer deals with radios and the physical transmission medium, the free space. The module used to model this layer is referred to as the Channel. The Channel provides only one service which is broadcasting of frames between Nodes. The Channel must know which nodes can hear other nodes, and this information is provided by the Connectivity Matrix. The two dimensional Connectivity Matrix is indexed by node identifiers. Node identifiers could be unique alphanumerics, as in Figure 3.2 or callsigns e. g. AF8EN, AF8LS, WPA4B . Each element would be a "1", (source can hear destination), or a "0", (source can not hear destination).

Xmt.\Recv.	A	B	C	D
A	1	0	1	0
B	0	1	0	1
C	1	0	1	1
D	0	1	1	1

Figure 3.2. Connectivity Matrix

In this example of the Connectivity Matrix, Figure 3.2, node A can hear itself and node C, but can not hear nodes B and D. This matrix holds the network's topology. Some might argue that this makes the Channel too smart since it should only be responsible for the transmission of frames to nodes. Perhaps this information should be kept by the simulation manager or the nodes. However, by examining a network which uses cabling to connect its nodes, it becomes obvious that it is the physical connections that determine the network's topology. Thus the Channel is the appropriate place for this information.

The frame is the basic unit sent over the transmission medium and the Channel must know its source in order to route it properly. Therefore, the frame must contain a field, Transmit_Source, which holds the node identification of the transmitter. When the Channel receives a request to broadcast a frame, it will examine the Transmit_Source field, find the appropriate row of the Connectivity Matrix, and pass a copy of the frame to all the nodes (except the source) that can hear it.

For example, Node C has a packet to send to Node D, see Figure 3.3. Node C sets the Transmit_Source field of the frame to "C" and places the packet in the frames data field. Node C requests the Channel to broadcast the frame, passes the frame to the Channel. The Channel examines the Transmit_Source field and finds "C" is the source. The Channel looks through the transmitter index of the Connectivity Matrix, Figure 3.2. When it finds the row corresponding to "C" as the transmitter, it indexes through the receivers to see which Nodes can hear Node C. It then makes a copy of the frame for each of these Nodes, Nodes A and D. A copy is not made for Node C because a Node can not receive frames while it is transmitting. It is necessary to make copies of the frame instead of passing it by reference to the nodes, otherwise they all would be modifying the same frame.

This method of passing frames using the Connectivity Matrix implies that the Channel has no propagation delay and is error free. Unfortunately, neither of these is a realistic assumption. There is a significant amount of time spent in transmitting and receiving a frame since we are working with radio data rates of 300 or 1200 baud. Furthermore, if the distance between two connected nodes is large enough, there will be a delay introduced by radio wave propagation through the free space.

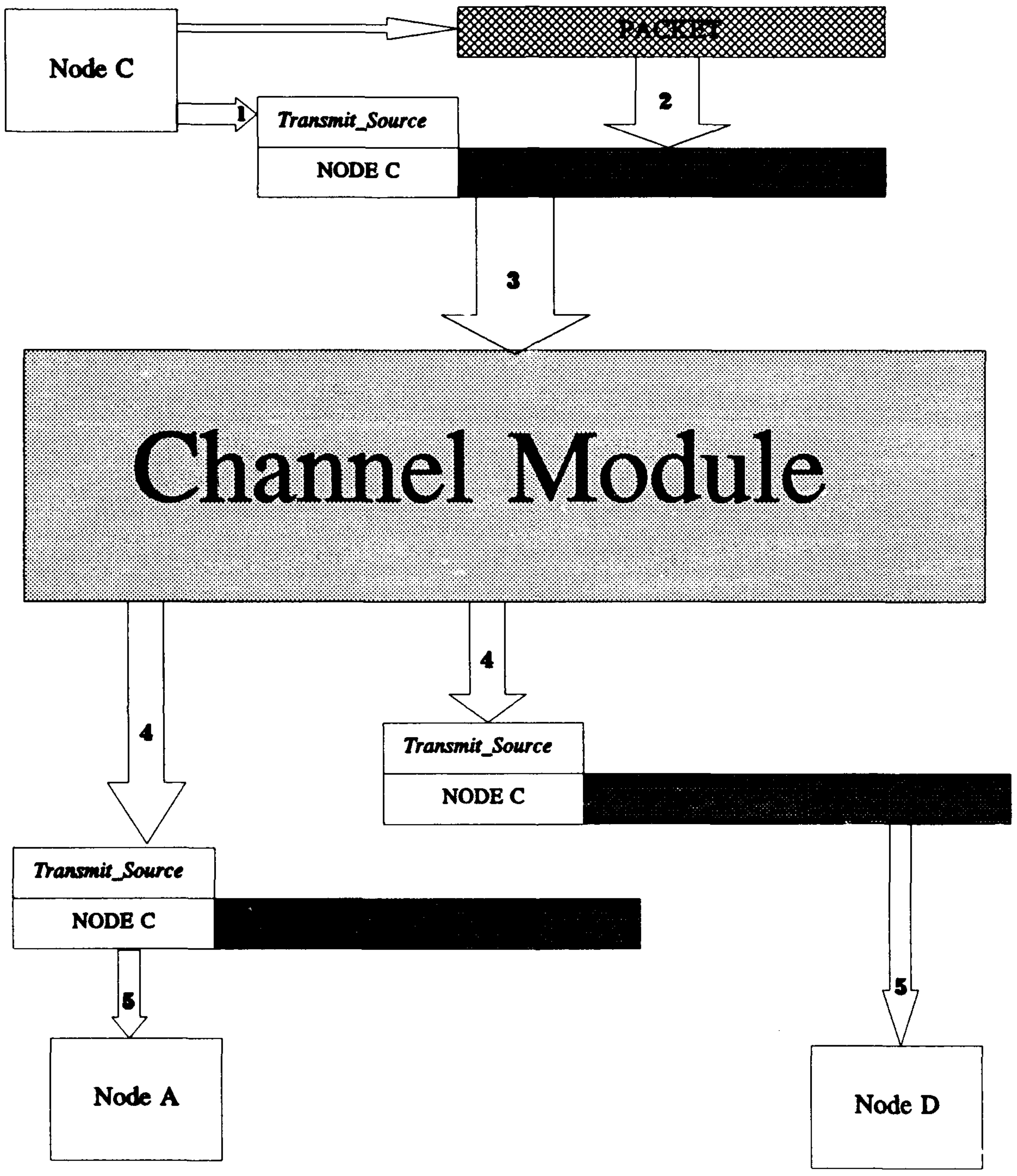

Figure 3.3. Example of Nodes passing a frame through the Channel, broadcasting

For the purposes of this thesis the time spent transmitting and receiving plus the propagation delay will be referred to as the delivery delay.

	Frame Size in Bits	
Radio Data Rate (Baud)	152 (Minimum)	2656 (Maximum)
300	0.506 seconds	8.85 seconds
1200	0.126 seconds	2.213 seconds

Table 3.2. Limits on Packet Transmission Delays

The transmission delays shown in Table 3.2 are large compared to the delays caused by radio wave propagation between nodes. Therefore the delivery delay need only consider the transmission delays. However, if the data rates increase to 9600 baud and possibly 56000 baud, the transmission delay will reach the same magnitude as the propagation delay and thus the delivery delay would need to consider both.

Assuming that propagation delay is negligible means that the moment a node begins transmitting another node begins receiving. Then the transmission delay needs to be counted only once and can be simply realized at the delivery stage of the Channel. At this point, the frame definition must go through another iteration, adding on a field containing its length, Frame_Length. The Channel would examine this field prior to passing the frame to the destination Node to determine the appropriate delivery delay. It would schedule the passing of the frame to the Node at the appropriate simulation time. By placing this delay in the Channel, the propagation delay can be included in the delivery delay calculation, if it becomes necessary.

The other assumption was the Channel was error free. To correct this erroneous assumption, the concept of the Probability of Bit Error (P_b) Matrix is introduced (please see Figure 3.4). The P_b Matrix is similar in construction to the Connectivity Matrix except the elements contain a P_b threshold for the transmitter receiver pair. The P_b Matrix works as follows: When the Channel receives a frame for broadcasting it reads the frames Frame_Length and Transmit_Source fields, calculates the probability of frame error for the given frame length and P_b Matrix entry, and then

Xmt.\Recv.	A	B	C	D
A	0	0	2.46E-10	0
B	0	0	0	3.67E-15
C	2.46E-10	0	0	2.171E-13
D	0	3.67E-15	2.171E-13	0

Figure 3.4. P_b Matrix

determines if the frame is in error. This is accomplished by using a random number generator with the probability distribution for the type of channel modeled. If the number returned when the random number generator is called exceeds the probability of frame error then the Channel will mark the frame. If only one copy of the frame is used for all the nodes, then the last calculation of frame error would apply to all the receiving nodes. Therefore, it is necessary to make copies of a frame for each receiving Node during the broadcasting process. In order to mark the frame as being in error, another field must be added to our definition of a frame. This new field will be called Frame_Error and can use a simple toggle switch to indicate error or no error.

There are other control items that need to be addressed concerning the interface between the Channel and Node modules. The first is that of carrier sensing. In the current scheme of packet radio, each node listens before it transmits. If the node does not hear anything, it transmits. The Channel must be able to let each Node know if another Node that it can hear is transmitting. This is accomplished by telling each node that can hear a particular frame transmission to set a carrier flag associated with that Channel. At the instant when the frame is received by the Channel, it sets a carrier flag, and makes a copy of it for the receiving Node. The carrier flag is cleared at the same time the Channel is scheduled to deliver the frame to the Node. The interaction between the Nodes and the Channel controls the Nodes access to the Channel.

Another control item to be addressed is that of frame collisions. If two or more nodes decide to transmit at exactly the same simulation time a collision occurs. Since the radio can not listen while it is transmitting, handling a collision is not as simple as with an Ethernet transceiver, which continues to listen on the cable while it is transmitting. If it hears another signal during its transmission a collision is detected, transmission is terminated, and a special signal is sent over the cable indicating a collision. In packet radio, however, the radio switches between transmit and receive so the only way a collision is detected is by not receiving an acknowledgement for a packet. However, for purposes of this project, the discrete event simulation frames are delivered and received sequentially and instantaneously. At the time a frame is scheduled to be delivered to a Node the whole frame is sent at once by the Channel. The whole intent of introducing the delivery delay was to simulate the real world delay encountered in transmitting and receiving. Yet striving for realism creates another problem, the problem of how to deal with frame collisions.

Once again the two-dimensional matrix appears to be a feasible solution. The Collision Matrix could be indexed by node identifiers and each element composed of a Frame_Delivery_Pending flag and the Time_of_Next_Delivery. Along with this matrix, a Collision Array indexed by node identifiers is needed. Each element of the array requires a Collision_Indicator flag and the Collision_End_Time. Each element of the matrix and array is initialized to zero. When a frame is received from Node C, by the Channel for broadcasting, the Channel sets the Frame_Delivery_Pending flag and sets the Time_of_Next_Delivery equal to the current simulation time plus the transmission delay (Figure 3.5).

At the same simulation time that the frame is originally processed by the Channel, it examines the row of the Collision Matrix that corresponds to the receiving node. In this particular case there are no frame deliveries pending for this destination node (Figure 3.5). This results in the Collision Array looking like Figure 3.6. When the next frame is transmitted from Node B, the Channel checks the Connectivity Matrix, Figure 3.2. The Channel would copy the frame and deliver to Node D. At the same simulation time the Channel updates the Collision Matrix as in Figure 3.7.

Xmt.\Recv.	A	B	C	D
A	0	0	0	0
	0.0	0.0	0.0	0.0
B	0	0	0	0
	0.0	0.0	0.0	0.0
C	1	0	0	1
	10.0	0.0	0.0	10.0
D	0	0	0	0
	0.0	0.0	0.0	0.0

Figure 3.5. Collision Matrix - transmission of frame from C

	A	B	C	D
Collision_Indicator	0	0	0	0
Collision_End_Time	0.0	0.0	0.0	0.0

Figure 3.6. Collision Array

Once the matrix has been updated, the Channel examines the entire column corresponding to the receiver to determine if there are any other frames pending delivery. This time there is another frame pending delivery to Node D. The Channel then compares the two Pending_Delivery_Times to see which is greater, it sets the Collision_End_Time equal to the greater time, and sets the Collision_Indicator in the Collision Array. This results in a Collision Array like Figure 3.8.

When the simulation time comes to deliver the pending frames, the Channel should check the Collision Array. If the Collision_Indicator is set, it should compare the Collision_End_Time with the current simulation time. If the current simulation time is less than the Collision_End_Time, the Channel should discard the frame. If the current simulation time is greater than the Collision_End_Time, the Channel should clear the Collision_Indicator, zero the Collision_End_Time, and deliver the frame to the receiver. Once the frame is delivered, the Channel should reset the Collision Matrix element corresponding to the transmitter receiver pair, unless there is another frame pending delivery from this same pair. The Channel could tell if the

Xmt.\Recv.	A	B	C	D
A	0	0	0	0
	0.0	0.0	0.0	0.0
B	0	0	0	1
	0.0	0.0	0.0	9.367
C	1	0	0	1
	10.0	0.0	0.0	10.0
D	0	0	0	0
	0.0	0.0	0.0	0.0

Figure 3.7. Collision Matrix - transmission of frame from B

	A	B	C	D
Collision_Indicator	0	0	0	1
Collision_End_Time	0.0	0.0	0.0	10.0

Figure 3.8. Collision Array - after collision is detected between the transmissions of nodes B and C

matrix reflected another frame delivery (other than the one just made) by comparing the Time_of_Next_Delivery with the current simulation time. If they are equal, then the frame just delivered was the one indicated by the matrix. If the simulation time is greater, there is another later frame awaiting delivery from the same source and the matrix element should not be changed.

The reasoning behind the design of the collision mechanisms is necessary for clarity. It is not unreasonable to change the Time_of_Next_Delivery every time the transmitter requests the Channel to broadcast a frame. This is because a node can only transmit one frame at a time, thus consecutive transmissions from the same node could not collide with one another. All of the frames transmitted prior to the last one transmit without collision. If, however, another node transmits while there are a number of pending delivers for this receiver, the Collision_Indicator is set at the start of the interfering transmission so that any frames pending delivery after this would be discarded. When a collision occurs at one receiving node, it does not have to occur at all the receiving nodes. If only one copy of the frame is used, discarding it leaves nothing to deliver to the clear nodes. Making a copy of the frame for each receiving node prior to delivery appears to be a sensible approach to prevent loss through automatic discard.

The collision mechanism also resolves the problem of hidden nodes in a packet radio network. Resolution occurs when the Channel examines the column of the Collision Matrix corresponding to the receiver and gets a broadcast request. In this example, Node B can talk to Node D and Node C can also talk to Node D, but Node B and Node C can not talk to each other, as shown in Figure 3.9. When

Xmt.\Recv.	A	B	C	D
A	1	0	1	0
B	0	1	0	1
C	1	0	1	1
D	0	1	1	1

Figure 3.9. Connectivity Matrix

Node C transmits to Node D, the Channel would ask Node D to set its carrier flag. Thus Node D would not be able to transmit until the carrier flag was cleared. However, Node B remains unaware that Node C is transmitting to Node D. As far as Node B is concerned, it is free to transmit. When it transmits, Node A will hear it and the Channel will have Node A set its carrier flag. Yet, Node D is currently receiving something from C and the two transmissions collide at Node D only. The collision affects only the receiving node involved.

An essential component of the Channel is a Node Array for keeping track of the Node modules' unique identifiers and memory reference variables. The Node Array is created at run-time and each element of the array consists of a unique Node identifier, possibly the callsign, and the corresponding memory reference variable (Figure 3.10). By creating the Node Array at run-time its size is governed by the number of Nodes the user requests, thus making the simulation's scenarios dynamic.

	A	B	C	D
identifier	WA3ULK-10	W8PUG-1	KA9R-13	OU8PI-2
ref. variable	0x54984h	0x328c9h	0x1ac4h	0x2ae9h

Figure 3.10. Node Array

Note that all of the matrices and arrays in this thesis use the indices of A, B, C, and D. This is not meant as an implementation detail, but to show that all the components should be indexed by the same data type. When the Channel searches for information concerning a given Node, it will use the same variable to search in each component. The Channel can locate the particular Node it wishes to communicate with in the Node Array. It will set a variable equal to the memory reference variable for that Node, and pass that variable to a generic communications routine. The Channel thus has only one set of routines to communicate with all of the Nodes in the simulation.

Another method for giving the Channel this knowledge about the Nodes is to program a unique identifier for each Node into the Channel. With this method the simulation would need to be recoded and recompiled every time a new scenario is analyzed. Unfortunately, this forces the user to be knowledgeable about the simulation language, and to have access to a compiler for the specific language. Thus far the researcher has attempted to keep the discussion of the Channel design at an abstract level to avoid specifying implementation details. This was done to facilitate the possibility of creating a simulation using any language or any hardware platform desired.

3.5 The Nodes

The Channel knows about the Nodes, through the Node Array, and will communicate with them in this simulation. In Section 3.4 the discussion of the Channel defined a field of the Node module, called the carrier flag. Recall from Section 3.3 that a Node is defined as a group of components, a computer, a TNC, and a transceiver.

Since the TNC is the component that will listen to the radio channel for a transmission, it is appropriate to assign it the carrier flag. In the real world the broadcast radio actually does the listening to the channel for the TNC. However, the radio only acts as an interface between the TNC and the channel. Also for the purposes of this project, it is not necessary to simulate the method of transmission, e. g. Frequency Modulation (FM), Amplitude Modulation (AM), etc.. This factor is taken into account in the Channel's Probability of Bit Error Matrix. Therefore, the radio's listening function and the carrier flag are given to the TNC.

3.5.1 TNC Module If both AFITPAC and NOS will be analyzed using this simulation, then two types of a TNC must be designed: 1) a full implementation which includes the AX.25 protocol[1] and 2) a limited implementation which includes the KISS protocol[2]. AFITPAC expects the TNC to provide all of the services in the AX.25 data link layer and expects to be able to interface with the TNC using the TNC's command syntax. The NOS package expects to have a TNC operating using the KISS protocol and all of the higher level protocols, e. g. IP, AX.25, High Level Data Link Control (HDLC), etc., are found in the NOS software modules. The original purpose of this thesis was an interface the AFITPAC software, therefore the focus will be placed on the first type of TNC design.

AX.25 Protocol Module

Control Field Handler Basically, AX.25 is a bit level protocol and depends on bit sequences to function. The Control Field processing is the only part of the AX.25 protocol necessary to implement at this level of detail. All other protocol parts can be implemented using higher level data structures and behavioral modeling. Since the control field is always eight bits long, it makes sense to create it as a fixed array of length eight. Each element of the array can be any data type

[1]For protocol information see Appendix A, Page A-1
[2]For protocol information see Appendix A, Page A-3

supported by the implementing language. For this project a boolean data type, TRUE or FALSE, is used, since this most closely resembles the behavior of a bit to be either 1 or 0. It is then possible to create and analyze these bit patterns for every packet transmitted and received. The receiving Node can respond according to the AX.25 protocol once the Control Field has been examined. Before a receiving TNC should bother to examine the packet's Control Field, it must determine if the packet is destined for this Node.

Packet Destination Address Handler When a Node's TNC receives a packet, it checks the addresses in the Destination Field and the elements of the Digipeater[3] Array. If the Destination Field matches the Node's address the TNC proceeds with processing the packet, compares it's Node's address with the elements of the Digipeater Array. The TNC uses the following algorithm for the comparison to assure that the function adheres to the AX.25 standard:

1. If there are no more elements in the Digipeater Array then exit
2. If there are elements in the Digipeater Array then either choose:
   (a) The selected element does not match this Node's address then
      i. If last character in the element is an "H" then select next element and go to 1
      ii. If last character in the element is not an "H" then discard the packet and exit
   (b) If the selected element does match this Node's address then
      i. Add an "H" to the end of this element and place back in the array
      ii. Set the Transmit_Source field of the frame to this Node's address

[3]*Digipeater* - "digital repeater, a device that receives, temporarily stores and retransmits (repeats) packet-radio transmissions that are specifically addressed for routing through the digipeater(12)

iii. Request the Channel to broadcast this frame

iv. Dispose of the local copy of this frame and exit

The "H" is added on to the end of the element that matches this Node's address to indicate that this packet Has-Been-Repeated. The "H" was chosen to coincide with the AX.25 name for this bit, the "H-bit". If the frame happens to return to this node, the TNC will not recognize this Digipeater Array element, because it is not an exact match for its own address.

The order in which the elements of the Digipeater Array are scanned is important because it corresponds to the order in which the packet is routed through the digipeating Nodes. If the Node's address does not match the selected element and this element ends in an "H", then TNC can go on to the next element. This procedure prevents the packet from being digitally repeated by two nodes at the same time.

For example, given the Connectivity Matrix in Figure 3.11 and that Node B needs to send a message to Node A, Node B has two routes to get the message to Node A: 1) from Node B to Node D to Node C to Node A ($B- > D- > C- > A$) and 2) from Node B to Node C to Node A $B- > C- > A$). Node B does not know about the direct connection to Node C and so it decides to use the first route. Figure 3.12 shows an example of the AX.25 packet header.

When Node B asks the Channel to broadcast this packet[4], Nodes D and C receive a copy. Node D finds that its address is the first in the Digipeater Array and therefore Node D digitally repeats the packet. Node C processes the packet and finds Node D's address is the first element in the array, discovers no "H" at the end of the element, and discards the packet. Node D now requests the Channel to broadcast this packet whose AX.25 packet header is like Figure 3.13. Nodes B and

[4]The word packet is used here to remain consistent with the level of the design. The Node would still send the Channel a frame and all the fields associated with it.

Xmt.\Recv.	A	B	C	D
A	1	0	1	0
B	0	1	1	1
C	1	0	1	1
D	0	1	1	1

Figure 3.11. Connectivity Matrix for Digital Repeating Example

Destination Address	Source Address	Digipeater Array	
A	B	D	C

Figure 3.12. AX.25 packet Address Header for Digital Repeater Example

Destination Address	Source Address	Digipeater Array	
A	B	DH	C

Figure 3.13. AX.25 packet Address Header for Digital Repeater Example

C receive a copy of the packet. Node B does not find its address anywhere in the header, because a receiving node does not examine the Source Address and discards it. Node C examines the Digipeater Array, finds DH in the first element, it finds that the element ends in an "H", and finds its address is the next element in the array. Node C processes the packet for digital repeating and makes a request to the Channel to broadcast this packet, whose AX.25 packet header looks like Figure 3.14. Nodes B, D, and A all receive a copy of the packet. Node B discards the packet

Destination Address	Source Address	Digipeater Array	
A	B	DH	CH

Figure 3.14. AX.25 packet Address Header for Digital Repeater Example

as before. Node D examines the Digipeater Array, it does not find its address in any of the elements and, therefore discards the packet. If Node C were allowed to digitally repeat the first packet (because its Node address was in the Digipeater

Array) Node A would have received two copies of the same packet. Node A only receives one copy of the packet and proceeds to analyze the Control Field.

Behavioral modules could be designed for the rest of the AX.25 protocol in a similar fashion to the digital repeating module. The digital repeating function of the TNC is the protocol's simplest part because it does not require any acknowledgements or control mechanism handling.

PC Interface Module This way of handling the digital repeating algorithm using the "H" bit is an example of how protocol parts can be behaviorally implemented using higher level data types. The method detailed only deals with the data link layer protocol found in the TNC. The interfacing of the TNC's data link layer with the rest of the model has not been addressed. AFITPAC expects to control the TNC by using the normal command line syntax. For example, when AFITPAC establishes a connection to another PRU it literally sends "c NODE1 via REPEATER1", one character at a time to the TNC. AFITPAC uses assembly language routines to communicate with the port on the PC connected to the TNC. Therefore, AFITPAC could be link to the simulation by constructing a PC interface module that would replace these functions, e.g. Ser_send(), Ser_recv(), etc., and then translate the TNC command syntax into service requests of the TNC simulator.

3.5.2 PC Module In the real world AFITPAC and NOS would operate on a personal computer. Thus the module in which these packages operate is called the PC Module. The PC module contains the software package to be analyzed, an interface to the programs for the simulator, and a simulated user. The simulator's interface to the programs is needed so that the parameters of the software packages can be changed. This interface will either require modifications to AFITPAC and NOS or use direct calls to the packages' internal functions.

The changing of these parameters would effect the performance of the packages automated functions and thus the amount of traffic generated on the network. It is not necessary however, to interact with the software to generate normal message traffic. The simulation employs a user to test total efficiency and efficacy, and is designed as follows:

1. A random number generator to simulate the rate of traffic generation. Using diverse random distributions, different traffic loads can be simulated. Most random number generators return a value between 0.0 and 1.0. When the returned value exceeds a defined threshold a message transmission occurs.
2. A random number generator to simulate the length of the traffic generated. In this case, the distribution is partitioned into bins from zero to a defined maximum message length. When the value returned by the random number generator falls in a bin, the size of the message is determined.

3.6 Summary

The chapter opened with a discussion of modularity and how it related to simulation design. The concept of discrete components was introduced to provided a mechanism by which a modular simulation could be designed. The layers of the OSI network model were associated with the discrete components to demonstrate how a computer communications network could be simulated. By examining the layers of the network model a design for a packet radio network simulation was proposed. This simulation model included a Channel module and a Node Module. The Channel Module was broken down in to its subcomponents: Connectivity Matrix, Probability of Bit Error Matrix, Collision Matrix, and Node Array. Each one of these was described in detail. In describing the Channel module a message unit called a frame was developed. The frame was the message unit passed between Nodes by the Channel. The Node module was also divided up into its subcomponents: TNC module and PC module. The TNC was decomposed further into an AX.25 module and a PC Interface module. The AX.25 module contained the Control Field Handler, which evaluated the contents of a packets control field. It also included an Packet

Destination Address Handler, that determined if the Node was the destination for a received packet or ıf the Node was to digitally repeat the packet. The remainder of the AX.25 protocol was not explicitly discussed, however it was mentioned that it could be implemented behaviorally using higher level data types. The PC Interface module was introduced as a gateway between the software packages being analyzed and the simulated TNC. A brief description was then given of the subcomponents of the PC module: Interface to Programs and simulated user.

This closes the discussion of a packet radio network simulator design. The Node module components are defined in less detail because of developmental problems in the early stages of their design. As the development cycle progressed the details of the design suffered when coding problems occurred. The exact nature of these problems is discussed next, in Chapter IV.

IV. Conclusions & Recommendations

4.1 Introduction

Chapter IV begins with a description of problems encountered with the PC-DOS version of MODSIM II^{TM}. These problems slowed the development of a working simulation and eventually led to the project's end. Following the discussion of problems, the author makes recommendations for further research and development.

4.2 Problems with MODSIM II^{TM} -

The first problem was that a major concept in the MODSIM II^{TM} language did not function as documented in the manual. The problematic concept was EXPORTing Object types in the definition module for use by other Objects. If, for example, a "parent" object was to tell another object, called "child", to perform a function, MODSIM II^{TM} requires that the reference variable used to command the child to do something be of type "child". It is desirous to define the "parent" as having a field of type "child" object. Of course, the "child" object will want to respond to the "parent" object, therefore, it will need to have a field of type "parent" object. The MODSIM II^{TM} manual states each object should EXPORT its definition for use by the other object. This does not work! Attempting to compile two objects, each having a field of the other object type, results in a cyclic definition error.

After some attempts to work around this problem, the manufacturer was contacted. CACI Products Company responded that this was indeed a known problem with the compiler. There are no plans to correct the problem because they no longer plan to support the DOS version of the package. They were able, however, to provide a workable solution to the problem. Define the "parent" and "child" objects as having a field of type ANYOBJ, which is a generic reference variable to an object. However, do not tell an ANYOBJ to do anything, because the information about its methods are unknown. In order to use this solution, IMPORT the object type into the implementation module. For example the "parent" object's implementation module would contain the line; *FROM childmodule IMPORT ChildObjectType;*. Then the "parent" object coerces ANYOBJ into a "child" object type, a concept similar to that of type casting in C^{++} and ADA. Then use this coerced variable to tell the child to do something. This solution did work and allowed for further simulation development.

The second problem encountered was with the use of the monitor objects. These are a special class of objects which MODSIM II^{TM} provides for the collection of data concerning records and fields. "Monitoring may be specified as being left, right, or left and right. Left monitoring means that any time the variable or field is updated monitoring methods that you have specified will be invoked. Right monitoring invokes the specified methods whenever the variable or field is referenced.(3)" There are three advantages of using monitor class objects; 1) less code is needed for monitoring variables and fields, 2) changes to the monitoring functions only have to be made in the monitor's definition, and 3) the monitoring functions can be enabled or disabled without effecting the simulation. The monitor objects were to be used to track frames in the PRNET simulation. An attempt to monitor two of the same type of variables in the same object resulted in the simulation test ending in error, specifically "Unidentified FPE". The MODSIM II^{TM} manual only repeated what the error messages said on the screen and provided no further explanation. Confidence is high that monitoring two variables is the problem because when one of the monitors is removed the test program executed properly.

The most challenging problem came when testing out a generic record queue and timer. The timer module tested fine, the record queue did also, but it was the combination of the two that presented difficulties. Every time the simulation was run it would run out of memory. A small C^{++} routine was written to show the amount of available memory and interfaced it into the simulation. The record queue was first tested to ascertain that when items were removed from the queue that the memory was being freed. The memory was being allocated and freed just as expected. The heap size declined as more items were created and added to the queue and increased as they were removed and disposed. The researcher then tested the timer module with similar results. The combined model was a simple traffic light having two queues (to represent the intersection), four complex records (to represent cars), and a timer (to make the lights change). A total of twenty-three car records and the other three objects were used. The simulation run with the memory checker resulted in more than half of the available memory consumed or tied up. As the simulation ran, the memory addresses were not being reused in the event tables as expected. They simply kept increasing until there were only 144 bytes left and the simulation halted. The computer used to execute this model has a total of four megabytes of memory, which seemed adequate. However, it appears the DOS extender, provided with MODSIM II^{TM}, is used only for compiling and has no effect on run-time modules. The entire simulation is loaded into the available base 640K of memory on the computer. This suspicion was confirmed by the manufacturer when they were called about running out of memory during compile time.

This simple traffic light model indicated that there would not be enough memory on a PC to run a PRNET simulation. The researcher believes that a simulation including at least ten Nodes, would be needed to produce results of any significance. A simulation of this size would require twenty or more record queues, twenty or more timers, and a large number of frame records. These would just represent the basic elements of the TNC module and does not include the remaining Node or Channel module components. Therefore, it was decided to stop development of the PRNET simulation on a personal computer.

4.3 Recommendations

The packet radio simulation that allows an application developer to analyze the performance impact of their work on a packet radio network is a valuable tool and should be pursued. Wireless networks are beginning to make their way into the local area network arena. As the technology improves, wireless wide area networks will become more popular. A packet radio network can provide the type of long distance connections necessary. There is great potential for today's advanced network applications to take advantage of this medium's ability to connect nodes large distances apart. Development of a PRNET simulation tool could be used in the development of more bandwidth efficient protocols and more advanced applications. Therefore, it is recommended that the development of a PRNET simulation be pursued on a hardware platform other than a personal computer. In the researcher's opinion, if MODSIM II^{TM} is available for use on a Sun-3, Sun-4, SPARC, VAX/VMS, or Unix system that it should be used because it provides a simple interface capability to C. If MODSIM II^{TM} is not available, the PRNET design proposed in this thesis could be used to develop a simulation in ADA or other languages, which provide an interface to C.

Once the simulation is working improvements could include elaboration of the Channel model. The matrices, Connectivity and P_b, currently used make the assumption of equal transmission power and equal reception capabilities. It would make the Channel more realistic if the Connectivity Matrix took into consideration transmitter power, propagation losses, and antenna gain to determine which nodes could hear each other. Also if the P_b Matrix reflected that a radio path between two nodes might not be two directional. This suggestion is the result of a recommendation by Phil Karn to use transmitter power control to increase frequency usage. Karn proposes that by using a channel access scheme that adjusts the power to a level necessary to just hear the next node you need to talk to, the throughput of a band-limited channel could be increased(13).

Another recommended modification of the simulation is to the Node modules. Each Node could keep an array similar to that of the Channel's Node Array, except this would be a Channel Array. By creating different channels, the simulation could take into account simultaneous transmission in different frequency bands. Furthermore, it could be used to simulate the ability of some Nodes to provide gateway services between different frequency channels.

An area of great interest is the security of a packet radio network. The tracing ability of the NOS package could also be used for spying on a packet radio network. An individual could turn tracing on and log all of the message traffic to disk for later analysis. Both NOS and AFITPAC provide the ability to transmit encrypted messages. However, the packet header information is transmitted in the clear, leaving the network subject to traffic flow analysis. This problem could be resolved if the entire transmitted packet was encrypted. The NOS package expects to see a TNC operating in Keep-It-Simple-Stupid(7) (KISS) mode, which allows the host's software to have control over the TNC functions at the lowest possible level(7). In this mode of operation the TNC "simply converts between synchronous HDLC, spoken on the full- or half-duplex radio channel, and a special asynchronous, full duplex frame format spoken on the host\TNC link.(7)" Therefore, it is recommended that an encryption software module or device be developed to place between the KISS TNC and computer to provide the security measures necessary.

Appendix A. Definition of Terms

A

AX.25 -	The link-layer packet-radio protocol based on the CCITT X.25 packet-switching protocol.(12) For complete details of the workings of the AX.25 protocol refer to , *AX.25 Amateur Packet-Radio Link-Layer Protocol, Version 2.0, October 1984*(9), which is available from the American Radio Relay League (ARRL).

C

Carrier Sense Multiple Access -	(a) a channel-access arbitration scheme in which packet-radio stations listen for the presence of a carrier on a channel before transmitting.(12) (b) a characteristic of network hardware that operates by allowing multiple stations to access to a transmission medium by listening to see if it is idle.(8) (c) when a station has data to send, it first listens to the channel to see if anyone else is transmitting. If the channel is busy, the station waits until it becomes idle. When the station detects an idle channel, it transmits a frame. If a collision occurs, the station waits a random amount of time and starts all over again.(19)
Channel -	the module used to represent a single radio frequency free space transmission medium.
Collision Matrix -	a matrix used to keep track of pending frame deliveries to a simulation Node. It is used to com-

	pensate for the instantaneous delivery of frames to nodes in a discrete event simulation.

D

Datagram protocol-	a Network-layer protocol that transfers each packet independently along the best available route; also called connectionless protocol.(12)
Delivery delay -	the time it take to pass the entire contents of a packet from transmitter to receiver. It includes transmission time, propagation delay, and reception time.
Digipeater -	digital repeater, a device that receives, temporarily stores and then transmits (repeats) packet-radio transmissions that are specifically addressed for routing through the digipeater.(12)

E

Ethernet Transceiver -	a device which clamped to a coaxial cable so that it makes contact with the inner core. The transceiver contains the electronics that handle the carrier detection and collision detection. When a collision is detected, the transceiver also puts a special invalid signal on the cable to insure that all other transceivers also realize that a collision has occurred.(19)

G

GILLAROO -	"cryptographic board encrypts data by using a cryptographic key (loaded from a paper tape) which converts the data into pairs of output characters; the first character is a capitol letter between A and P, and the second is a lower case letter between a to p. No

control characters other than pause transmission (control S), resume transmission (control Q), and end of file (control Z) are output from the device while in secure mode.(15)"

H

HDLC - (*High-Level Data Link Control*) a link level protocol standard by ISO. CCITT later adapted HDLC for its link access protocol (LAP) used with X.25 networks. HDLC is increasingly important to the Internet because Packet Switch Node (PSN) interfaces now use it to transfer frames between the host and PSN.(8)

Hidden Nodes - a packet radio station that can be heard by only one of two other stations that are connected; in such a situation, the two stations that cannot hear each other transmit simultaneously, which results in the reception of interference or a packet collision by the third station(12).

HOP - the equivalent of an edge on a network graph. The number of point-to-point transmissions used to deliver a message from source to destination.

K

KISS - an acronym for "Keep It Simple Stupid," a Link-layer nonprotocol for serial input and output that supports Serial Line Interface Protocol (SLIP), written by Mike Chepponis.(7, 12)

M

MODSIM II^{TM} - "the Modular Simulation language is a general-purpose, modular, block structured language which provides support for object-oriented programming and discrete event simulation. It is intended to be used for building large process-based discrete event simulation models through modular and object-oriented development techniques.(5)"

Module -	any independent unit which is part of a large system. Microcomputer and other systems may be made from several modules. a piece or segment of a whole; an incremental block(6).

N

Node -	(a) the name of a module in this thesis that represents the collection of a TNC, a computer, and a simulated user. (b) any terminal, station, or communications computer in a computer network(6). (c) a junction point within a network(12).

P

Packet -	(a) In communications, a short (1000-2000 bits) block of data prefixed with addressing and other information for control that is used to carry information through a packet-switching-network(6).(b)The unit of data sent across a packet switched network. The term is used loosely. While some Internet literature uses it to refer specifically to data sent across a physical network, other literature views the Internet as a packet switching network and describes IP datagrams as packets(8).
PING -	(*Packet InterNet Gropper*) The name of a program used in the Internet to test reachability of destinations by sending them an ICMP echo request and waiting for a reply. The term has

	survived the original program and is now used like a verb as in, "please ping host A to see if it is alive."(8)
PRNET -	Packet Radio Network, a computer network consisting of PRUs.
Probability of Bit Error Matrix -	a component of this project used to hold the probability of bit error between a pair of PRUs. Probability of bit error is the chance that a transmitted bit will be received incorrectly.
PRU -	Packet Radio Unit, each unit consists of a terminal or computer, a TNC, and a radio transceiver.

R

Routing Table -	a table used to track assignments of communications paths for message delivery(6)

T

Terminal Node Controller (TNC) -	an Amateur Radio packet assembler/disassembler; it may or may not include a modem(12)
Topology -	In network terminology, describes the physical or logical placement of nodes (stations) in a computer network system or configurations(6)
Transmit Queue -	a list used to keep track of the simulation frames waiting to be transmitted by the TNC.

U

Uuencode\Uudecode -	**encode/decode a binary file for transmission via mail. Output from uuencode is all ASCII printable text from a binary input file. Output from uudecode is a binary file from an uuencoded file.**

Bibliography

1. Bausbacher, Pete and David Young. *PC-NETSIM: A PC Based Network Simulation*. SRNTN N00140-87-C-8903, Richardson, Texas: Rockwell International Collins Defense Communications, July 1989 (AD-A213-203).
2. Bausbacher, Peter E. *The Link Performance Model of a Packet Radio Network Simulator*. SRTNTN N00140-87-C-8903, Richardson, Texas: Rockwell International Collins Defense Communications, March 1989 (AD-B132-447Y).
3. Belanger, Ron, et al. *MODSIM II^{TM} The Language for Object-Oriented Simulation: User's Manual*. CACI Products Company, La Jolla, California, January 1990. Revision 10.
4. Belanger, Ron, et al. *MODSIM II^{TM} The Language for Object-Oriented Simulation: Reference Manual*. CACI Products Company, La Jolla, California, June 1990. Revision 6.
5. Belanger, Ron and Alasdar Mullarney. *MODSIM II^{TM} The Language for Object-Oriented Simulation: Tutorial*. CACI Products Company, La Jolla, California. January 1990. Revision 8.
6. Bolander, Donald, et al., editors. *The New Lexicon Webster's Dictionary of the English Language* (Deluxe encyclopedia edition Edition). New York: Lexicon Publications, Inc., 1990.
7. Chepponis, Mike and Phil Karn. "The KISS TNC: A simple Host-to-TNC communications protocol." In *ARRL/CRRL Amateur Radio 6th Computer Networking Conference*, pages 38–43, Newington, CT: The American Radio Relay League, Inc., 1987.
8. Comer, Douglas E. *Internetworking with TCP/IP: Principles, Protocols, and Architecture* (First Edition). Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1988.
9. Fox, Terry L. *AX.25 Amateur Packet-Radio Link-Layer Protocol: Version 2.0, October 1984*. Newington, CT: American Radio Relay League, Inc., 1984.
10. Geier, Captain James T. *Automatic/Adaptive Routing Algorithm for the Air Force Logistics Command Packet Radio Network*. MS thesis, AFIT/GE/ENG/90S, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, July 1990.
11. Geier, Captain James T., et al. "Network Routing Techniques and Their Relevance to Packet Radio Networks." In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 105–117, Newington, CT: The American Radio Relay League, Inc., 1990.

12. Horzepa, Stan. *Your Gateway to Packet Radio* (Second Edition). Newington, Connecticut: American Radio Relay League, 1991.

13. Karn, Phil. "MACA - A New Channel Access Method for Packet Radio." In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 134–140, Newington, CT: The American Radio Relay League, Inc., 1990.

14. Lebano, Captain Tito Nicola. *A TCP/IP Gateway interconnecting AX.25 Packet Radio Networks to The Defense Data Network.* MS thesis, AFIT/GCS/ENG/88D-25, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988.

15. Marsh, Captain Steven L. *Integration of the Air Force Logistics Command Packet Radio Network..* MS thesis, AFIT/GE/ENG/89D-29, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

16. Spiegel, Mitchell G. "Modeling Computer Networks," *IEEE Network: The Magazine of Computer Communications*, *2*(4):5 (July 1988).

17. Stallings, William. *Department of Defense Protocol Standards*, Volume 3 of *Handbook of Computer-Communications Standards.* Indianapolis, Indiana: Howard W. Sams & Company, 1988.

18. Stallings, William. *The Open Systems Interconnection [OSI] Model and OSI-Related Standards*, Volume 1 of *Handbook of Computer-Communications Standards.* Indianapolis, Indiana: Howard W. Sams & Company, 1989.

19. Tanenbaum, Andrew S. *Computer Networks* (Second Edition). Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1988.

20. Taris, Captain William J. *Design and Development of a Computer-Based Message Transfer System for The Air Force Logistics Command Packet Radio Network.* MS thesis, AFIT/GE/ENG/88D-45, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988.

21. Wade, Ian. "The Definitive Primer for the KA9Q Network Operating System." A published book by the same author and title is due out during the fourth quarter of 1991, September 1991.

22. Zeigler, Bernard P. "Hierarchical, modular discrete event modelling in an object-oriented environment," *SIMULATION*, *49*(5):219–230 (November 1987).