A TCP/IP GATEWAY INTERCONNECTING AX.25 PACKET RADIO NETWORKS TO THE DEFENSE DATA NETWORK

THESIS

Tito Nicola Lebano
Captain, USA

AFIT/GCS/ENG/88D-25

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY
Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/88D-25

A TCP/IP GATEWAY INTERCONNECTING AX.25 PACKET RADIO NETWORKS TO THE DEFENSE DATA NETWORK

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 Computer Systems

Tito Nicola Lebano, B.A.
Captain, USA

December, 1988

Preface

I would like to gratefully acknowledge the support and guidance provided by my thesis advisor, LTC Garcia. His patience, understanding and concern made this thesis effort possible. I would also like to thank my family; my wife Laura and my son Nicholas; for making the last eighteen months bearable especially those times that I communicated more with my computer than with them. Their love and support was crucial in keeping the last eighteen months in perspective. Finally, I would like to thank my Lord and Saviour, Jesus Christ, who saw fit to allow me to complete my studies (Proverbs 3:5-6), has blessed me with a wonderful family (Proverbs 18:22) and has given me hope for the future (Romans 6:23, I Thessalonians 4:16-17).

Tito Nicola Lebano

Table of Contents

List of Figures

Abstract

The packet radio, in recent years has been gaining popularity as a transmission medium for digital traffic. It provides its user with the capability of accessing computer resources while in a mobile configuration or in a remote location where access to cable based computer networks are either unreliable or non existant. As with any communications network, communications protocols must be in place to manage the transmission of data. Currently, the Transmission Contol Protocol (TCP) and Internet Protocol (IP) are used in the cable-based network, Department of Defense Network (DDN). With the capability that the packet radio provides, this effort's objective was to implement the TCP/IP protocols within a packet radio network and build a gateway through the Air Force Institute of Technology Network (AFITNET) to the DDN. Research for the project revealed that the software package, The KA9Q Internet Software Package, had already implemented TCP/IP and associated protocols such as Telnet and FTP for the packet radio. Unfortunately, when using the Telnet protocol within this package to communicate with a mainframe computer, passwords are transmitted in the clear. Any station monitoring the frequency can acquire a user's password to a particular mainframe. In order to protect the integrity of a user's computer resources, a method of a passwordless login is necessary. This effort undertook implementing the Data Encryption Standard (DES) within the KA9Q software, such that, encrypted strings are used to validate users. The DES process was implemented as a telnet option that is negotiated when the remote station requests it prior to telneting to the desired host. The source code for Telnet on the mainframe is proprietary so all modifications where made to the KA9Q package. The gateway consisted of an IBM AT connected to the AFITNET via an ethernet connection.

A TCP/IP GATEWAY INTERCONNECTING AX.25 PACKET RADIO NETWORKS TO THE DEFENSE DATA NETWORK

I. Introduction

1.1 Background Information

Computer networks, until 1970, were wire-based and imposed limitations in the design of computer networks. Mobile users and users located at remote sites, inaccessible by cables could not be interconnected into the network. In 1970, Norman Abramson and his colleagues at the University of Hawaii recognized the need to interconnect their remote campuses on the outlying islands to the main computer. The installation of cables from island to island was not economically feasible, consequently, another transmission medium was required. The packet radio was selected as the new mode for digital data transmission. The system was called the Aloha System [1:25]. The Defense Advanced Research Projects Agency (DARPA) soon followed with its own research and development effort of a multi-hop packet radio network called PRNET [1:26]. In recent years, the packet radio has also become widely used by the amateur radio community as its digital data transmission medium [2:11].

The packet radio has experienced vast growth in recent years, both in the military, as well as, civilian sectors. In the civilian community alone, packet radio stations have increased 10,000 percent over the past five years [2:5].

With the increased popularity of the packet radio, it is reasonable to desire to utilize this transmission medium to provide mobile and remotely located users a

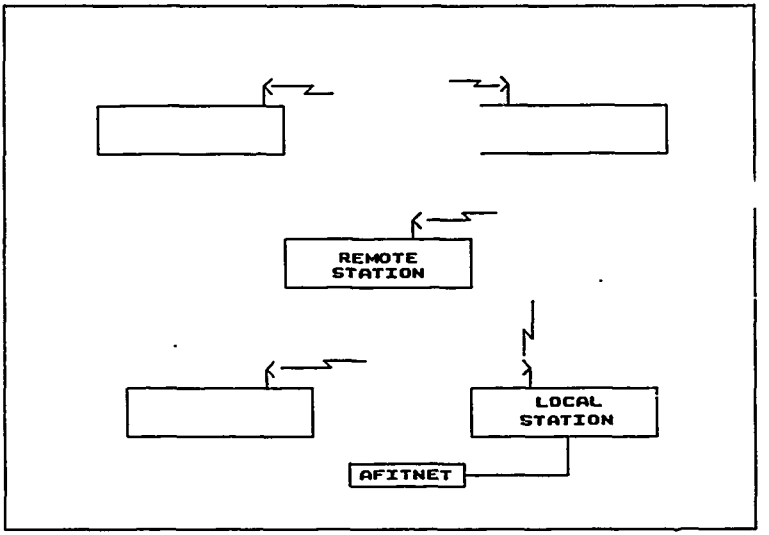


Figure 1.1. A Packet Radio Network Interconnected with AFITNET

means of interconnecting to largely, cable-based networks, such as the Defense Data Network (DDN). This application can be extended beyond the civilian community to that of the military. Mobile command and control posts, airmobile units, and rapid deployment forces, to mention a few, will require access to cable-based networks in the rear battle area to provide for command and intelligence information. The purpose of this thesis was to design and implement a gateway so that packet radio networks consisting of micro-computers (personal computers), could access the DDN through the AFITNET. Figure 1.1 provides a visual representation of such a network.

1.1.1 The Packet Radio. The packet radio's popularity can be attributed to its speed of transmission, networking capability, error-free transmissions, and efficient use of the frequency space. By utilizing a standard digital communications networking technique known as Carrier-Sense Multiple Access with Collision the frequency is busy but will wait until it is clear and then transmit a short burst

of information. These bursts are short and sporadic so, many packet stations can operate on the same frequency without interference. If transmissions collide, each station will wait a random amount of time and then retransmit the packet. This process will continue until all data has been successfully transmitted.

The typical configuration of a packet radio station includes a digital terminal equipment (computer), transceiver (radio), and Terminal Node Controller (TNC), shown in Figure 1.2. The TNC provides the interface between the transceiver and the computer and uses the AX.25 protocol which is a modified version of the Open Systems Internet (OSI) protocol X.25 [2:12]. The AX.25 protocol defines the format of the information sent in a packet, instructs the TNC what steps are to be taken under different circumstances, and defines networking procedures.

The goal for any computer network is to facilitate the exchange of data and sharing of computer resources. The extent to which a packet radio network can achieve this goal is dependent on the software running on the computer connected to the TNC. These programs are referred to as terminal programs and range from a simple communications software package, which provides only real time communications (no file transfer capability), to packages which perform services such as file transfer and remote login. One such package is the KA9Q Internet Software Package written by Phil Karn [3].

The KA9Q package was selected because it is specifically designed for the amateur packet radio community to provide networking services such as file transfer and remote login, typically found in cable-based, wide and local area networks (LAN). This package implements the internet protocols, Transmission Control Protocol (TCP) [4] and Internet Protocol (IP) [5]. The remote login protocol, Telnet [6], and the File Transfer Protocol, FTP [7], which are dependent on a TCP/IP connection, are also implemented. The TCP/IP family of protocols is found in many commercial LANs today [8:111]. In addition, the implementation of IP provides the ability to transmit data to networks other than the packet radio network such as

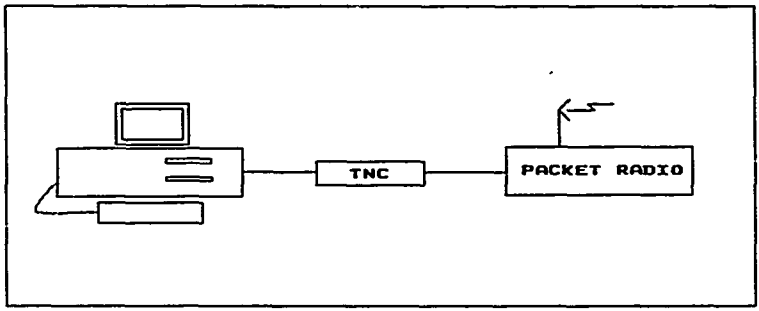


Figure 1.2. Components of a Packet Radio Station

the Defense Data Network, which has also implemented TCP/IP. In addition, Phil Karn, has made the source code for his package, available to those who wish to use it or modify it to meet user specific needs.

1.2 Nature of Problem and Problem Statement

The use of packet radios as a transmission medium is governed by the FCC. The license class used to operate a packet radio prohibits the use of data encryption except as a verification method. The KA9Q software currently does not implement any encrypted verification method; consequently, when the Telnet protocol is used to remotely log into a mainframe (time-sharing computer) such as one in the AFITNET, the password that is entered is transmitted in the clear. In other words, the password will be displayed to the screen of any station monitoring the frequency. If the user is to ensure the integrity of his or her computer resources the password cannot be allowed to be transmitted in the clear.

Phil Karn, the author of the KA9Q software, has implemented in a separate package, a commonly used encryption algorithm called the Data Encryption Standard (DES) [9]. This algorithm uses a binary number, called a key, to encrypt and decrypt data. This package is currently separate from the KA9Q software and includes the DES algorithms, as well as, driver programs which enable the algorithms to be used. To use the DES program in conjunction with the KA9Q software, both must be installed under DoubleDos or DesqView which are multitasking utility programs for MS-DOS. In addition, a program called Radlogin must be present on the mainframe. When a remote station attempts to telnet into the mainframe, it recognizes the login as one from a radio and calls up Radlogin. This program then asks for the user's login name and then sends the remote user a plain text string to encrypt. The remote user must then temporarily leave the KA9Q program, enter the DES program, manually enter the string to encrypt, encrypt the string, re-enter the KA9Q program and manually enter the encrypted string. This process is time consuming, cumbersome, and introduces the possibility of human typographical errors.

1.2.1 Problem Statement. The problem then is to integrate the DES software and the KA9Q software into one package so that the encryption and validation process and login process to the mainframe are invisible to the user. There will no longer be a need for additional programs on the mainframe, as well as, the requirement to run DesqView or DoubleDos on the remote packet station. Ideally, the remote user should only have to enter *telnet requested machine* and the next item that appears on the monitor is a prompt from the requested machine.

1.3 Scope

There are several issues that can be investigated when dealing with the security of computer networks. This thesis secure means for a remote user, in a packet radio network, to log into a computer on the AFITNET which will then allow for connection to the DDN. This will be accomplished as a passwordless login for the

user. All user verification processes will be completed invisible to the user by way of the DES algorithms. This effort will involve the integration and modification of the KA9Q and DES source code.

1.4 Assumptions

The only assumption is that the TCP/IP and Telnet protocols, as well as the DES algorithms implemented by Phil Karn, are correct and in compliance with the specification documents.

1.5 Summary of Current Knowledge

The summary of current knowledge can be addressed by answering three questions:

1. Why use the TCP/IP family of protocols to be implemented in a packet radio network?
2. What capability does Telnet provide us?
3. Why select the Data Encryption Standard as a means to provide secure login?

1.5.1 Transmission Control Protocol/Internet Protocol. The TCP/IP family of protocols was first introduced in the mid 1970s and were mandated in 1978 as DOD standards. TCP/IP provides the capability to interconnect heterogeneous computers in a network. In 1982, TCP/IP became associated with two industry accepted systems, the Unix 4.2 BSD operating system and the Ethernet, and became accepted by the private sector. In the early 1980s, the International Standards Organization (ISO) introduced the Open Systems Interconnection seven layer model; however, the X.25 protocol, which defines the first three layers, is the only available protocol under this model. Consequently, manufacturers are implementing TCP/IP to meet their networking needs until ISO standard products are widely available [8:111].

1.5.2 Telnet. The Telnet protocol is dependent on a TCP connection. Its primary purpose is to provide a standard method of interfacing terminal devices to each other as in a remote 'Network Virtual Terminal' (NVT); the principle of negotiated options; and a symmetric view of terminals and processes [6:1]. The NVT is a conceptual device which provides a standard for all the terminals in the network. Each terminal handles its own mapping of local device characteristics and assumes that the other terminals do the same thing. The principle of negotiated options extends how a terminal is defined during a telnet session. All terminals will operate at the default parameters established by the NVT unless option negotiation is requested by either the remote or local station. Such options include the echoing of characters. In addition, user defined options may be implemented to support the requirements of a particular network. Telnet options are negotiated immediately after the TCP connection is established and before the actual telnet session begins. The capability to define user specific options provides the mechanism with which to effectively integrate DES into the KA9Q package.

Telnet is implemented as a client half and a server half and when fully implemented can negotiate options. The client is activated when a remote station attempts to establish a telnet session with a host computer (server or local station). In the KA9Q package, only the Telnet client is fully implemented. The server is not, because MS-DOS is not a timesharing system; only one user can have functional access to the machine's resources. Consequently, when telneting from PC to PC, once a TCP connection has been established under a Telnet request, both stations go into a keyboard to keyboard mode where the operators can pass text traffic back and forth. When a user of this package telnets to a mainframe running UNIX, it relies on the mainframe telnet server.

1.5.3 The Data Encryption Standard. The problem of providing a secure means of logging into another computer system can be addressed through the implementation of data encryption. Data encryption is a process used to hide the real

meaning of data and can be achieved through the use of an algorithm that transforms data from it intelligible form to a cipher. An encryption algorithm must meet the following requirements to be accepted as a federal standard:

- It must provide a high level of security.
- It must be completely specified and easy to understand.
- The security provided by the algorithm must not be based upon the secrecy of the algorithm.
- It must be available to all users and suppliers.
- It must be economical to implement in electronic devices and be efficient to use.
- It must be amenable to validation.
- It must be exportable [10:7].

Several encryption algorithms are in existence, however, in 1977, the National Board of Standards (NBS) adopted the Data Encryption Standard as the data encryption algorithm to be used for all unclassified U.S. government applications [9]. In addition to its use within the federal government, it has been approved as a standard by the American National Standards Institute and recommended for use by the American Bankers Association. The selection of the DES provides the packet radio community with an accessible, inexpensive, easily understood, secure means of encryption. A complete discussion of the algorithm and how it will be implemented for this application can be found in chapter 2 of this document.

1.6 Standards

The implementation of secure login with DES will comply with the standards for the Data Encryption Standard set forth by the National Bureau of Standards

(NBS) [10]. Any modification to the Telnet code within the KA9Q package will comply with the Telnet specifications [6].

1.7 Approach

The approach taken to solving the problem of secure login can be broken down into the following three phases:

Phase 1: Installation, Verification, and Analysis of the KA9Q Software Package and the Telnet Protocol.

The KA9Q software package had to be installed on the resources available and performance tests had to be conducted to insure that a remote station could gain access to the AFITNET and the DDN using the Telnet protocol as currently implemented. Then, an analysis of the source code for KA9Q was done to gain a full understanding of the process taking place when a Telnet session to the AFITNET is invoked by a remote user. A study of the protocol and the options specification was completed to establish whether designing and implementing a DES option would be successful in integrating the DES code into the KA9Q package. This study in the AFITNET is proprietary, so any modifications to the software had to be made in the KA9Q implementation of the protocol.

Phase 2: Design of the DES Telnet Option

In this phase, the DES telnet option was designed and implemented. This included identifying the specific modules in the KA9Q software package to be modified, as well as, designing the DES telnet option software to accommodate the integration of the DES software. The modifications were based on detecting incoming data, not echoing the incoming data to the screen, and interpreting the nature of the data so that essential data may be passed to the appropriate module for encryption or decryption.

Phase 3: Design and Implementation of Software Necessary to Conduct a Packet Radio Station to Mainframe Login via an Ethernet.

In this phase, once a remote user had been validated by the local packet station (server), the KA9Q package had to be modified so that the server would perform the login procedure to the specified computer on the AFITNET. Once the connection to the AFITNET had been established, both connections at the local station had to be united to generate one connection.

1.8 Material and Equipment

There are several pieces of hardware, as well as, two software packages that were necessary for the completion of this thesis effort. The installation of both the hardware and software is in Appendix A.

1.8.1 The Hardware Configuration. The hardware consisted of the following equipment:

1. One Zenith Z-248 computer with monitor.
2. Two IBM AT computers with monitors.
3. Two MFJ-1270 Terminal Node Controllers (TNC).
4. Two power supplies
5. An Ethernet connection
6. A 3COM501 ethernet controller card
7. Two 27256 EPROMS with the KISS code (Installed in the TNCs, see KA9Q User's manual)

All the equipment was on hand and located in the Communication - Electronics Laboratory, Rm 225, Bldg 640, Air Force Institute of Technology.

The equipment was broken down into the following configurations:

Local Station or Server Host

This packet radio station serves as the local station connected to the AFITNET via the Ethernet. It consists of an IBM AT, TNC, radio, 3COM501 card, and power supply. The ethernet card can accommodate either a transceiver cable or an RF cable. The card is currently set for a transceiver cable. If an RF cable is to be used, the jumper on the card needs to be moved.

Remote Station or User Host

This packet radio station serves as the remote station. It consists of an IBM AT, TNC, radio, and power supply.

The Zenith Z-248 was initially used in the development process. It was needed because the source code, when compiling under AZTEC C on a Zenith Z-248, would compile satisfactorily but would not link. This problem was alleviated when the IBM AT was used.

1.8.2 Software. The software required to complete this thesis effort consisted of the KA9Q Internet Software Package and a package which contained the Data Encryption Standard algorithms in software, both of which were written by Phil Karn. In addition, the AZTEC C Compiler, version 4.10c by MANX Software was used for all software development. This software AZTEC C compiler may be obtained from CPT Dean Campbell, located in room 238. A complete list of all the files and how to compile all the files to generate an executable file is in Appendix A.

1.9 Sequence of Presentation

In chapter two, a more in-depth examination of the KA9Q package is presented, concentrating on how a telnet session is established between to PCs. A discussion of telnet option negotiation and the Data Encryption Standard is also presented.

The design of the modifications to the KA9Q source code that were required to integrate DES is presented in chapter three.

Chapter four discusses how to use the DES enhanced KA9Q package. Also included is a discussion of the testing that was performed and the results of this testing.

The conclusions of this project and the recommendations for future research are presented in chapter five.

II. Description of the KA9Q Internet Software Package and the Data Encryption Standard

2.1 Introduction

The KA9Q Internet Software Package written by Phil Karn [3] provides the amateur packet radio community with a network software package that provides similar networking services found in many cable-based networks. These services include file transfer and remote login. The KA9Q package was designed to follow the Open Systems Interconnection seven layer architecture. There are various protocols implemented at the different layers. The ones of interest for this effort are:

2.1.1 Physical/Link Layers. At this layer several protocols have been been implemented. The two of interest are the AX.25 protocol which is a modified version of the Open Systems Internet protocol, X.25 and the Ethernet. Several drivers for various ethernet cards are included. This effort uses the 3-Com 3C500 ethernet card and driver.

2.1.2 Network Layer. The network layer or Internet layer implements the Internet Protocol (IP) [5]. This implementation supports IP datagram fragmentation and reassembly and IP options to include: Loose Source Routing, Strict Source Routing, and Record Route. In addition, a manually maintained routing facility which contains the list of host-specific destinations and a default entry has been implemented. This table is generated manually by the user.

2.1.3 Transport Layer. The transport layer or host-host layer is implemented with the Transmission Control Protocol (TCP) [4]. This protocol provides a reliable, sequenced *virtual circuit* or *connection-oriented* service atop IP on an end-to-end basis. The use of port numbers allows for multiple connections within a single TCP

module. This ability provides the functionality of the Session layer so this layer has not been defined in this package.

2.1.4 Application Layer. The Application layer is defined by three protocols: File Transfer Protocol (FTP) which provides the ability to transfer binary and/or text file between computers [7], the Remote Login Protocol (TELNET) which allows a user to log into a remote computer and negotiate options such as remote versus local echoing, and the Simple Mail Transfer Protocol (SMTP) which provides a means of transferring electronic mail between computers. All three protocols rely on a successful TCP connection. A more complete description of these protocols can be found in the KA9Q Users Manual [3] and the protocol's specification document.

2.2 Mechanics of the KA9Q Package

The mechanics of how the software functions, data structures, and how the code is structured is well defined in the users manual and the intention is not to redefine what already exits. It is, however, important to briefly examine some of the constructs involved in establishing a TELNET connection which is dependent on a successful TCP connection.

The overall concept which the program relies on is that of the finite state machine. Any on going process, whether it be establishing a TCP connection or negotiating telnet options, is in a given state at a given time. For example, when establishing a TCP connection, the SYN flag in the TCP header of the first packet to be transmitted from the remote station is set. The state of the connection at the remote station is then set to SYN_SENT and will cause a specific section of code to be executed when the acknowledgment is returned.

The KA9Q package, being written is the C language, relies heavily on the data structure *struct. Struct* is a data structure where data elements of different types are lumped together. It is very similar to the record structure found in Pascal or Ada.

The struct concept is used extensively within the program. Ones of particular interest are those used to define the TCP and IP headers, the TCB (Transmission Control Block) which records the state of a connection, the session control structure which records the session type (telnet, ftp...), a telnet protocol control block structure, and mbuf which defines the memory buffer used for incoming and outgoing data and for holding the actual packets. The definitions for these structures can be found in Appendix B.

Finally, the KA9Q program uses pointers extensively to make function calls. This is done by using the C Language syntax to define a variable name as a pointer to a function. This is done by assigning the variable name, the name of a user specified function whenever a particular function is needed. Then, whenever a function pointer variable is encountered and is non nuil, the function that is being pointed to will be called. A good example of this is seen at the beginning of the function Open_TCP shown below:

```
open_tcp(lsocket,fsocket,mode,window,r_upcall,
        t_upcall,s_upcall,tos,user)

struct socket *lsocket; /* Local socket */
struct socket *fsocket; /* Remote socket */
int mode;               /* Active/passive/server */
int16 window;           /* Receive window
                           (and send buffer) sizes */
void (*r_upcall)();     /* Function to call when
                           data arrives */
void (*t_upcall)();     /* Function to call when ok
                        to send more data */
void (*s_upcall)();     /* Function to call when
                           connection state changes */
```

Here the variables r_upcall, t_upcall, and s_upcall are defined as pointers to three different functions. When the function 'Open_TCP' is called, the actual function names will appear in the function call (Rcv_Char,Tn_Tx,T_State). Then, within

Open_TCP, the function that was assigned to one of these pointers will be called when the pointer is encountered.

```
tcb = open_tcp(&lsocket,&fsocket,TCP_ACTIVE,0,
        rcv_char,tn_tx,t_state,0,(char *)tn);
```

These concepts described above are used throughout the KA9Q package and are used in the implementation of TCP, IP, and TELNET. With this basis of knowledge, a closer examination of how these protocols are implemented within the KA9Q package will provide a clearer understanding of the interfaces between the different network layers and establish a foundation of understanding from which the modifications of the code were based.

2.3 Establishing a Telnet Connection from PC to PC Using TCP/IP Over an Ethernet.

The following discussion outlines how a Telnet connection using TCP/IP over an Ethernet is established.

2.3.1 Tracking a Packet in the Remote Station. The initiation of a telnet session commences when the remote user enters the command 'telnet *desired machine*'. The main program monitors the keyboard, the input port, and the backlog queue for input. When it detects input from the keyboard, it reads it, and calls the commandparse function which evaluates the command and calls the appropriate function. In this case, it calls the function *Dotelnet.* Figure 2.1 should be used as a reference.

Dotelnet() Function.

The purpose of this function is to determine if the destination address is in the routing table. If it is, a session structure is allocated, a Telnet protocol structure is

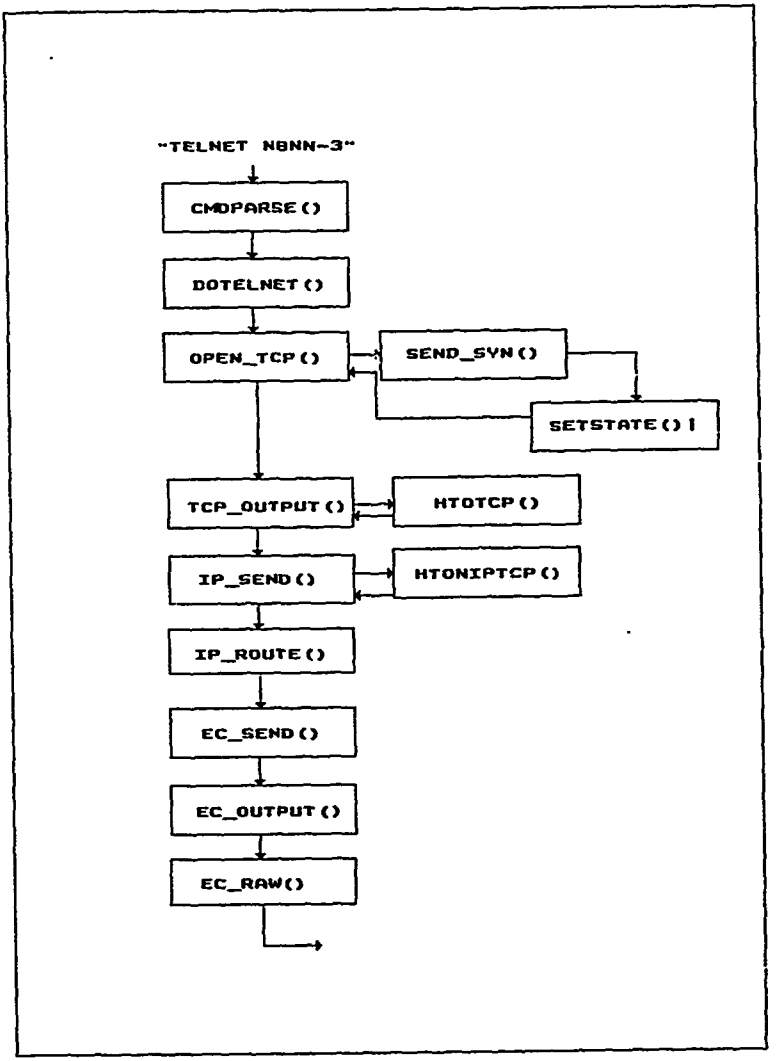


Figure 2.1. Outgoing Telnet Connection Request

created and initialized, and the function Open_TCP is called to begin establishing the TCP/IP connection. This function is passed the local and remote addresses and ports, the TCP state, window size, the name of three functions, time of service and a structure called *user*.

Open_TCP() Function.

The function Open_TCP sets the local and remote addresses and ports, initializes a TCB connection control block and the various fields within the TCB. These fields include pointers to functions that are to be called during a telnet session when data has been received, when file uploading is done, or when the telnet state changes.

A TCP connection can be in one of three states: TCP_Passive, TCP_Active, and TCP_Server. When establishing a TCP connection the remote station's state is set to TCP_Active. This causes the Send_Sync(), Setstate(), and TCP_Output() functions to be called.

Send_Sync() Function.

This function generates the initial TCP sequence number and puts a 'SYN' on the output queue. In other words, the SYN flag in the TCP header is set.

Setstate() Function.

This function sets the TCB state initially to 'SYN_Sent' and calls the specified state change upcall function. The upcall routine associated with the telnet client will print to the screen the current state of the connection. The upcall routine associated with the telnet server waits until the connection has been established before putting a message to the screen.

TCP_Output() Function.

This function checks for various conditions and computes a pseudo IP header (this is done in order to compute information needed when IP functions are called). If everything is correct, values are assigned to fields that will be used to build the TCP header. This includes the control flags of which the 'SYN' flag will be set. These values are passed to a function called 'HTONTCP' which generates the TCP header, computes the checksum, links in the data, if any, and formats the header to that of the mbuf structure. The function IP_Send is then called and passed the source and destination addresses, the type protocol which is calling the IP function (in this case TCP), time of service, time to live, the TCP header and data which is stored in a mbuf structure, the length of the TCP header and data, and some optional IP information.

IP_Send() Function.

This function fills in the IP header and calls 'HTONIPTCP' which generates the IP header, computes the checksum, links in the data (in this case the data field consists of the TCP header and data), and formats the header to that of the mbuf structure. The function IP_Route is then called and passed the IP header and data, again in the mbuf structure.

IP_Route() Function (send).

The IP_Route function initially checks to see if any of the following conditions exist:

1. The packet is shorter that a legal IP header.
2. The IP header length field is too small.
3. The IP header checksum is bad.
4. Wrong version of IP.

If any one of these conditions exists, the connection process is halted, the allocated memory containing the IP header and data is de-allocated, and control is returned to the main program with the appropriate error message displayed. If these conditions don't exist, the length of the data field is trimmed, if needed (fragmentation), and the length of a secondary IP header is computed in case of fragmentation. Any IP options are also handled at this point.

Finally, the destination address is compared against the station's address. If they are different, the packet is outgoing and the specified interface function is called (A pointer to this function is initialized and set when the KA9Q package is installed and the required device drivers is initialized). For this example, the function for the 3COM 3C501 card will be called: EC_Send(). This function along with the functions EC_Output() and EC_Raw build an ethernet packet containing the IP packet and put the total packet out on the ethernet.

2.3.2 Tracking a Packet in the Local Station. This station, being in a listening state, detects an incoming packet and calls the function EC_Recv which is part of the ethernet card driver. Figure 2.2 should be used as a reference.

EC_Recv() Function.

This function strips off the ethernet header and sends the packet to the next layer which is the network layer and function IP_Route.

IP_Route() Function (receive).

IP_Route performs the same as in the send mode until it compares the destination address with its own address. It will recognize that the packet is for itself and pass the packet to another IP level function, IP_Recv.

IP_Recv Function.

This function assigns the appropriate upcall routine depending on what value is in the protocol field of the IP header. In our case, TCP is indicated so a pointer to the TCP function, TCP_Input, is initialized. Following this, the packet is checked to see if it is complete or part of a fragmented packet and then the specified upcall function is called.

TCP_Input() Function.

This function performs many functions the most important of which is the processing of a valid packet. If this is an initial request for connection, then no TCB has been established. If this is the case, a TCB is initialized and the data fields are assigned values. If the TCB exists for this connection, this part would not be negotiated. The control flags within the TCP header are then checked. The SYN flag has been set so the functions, Proc_Syn(), Send_Syn(), are called to process the received 'SYN' and Setstate() is called to set the new state which is now 'SYN_RECEIVED.' The function TCP_Output is then called and the sequence of events as outlined earlier will occur to get this packet out. The one difference is that in TCP_Out, the ACK control flag will be set indicating that the SYN from the remote station has been acknowledged.

When the packet returns to the remote station, it will be passed to TCP_Input which will check the control flags, see that the ACK flag has been set and recognize that it's SYN has been acknowledged. The TCB state is set to "Established" and an ACK to the ACK is sent back. In addition, the remote station enters the keyboard to keyboard telnet mode. The local station receives the acknowledged ACK, sets its state to established, and also enters the keyboard to keyboard telnet mode. Only in this case the operator would have to select this particular session in order to communicate with the remote user.

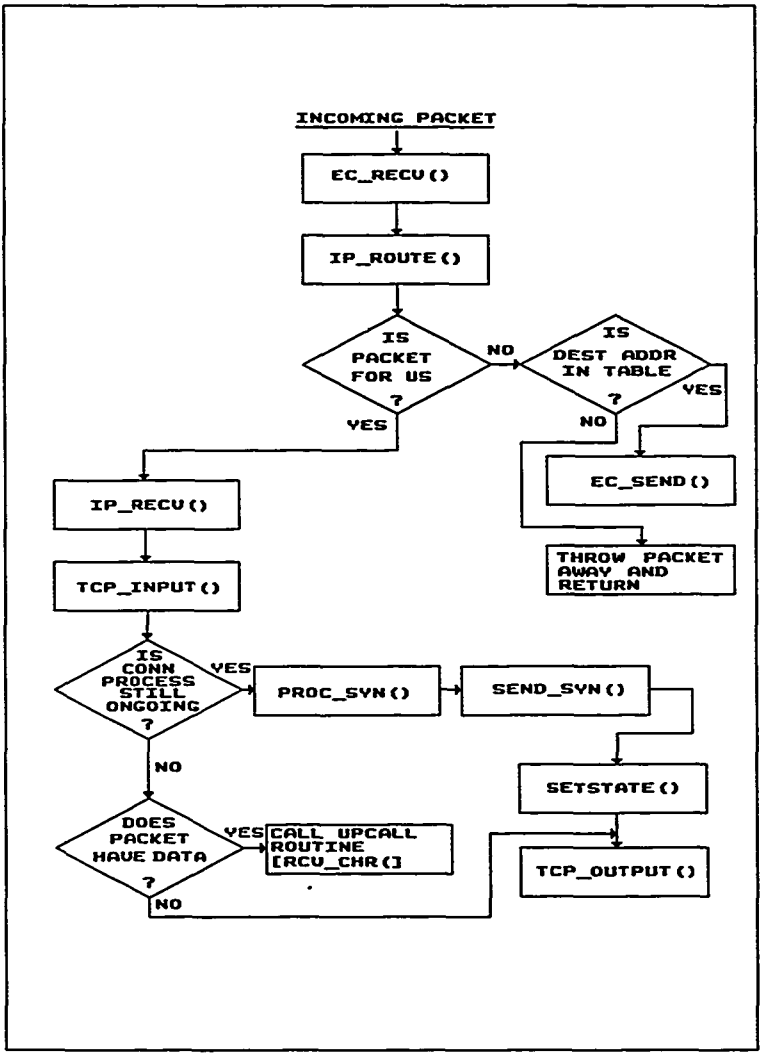


Figure 2.2. Incoming Telnet Connection Request

2.4 Telnet Options Negotiation

Telnet options can only be negotiated if both the client and server are fully implemented. As mentioned earlier, this is not the case in the KA9Q package; only the client is fully implemented and can accommodate options negotiation. When this package is used to telnet to a timesharing system such as UNIX, the options section of the client can be activated if option negotiation is requested by either the remote or local station.

The option negotiation process is a handshaking process consisting of passing specific well defined telnet commands between two processes. A command consists of a three byte sequence: the 'Interpret as Command' (IAC) escape character, the code for the command, and the code for the option itself. Listed below are the commands of interest, their codes and meaning, and example of a telnet option command line. [6:14].

COMMAND	CODE	MEANING
Will	251	Indicates the desire to begin performing, or confirmation that you are now performing, the indicated option.
Won't	252	Indicates the refusal to perform, or continue performing, the indicated option.
Do	253	Indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the indicated option.
Don't	254	Indicates the demand that the other party stop performing, or confirmation that you are no longer expecting the other party to perform, the indicated option.
IAC	255	Data byte 255.

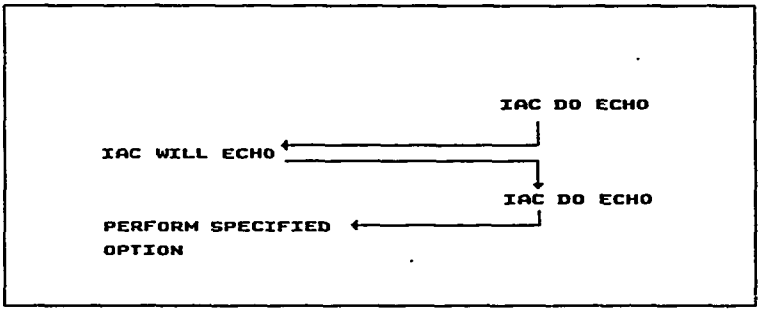


Figure 2.3. Telnet Option Negotiation Process

Using the telnet option 'ECHO' for the example, a command line would look like this (ECHO has a value of 1):

IAC DO ECHO

A telnet option request may be initiated by either the client or the server. A typical telnet negotiation session is shown in Figure 2.3 where the echo option is being negotiated.

2.5 Understanding The Data Encryption Standard

2.5.1 Encryption Concepts and Terminology. Prior to presenting the DES algorithm, it is important to understand some basic data encryption concepts and terminology. Encryption is an algorithmic computation which transforms data from an understandable form to that of a non understandable cipher form [10]. The cipher which is generated is normally the same length as the data which was enciphered. Ciphers generally operate on data elements of fixed length which consist of syntactic entities such as letters or groups of letters. In computer applications, bits or bytes are used in data encryption.

The transformation of data comprises of two processes: Permutation and Substitution. Given a specified bit stream, permutation is a process which re-arranges the positions of the individual bits. Conversely, substitution replaces each bit with another bit which may or may not be the same value. When a group of bits is transformed into a group of cipher bits it is called a block cipher. Additionally, when permutation and substitution produce a complex transformation it is called a product cipher. When it is applied to a block of bits it is called a block product cipher.

The final concept needed for complete understanding of the DES is that of the recirculating block product cipher. The recirculating block product cipher is generated when a block product cipher is constructed by using a permutation operation and a substitution operation alternately and recirculating the output of one pair of operations back as input for a specified number of times. This, as well as, the other concepts mentioned can be seen in the encryption computation diagram for the DES algorithm in Figure 2.4.

2.5.2 DES Specification and Practical Example. The DES algorithm consists of a recirculating, 64-bit, block product cipher whose security is dependent a 64-bit secret key. A key consists of 56 information bits which are used for the enciphering and deciphering operations and 8 parity bits which are used for error detection. The algorithm includes an initial permutation of the 64 data bits, 16 iterations of computation with the function f(Rn,Kn+1), and a final permutation.

The encryption computation process consists of the following steps. An example that works through the initial iteration of the algorithm is included.

1. Given a 64-bit data input, perform an Initial Permutation (IP). The IP rearrangement sequence is given in Appendix C.

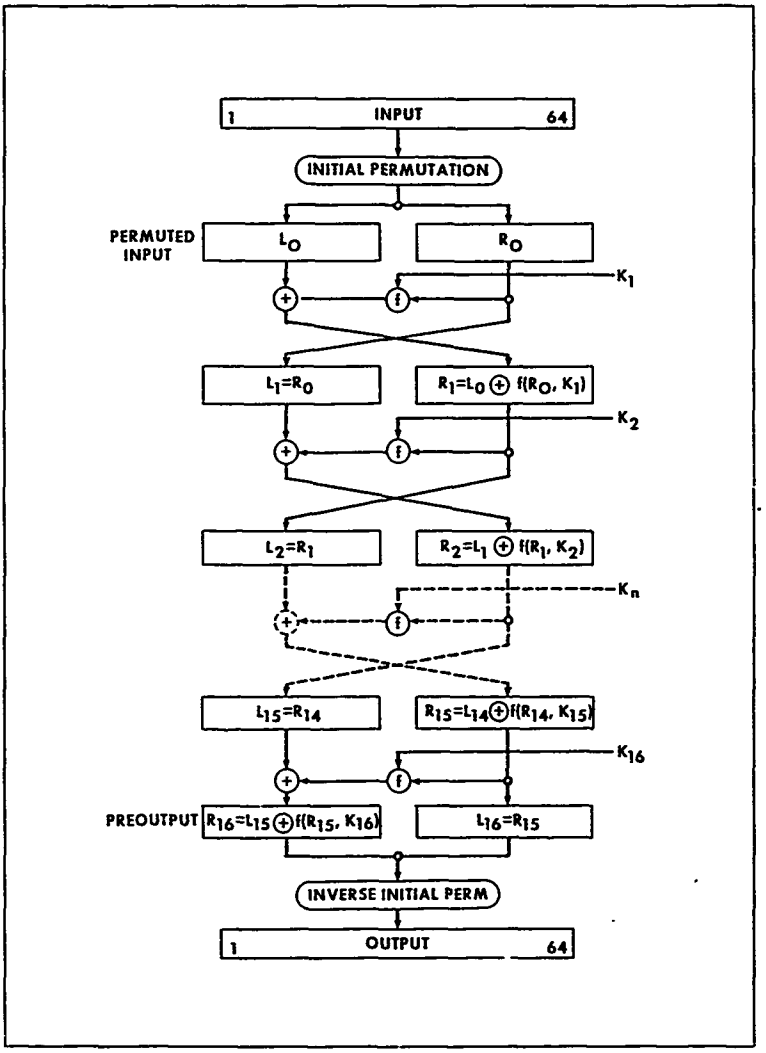


Figure 2.4. Enciphering Computation [5]

Initial 64 bit data stream (51 45 4b 58 2d df 44 0a) is

```
0 1 0 1 0 0 0 1
0 1 0 0 0 1 0 1
0 1 0 0 1 0 1 1
0 1 0 1 1 0 0 0
0 0 1 0 1 1 0 1
1 1 0 1 1 1 1 1
0 1 0 0 0 1 0 0
0 0 0 0 1 0 1 0
```

When the IP table in is applied, the resulting 64-bit string is:

Initial Permutation

```
0 1 1 0 1 1 1 1
0 0 1 0 1 0 0 1
0 1 1 1 0 0 1 0
0 0 1 1 0 1 1 1
0 0 1 0 1 0 0 1
0 1 1 1 0 0 1 0
0 0 1 1 0 1 1 1
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
1 0 1 1 1 1 0 0
1 0 1 0 0 1 0 0
```

2. Divide the 64-bit block of data input into 2-32 bit blocks, L0 and R0. L1 equals R0. R0 will be used as input to a non linear function, f(Rn,Kn+1), and the result will be exclusive OR'ed with L0 to form R1. The secret key is used in the computation of the function to manipulate the data bits.

```
     0 1 1 0 1 1 1 1
L0 = 0 0 1 0 1 0 0 1
     0 1 1 1 0 0 1 0
     0 0 1 1 0 1 1 1
```

```
     0 0 1 0 0 0 0 0
R0 = 0 0 0 1 0 0 0 0
     1 0 1 1 1 1 0 0
     1 0 1 0 0 1 0 0
```

Computing the L1 is simple because it equals R0.

```
        0 0 1 0 0 0 0 0
        0 0 0 1 0 0 0 0
L1 =    1 0 1 1 1 1 0 0
        1 0 1 0 0 1 0 0
```

Calculation of f(Rn,Kn+1).

Computation of R1 is more complex because of the computation involving the key (K) and R. First, the key will be addressed. As was mentioned earlier, only 56 of the 64 bits of the key are used. For each of the 16 iterations a new sequence of key bits is generated, K1..K16, called the key schedule. Figure 2.5 depicts the key schedule calculation and shown below is the derivation of K1, given the key:

Initial Key : 38 49 67 4c 26 02 31 9e (hex)

```
0 0 1 1 1 0 0 0
0 1 0 0 1 0 0 1
0 1 1 0 0 1 1 1
0 1 0 0 1 1 0 0
0 0 1 0 0 1 1 0
0 0 0 0 0 0 1 0
0 0 1 1 0 0 0 1
1 0 0 1 1 1 1 0
```

The first calculation to be accomplished is to apply the Permuted Choice 1 table found in Appendix C to the original 64 bits of the key to select the 56 bits to be used. The 56 bits are then divided into 2 halves labeled C0 and D0.

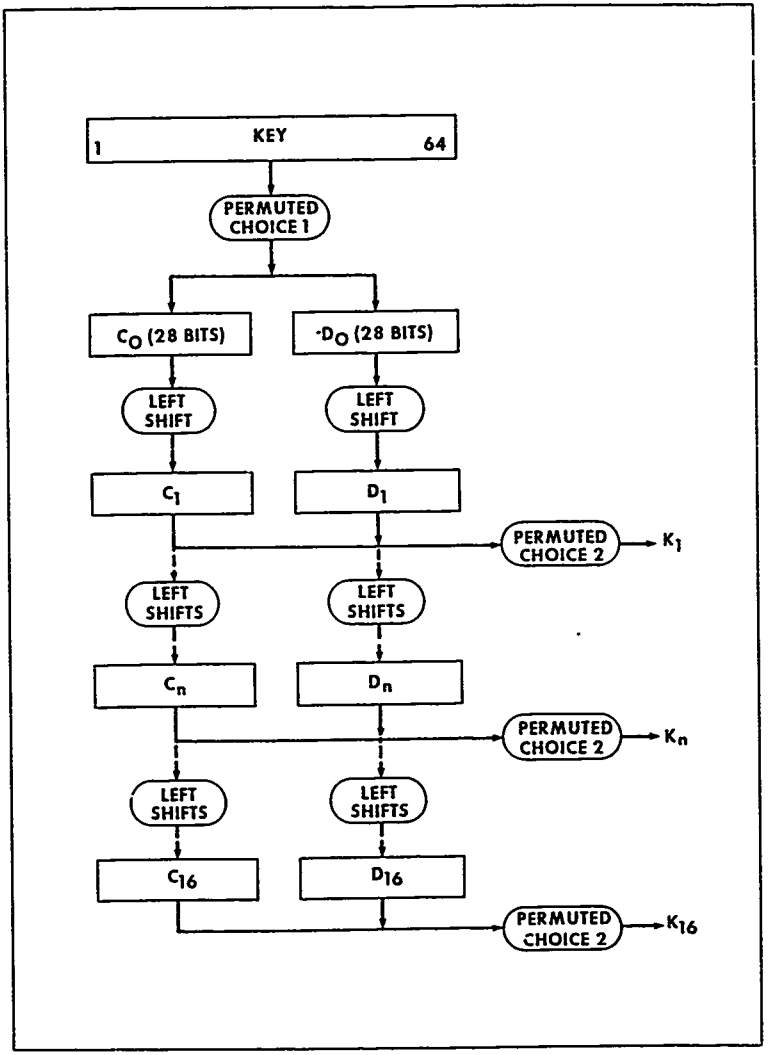


Figure 2.5. Key Schedule Calculation [5]

```
      1 0 0 0 0 0 0                1 0 1 1 0 1 0
C0 =  0 0 0 0 0 1 1          D0 =  0 1 0 0 1 1 1
      1 0 0 1 0 1 0                0 0 1 0 0 0 1
      1 0 1 1 1 0 0                0 1 1 0 0 0 1
```

Next, to derive C1 and D1, the blocks are shifted either 1 or 2 bits to the left depending on the Left Shift Table found in Appendix C. C1 and D1 require 1 bit shifted left.

```
      0 0 0 0 0 0 0                0 1 1 0 1 0 0
C1 =  0 0 0 0 1 1 1          D1 =  1 0 0 1 1 1 0
      0 0 1 0 1 0 1                0 1 0 0 0 1 0
      0 1 1 1 0 0 1                1 1 0 0 0 1 1
```

Now, concatenate C1 and D1 and apply the Permuted Choice 2 table found in Appendix C to derive Kn+1 or this case, K1.

```
      1 1 0 1 0 0
      0 1 0 0 1 0
      1 0 1 0 0 0
      0 0 0 0 1 0
K1 =  1 0 1 0 0 1
      1 1 1 0 1 1
      1 0 1 1 0 0

      0 0 1 1 0 0
```

Before R0 can be used, it must be expanded from 32 bits to 48 bits using the E-Bit Selection Table in Appendix D.

R0	(E)xpanded R0
0 0 1 0 0 0 0 0	1 0 1 1 0 1
0 0 0 1 0 0 0 0	0 1 1 1 1 0
1 0 1 1 1 1 0 0	1 0 0 1 0 1
1 0 1 0 0 1 0 0	0 1 0 0 1 0
	1 0 1 1 1 0
	1 0 0 1 0 0
	0 0 0 1 1 0
	1 0 1 1 1 f

Next, exclusive OR K1 and ER

```
  K1 - 110100010010101000000010101001111011101100001100
+
  ER - 101101011110100101010010101110100100000110101110
  ----------------------------------------------------------
       100001110001001101010101011000011111110010111010
```

The resulting 48-bit string is then divided into 8 blocks of six bits each. Then functions S1..S8 which are found in Appendix C are applied to the 8 blocks to produce the 32 bit string to be exclusive OR'ed to L0. The end result will be R1. The S function operates in the following manner. The first and last bit a block represent a base 2 number in the range 0 to 3. Let it be 'i'. The middle four bits represent a base 2 number in the range 0 to 15. Let this number be 'j'. By looking in the specific S table in the ith row and jth column, a four bit number can be found and this number will replace the 6 bit block.

Example of how a six bit string is converted to a four bit string.

```
  R0 + K1

1 0 0 0 0 1 ----------> 1 0 0 0 0 1
1 1 0 0 0 1
0 0 1 1 0 1              11    0000
0 1 0 1 0 1                                        S1  = 15 or
0 1 1 0 0 0
0 1 1 1 1 1         row   3 ,   0   column         1111
1 1 0 0 1 0
1 1 1 0 1 0

  100001 110001 001101 010101 011000 011111 110010 111010

    S1     S2     S3     S4     S5     S6     S7     S8

   1111   1011   0110   0010   1101   1000   1111   0011
```

The final step to deriving the value of the function is to apply permutation P (Appendix C) the new 32 bit stream.

```
f(R,K) =  0111 0011 1101 1111 1100 1110 0000 0111
```

Deriving R1.

To derive R1, exclusive OR the value of the function with L0.

```
   L0    =  0110 1111 0010 1001 0111 0010 0011 0111
    +
 f(R,K) =  0111 0011 1101 1111 1100 1110 0000 0111
-------------------------------------------------------
   R1    =  0001 1100 1111 0110 1011 1100 0011 0000
```

3. Repeat the second half of step two with the new values of L and R as input until L16 and R16 have been computed.

4. Concatenate R16 and L16 and perform the Inverse Initial Permutation (IIP). The IIP rearrangement sequence is given in Appendix C. The result is encrypted version of the original data information.

2.5.3 Implementation of The DES for Secure Login. The DES will not be used to actually encrypt the password that is transmitted to gain access to a remote computer resource. Instead, it will be used to encrypt a randomly generated 64 bit data stream. When a remote station requests to log into a computer resource, the local station will request the users login name and then transmit a randomly generated 64 bit data stream to the remote user. Both stations will then encrypt the bit stream with the user's key which will be located in a file at both stations. The remote user will then transmit the encrypted string to the local user, at which point the local user will compare both encrypted strings. If they are equal, the remote user will be granted access to the computer resource.

III. Design for Integration of DES into the KA9Q Package

3.1 Introduction

The objective of this effort was to develop a TCP/IP gateway interconnecting the AX.25 packet radio network to the AFITNET network. In chapter one, a software package was identified which provided the required network protocols; the KA9Q Internet software package. However, when using this software to telnet to a mainframe, it was revealed that by responding to the password prompt, the password was transmitted in the clear for any station monitoring the frequency to see. The focus of this effort, consequently, shifted to developing a means of providing a passwordless login to a mainframe using the KA9Q code as a basis.

A password is a method of validating whether the requesting user is allowed access to a specific computing resource. If no password is to be transmitted over the packet radio network, another method of validation is required. The Data Encryption Standard was identified as an acceptable means of validation along with a package with the DES encryption/decryption algorithms implemented in software. Thus, the final problem definition: Design and implement a way to integrate the existing DES software into the KA9Q software so that the encryption and validation process and login process to a mainframe are invisible to the user.

3.2 Methodology

In chapter one, the ability to negotiate user defined options under telnet was discussed as a method of implementing DES as a validation process. With this in mind, the solution appeared obvious; develop a DES telnet option and modify both the telnet client in the KA9Q code and telnet server on the mainframe. A general concept of operation of an end product is as follows (Figure 3.1):

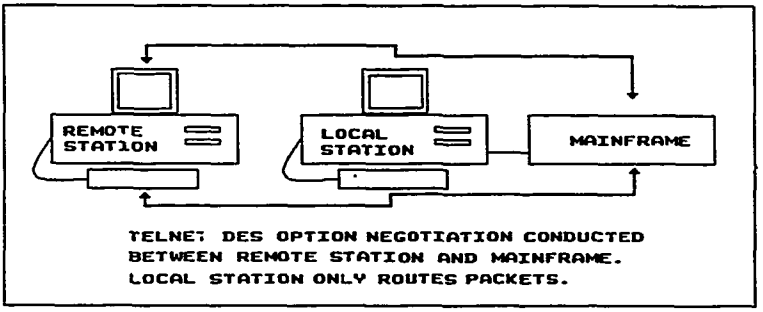


Figure 3.1. Ideal Design for Implementing a Telnet DES Option

1. The remote packet radio user enters *telnet mainframe.*
2. Once the TCP connection has been established, the mainframe requests the users login name. Upon receiving it, the mainframe passes the remote user a plain text string to encrypt, while encrypting the string itself. (Both the remote and mainframe have the user's DES key in a file.)
3. The remote user then passes the encrypted string back to the mainframe where it compares both encrypted strings. If they match, the mainframe retrieves the user's actual password from a file and logs the user in.

Unfortunately, this solution is not feasible because the telnet software is proprietary software. It became clear that the design for a passwordless login to a mainframe would be depend on making modifications to the KA9Q source code.

Two issues had to be addressed when developing a design. They were:

1. The user must only have to enter *telnet mainframe.*
2. The packet radio station performing the function of a gateway to the mainframe will have to negotiate the telnet DES option with the remote machine and will be responsible for logging the remote user into the mainframe.

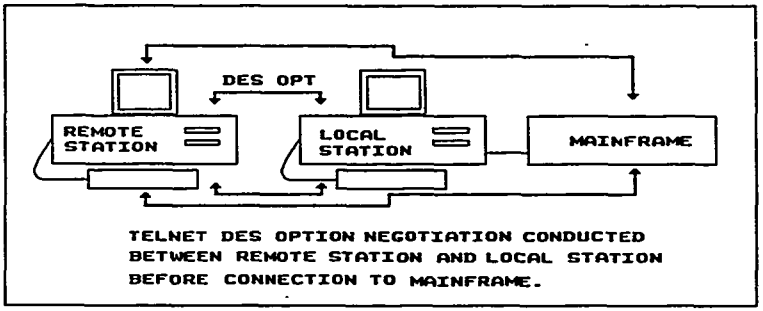


Figure 3.2. Design Option One

With these design considerations, in mind, two designs were developed for consideration.

3.2.1 Design Option One. The first design option is represented in Figure 3.2. In this design, the user enters *telnet mainframe.* The remote machine temporarily saves the IP address of the mainframe and instead, performs a telnet session with the local or gateway machine using the IP address of the local machine. Once the TCP connection is established, the DES telnet option is negotiated. When the local machine has validated the remote user, the remote machine automatically initiates a telnet session to the mainframe using the mainframe's IP address that it previously saved. The local machine, performing the function of a gateway, routes each packet to the mainframe and back. This is where a problem arises.

The remote machine is now directly connected to the mainframe and will receive the request for the login name and password. If the remote user responds to these prompts, the password will be transmitted over the airwaves for all to see. To resolve this, the remote machine will respond to the prompts by sending a flag in the data field which indicates which prompt has been received. The local machine will intercept each packet to the mainframe, inspect the data field, and look for the

prompt flags. Depending on the flag, the required login name or password will be retrieved from a file, stuffed into the data field, and transmitted to the mainframe, accomplishing the login process. The local machine will then stop intercepting packets and resume normal gateway functions.

This appears to be a reasonable solution. However, in order to accomplish it, the local machine, must disassemble each packet by stripping away the IP and TCP headers, inspect the data field for the prompt flags, and then reassemble each packet. When stuffing information into the data field, the integrity of the original packet is changed, thus changing its checksum. In addition, careful examination of the code revealed that new functions have to be created to manipulate the packets while in the monitoring mode.

3.2.2 Design Option Two. This option is similar in appearance to the first one but, in fact, is very different (Figure 3.3). In this design, the remote user enters *telnet mainframe*. The remote machine processes the telnet command and transmits the IP address of the mainframe. The local machine intercepts the packet, recognizes that the connection is meant for a mainframe and responds as if it were the mainframe. It negotiates the telnet DES option with the remote machine and validates the user. The local machine then takes the IP address of the mainframe, telnets to it, and responds as if it were the remote machine. When the requests for the login name and password are received, the local machine retrieves the required information from a file and transmits it to the mainframe. The local machine then returns to performing the functions of a gateway by routing packets from either the remote station or mainframe to the other without intercepting the packets.

A thorough review of both designs was conducted and the second design was selected for implementation. There were two contributing factors for its selection. The first one being that the integrity of a packet transmitted by the remote station would be retained. No examination of data fields by the local machine would be

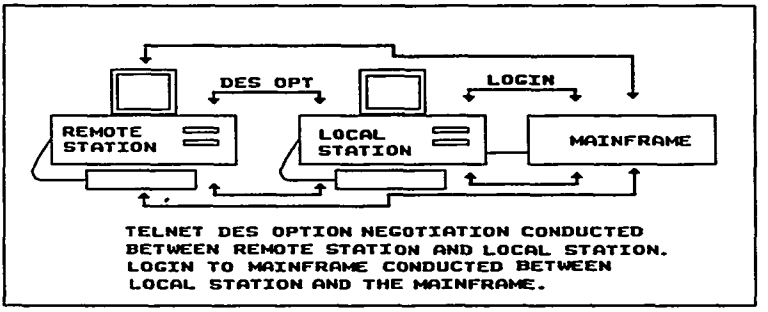


Figure 3.3. Design Option Two

necessary. The second factor is that this design could be supported by the existing code with only modifications being made to selected functions.

3.3 Original Source Code Design Considerations

The analysis of the source code performed when checking its compatibility with the two designs revealed three concepts that needed to be considered prior to developing the chosen design. The three concepts are the Transmission Control Block (TCB) data structure, the finite state machine, and sequence and acknowledgment numbers within a packet header.

3.3.1 The Transmission Control Block (TCB). The TCB is a data structure that maintains current information on each TCP connection (see Appendix C for TCB structure definition.) When the KA9Q program is installed, a generic TCB is generated with the machine's IP address in the local address field and a locally generated port number in the port field. The remote address and port are set to NULL. A hash function is then called to determine the index number for the array which maintains the active TCBs, to include the generic TCB. The hash function uses the local and remote addresses and ports to determine the array index.

The generic TCB is important when a TCP connection is established. When a station receives a request for a TCP connection, the local address and port number fields of the CONN struct data structure receive the values of the destination address and port number found in the IP header. The CONN data structure maintains the current source and destination address and port number of the current packet being processed (see Appendix C for CONN structure definition.) These numbers will be the same as the machine's IP address if, in fact, the packet is for the receiving station. The remote address and port numbers are set to the source address and port numbers found in the IP header. The hash function is then called and passed the CONN data structure. The hash function generates an index number for the TCB array and the array is inspected to see if a TCB with those addresses has been generated. If none is found, the remote address and port numbers in the CONN data structure are set to NULL and the hash function is called again. This time the generic TCB will be retrieved so that a copy of it can be made for the new TCP connection. The remote address and port fields are changed in the new TCB to reflect the actual address and is then stored in the TCB array. It is at this point that the second design encounters a problem.

In the second design, the local machine responds as if it were the mainframe, as well as, the remote station. When an initial connection request is made, the destination address (mainframe address) in the IP header is assigned to the local address field of CONN and a search for a TCB is made. No TCB will be found. The next step is to retrieve the generic TCB. Unfortunately, the local address in the CONN structure is not that of the local machine so the generic TCB cannot be retrieved because the index number will be different than expected. Instead, an error message will be returned and the connection request denied. Thus, in designing option two, a method of capturing all address and port numbers at the local machine needed to be developed so that the appropriate addresses could be substituted when retrieving TCBs.

3.3.2 The State Machine Concept. The concept of the finite state machine was briefly mentioned in chapter one. The KA9Q package has several data structures which implement this concept to record the current state the program is in when establishing a TCP connection or negotiating a telnet option. In the TCB block, there is a field called *state*. Different sections of code are executed depending on what state the TCP connection is in. The state field in the generic TCB is initialized to the *Listen* state. This state changes to *Established* when a connection is established.

There is also a telnet control structure with a state field. Its default value is 'TS_DATA' and this allows for normal processing of the data in the data buffer of a packet. When telnet options are negotiated, the state changes to reflect what option is being negotiated.

In designing the integration of the DES software into the KA9Q code this feature proved invaluable because a new state could be created for a DES option (TS_DES) which would trigger code related to the DES to be executed.

The concept of the state machine also proved useful in manipulating the DES code itself, as well as, in the routing of packets. These state machines will be addressed in more detail in the discussion of the detail design. In Appendix D, there is a table which reflects the different states the program transitions to when establishing a passwordless login from a remote packet station to a mainframe. It provides a detailed view of all pertinent state changes and what events trigger the changes.

3.3.3 Sequence Numbers and Acknowledgment Numbers. This concept is not unique to the KA9Q package, however, an understanding of them is important. When a TCP connection is established, both the transmitting and receiving station synchronize the sequence numbers that will be used to keep track of the packets. As packets are built and transmitted, the sequence number increments. The acknowledgment number indicates to a station what sequence number it should anticipate

from the distant end. If the sequence number of a packet is not the same as the acknowledgment number or is not in the window of acceptable numbers, the packet is discarded.

In the second design, although the final result is a connection between the remote station and the mainframe, two intermediate connections were made prior to this. They are, the connection between the local station and the remote station which negotiates the telnet DES option and the connection between the local station and the mainframe which performs the login to the mainframe. When the local machine establishes these connections, it has synchronized two different sets of sequence numbers with the remote station and mainframe (recall that the end machines believe they are transmitting to each other.) When the local station returns to normal gateway functions and routes packets between the two end stations, the packets are rejected because the sequence and acknowledgment numbers are different. The two end machines never had an opportunity to synchronize their numbers with each other. In the detailed design, a method had to be developed so that the sequence and acknowledgment numbers between the two end processes could be synchronized.

3.4 Detailed Design of Option Two

The design of option two was broken down into three phases because of three significant events which occur during the connection process. These three phases are:

Phase (1) The negotiation of the DES telnet option between the remote machine and the local machine.

Phase (2) The login process between the local machine and the mainframe.

Phase (3) The removal of the local machine and the connecting of the remote station with the mainframe.

A discussion of the design for each phase follows. Flowcharts for all the major functions which were modified are included in this chapter with other supporting flowcharts in Appendix E.

Phase 1: Design of the Remote Station to Local Station DES Telnet Option Negotiation Connection

The validation of the remote user is accomplished in this phase. Although the remote machine believes it is connected to the mainframe, in fact, a connection has been established to the local machine so that the telnet DES option can be negotiated. The design had to address the following issues:

- How to intercept packets at the local machine addressed for a mainframe?
- How to manage the retrieval of the correct TCB when intercepting packets at the local machine?
- How to enter the telnet DES option?
- How to conduct the DES encryption process between the two machines?

Issue One: Intercepting Packets.

In the KA9Q package, the routing of packets is handled in a function called IPROUTE. In the original source code, when a packet is received, the IP header is stripped off and the destination address is inspected. If the address matches that of the receiving station, the packet is accepted and passed to the TCP protocol function TCPIN. If there is no match, the function checks its routing table and sends the packet out the appropriate interface. It is at this point that the packets must be intercepted. If the local machine identifies a packet for a mainframe and no previous connection has been established, the local machine must be able to respond as if it were the mainframe in order to negotiate the telnet option with the remote station. In addition, the local machine, when telneting to the mainframe, must be

able to intercept packets addressed for the remote machine in order to perform the login process.

The procedure to intercept packets had to be flexible enough to operate for both a remote station, that doesn't need to intercept packets, as well as, for the local station. The flowchart of the IPROUTE function in Figure 3.4 reveals the required modifications. A state manager called *current_machine* was developed that would monitor the state of connection the process was in. The four states are:

CURRENT_MACHINE DEFINITION

Remote_Machine	The state a station is in when initiating a telnet connection.
Local_Machine	The state a station is in when it is receiving a telnet request.
Local_To_Remote	The state a local station is in when negotiating a telnet DES option.
Local_To_Mf	The state a local station is in when logging into the mainframe.

The *current_machine* state manager is a variable that maintains the current state a machine is in during a telnet connection to a mainframe. When a station is in the listening TCP state, the *current_machine* state manager is in the Local_Machine state. When a station requests a telnet connection, its *current_machine* state manager moves into a Remote_Machine state. A machine that receives a telnet request is the local machine and remains in a Local_Machine state if the connection is for itself or some other station other than a mainframe. If the connection is for a mainframe, the *current_machine* state moves into a Local_To_Remote state in order to negotiate

the DES telnet option. If the packet is retained, the function IP_RECV is called which in turn calls the function TCPIN to process it.

Issue Two: Searching for the Correct TCB.

In the function TCPIN (figure 3.5), the issue of changing the source and destination addresses is resolved. If the *current_machine* state manager is in a Remote_Machine or Local_Machine state, no resetting of addresses is required because the incoming packets are for the receiving station. If the *current_machine* state manager is in Local_To_Remote or Local_To_Mf state, we are in the local machine. The connection is intended for a mainframe or the remote station and the local station must substitute its IP address into the CONN structure before the TCB is retrieved.

The TCB is used when forming the TCP header. When the local station is intercepting packets from the remote station and mainframe, it has had to temporarily place its IP address and port number in the local address and port field of the TCB. If these fields are not reassigned the address and port number of the station which initiated the connection, the end receiving station of the packet will not find the correct TCB. This situation is handled in the function TCP_Output which generates the TCP header (Figure 3.6). The correct addresses will be placed in the TCB based on the state of the *current_machine* state.

Issue Three: Negotiating a DES Telnet Option.

A telnet option may be negotiated by either station once the TCP connection has been established. The remote station was selected to initiate the DES request because it desires a connection to a mainframe. Once the TCP connection is established, the remote station places the telnet option command string, *IAC DO TN_DES*, in the data buffer of an outgoing packet. The local station detects that the received packet has data and passes it to a function called RCV_CHAR. The

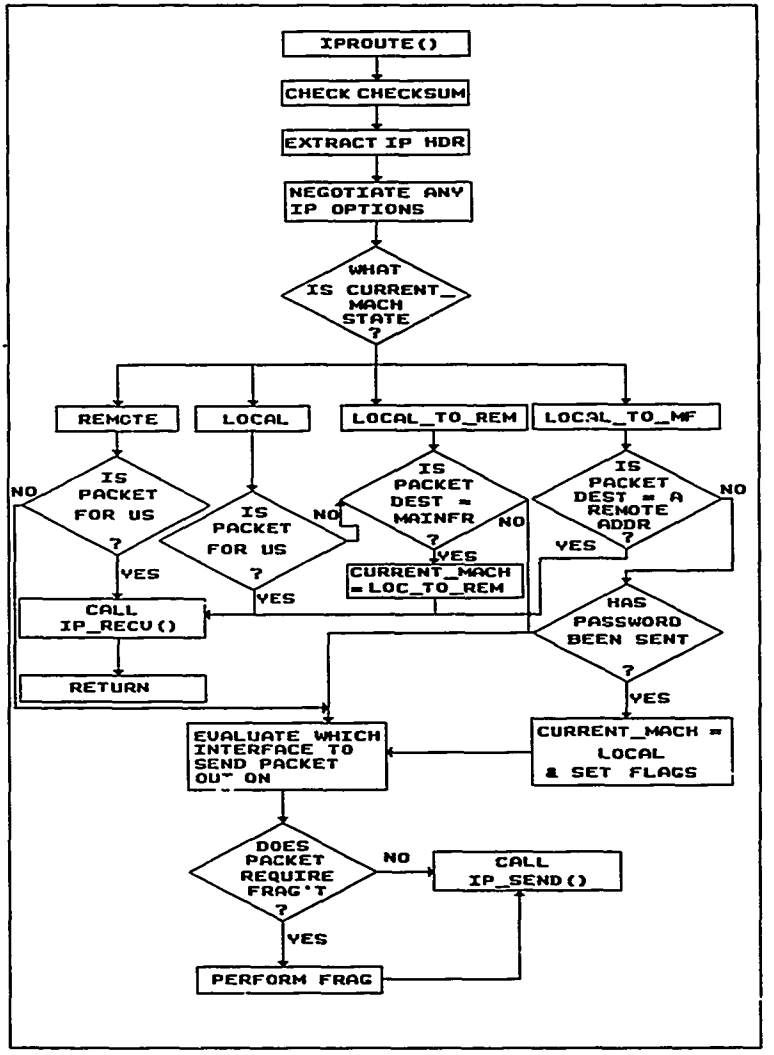


Figure 3.4. Modified IPROUTE() Function

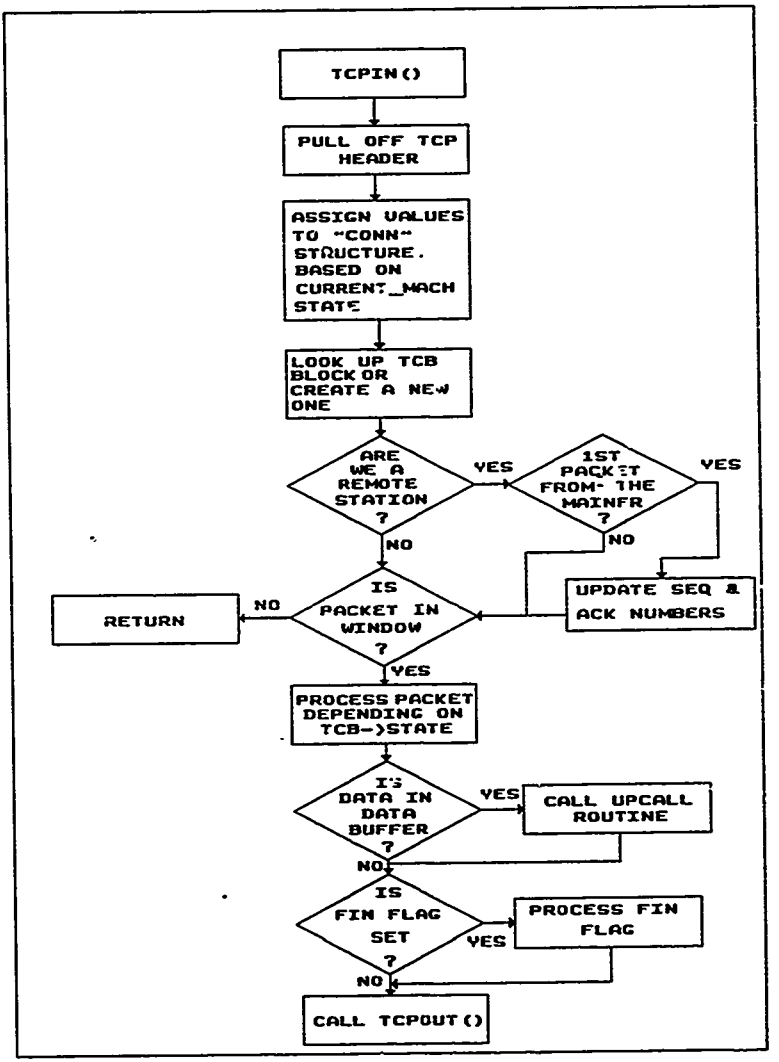


Figure 3.5. Modified TCPIN() Function

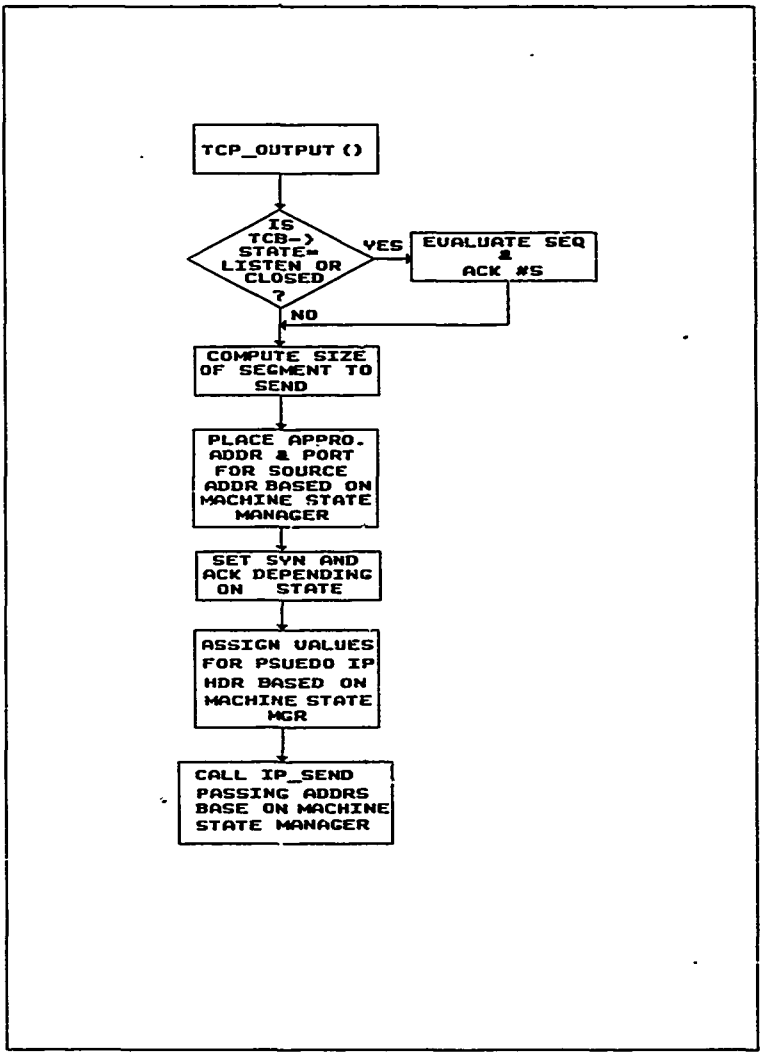


Figure 3.6. Modified TCP_OUTPUT() Function

function RCV_CHAR determines what function will receive the data buffer depending on what state the telnet control block is in. In this case, it is initially in the TS_DATA state so the data is passed to a function called TEL_INPUT (Figure 3.7). In TEL_INPUT, the telnet option command string is interpreted and responds with *IAC WILL TN_DES*. When the local station receives a second *IAC DO TN_DES*, the state variable in the telnet control block changes from *TS_DATA* to *TS_DES*. This change will cause the function, RCV_CHAR, to pass future data to a called function TEL_DES which will conduct the encryption process with the remote station (Figure 3.8). The data to be passed includes the login name, the plain text string, and encrypted string. The remote station will enter a TN_DES state after it receives the command *IAC WILL TN_DES*.

Issue Four: Conducting the DES Encryption Process.

The DES encryption process commences when the telnet state is set to TS_DES and the function TEL_DES is called (Figure 3.9). In the function TEL_DES, there is a state manager called *des_state* which maintains which state the encryption process is in depending on whether the station is the remote station or local station. Listed below are the five states, their definitions, and which machine enters which states.

DES_STATE	DEFINITION	STATION
DES_NAME	Requests login name	Local
DES_SEND_NAME	Sends login name	Remote
DES_ENC_LOCAL	Generates plain text string, sends to remote, and encrypts string.	Local
DES_ENC_REM	Encrypts plain text string and sends to Local.	Remote

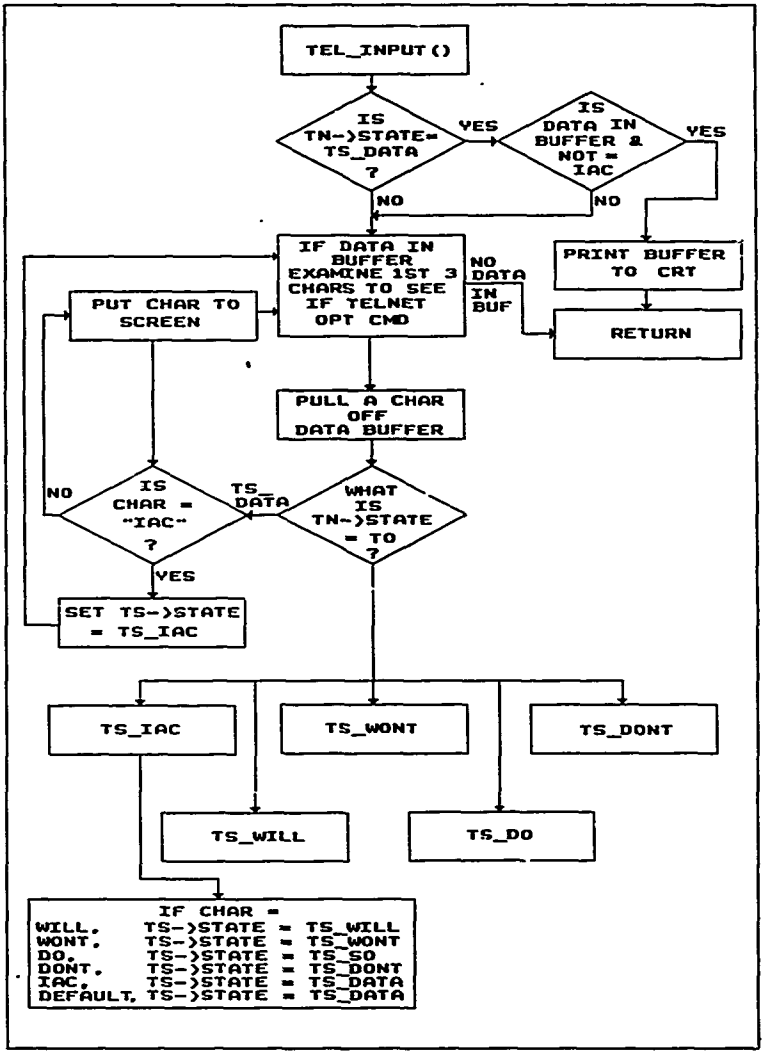


Figure 3.7. Modified TEL_INPUT() Function

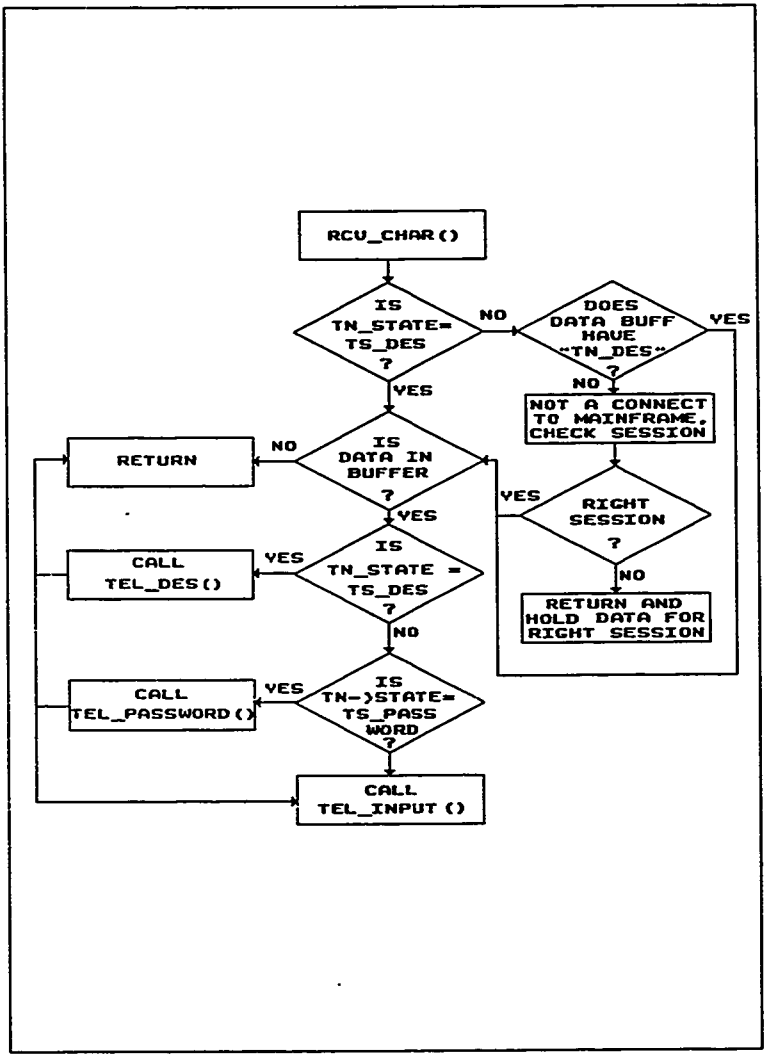


Figure 3.8. Modified RCV_CHAR Function

DES_VALIDATE	Compares two encrypted string and validates remote user.	Local
USER_VALIDATED	Receives message from local as to status of user.	Remote

The local station begins the encryption process in the DES_NAME state and the remote station starts in the DES_SEND_NAME state. These states are set before each station completes processing the telnet option command. The local station begins the process by requesting the remote user's login name while in the DES_NAME state and then moves into the DES_ENC_LOCAL state. The remote station, in the DES_SEND_NAME state, retrieves the name from a file, sends it back to the local station, and then moves into the DES_ENC_REM. The local station, with the login name, retrieves the user's key from a file, generates a plain text string, sends it to the remote station and also encrypts it. It then moves into the DES_VALIDATE state. The plain text string is generated using a standard C function that takes an instant of time based on a given seed. The remote station, upon receiving the string will encrypt it, send it back to the local station, and move into the USER_VALIDATED state. The local station, after it receives the encrypted string, will compare them. If they are equal, a message is sent to the remote station informing him that the remote user has been validated and login to the mainframe will continue. This also allows the remote station to prepare for a packet from the mainframe. If they aren't equal, the local station will close the session down. The completion of the event in a given state triggers the state to change to the next one. When the event in the last state has been completed, the telnet state is changed to TS_DATA to allow normal traffic to resume.

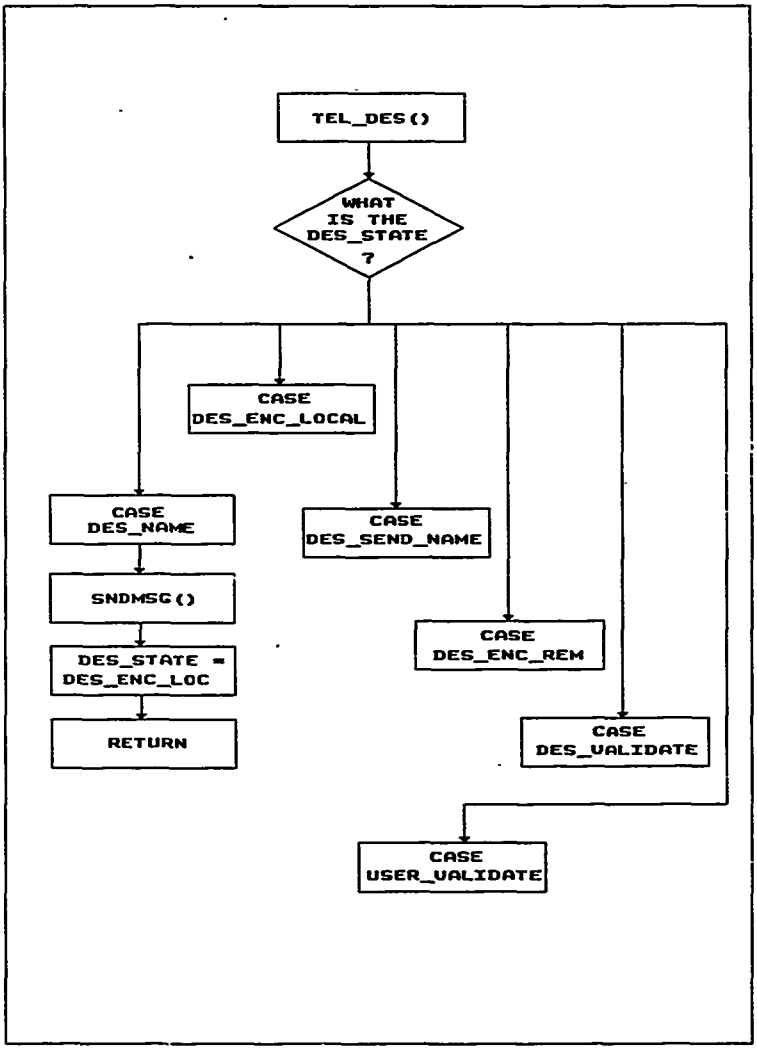


Figure 3.9. TEL_DES() Function

Phase 2: The Login Process Between The Local Machine and The Mainframe.

In this phase, the local machine will direct its packets to the mainframe by telneting to the selected mainframe. The local machine is triggered to perform the telnet operation during the validation process of the two encrypted strings. If they are equal, the function DOTELNET is called with the IP address of the selected mainframe being passed to it (Figure 3.10). In the function DOTELNET, the request is recognized as one for a mainframe so the machine state manager is set to LOCAL_TO_MF which will allow the local station to respond as the remote station. In addition, the telnet state variable is set to TS_PASSWORD which allows the local station to capture the login and password prompts from the mainframe (Figure 3.11).

Phase 3: The Removal of the Local Machine and the Connecting of the Remote Station with the Mainframe.

In this phase the local machine must cease to intercept packets from the mainframe and remote machine and resume normal gateway functions of routing packets. Once the login name and password have been sent, the telnet state variable is set to TS_DATA, the *current_machine* state manager is set to LOCAL_MACHINE, and a flag is set so that the local station will cease capturing packets for the mainframe or remote station. Three problems became evident at this point of the design.

1. By setting the connection flag to allow the remote and mainframe station to be directly connected, there must be a way to reset the flag once the connection is broken down. If the flag is not reset, any attempt to connect to a mainframe through this gateway will result in a direct connection without the local station ever intercepting packets. To resolve this condition, the remote station, upon closing the connection with the mainframe, immediately telnets directly to the local station. The local station is anticipating this connection and upon

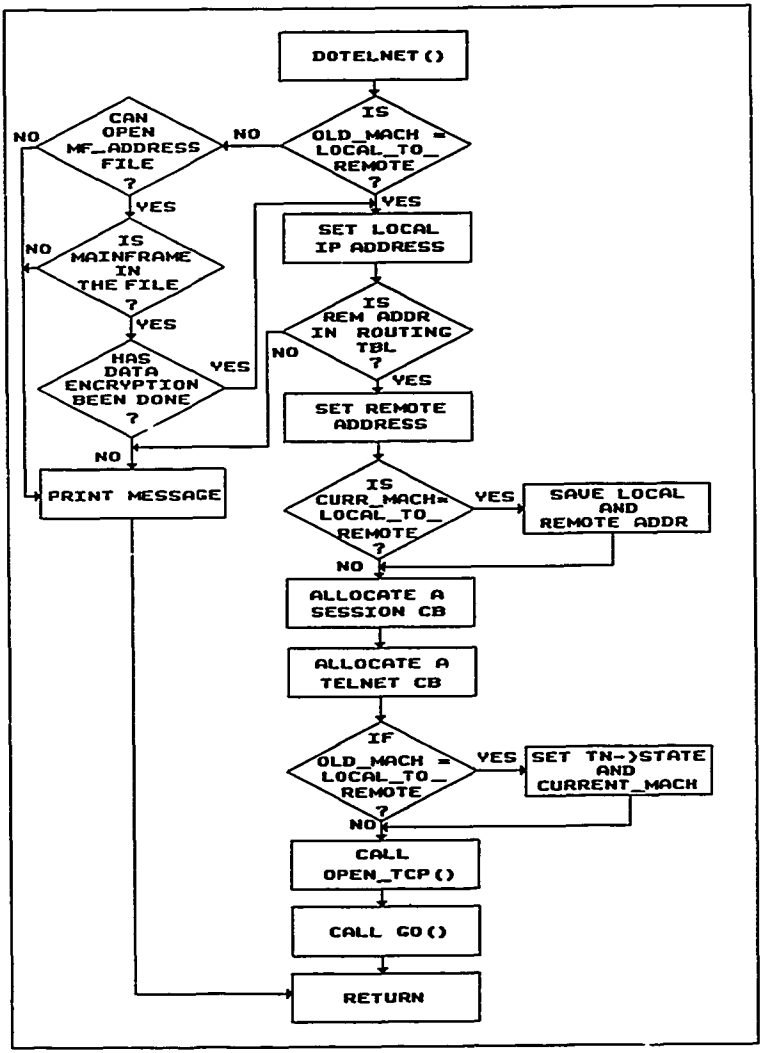


Figure 3.10. Modified DOTELNET() Function

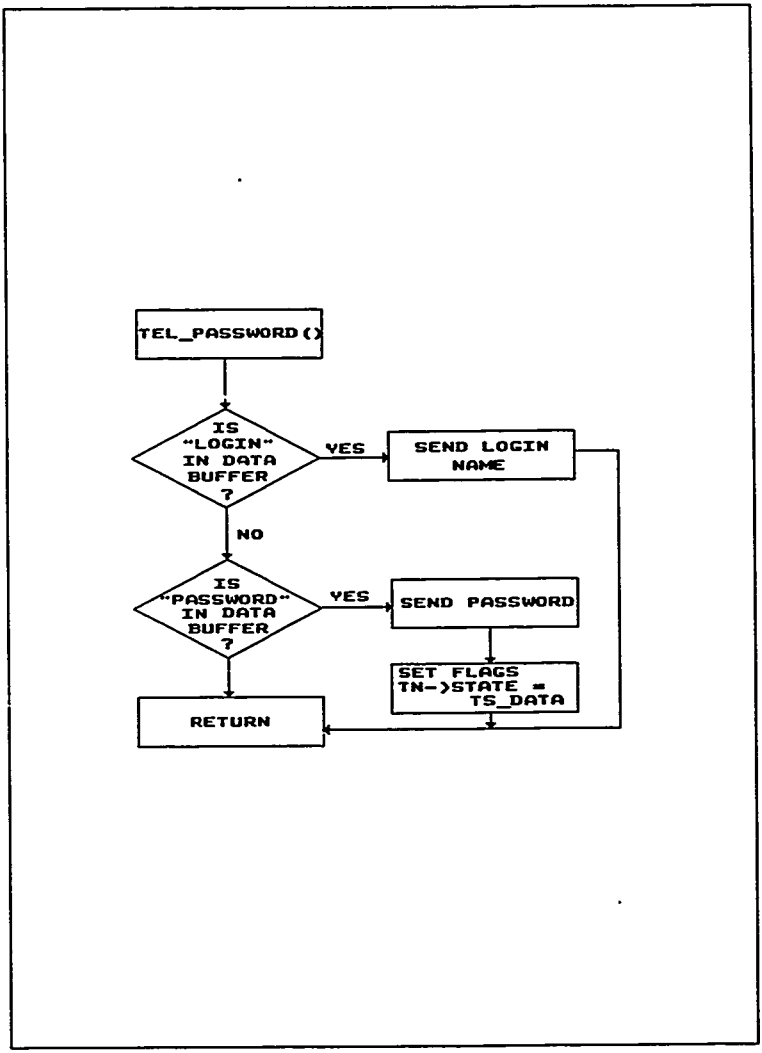


Figure 3.11. TEL_PASSWORD() Function

establishing the connection, resets the connection flag. The remote station, upon a successful connection, closes it down.

2. When the connection between the remote station and the mainframe is broken down, there still remains two active TCBs that were established in the local station. To resolve this situation, the local station when it resets the connection flag discussed in problem one, also removes the two TCBs.

3. The most critical problem is that of the mismatching sequence and acknowledgment numbers. The remote station is the first of the remote-to-mainframe connection to receive a packet once the connection flag has been set. Without resetting the sequence and acknowledgment numbers within the TCB, the incoming packet will not be accepted. This situation is handled in the function TCPIN. When the first packet from the mainframe is received, the sequence and acknowledgment numbers in the TCP header are used to reset the sequence and acknowledgment numbers in the TCB for that connection.

3.5 Limitations

There is one limitation to the chosen design. The original source code for the KA9Q package allowed up to 10 concurrent active TCP/IP connections. This was possible because an incoming packet was either for the receiving station or it was not. No states had to be set as is done in the modified design. When the local station is performing the modified gateway functions for a remote station desiring a connection to a mainframe, no other station may access the local station until the connection between the two end machines is broken down. This limitation makes the modified KA9Q package a single user system w connection to a mainframe is requested.

IV. Implementation of the KA9Q Package with DES

4.1 Introduction

The design and implementation of the DES into the KA9Q package underwent an evolutionary process during its implementation in software. Although a thorough review of the source code was performed and a good understanding of the flow and control of the software was achieved, revisions were necessary to accommodate the existing software. The KA9Q package had limited internal documentation, consequently, trial and error was often used to achieve an understanding of how sections of code functioned. The resulting product, the DES Enhanced KA9Q Internet Software Package provides passwordless logins to specified mainframe computers. The original functionality of the original software was retained.

4.2 Operation of the DES Enhanced KA9Q Package

The enhanced KA9Q package functions much like that of the original version. The installation and operation is essentially the same as described in the original user's manual except for two additional commands and four additional files. The enhanced version can communicate with the original version, however, if a remote station desires to log into a mainframe, both the remote and local station, must have the enhanced version installed on them. If they don't, the remote station will be connected directly to the mainframe without the DES validation process taking place.

The operation of the enhanced package is broken down into two categories:

- Remote Station Installation and Operation
- Local Station Installation and Operation

4.2.1 Remote Station Installation and Operation. The DES Enhanced KA9Q package consists of seven files. Listed below is a brief description of each file and on which drive and directory each should be located.

- NETDES.EXE - This is the executable file for the enhanced version of the KA9Q package. For a system with a hard drive, a subdirectory should be created called 'DES' and this file should be placed there. For a system with only floppy drives it must be on a diskette in the 'A' drive.
- HOSTS.NET - This file is the same as in the original package. It contains the call signs and IP addresses of all valid packet radio and mainframe stations. It is used to convert alphanumeric callsigns into IP addresses. This file must either be in the default drive of the hard drive or on the same floppy diskette as NETDES.EXE. This can be modified to meet individual requirements. See the original user's manual for details.
- AUTODES.NET - This file is also the same as the one in the original package. It can be modified to meet individual requirements. See the original user's manual for details. This file should be in DES directory if using a hard drive or on the same floppy as NETDES.EXE.
- LOGNAME - This file contains the user's login name for the desired mainframe. If a system with a hard drive is being used, this file may either reside in the DES directory or the user may place this file on a floppy diskette and place it in drive 'A'.. If using only a floppy drive system, it must be on the same floppy as the NETDES.EXE file.
- KEYFILE - This file contains the user's DES key. If a system with a hard drive is being used, this file may either reside in the DES directory or be placed on a floppy diskette and read from drive 'A'. If using only a floppy drive system only, it must be on the same floppy as the NETDES.EXE file.

- MF_FILE - This file contains the list of available mainframes and their IP addresses. The format of the entries in the files is *mainframe name, IP address.* If using a system with a hard drive, this file must reside in the DES directory. If using only a floppy drive system, it must be on the same floppy as the NETDES.EXE file.

- MF_PSWD - This file resides in the local station and contains all valid users and their passwords for the all available mainframes. If using a system with a hard drive, this file must reside in the DES directory. If using only a floppy drive system, it must be on the same floppy as the NETDES.EXE file.

The two commands which were added to the package are 'des' and 'driveset'.

- DES - This command is used to set the telnet DES option. By entering the command 'des accept' or 'des refuse' a flag is turned on or off to inform the telnet client whether to request negotiation of the DES option or not. The default state for the flag is 'refuse'. If the user desires to log into a mainframe he/she will have to enter 'des accept' prior to entering 'telnet *mainframe*'. If the user attempts to telnet to a mainframe without first entering the accept command, an error message will be displayed notifying the user to first enter the accept command. The command 'des refuse' will turn the flag off. Conversely, if the user attempts to telnet to another packet radio station while the des flag is in the accept mode, an error message will be displayed notifying the user to first enter the refuse command. By entering 'des' only, the current state of the flag will be displayed. When a connection to a mainframe is disconnected the des flag automatically is reset to the default setting.

- DRIVESET - This command allows the user to specify which drive the LOGNAME and KEYFILE are located. The default location is the on the hard drive in the DES directory. If the user desires to read these files from a floppy, the user enters 'driveset a' which will tell the program to read these files off the

'a' drive. The files must have the same names as mentioned above and may not be in any directories on the floppy diskette. By entering only 'driveset', the current drive setting will be displayed. Currently, the only drives that are implemented are the hard drive and the floppy, drive 'A'. The user must manually reset the drive to the default value if it is currently set for drive 'a'. This is accomplished by entering *driveset c.*

To install the enhanced version, insure the above mentioned files are in place, and enter *netdes autodes.net.* The program will return a banner notifying the user that he/she is using the DES enhanced version of the KA9Q package and the user will then be ready to communicate with other packet stations or valid mainframes.

4.2.2 Local Station Installation and Operation. The installation and operation is the same as for the remote station except for one additional file. This is the file called MF_PSWD. This file contains all the valid users to a specific mainframe along with their actual password. In addition, appropriate changes must be made to the the AUTODES.NET to reflect the different call sign, IP address and ethernet link to the mainframe. The original user's manual provides information on how to do this. The des flag does not need to be set when this package is used in a gateway, it will always accept a telnet DES option request.

4.2.3 Sample Session. This is a sample telnet session with a mainframe.

```
net> des a
net> telnet ssc
     SYN Sent
     Established
     USER VALIDATED
   <Banner of the day for the SSC>
ssc>
   <Pass traffic>
ssc> logout
logout
```

```
Close wait
Last ACK
Closed (Normal)
```

The first task to do is to set the des flag to accept and then to issue the telnet command for the mainframe, SSC. The user will receive two notification messages. The first one indicates that the SYN has been sent and the second one indicates that the connection has been established. If the original package were being used, 'Established' would mean that the desired connection had been established and traffic to the distant end could commence. In the enhanced version, *Established* means that the local machine has intercepted the packets for the mainframe in order to negotiate the DES telnet option. Next, if the encrypted strings matched, the local station will transmit to the remote station the message *USER VALIDATED* which indicates that the encryption validation was successful. The remote user will only know if the connection to the desired mainframe was successful, by receiving packets from the mainframe itself, with the welcome banner and prompt from that particular machine put to the screen. Once this occurs, the remote user is directly connected to the mainframe. If a login to the mainframe was not successful an invalid login message would be transmitted to the remote station and the connection to the mainframe would be disconnected. One should note that the mainframe will echo all commands back to the remote user as seen above with the logout command.

To terminate the connection to the mainframe, the mainframe logout command is given. The user is notified what stage the disconnection is in by the messages, Close wait, Last ACK, and Closed (Normal). The Closed (Normal) message signals that the disconnection to the mainframe is complete. The local station, however, still has two open TCBs which must be deleted so the remote station automatically telnets to the local station and then closes the session down. This action triggers the local station to reset the connection flag which prohibited it from intercepting packets. The last seven lines that appear on the screen.

```
net> SYN Sent
Established
FIN wait 1
FIN wait 2
Time wait
Closed (Normal)
net>
```

When the net prompt has returned, the entire connection process to a mainframe has been completed.

4.2.4 Error Messages. Error messages are provided to the user when one of the following conditions occurs:

- A file that needs to be opened, cannot be.
- The login name is invalid.
- The encrypted strings don't match.
- The mainframe is invalid.
- The remote user attempts to telnet to a mainframe without setting the des flag to accept.
- The remote user attempts to telnet to another packet radio station without setting the des flag to refuse.
- The login to the mainframe from the local station fails.

The first four conditions will only return the error message: Login Failed. This is done in order not to provide an unauthorized user any additional information as to why the connection attempt failed. An authorized user with experience or access to this document will be able to troubleshoot the system if this message is returned. The next two conditions return messages that simply inform the user that the des flag must be set appropriately prior to connecting to a specific station. The last

error message informs the remote user that the login process from the local station to the mainframe was not successful and that something is incorrect in the KEYFILE located in the local station.

4.3 Testing

Testing of the DES enhanced KA9Q software broke down into two phases. The first phase involved testing the modified software to ensure that it retained the same functionality as the original code. All commands that were in the original package were tested and performed as expected. When no mainframe connection was active, establishing all ten concurrent TCP connections was achieved.

Phase two involved testing the DES enhanced modifications to the software. The following criteria were tested to ensure that the software performed as designed:

1. Can a remote user telnet to a mainframe without setting the des option to *accept*?
2. Can a remote user telnet to another PC with the des option set to *accept*?
3. Will a remote user be validated by the DES option when the remote users key is invalid or missing?
4. Will a remote user be validated by the DES option when the remote users login name is incorrect or missing?
5. Does the user ever have access to the mainframe's login and password prompt?
6. Can another station use the local station (gateway) while it is servicing a mainframe login?
7. Does the local station ever use the same plain text string more than once when servicing two consecutive mainframe logins?

Each criteria was fully tested and in each case no errors could be detected except for in case 5. In this case, an invalid password was placed in the MF_PSWD file

in the local machine When the invalid password was transmitted to the mainframe it rejected it by sending an invalid login message and requested the login name and password again. Unfortunately, this message was passed to the remote user and he/she was now being asked to re-enter the login name and password. This, of course, is unacceptable; an incorrect login was not being trapped by the local machine because as soon as it transmits the password to the mainframe it sets its flag which prohibits it from intercepting any further packets, consequently, the remote user gains access to the login and password prompts.

To correct this error an initial attempt was made to trap the invalid login message at the local machine and then to retransmit the login name and password in case they, in fact, were correct but had been corrupted during transmission. Since Unix will only allow five login attempts before breaking the connection, it was felt that at worst case the incorrect login name and/or password would be retransmitted five times before the connection would be broken. This process proved to be difficult to implement because of how the local machine managed the two connections it was maintaining between the remote machine and the mainframe. Instead, the invalid login message was allowed to be transmitted to the remote station. The remote station when it starts to receive packets from the mainframe inspects the packets, if the first packets contain the incorrect login message, the message is not be put to the screen and instead the process is placed in a wait state for sixty seconds. At the end of sixty seconds the Unix system automatically breaks the connection because the login process is timed out. To the remote system this termination of the connection triggers it to telnet to the local station so that it may remove the two TCB blocks that are no longer required.

V. *Conclusions and Recommendations*

5.1 Introduction

The design and implementation of the DES Enhanced KA9Q 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 package, although very well structured, did not have any significant 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 any design modifications could be achieved. As an understanding of these areas increased, the design for the modifications underwent changes during the implementation phase. Because of the extensive use of pointers in the original implementation, modifications that were made did not always work as expected. Many hours were spent understanding the use of pointers in the C programming language and an appreciation for individuals who perform maintenance on source code which in not there own, was realized. These obstacles, however, were overcome and the DES Enhanced KA9Q Internet package was produced.

The DES Enhanced KA9Q Internet package when used in a packet radio network, in which one station performs as a gateway to a mainframe, provides the remote station the capability to telnet to the mainframe without transmitting the user's password. Instead, the Data Encryption Standard is used to validate the remote user.

The enhanced version of the KA9Q package retained all the functionallity of the original source code except for the packages ability to handle up to ten concurrent TCP/IP sessions. When one of the stations in the packet radio network has an open connection to the mainframe through the gateway machine, all other connection requests to the local station or mainframe are denied. This situation exists because

the local machine must intercept packets from both the mainframe and the remote user during the login process and the IPROUTE function had to be modified. This modification does not allow the software in the gateway machine to handle more that one connection to the mainframe at a time. This means that the ability of TCP/IP to provide up to ten sessions has been reduced to one. This modification does not affect the ten session capability if packet radio stations are transmitting between themselves. This degradation in capability may not be significant if traffic to a mainframe is expected to be low, however, it could be, if timely access to the mainframe is required.

An approach to solving this problem would be to establish an array of size ten of connection records. Each record would contain pertinent information about each connection such as the numbers, the current state of the *current_machine* variable and the *des_state* variable. When an incoming packet was received the array would be inspected to see if that particular connection existed. If it did then the two state variables would be updated before further processing of the packet was done. Another approach would be to add additional fields to the TCB so that each block could maintain all pertinent information on its own connection. A problem here though is that the TCB block is not retrieved until after the IPROUTE routine which is where validation of a connection is done. Consequently, the establishment of a TCB like structure could be placed in the IPROUTE function to monitor ten concurrent sessions to a mainframe.

5.2 Design Summary

The design that was selected for implementation proved to be straight forward. The problems that were encountered were due to an inadequate understanding of the original source code, as well as, the C programming language in the beginning of the development process. There were four areas in the original source code that were modified. They were available user commands, the telnet code, the IP routing

function, and the deletion of the two TCBs in the local station. A summary of the modifications in each area follows.

User Commands. The modifications to add the two commands *des accept/refuse* and *driveset c or a* were made in the MAIN.C file. These commands were added to the list of possible commands and then the functions for them were added. The functions were modeled after the other existing command line functions.

Telnet Code. Both the telnet server and client were modified. The associated files were TNSERV.C and TELNET.C. The file TELNET.C underwent the most significant changes. The following functions were either modified or created:

1. Dotelnet() - This function was modified so that a telnet session from the local machine to the mainframe could be initiated.
2. Tel_Input() - This function was modified so that it could identify the telnet option request for DES and conduct the negotiation.
3. Rcv_char() - This function was modified to perform the duties of a data buffer router. If the telnet session in progress has not been identified as one to a mainframe, the data buffer (from the remote user) is passed to Tel_Input(). negotiated, any future packets (from the remote user) are passed to Tel_Des() where the encryption and validation process occurs. If the remote user is validated, then this function must pass any data buffers (from the mainframe) to the function Tel_Password() which performs the login to the mainframe.
4. Tel_Des() - This function was created to perform the DES encryption and validation. There are six states of which three are active in the remote machine and three within the local machine.

5. Tel_Password() - This is a new function. Its purpose is to conduct the login session between the local machine and the mainframe. Once the login has been successful, it changes the telnet state to TS_DATA so that the function Rcv_Char will pass any future data buffer to Tel_Input for evaluation.

6. T_State() - This function was modified so that when the remote machine believes a connection has been established to the mainframe, it will request the telnet DES option to be negotiated.

7. Willopt() - This one was modified to handle the DES option.

8. Doopt() - This one was also modified to handle the DES option.

IP Routing Function. The most significant modification involved the intercepting of packets from both the remote station and the mainframe by the local station. The design specified that the local station would establish a session with the remote station while imitating the mainframe in order to negotiate the DES telnet option. It would also have to establish a connection with the mainframe and imitate the remote station in order to log into the mainframe. To accomplish this, different states were established which dictated where the local machine was in the login process from the remote station to the mainframe.

The concept of different states was effective, however, this design called for changes in other files and functions. The file TCPIN.C with the function Tcp_Input() had to know what state the local machine was in in order to substitute the appropriate addresses for retrieval of the appropriate TCB. The function Tcp_Output() in the file TCPOUT.C also required this information so that the appropriate addresses could be place in the TCP header before the packet was

Deletion of TCBs. During the login process to a mainframe, the local station will have one or two active TCBs. If at any time the connection is denied or the connection has been successful and the remote station later logs out

from the mainframe, the active TCBs in the local station must be removed. The file SESSION.C contains functions that handle the creation of sessions and TCPUSER.C contains the function Close_TCB() and Reset() which can remove TCBs. These two files were modified so that the appropriate session number could be determined and the appropriate TCB deleted.

The design and implementation of the DES Enhanced KA9Q package proved successful in demonstrating that DES can be used as a validation method of a remote user in a packet radio network versus transmitting a password over the air waves.

5.3 Recommendations for Future Research

The enhanced version is limited, in that, only one remote user may access the mainframe at one time. This limitation was a result of not having access to the telnet source code on the mainframe and having to implement the DES telnet option totally within the telnet code of the KA9Q package. This, in turn, dictated that the local machine intercept packets and imitate both end machines. These additional transmissions combined with the effective baud rate of the radios at 300 baud make communication between a remote station and a mainframe cumbersome and slow.

The ideal solution is to implement the DES telnet option on the mainframe so that the local station functions only as a gateway and no longer is required to intercept packets. A recommendation for further research is to make a Sun workstation the host mainframe. AFIT owns the telnet code for the suns so modification of the code would be possible.

If the current enhanced version is to be maintained, further research can be done to solve the problem of a single user system when communicating with a mainframe. One possible direction would be to establish an array of records where each record would contain information which identifies it as a unique connection. The information that might be retained is the remote and destination addresses, the *current_machine* state, and the session number.

The addition of the DES to the KA9Q package now permits a remote user to log into a mainframe without transmitting a password. A problem now arises that if there were another station monitoring the same frequency and had a more powerful radio, this station could conceivably was logged into the mainframe. A recommendation for further enhancement of this package would be to add a *DES Checksum* to each packet as an IP level option (encrypt the checksum.) This would allow individual verification of each frame at the receiver. Since the intruding station would not be able to encrypt the checksum with the same agreed upon KEY, packets from the intruding station would be d sregarded.

Teln t is the only protocol in the KA9Q package which is supported by DES. Further research could be done in the area of supporting FTP by using DES to validate the requesting station instead of transmitting a password.

Appendix A. *Compilation Instructions*

In order to generate an executable version of the Enhanced KA9Q package, two sets of files are required. The first set consists of 118 files which contain the source code for the package. The second set consists of the minimal number of Aztec C files that are required to compile the enhanced package. Once both sets are stored in a common directory on the hard drive of an IBM AT, enter "make". This command will execute the MAKE.EXE file which will commence to compile each file as they are listed in the MAKEFILE. With 118 files this process will take approximately 20 to 25 minutes. Once the process is complete the executable will be placed in NETDES.EXE.

Required files of the Enhance KA9Q Internet Package

ALLOC.C	ECVECAT.ASM	MAC_AT H	TELNET.C
AMIGA.C	ENET.C	MAC_IO.C	TRACE.H
AMIGA.H	ENET.H	MISC.C	FILES.C
AMIGA_UT.C	ENETDUMP.C	PC100.C	UDP.C
ARP.C	FTP.C	PC100.H	UDP.H
ARP.H	FTP.H	MBUF.H	UDPCMD.C
ARPCMD.C	FTPCLI.C	PC100VEC.ASM	UDPDUMP.C
ARPDUMP.C	FTPSERV.C	NETUSER.H	UNIX.H
ASY.C	ICMP.C	PATHNAME.C	TELNET.H
ASY.H	GLOBAL.H	EC.C	X.C
ASYVEC.ASM	HAPN.C	PC.H	DES.C
ATDUMP.C	HAPN.H	PCGEN.ASM	TIME.H
AUDIT.C	HAPNVEC.ASM	PING.H	TCPIN.C
AX25.H	ICMP.H	SLIP.C	IPROUTE.C
AX25CMD.C	ICMPCMD.C	SLIP.H	AX25.C
AX25DUMP.C	ICMPDUMP.C	SMISC.C	SESSION.C
AX25SUBR.C	ICMPMSG.C	SMTP.H	MBUF.C
AX25USER.C	IFACE.C	SESSION.H	TCPSUBR.C
BSDUNIX.C	IFACE.H	SMTPCLI.C	TCPOUT.C
BSD_IO.C	INTERNET.H	SMTPSERV.C	TCPUSER.C
CMDPARSE.C	IPCMD.C	SYS5UNIX.C	TTYDRIV.C
CMDPARSE.H	IPDUMP.C	SYS5_IO.C	IP.H

CONFIG.H	KISS.C	TCPDUMP.C	PC.C
DIRUTIL.C	KISS.H	TCPTIMER.C	IP.C
EAGLE.C	LAPB.C	TIMER.C	NETUSER.C
EAGLE.H	LAPB.C	TIMER.H	VERSION.C
EAGLEVEC.ASM	LAPB.H	TCP.H	MAIN.C
ECCMD.C	LAPBTIME.C	TCPCMD.C	TNSERV.C
EC.H	MAC.H	TRACE.C	
ECVEC.ASM	MAC_AT.C	MAKEFILE	

Required Aztec C Files

AS.EXE	COLOR.H	LIMITS.H	SIGNAL.H
CC.EXE	CTYPE.H	LMACROS.H	STAT.H
C.LIB	DIOCTL.H	LOCALE.H	STDARG.H
CGEN.EXE	ERRNO.H	MATH.H	STDDEF.H
LN.EXE	FCNTL.H	MEMORY.H	STDIO.H
CLC.LIB	FLOAT.H	MODEL.H	STDLIB.H
MAKE.EXE	IO.H	SETJMP.H	STRING.H
ASSERT.H	LIBC.H	SGTTY.H	TIME.H

Appendix B. *Structure Definitions in C*

Telnet Protocol Control Block

```
struct telnet {
        struct tcb *tcb;
        char state;

#define TS_DATA 0       /* Normal data state */
#define TS_IAC  1       /* Received IAC */
#define TS_WILL 2       /* Received IAC-WILL */
#define TS_WONT 3       /* Received IAC-WONT */
#define TS_DO   4       /* Received IAC-DO */
#define TS_DONT 5       /* Received IAC-DONT */
#define TS_DES  6       /* DES encryption state */

   char local[NOPTIONS];   /* Local option settings */
   char remote[NOPTIONS];  /* Remote option settings */

   struct session *session;/* Ptr to session structure */
};
```

Session Control Structure

```
struct session {
        int type;
#define FREE    0
#define TELNET  1
#define FTP     2
#define AX25TNC 3
        char *name;     /* Name of remote host */
        union {
                struct ftp *ftp;
                struct telnet *telnet;
                struct ax25_cb *ax25_cb;
        } cb;
        int (*parse)();         /* Where to hand typed input
                                   when conversing */
```

```
        FILE *record;           /* Receive record file */
        char *rfile;            /* Record file name */
        FILE *upload;           /* Send file */
        char *ufile;            /* Upload file name */
};
```

IP Header

```
struct ip {
        char version;           /* IP version number */
        char tos;               /* Type of service */
        int16 length;           /* Total length */
        int16 id;               /* Identification */
        int16 fl_offs;          /* Flags+fragment offset */

#define F_OFFSET        0x1fff  /* Offset field */
#define DF      0x4000          /* Don't fragment flag */
#define MF      0x2000          /* More Fragments flag */

        char ttl;          /* Time to live */
        char protocol;     /* Protocol */
        int32 source;      /* Source address */
        int32 dest;        /* Destination address */
        char options[44]; /* Options field */
        int16 optlen;      /* Length of opts field, bytes */
};
```

TCP Segment Header – Internal Representation

```
/*
 * Note that this structure is NOT the actual header as it
 * appears on the network (in particular, the offset and
 * checksum fields are missing).  All that knowledge is in
 * the functions ntohtcp() and htontcp() in tcpsubr.c
 */
struct tcp {
        int16 source;   /* Source port */
```

```
        int16 dest;     /* Destination port */
        int32 seq;      /* Sequence number */
        int32 ack;      /* Acknowledgment number */
        char flags;     /* Flags, data offset */
#define URG     0x20    /* URGent flag */
#define ACK     0x10    /* ACKnowledgment flag */
#define PSH     0x08    /* PuSH flag */
#define RST     0x04    /* ReSeT flag */
#define SYN     0x02    /* SYNchronize flag */
#define FIN     0x01    /* FINal flag */
        int16 wnd;      /* Receiver flow control window */
        int16 up;       /* Urgent pointer */
        int16 mss;      /* Optional max seg size */
};
```

TCP Connection Control Block

```
struct tcb {
        struct tcb *prev;  /* Linked list pointers for
                              hash table */
        struct tcb *next;

        struct connection conn;

        char state;     /* Connection state */
#define CLOSED          0       /* Must be 0 */
#define LISTEN          1
#define SYN_SENT        2
#define SYN_RECEIVED    3
#define ESTABLISHED     4
#define FINWAIT1        5
#define FINWAIT2        6
#define CLOSE_WAIT      7
#define CLOSING         8
#define LAST_ACK        9
#define TIME_WAIT       10

        char reason;            /* Reason for closing */
#define NORMAL   0   /* Normal close */
#define RESET    1   /* Reset by other end */
```

```
#define TIMEOUT  2   /* Excessive retransmissions */
#define NETWORK  3   /* Network problem (ICMP message) */

/* If reason == NETWORK, the ICMP type and code values are
   stored here */
        char type;
        char code;

/* Send sequence variables */
 struct {
   int32 una;   /* First unacknowledged sequence number */
   int32 nxt;   /* Next sequence num to be sent for the
                   first time */
   int32 ptr;   /* Working transmission pointer */
   int16 wnd;   /* Other end's offered receive window */
   int16 up;    /* Send urgent pointer */
   int32 wl1;   /* Sequence number used for last window
                   update */
   int32 wl2;   /* Ack number used for last window update */
        } snd;
    int32 iss;              /* Initial send sequence number*/
    int16 cwind;            /* Congestion window */
    int16 ssthresh;         /* Slow-start threshold */
    int32 resent;           /* Count of bytes retransmitted */

        /* Receive sequence variables */
        struct {
             int32 nxt;     /* Incoming sequence number
                               expected next */

             int16 wnd;     /* Our offered receive window */
             int16 up;      /* Receive urgent pointer */
        } rcv;
        int32 irs;    /* Initial receive sequence number */
        int16 mss;    /* Maximum segment size */
        int32 rerecv; /* Cnt of duplicate bytes received */
        int16 window; /* Rcvr window and send queue limit */

        char backoff;           /* Backoff interval */
        void (*r_upcall)();     /* Call when "significant"
                                   amount of data arrives */
        void (*t_upcall)();     /* Call when ok to send
```

```
                                        more data */
        void (*s_upcall)();             /* Call when connection
                                           state changes */
        char flags;                     /* Control flags */
#define FORCE   1                       /* We owe the other end an
                                           ACK or window update */
#define CLONE   2                       /* Server-type TCB, cloned
                                           on incoming SYN */
#define RETRAN  4                       /* A retransmission has
                                           occurred */
        char tos;                       /* Type of svc (for IP) */

        struct mbuf *rcvq;              /* Receive queue */
        int16 rcvcnt;

        struct mbuf *sndq;              /* Send queue */
        int16 sndcnt;                   /* Number of unacknowledged
                                         * sequence numbers on
                                         * send queue. NB: includes
                                         * SYN and FIN, which don't
                                         * actually appear on sndq!
                                         */
    struct reseq *reseq;        /* Out-of-order segment queue */
    struct timer timer;         /* Retransmission timer */
    struct timer rtt_timer;     /* Round trip timer */
    int32 rttseq;               /* Sequence num being timed */
    int32 srtt;                 /* Smoothed round trip time,
                                   milliseconds */
    int32 mdev;                 /* Mean deviation, millisnds */

    char *user;                 /* User parameter (e.g., for
                                 * mapping to an application
                                 * control block
                                 */
};
```

Basic Message Buffer Structure

```
struct mbuf {
        struct mbuf *next;        /* Links mbufs belonging
                                     to single packets */
        struct mbuf *anext;       /* Links packets on queues*/
        char *data;               /* Active working ptrs */
        int16 cnt;
};
```

Appendix C. *DES Encryption Tables [5]*

Initial Permutation

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	45	7

Inverse Initial Permutation

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

E Bit Selection Table

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Permutation Function P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

S Function (S1)

Row No.	Column Number															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Permuted Choice 1 (PC-1)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Permuted Choice 2 (PC-2)

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Left Shift Table

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Appendix D. *Remote Station to Mainframe Telnet State Table*

REMOTE MACHINE				LOCAL MACHINE	
State Variables	Current State	New State		Current State	New State
tcb->state tn->state des->state current_mach	listen TS_DATA none remote	syn_sent TS_DATA none remote	--- SYN--->	listen TS_DATA none local	
tcb->state tn->state des->state current_mach	syn_sent TS_DATA none remote		<--- ACK ---		syn_rcv TS_DATA none local
tcb->state tn->state des->state current_mach		estab TS_DATA none remote	--- ACK --->	syn_rcv TS_DATA none local	estab TS_DATA none local
tcb->state tn->state des->state current_mach	estab TS_DATA none remote		- IAC DO TS_DES ->	estab TS_DATA none local	
tcb->state tn->state des->state current_mach	estab TS_DATA none remote				estab TS_IAC none loc_to_rem

REMOTE MACHINE				LOCAL MACHINE	
	Current State	New State		Current State	New State
tcb->state tn->state des->state current_mach	estab TS_DATA none remote				estab TS_DO none loc_to_rem
tcb->state tn->state des->state current_mach	estab TS_DATA none remote		IAC WILL - <- TS_DES		estab TS_DATA none loc_to_rem
tcb->state tn->state des->state current_mach		estab TS_IAC none remote		estab TS_IAC none loc_to_rem	
tcb->state tn->state des->state current_mach		estab TS_WILL none remote		estab TS_DO none loc_to_rem	
tcb->state tn->state des->state current_mach		estab TS_DES des_send_name remote	- IAC DO TS_DES ->	estab TS_DATA none loc_to_rem	

REMOTE MACHINE				LOCAL MACHINE	
	Current State	New State		current State	New State
tcb->state tn->state des->state current_mach	estab TS_DES des_send_name remote				estab TS_IAC none loc_to_rem
tcb->state tn->state des->state current_mach	estab TS_DES des_send_name remote				estab TS_DO none loc_to_rem
tcb->state tn->state des->state current_mach	estab TS_DES des_send_name remote		send request - <- for name "n"		estab TS_DES des_local loc_to_rem
tcb->state tn->state des->state current_mach		estab TS_DES des_remote remote	- user login name ->	estab TS_DES des_local loc_to_rem	
tcb->state tn->state des->state current_mach	estab TS_DES des_remote remote		text string - <- to encrypt		estab TS_DES des_val loc_to_rem

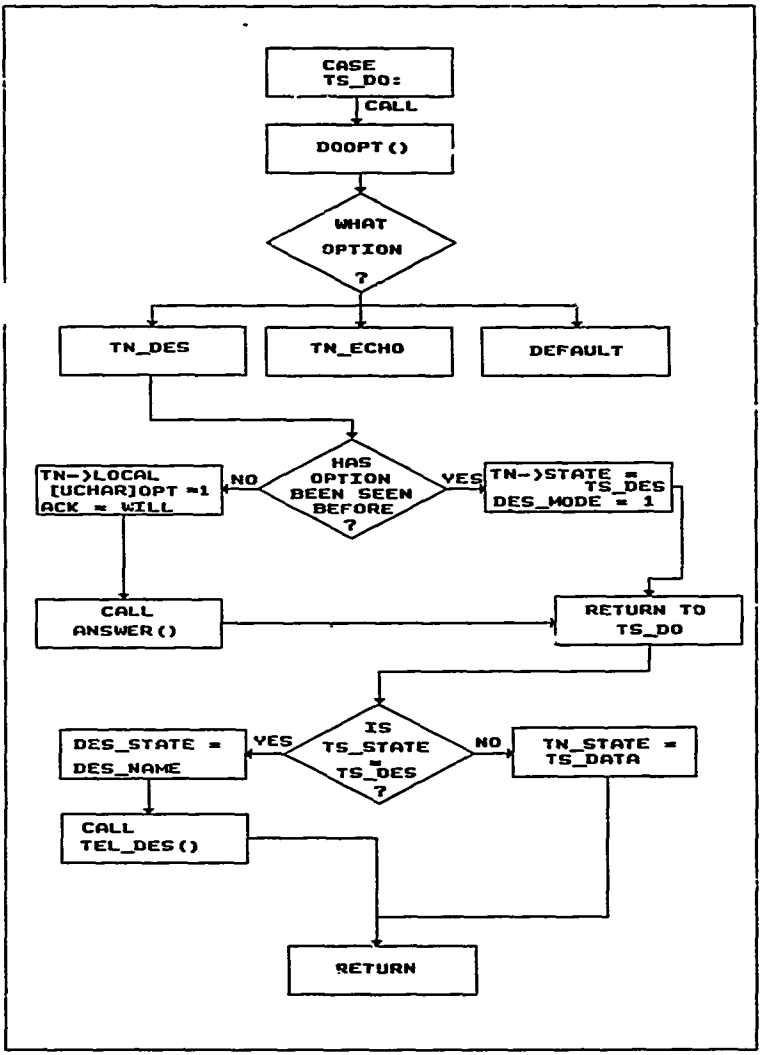
REMOTE MACHINE				LOCAL MACHINE	
	Current State	New State		Current State	New State
tcb->state tn->state des->state current_mach		estab TS_DES none remote	- encrypted string ->	estab TS_DES des_validate loc_to_rem	

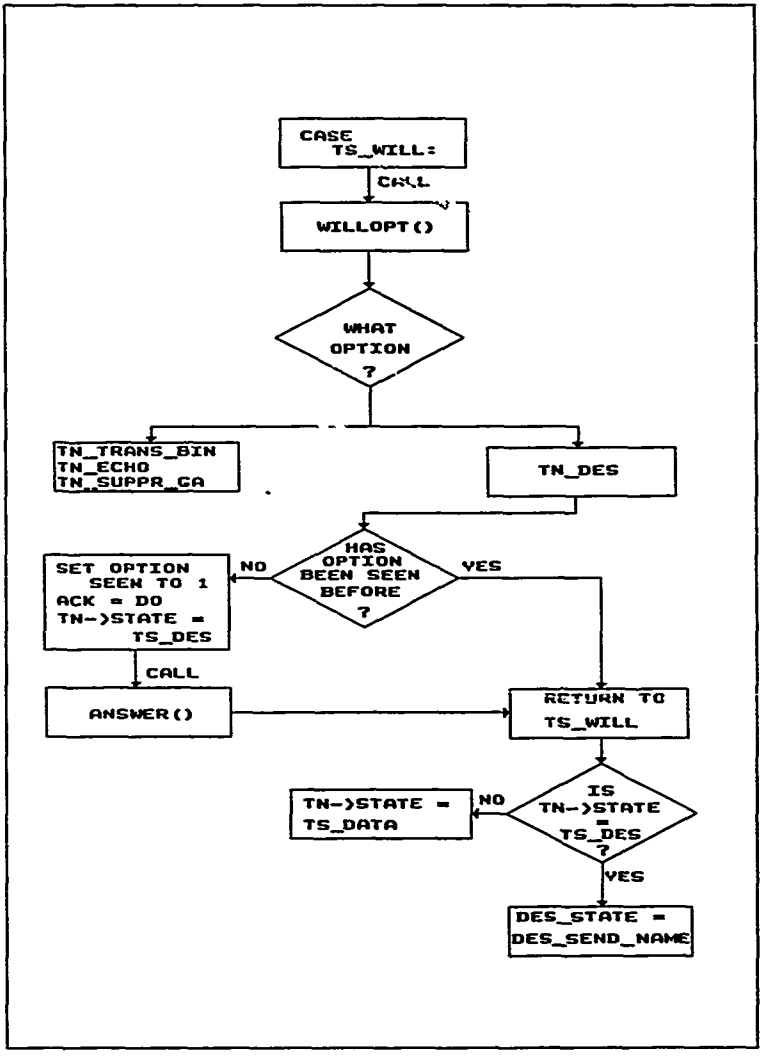
LOCAL MACHINE				MAINFRAME MACHINE	
	Current State	New State		Current State	New State
tcb->state tn->state des->state current_mach		estab TS_PASSWORD none loc_to_mf	- telnet to mainframe ->		
tcb->state tn->state des->state current_mach	estab TS_PASSWORD none loc_to_mf		<- Login prompt -		
tcb->state tn->state des->state current_mach		estab TS_PASSWORD none loc_to_mf	- Login name ->		

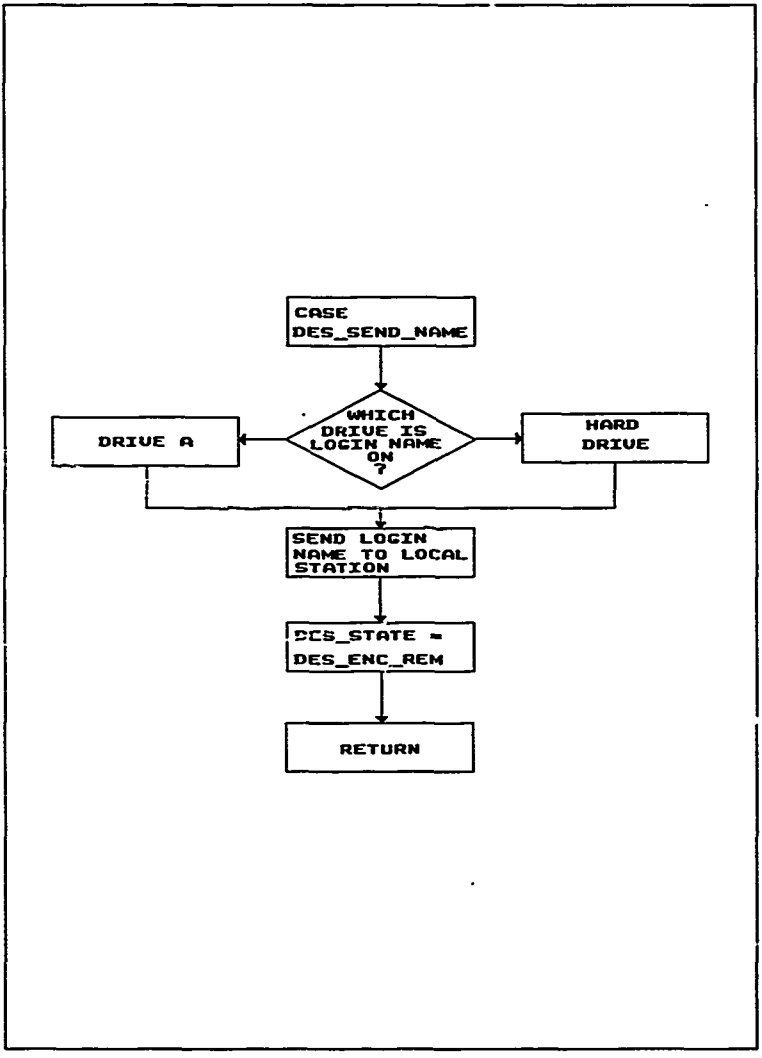
LOCAL MACHINE				MAINFRAME MACHINE	
	Current State	New State		Current State	New State
tcb->state tn->state des->state current_mach	estab TS_PASSWORD none loc_to_mf		Password - <- prompt		
tcb->state tn->state des->state current_mach	estab TS_DATA none local		- Password ->		

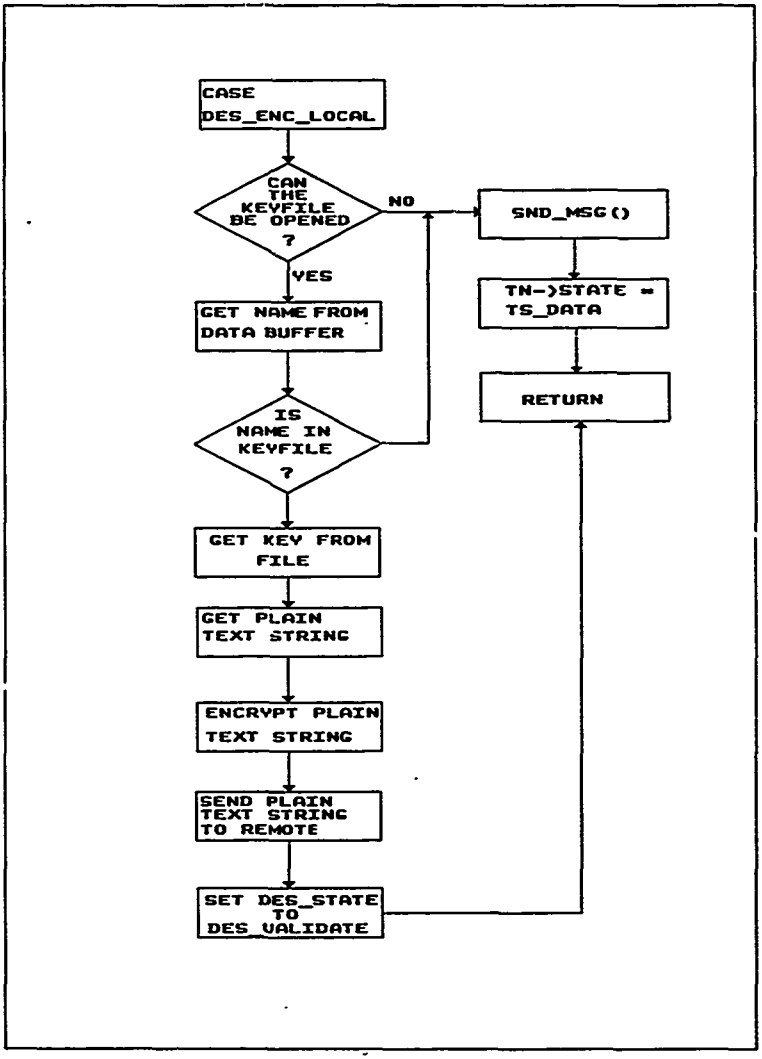
REMOTE MACHINE				MAINFRAME MACHINE	
	Current State	New State		Current State	New State
tcb->state tn->state des->state current_mach	estab TS_DATA none remote		- traffic -> <- traffic -		

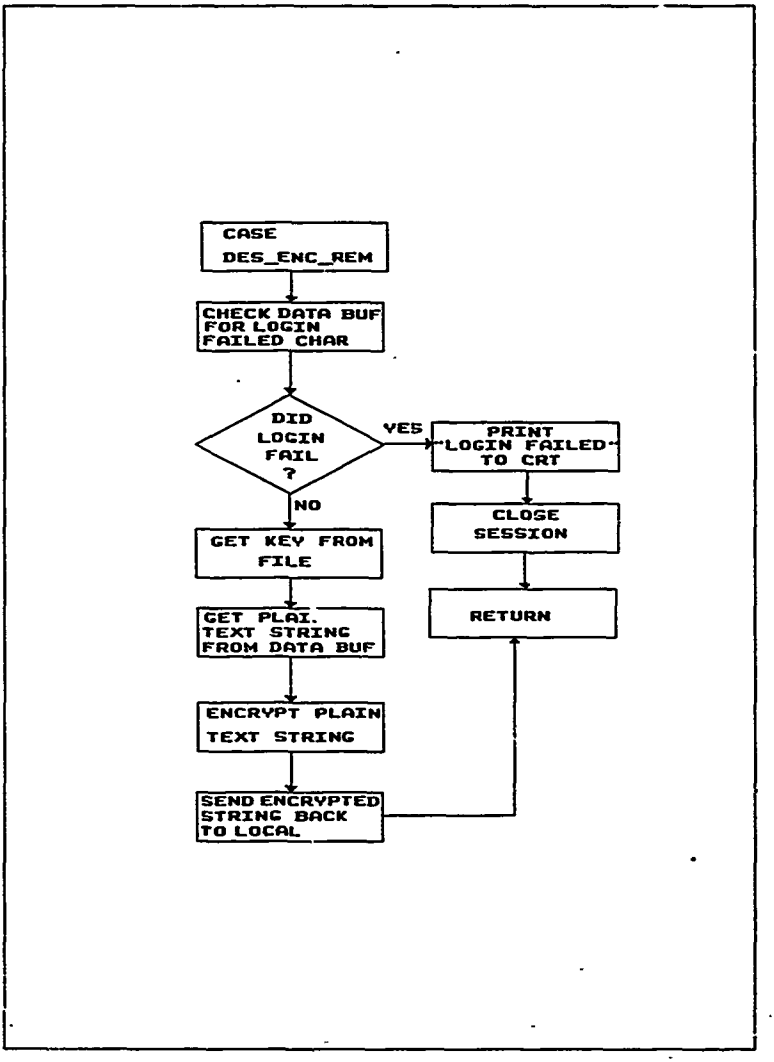
Appendix E. *Supporting Flowcharts for Modified Functions*

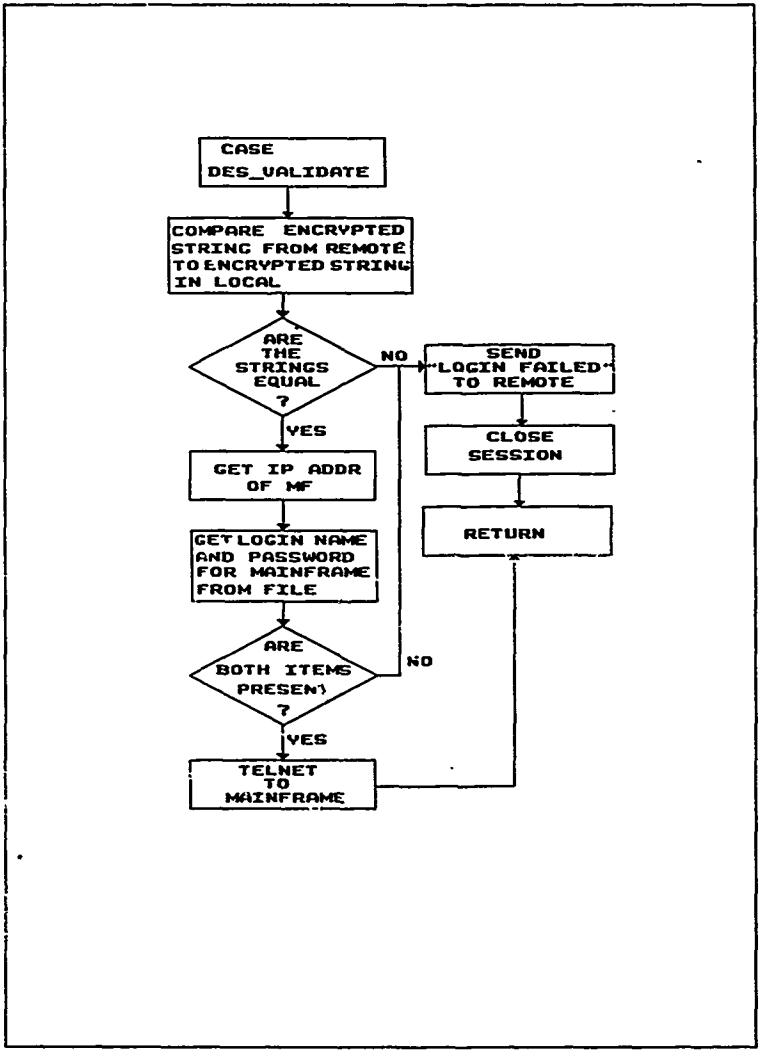

CASE
TS_DO:
CALL
DOOPT()
WHAT
OPTION
?
TN_DES
TN_ECHO
DEFAULT
HAS
OPTION
BEEN SEEN
BEFORE
?
NO
YES
TN->LOCAL
[UCHAR]OPT =1
ACK = WILL
TN->STATE =
TS_DES
DES_MODE = 1
CALL
ANSWER()
RETURN TO
TS_DO
IS
TS_STATE
=
TS_DES
?
YES
NO
DES_STATE =
DES_NAME
TN_STATE =
TS_DATA
CALL
TEL_DES()
RETURN


CASE
TS_WILL:
CALL
WILLOPT()
WHAT
OPTION
?
TN_TRANS_BIN
TN_ECHO
TN_SUPPR_GA
TN_DES
HAS
OPTION
BEEN SEEN
BEFORE
?
NO
YES
SET OPTION
SEEN TO 1
ACK = DO
TN->STATE =
TS_DES
CALL
ANSWER()
RETURN TO
TS_WILL
IS
TN->STATE
=
TS_DES
?
NO
TN->STATE =
TS_DATA
YES
DES_STATE =
DES_SEND_NAME


CASE
DES_SEND_NAME
WHICH
DRIVE IS
LOGIN NAME
ON
?
DRIVE A
HARD
DRIVE
SEND LOGIN
NAME TO LOCAL
STATION
DES_STATE =
DES_ENC_REM
RETURN


CASE
DES_ENC_LOCAL
CAN THE KEYFILE BE OPENED ?
NO
SND_MSG()
YES
TN->STATE = TS_DATA
GET NAME FROM DATA BUFFER
RETURN
IS NAME IN KEYFILE ?
GET KEY FROM FILE
GET PLAIN TEXT STRING
ENCRYPT PLAIN TEXT STRING
SEND PLAIN TEXT STRING TO REMOTE
SET DES_STATE TO DES_VALIDATE


CASE
DES_ENC_REM
CHECK DATA BUF FOR LOGIN FAILED CHAR
DID LOGIN FAIL ?
YES
PRINT "LOGIN FAILED" TO CRT
CLOSE SESSION
RETURN
NO
GET KEY FROM FILE
GET PLAI. TEXT STRING FROM DATA BUF
ENCRYPT PLAIN TEXT STRING
SEND ENCRYPTED STRING BACK TO LOCAL


CASE
DES_VALIDATE
COMPARE ENCRYPTED STRING FROM REMOTE TO ENCRYPTED STRING IN LOCAL
ARE THE STRINGS EQUAL ?
NO
SEND "LOGIN FAILED" TO REMOTE
CLOSE SESSION
RETURN
YES
GET IP ADDR OF MF
GET LOGIN NAME AND PASSWORD FOR MAINFRAME FROM FILE
ARE BOTH ITEMS PRESENT ?
NO
YES
TELNET TO MAINFRAME

Vita

Captain Tito N. Lebano was born on 14 November 1957 in Washington D.C. He graduated from high school in Bloomington, Indiana, in 1975 and attended Indiana University, from which he received the degree of Bachelor of Arts in Computer Science in May 1979. Upon graduation, he received a commission in the US Army through the ROTC program. Between June 1979 and December 1982 he was stationed at Ft Gordon, Georgia where he served as a staff officer for the Directorate of Training Developments and as a Training Officer in the 6th Advanced Individual Training Battalion. In late December 1982, he was reassigned to Ft Richardson, Alaska and was stationed there until May 1987. While at Ft Richardson, he served as the battalion Communications-Electronics Staff Officer for the 37th Field Artillery and as Company Commander for Bravo Company, 6th Signal Battalion, 6th Light Infantry Division. In June 1987, he entered the School of Engineering, Air Force Institute of Technology.

Permanent address: 528 Carters Grove Road
Centerville, Ohio 45459

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/88d-25	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology (AU) Wright-Patterson AFB, Ohio 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) DCLD Griffiss AFB, NY 13441-5700	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) A TCP/IP GATEWAY INTERCONNECTING AX.25 PACKET RADIO NETWORKS TO THE DEFENSE DATA NETWORK UNCLASSIFIED	
12. PERSONAL AUTHOR(S) Tito N. Lebano, CPT, USA	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) 1988 December 15. PAGE COUNT 104
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Communications Networks, Communication and Radio Systems, Computer Communications
12	07		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Advisor: Albert B. Garcia, Ph.D
Lieutenant Colonel, U.S. Army
Assistant Professor of Electrical
and Computer Engineering

Approved for release in Accordance with AFR 190-1
10 Jan. 1989

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Albert B. Garcia, LTC, USA	22b. TELEPHONE (Include Area Code) (513) 255-3576 22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Block 19 con't

Abstract

The packet radio, in recent years has been gaining popularity as a transmission medium for digital traffic. It provides its user with the capability of accessing computer resources while in a mobile configuration or in a remote location where access to cable based computer networks are either unreliable or non existant. As with any communications network, communications protocols must be in place to manage the transmission of data. Currently, the Transmission Contol Protocol (TCP) and Internet Protocol (IP) are used in the cable-based network, Department of Defense Network (DDN). With the capability that the packet radio provides, this effort's objective was to implement the TCP/IP protocols within a packet radio network and build a gateway through the Air Force Institute of Technology Network (AFITNET) to the DDN. Research for the project revealed that the software package, The KA9Q Internet Software Package, had already implemented TCP/IP and associated protocols such as Telnet and FTP for the packet radio. Unfortunately, when using the Telnet protocol within this package to communicate with a mainframe computer, passwords are transmitted in the clear. Any station monitoring the frequency can acquire a user's password to a particular mainframe. In order to protect the integrity of a user's computer resources, a method of a passwordless login is necessary. This effort undertook implementing the Data Encryption Standard (DES) within the KA9Q software, such that, encrypted strings are used to validate users. The DES process was implemented as a telnet option that is negotiated when the remote station requests it prior to telneting to the desired host. The source code for Telnet on the mainframe is proprietary so all modifications where made to the KA9Q package. The gateway consisted of an IBM AT connected to the AFITNET via an ethernet connection.