WN 00B0000039

WORKING NOTE

Performance and Setup Guide for the NOS TCP/IP Protocol Used on HF Near-Vertical-Incidence-Skywave (NVIS) Radio Paths

2/2000

Wickwire, Kenneth
Levreault, Robert
Bernock, Michael

Dept. No.: D620 **Project No.:** 03006113-GA and 03006113-MC

Approved for public release; distribution unlimited.

©2000 The MITRE Corporation

MITRE
Center for Air Force C2 Systems
Bedford, Massachusetts

Abstract

This note describes two years of testing of the Transmission Control Protocol and Internet Protocol (TCP/IP) protocol stack over near-vertical-incidence-skywave (NVIS) propagation paths in Massachusetts and New Hampshire. Results of the testing bear on the evaluation of a number of COTS and developmental systems that transfer data over high frequency (HF) radio.

Among these are systems being considered in “dual-use” HF radio configurations on board military aircraft being modified under the US Air Force Global Air Traffic Management (GATM) upgrade program. One use of such radio systems is communication of civilian air traffic control and “airline operational control” data over commercial networks. A second use is communication of encrypted military command and control data over purely military networks. Our results provide throughput and probability of correct message delivery baselines for both modes of use.

Our study also allows baseline performance comparison for systems that exchange data between tactical (“short-haul”) military users over beyond-line-of-sight radio links, and for systems that can communicate IP-addressable data among platforms being integrated into the worldwide US military Global Grid. In the last case, the TCP/IP protocol stack we have evaluated can itself be used in preliminary studies of direct transmission of TCP/IP datagrams over half-duplex radio.

To aid the use by others of the shareware stack we evaluated, we have expanded the scope of our report to include not only performance results but also *selected* information on how to set up and effectively operate the stack over error-prone radio links. The report is not, however, a comprehensive manual on configuration and operation of TCP/IP over radio.

NVIS paths are relatively short radio paths (ground distances from about 20 to 400 miles) used for tactical military communications and various civilian purposes. Difficult conditions are often encountered during NVIS communications. Among these conditions are multipath interference at sunrise and sundown, high D-layer signal absorption at midday and strong interference from long-distance broadcast stations at night. Protocol performance evaluations over NVIS links therefore provide conservative (i.e., lower-bound) assessments of the long-distance HF skywave communications normally associated with the concept of “shortwave” radio.

The tests are part of an on-going, multi-year program to assess and compare reliable, modern systems for communicating data over HF radio. Such systems comprise, at each station, an HF radio and antenna, a controller (usually a computer) that runs a user interface, and an HF modem that does waveform generation, signal processing and forward error correction. The software that implements a particular data transmission protocol runs either on the controller or in the modem or both.

“Reliable” in the sense of this report means that the protocol includes an automatic repeat request (ARQ) sub-protocol that causes a station receiving erroneous and uncorrectable data to ask the transmitting station to repeat it.

Although the standard TCP/IP stack was not developed for use on radio channels, its performance (in half-duplex modes) is still good enough for fielded military use and to serve as a baseline for the performance of (non-TCP) ARQ protocols that are tailored to HF channels. The TCP/IP stack used for these tests is the Net Operating System (NOS), a 16-bit, DOS-based stack developed by amateur radio operators and subsequently used commercially and by the military. By installing special software “shims” we have interfaced IP-based 32-bit Windows 95 applications such as E-mail clients and Worldwide Web (WWW) browsers with NOS and have sent E-mail and other TCP data over NVIS links using standard Windows 95 applications. This process is described in the note.

Because of their inability to distinguish between contention and channel errors (to be discussed below), standard (RFC-1122-compliant) TCP/IP stacks like NOS are ill suited to highly efficient use over radio channels. In spite of this, performance over HF links that is acceptable in many data-communication applications can be achieved by adapting the stack’s window- and segment-size parameters, and the HF modem’s data rate and interleaver settings, to noise and propagation conditions on the channel. Our experience with this approach, which is meant to defeat standard TCP’s unsuitable contention-control protocols, is also discussed in the note.

KEYWORDS: TCP/IP over HF Radio, IP-based Applications over HF Radio, Tactical Data Communications, Near-Vertical-Incidence-Skywave, NVIS, MIL-STD-188-110A Modems, Serial-tone Modems, NOS, Pinging, Radio Data Networks, SMTP, ICMP, FTP.

Acknowledgments

We are grateful to Terry Danielson of the Naval Ocean Systems Command (NOSC), for providing us with several of the synchronous DRSI I/O cards that drive a MIL-STD-188-110A serial-tone modem, and to Steve Barnett, also of NOSC, for information on setup of the cards.

Table of Contents

Section	Page
1. Introduction	**1**
2. Layout and Data Flow at our HF NVIS Stations	**5**
2.1 Hardware Layout	5
2.2 Data Flow	8
3. Setup of NOS for the NVIS Network	**9**
4. Description of the On-air Performance Testing	**13**
5. Description of NOS Operation over an HF Channel	**17**
6. Data Collection and Data Format for Point-to-Point Tests	**27**
7. Data Analysis Software and Analysis Output	**31**
8. Summary of Point-to-Point Performance Results	**35**
9. NOS Performance in a Network	**39**
10. Setup of Trumpet Winsock and the Pipe and Winpkt TSR Shims for 32-bit Operation via NOS	**47**
11. Addressing and Routing in an HF Network	**51**
12. Running 32-bit Applications over the HF NVIS Network	**53**
12.1 A PING Example	53
12.2 An FTP Example	55
12.3 An E-mail Example	60
13. Interfacing a NOS HF Network to a LAN	**65**
13.1 Connecting the HF Network to the Ethernet	65

14. Findings **69**
List of References **73**
Appendix A **75**
Appendix B **79**
Appendix C **81**
Glossary **85**

List of Figures

Figure	Page
1. Layout of Hardware at an HF NVIS TCP/IP Station	5
2. Data Flow at an HF NVIS TCP/IP Station	7
3. Geographical Layout of the Three HF NVIS Stations	13
4. Total Octets Sent and SRTT for an Easy FTP Transfer	23
5. Congestion Window and Slow-start Threshold for an Easy FTP Transfer	24
6. Total Octets Sent and SRTT for a Difficult FTP Transfer	25
7. Congestion Window and Slow-start Threshold for a Hard FTP Transfer	26
8. 32- and 16-Bit Data Flow at a Station in an IP-Based Radio Network	49

List of Tables

Table **Page**

1. Statistical Summary of HF NVIS SMTP Throughput Data 37

2. Statistical Summary of HF NVIS FTP Throughput Data 38

3. FTP Transfers from One Station to Two Other Stations 41

4. FTP Transfers Between Two Independent, Contending Pairs of Stations 43

5. FTP Transfers Through a Relay Station 44

Section 1

Introduction

This note documents a performance evaluation of the NOS TCP/IP protocol stack when it is used to send and receive text files over near-vertical-incidence-skywave (NVIS) HF radio propagation paths. This assessment bears on the evaluation of a number of COTS and developmental systems that transfer data over HF radio.

Among these are systems being considered in "dual-use" HF radio configurations on board military aircraft being modified under the US Air Force GATM upgrade program. One use of such radio systems is communication of civilian air traffic control and "airline operational control" data over commercial networks. A second use is communication of encrypted military command and control data over purely military networks. (NOS has been used by the US Navy and others with the KG-84C and KIV-7 devices to encrypt E-mail messages.) Our results provide throughput and probability of correct message delivery baselines for both modes of use.

Our study also allows baseline performance comparison for systems that exchange data between tactical ("short-haul") military users over beyond-line-of-sight radio links, and for systems that can communicate IP-addressable data among platforms being integrated into the worldwide US Military Global Grid. In the last case, the TCP/IP protocol stack we have evaluated can itself be used in preliminary studies of direct transmission of TCP/IP datagrams over half-duplex radio.

To aid the use by others of the shareware stack we evaluated, we have expanded the scope of our report to include not only performance results but also certain information on how to set up and effectively operate the stack over error-prone radio links. Although the report is not, therefore, a comprehensive manual on configuration and operation of TCP/IP over radio (no such manual will probably ever exist), we expect that some of the information in it will ease the sometimes painful process of setting up NOS and its variants for HF radio use.

The NVIS radio paths connect stations between 30 and 60 miles apart located in Massachusetts and New Hampshire. The text files, of various sizes, were sent in both compressed and uncompressed format from the standard TCP/IP-based Simple Mail Transfer Protocol (SMTP) and File Transfer Protocol (FTP) applications.

NVIS paths are relatively short paths (linking stations from 20 to 400 miles apart) that are used extensively for tactical military and various civilian communications. NVIS communications require antennas that can launch energy at high takeoff angles (60 or more degrees). Among the types of antennas that can do this are dipoles and sloping longwires.

NVIS communications occupy a transitional place on the range of radio communication modes running between line-of-sight and long-distance (one or more hop) skywave communications. Difficult conditions are often encountered during NVIS communications. (Among these are multipath interference at sunrise and sunset, high D-layer signal absorption at midday and strong interference from long-distance broadcast stations at night.) Protocol performance evaluations over NVIS links provide conservative (i.e., lower-bound) assessments of the usually longer-distance communications normally associated with the notion of "shortwave" or "skywave" radio, since properly chosen longer-distance modes are less affected by these phenomena than NVIS.

In addition to providing conservative performance estimates, NVIS networks such as ours provide a crucial additional advantage: they are easier to set up and control than networks with stations hundreds of miles apart that demand time-consuming and expensive travel or coordination to keep them on the air for the extended periods that realistic HF testing demands.

These NOS tests are part of an on-going, multi-year program to assess and compare the performance of reliable, modern systems for communicating data over HF radio. Such systems comprise, at each station, an HF radio and antenna, a controller (usually a computer) that runs a user interface, and an HF modem that does waveform generation, signal processing and forward error correction. The software that implements a particular data transmission protocol runs either on the controller or in the modem or both.

"Reliable" in the sense of our assessment program means that a protocol includes an ARQ sub-protocol that causes a station receiving erroneous data to ask the transmitting station to repeat it.

The performance of a data communications system used over radio links includes a wide range of attributes. Among these are

- Throughput,
- Probability of successful message file transfer,
- Day versus night differences (often pronounced in HF communications),
- User-friendliness of the operating interface,
- Ease of setup, learning and use,
- Interface to Local-area networks (LAN) and standard IP-based E-mail clients or servers,
- Usability with a variety of PC-controller operating systems and
- Usability and interface with cryptographic equipment.

Users often emphasize performance measures like these differently, so no fixed order of importance can be assigned to them. However, in this note we shall emphasize throughput and probability of successful message file transfer as a basis for comparisons of NOS point-to-point performance with that of other HF protocols like CLOVER-2000, FED-STD-1052, NATO STANAG 5066 and the Space Communications Protocol Specification (SCPS) modification of TCP/IP for satellite and other radio channels. Comments on other aspects of TCP/IP performance with NOS and the MIL-STD-188-110A serial-tone modem will be made throughout the report.

We shall show that although the standard TCP/IP stack is not ideally suited for use with radio channels, its performance is still good enough to serve as a baseline for assessing the performance of (non-TCP) ARQ protocols, such as those just cited, that are tailored to HF channels. The standard stack has the further advantage that it allows TCP/IP datagrams (data frames plus headers) to be passed directly to and from TCP-based applications (mail clients, browsers, etc.) without the need for translation into and out of a proprietary frame format especially designed for communication over radio channels.

The NOS stack used for our tests is a 16-bit DOS shareware suite developed by amateur radio operators about ten years ago. Since that time, the stack, whose C source code is freely available, has been developed further and has been used both commercially and by the US Navy, the US Army Corps of Engineers, several NATO navies, the Australian Armed Forces and several South American countries involved in projects with the US Government.

The NOS version we used, called JNOS after the name of its first developer (Johan Reinalda), conforms to the Internet Request for Comments (RFC) 1122, the specification for "standard" TCP/IP-stack operation. NOS can be used to transfer data from most popular TCP/IP applications, such as the internet control message protocol (ICMP), the FTP, SMTP and the hyper-text transfer protocol (HTTP). It can also act as an IP-switch and an IP-router and can thus directly transfer TCP/IP data between radio networks and wireline or wireless LANs.

Although NOS is a 16-bit program, we employed special software "shims" to interface 32-bit IP-based Windows 95 applications such as E-mail clients and Worldwide Web browsers with it, and have sent E-mail, etc., by that means over our NVIS links using 32-bit clients and servers. This process and its setup are described in the note.

Because of their inability to distinguish between contention and channel errors (a phenomenon we shall discuss below), standard (i.e., no more than RFC-1122-compliant) TCP/IP stacks like NOS are inherently unsuited to highly efficient use over radio channels. In spite of this, what is often *acceptable* performance over HF links can be achieved with a standard TCP/IP stack by adapting the stack's window- and segment-size parameters, and the HF modem's data rate and interleaver settings, to changing noise and propagation conditions on

the channel. Our experience with this approach, which has involved manual adjustment of such parameters between file transfers, has shown that it offsets the effects of standard TCP's contention-control protocols to some extent. This is also discussed further in the note.

The rest of the paper covers the following topics: layout and data flow at our HF NVIS stations, setup of NOS for the NVIS network, description of the on-air performance testing, description of NOS operation over an HF channel, point-to-point data collection and format, software for point-to-point data analysis and analysis output, summary of point-to-point performance, NOS operations in a network of more than two stations, setup of Trumpet Winsock and the pipe and winpkt terminate-and-stay-resident (TSR) shims for 32-bit Operation via NOS, addressing and routing in an HF sub-network, running 32-bit applications over an HF NVIS network, interfacing a NOS HF network to a LAN and a summary of our findings.

Section 5 contains relatively detailed material on how TCP operates as a protocol in the presence of contention and channel errors. We have included it as a guide for potential users and others who may have found it hard to locate such an account. Other readers may skip this material without losing the thread of the narrative. Readers interested in performance (throughput and probability of correct message reception) should read Sections 8, 9 and 14 (Findings). Finally, potential NOS users interested in running familiar 32-bit applications over radio via 16-bit NOS can read about how we accomplished that in Section 10.

Section 2

Layout and Data Flow at our HF NVIS Stations

2.1 Hardware Layout

Each of the three stations used in most of our tests of TCP/IP over NVIS links was assembled from the following hardware components (see Figure 1):

- An HF radio,
- A wire antenna (dipole or longwire) and associated RF cables,
- A Rockwell MDM-3001 or Harris RF-5710 MIL-STD-188-110A serial-tone HF modem and associated audio and data cables,
- A 90 MHz or faster PC running the Windows 95 operating system and the 16-bit JNOS TCP/IP stack software in a DOS window and
- A custom-made DRSI synchronous I/O serial interface card (provided to us by the US Navy Ocean Systems Command) installed in an ISA slot of the windows PC.

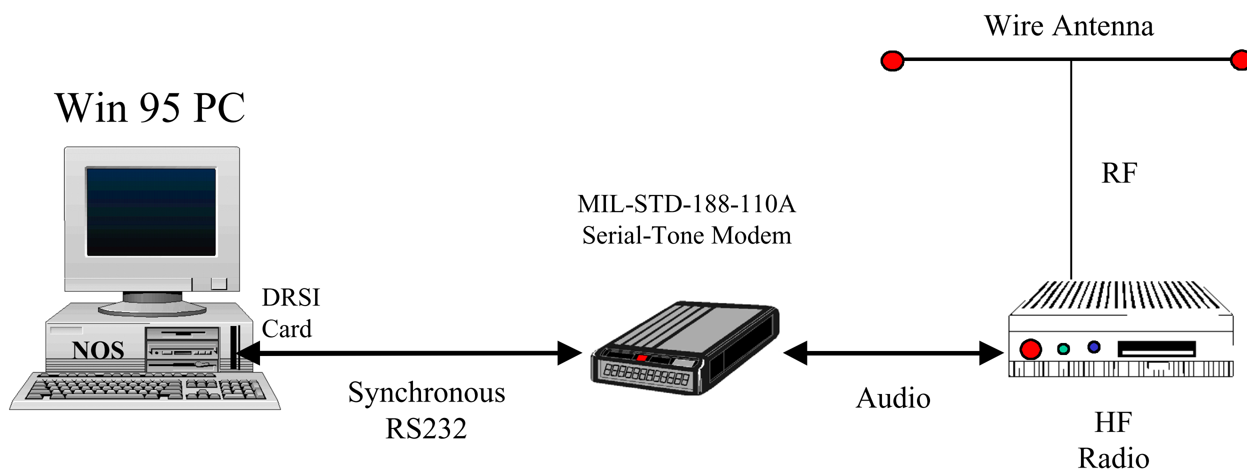

Figure 1. Layout of Hardware at an HF NVIS TCP/IP Station

2.2 Data Flow

The flow of data at an operating HF TCP/IP station is sketched in Figure 2. The basic unit of data, sent from station to station in the NVIS network is a TCP/IP "datagram," encapsulated in an AX.25 link-layer packet. The datagram consists of a twenty-byte IP header, a twenty-byte TCP header (for a total of forty header-bytes) and a block of data bytes, called a "segment," whose size is determined by software configuration parameters and the operation of the TCP/IP protocol itself. The AX.25 packet contains To and From link-layer addresses (radio callsigns), start- and stop-flags, the TCP/IP datagram, cyclic redundancy check (CRC) bytes used for the data-integrity checking performed by the AX.25 link layer and various control settings. The datagram headers contain IP To- and From-addresses, various control option settings and CRC bytes used for the data-integrity checking performed by the transport (TCP) layer of a standard stack. The datagrams contain data sent to or from applications such as E-mail clients, FTP clients or Web browsers.

Outgoing TCP/IP datagrams (whose data contents originate in a DOS or Windows 95 application that uses TCP/IP) are assembled by a TCP/IP stack (in our case, NOS or Trumpet Winsock) and passed to the DRSI card as AX.25 frames. These frames then go as synchronous data to the serial-tone modem. The modem formats the resulting data stream according to the framing, frame-synchronizing and equalizer-probe specifications of the MIL-STD-188–110A waveform and modem specification. Note that in data-transmission systems based on NOS, the modem performs primarily link- and physical-layer functions. Data processing needed to run network-, transport- and application-layer protocols takes place in the Data terminal equipment (DTE, usually a computer) that runs NOS, rather than in the modem, as in some other systems (e.g., CLOVER).

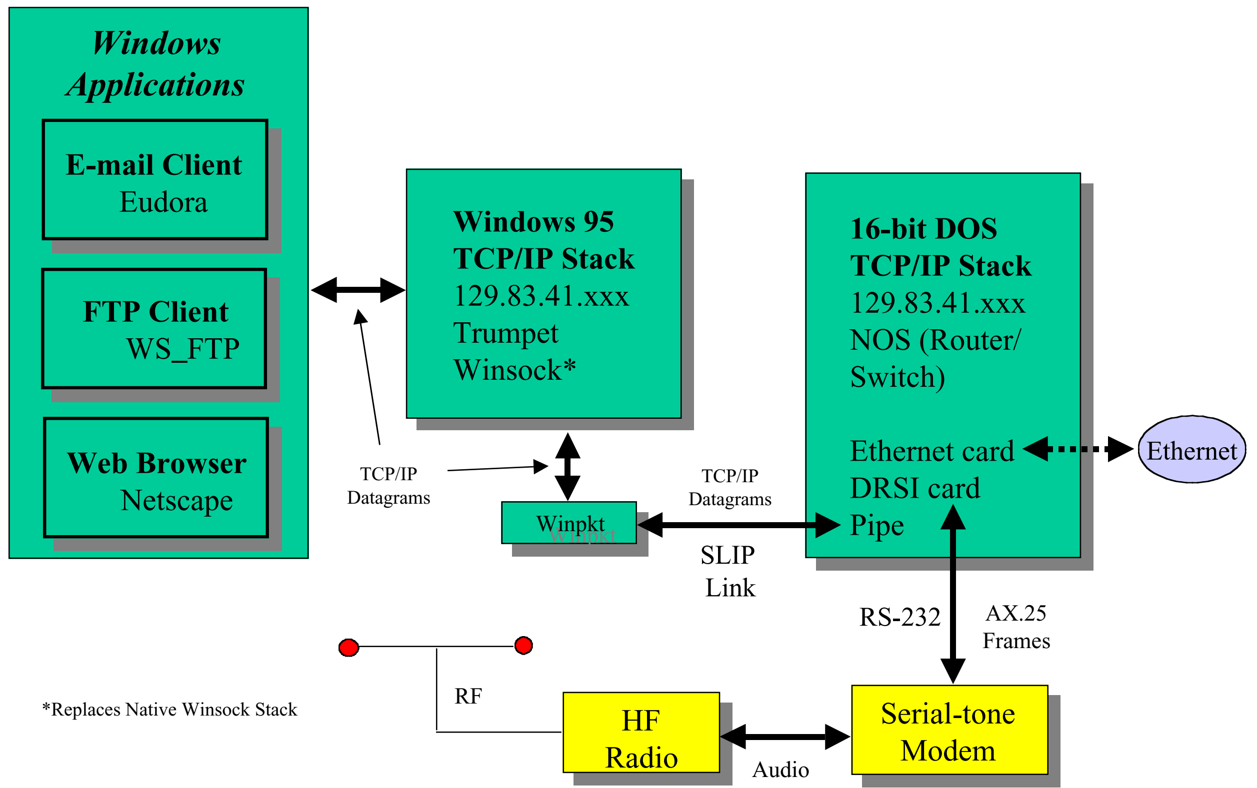

Figure 2. Data Flow at an HF NVIS TCP/IP Station

Section 3

Setup of NOS for the NVIS Network

The NOS TCP/IP software suite comprises a 16-bit DOS executable (binary) file accompanied by a number of setup files that the executable file reads at start up or during its operation. The NOS package is installed by simply copying it and its setup files to a desired directory on the PC-controller's hard disk. After installation, NOS can run in a DOS window and is launched by double clicking with a mouse on its icon from the Windows 95/98 graphical user interface (GUI).

It is important to realize that running NOS with a piece of hardware (an Ethernet card or a modem, for example) usually requires installation of driver software used in conjunction with NOS. In our case, the hardware in question is a MIL-STD-188-110A HF serial-tone modem, and the driver software for the DRSI card (drsi.c and the accompanying header file drsi.h) that we use came with the HF version of JNOS we obtained from the Navy. We have not investigated what it would take to incorporate the DRSI drivers (and perhaps other software) associated with use of a serial–tone modem into another version of NOS.

Basic setup of NOS is accomplished by editing an American standard code for information interchange (ASCII) text file called *autoexec.net*. For each of our stations, this file contains about 140 lines. Each line states the name of a command or parameter followed by its setting. An annotated copy of one of our HF autoexec files (for station MB1 in Bedford, Mass.) is shown in Appendix A. For a good tutorial on the content and set up of an autoexec file for NOS see References 3 and 4.

NOS opens and reads the autoexec file at startup to set its operating parameters. After startup, some of the parameters can also be changed on the fly from the NOS command line, although the changes don't take effect until a message transfer has finished. The settings, some of which we will discuss further below, can be divided into four general categories:

- Callsign or address assignment commands,
- Interface configuration commands,
- Routing and address notification commands and
- Miscellaneous setup and activation commands.

Examples of settings in the first category are those of a station's IP address and host name:

```
ip address   44.56.8.103
hostname     mb1.mbpr.org
```

TCP/IP stacks normally use IP addresses (four sets of up to three integers separated by periods) to establish and keep track of connections with each other, but using IP addresses as callsigns causes obvious inconveniences to users, who prefer callsigns they can remember. Mappings between callsigns (which are not case-sensitive) and IP addresses are therefore kept in *host tables*, which are ASCII files of host-name-to-IP-address correspondences that are accessible to NOS. In normal NOS operations, a calling station enters a receiving station's callsign at a user interface and NOS then consults its local host table to retrieve the IP address corresponding to the callsign.

The host names for this study were chosen as combinations of callsigns ("mb1" = MITRE Bedford 1, etc.) and domain names ("mbpr.org" for MITRE Bedford packet radio.org). Because we sometimes also use NOS for testing on amateur radio frequencies in the amateur packet radio network, whose domain name is ampr.org, and find it convenient to use the same host tables for all of our testing, we have found it necessary to use "full host names" as callsigns. Hence, rather than calling the Derry station with the callsign *der*, the usual practice, we call it with the callsign (full host name) der.mbpr.org.

Turning next to interface configuration, consider the command

attach drsi 0x310 5 ax25 dr0 2700 2600 2400 2400

This command "attaches" and configures the synchronous serial interface between NOS and the DRSI I/O card that passes both flow-control commands and message data between the NOS stack and the serial-tone modem (and from there to and from a radio). The interface parameters in this case are the card's I/O address (0x310) and interrupt number (5), the protocol used on the interface (AX.25, a "packet radio" variant of X.25), the interface label (dr0), the receiving buffer size in bytes (2700), the so-called maximum transmission unit (MTU) in bytes for TCP datagrams ("packets") sent over the interface (2600) and the data transmission rates for the two channels the card can process (2400 bps; in our case, we use only one channel).

As an example of a routing and address notification command, consider

route add 44.56.4.57 dr0

This command tells NOS that all datagram traffic intended for the station with IP address 44.56.4.57 is to be routed to the dr0 interface and from there to the serial-tone modem and the radio.

A miscellaneous setup command is

ifconfig dr0 tcp irtt   15000

This command specifies (configures the interface for) the "initial round-trip time" (IRTT) in milliseconds (ms) for TCP datagrams sent to another station via to dr0 interface. The IRTT is used by a collision-control algorithm built into TCP that decides when to re-send a datagram that has not been acknowledged in timely fashion owing—presumably—to a collision between that datagram and one sent at the same time by another station. (On radio links, noise, multipath, or interference can also cause a missed acknowledgment.) Collision control and its effect on throughput will be discussed below.

Other miscellaneous commands set up mailboxes, turn on the built-in FTP or mail server, and so on. See References 3 and 4 for more discussion of NOS commands, since we have only scratched the surface here.

As will be evident from this section and the discussion below of the setup of the Trumpet Winsock stack and the pipe and winpkt shims, correct configuration of NOS and the helper programs needed to make it operate with the 32-bit Trumpet Winsock stack on the one hand, and with a network of other TCP-equipped radio stations on the other, is complicated. Part of this complication comes from the fact that NOS can be used as a network server, rather than just a network client. The intricacies of correct server configuration are well known to anyone who has administered a Windows NT or similar server.

Most of the rest of the setup complications arise from our use of the standard TCP/IP stack in a radio network over half-duplex channels with rapidly and randomly changing propagation conditions on them. These changing conditions cause bit errors in received datagrams that "confuse" the standard stack's collision control mechanisms, causing them to react to a datagram unacknowledged because of a channel error as if the failed acknowledgment had been caused by a collision.

Thus, instead of trying to re-send the datagram as soon as the failed acknowledgment is noted, which would generally be the best strategy, the stack's algorithms invoke a "back off," causing the transport layer to wait longer to re-send the datagram than it would have in the absence of a channel-induced acknowledgment failure. This is a sensible reaction to congestion, but not to channel errors. Coping with this behavior in a way that leads to high throughput when there are channel errors requires a certain amount of dynamic and creative TCP parameter adjustment between message or file transfers. We will have much more to say about this adjustment process below.

Section 4

Description of the On-air Performance Testing

Most of our testing involved three stations, one each in Bedford and Norfolk, Massachusetts, and a third in Derry, New Hampshire. The stations lie roughly on a line running north-south, with Derry (42°53¨N, 71°16¨W) in the north, Norfolk (42°07¨N, 71°20¨W) in the south and Bedford (42°20¨N, 71°25¨W) in the middle, about 30 miles from both Derry and Norfolk. The configuration is shown in Figure 3. Most of our tests involved point-to-point (two-station) testing. Roughly a third of the point-to-point tests were transmissions from Bedford to Derry, a third from Bedford to Norfolk and the rest over the 60-mile ground path from Derry to Norfolk.

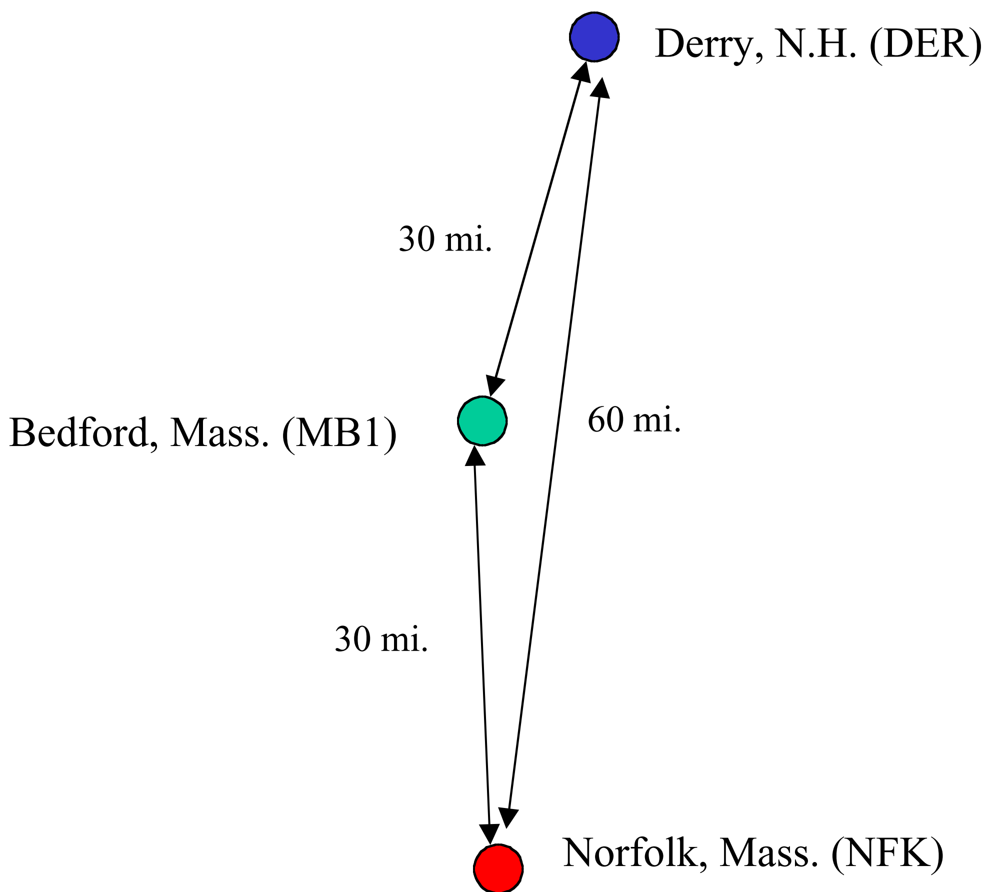

Figure 3. Geographical Layout of the Three HF NVIS Stations

We also ran a few point-to-multipoint tests, a few pairs of simultaneous point-to-point tests (to assess performance in networked and contention environments) and a few tests of file transfer through a relay. These multipoint tests produced only small sample sizes and the data (discussed in Section 9) were not submitted to the statistical analysis described below.

The three stations used 100-to-125-watt military (Harris RF-5000 and RF-280) and ham radios (Yaesu FT-1000MPs), with most tests using the military radios. Antennas were dipoles or sloping, resistively terminated longwires. Both types are appropriate for NVIS operation, which requires that energy be radiated at high (~70-90°) launch angles. The serial-tone modems were Rockwell-Collins MDM-3001s and Harris RF-5710s.

The tests ran from January 1998 to the fall of 1999 and thus covered about seven changes of season during a period of rising sunspot number, which lay between 30 to 130. Maximum Usable Frequencies (MUFs) on the links were between about 2 and 9 megahertz (MHz) during the testing period and we chose our operating frequencies from an available set covering 2 to 8 MHz.

With the exception of some tests at around 5 MHz toward the end of the period, all the tests used frequencies (most around 3 MHz) that were of only average quality from the standpoint of noise, interference and distance from the so–called optimal operating frequency (about 80 percent of the MUF). This means that our results reflect conservative performance. An automatic link establishment (ALE) system or a judicious choice of frequencies from a set free of interference (a great luxury for most users) might have produced average throughput perhaps 20 or 30 percent higher than ours.

The point-to-point tests covered both day- and nighttime operating conditions, including mid-day absorption by the D-layer, multipath fading at sunrise and sunset, thunderstorm noise and sporadic-E-layer propagation during the summer, and interference from broadcast stations at night. Because the data transfer process involved manual parameter adjustments between file transfers, and could thus not be automatically scheduled, few if any tests were run between local midnight and six in the morning local time. However, numerous tests with similar protocols that do allow scheduled data transfers show that because of increased levels of broadcast interference between local midnight and local sunrise, and occasional operation too close to the MUF, throughput between those times is often 10 or 20 percent lower than in the evening. This means that our nighttime results are probably slightly better than those derived from all-night operation.

The point-to-point throughput tests involved compressed and uncompressed text-message transfers using the standard SMTP application implemented by NOS and compressed and uncompressed text-file transfers using the standard FTP that NOS also implements. In both cases, the dictionary-based Lempel-Ziv-Welch (LZW) algorithm built into NOS carried out compression and decompression. The parameters of the algorithm were set for maximum compression (with the commands “LZW bits 16” and “LZW mode compact”).

In the case of SMTP ("E-mail") message transfers, compression was turned on and off with the NOS "smtp sendlzw on/off" command. In the case of FTP file transfers, compression was turned on and off by sending files in the ASCII mode (i.e., compressed) with the NOS command "type a" (sent to the receiving station at the start of an FTP session), or by sending files in "binary mode" with the command "type i" sent at the corresponding time. (The "i" here stands for "image" and comes from earlier days when graphics files were apparently the most common types of binary file sent with FTP.) Compression is accomplished in the ASCII mode because our version of NOS automatically compresses text files sent in that mode.

After a successful FTP transfer, NOS reports the transfer time (in seconds) to the screen. We entered this time and other particulars of point-to-point transfers (see below) into an ASCII archive file for later analysis. If a transfer was unsuccessful (NOS timed out), this was also recorded in the archive file. After a successful SMTP message transfer, NOS does not report the transfer time or throughput. To get this time we either monitored the "TCP status" or turned on the "trace" utility, which shows the size and time of arrival of datagrams along with the contents of datagrams whose content is associated with the beginning ("SYN," etc.) and end of a transfer ("FIN WAIT," etc.). The corresponding SMTP transfer time (in seconds) was then entered into the archive file as above.

We normally started each experimental point-to-point session with conservative parameter settings (TCP segment size and modem data rate). We found that a segment size of 216 octets and a modem data rate of 1200 bps were often good starting values. (With our manual adaptation technique it rarely seemed to be effective to use the modem's long interleaver setting, but the interplay of data rate and interleaver depth in our approach is a topic that deserves much more study than we were able to give it.)

If the throughput for the first few transfers of a session indicated that the channel conditions were good (often in mid-morning and late afternoon when absorption and noise were relatively low, or the operating frequency was near the optimal one, or some combination of the three), we raised the datagram size. If this led to higher throughput, we raised the modem data rate. If, on the other hand, the initial results were poor, we reversed the process, lowering the datagram size and then, if necessary, the data rate or even the interleaver depth until we got what appeared to be the best results achievable in the prevailing channel conditions.

We generally kept the window size fixed at 2592 bytes at Bedford and Norfolk and 1728 bytes at Derry, and chose our segment (frame) sizes in point-to-point tests to be factors of the window size. (See the next section for the definitions of a window and segment.) According to the TCP protocol, this generally caused NOS to send integer numbers of segments, up to the number of segments comprised by a window. To send traffic to stations with window size 2592 (= 12 x 216), we generally chose segment sizes of 216, 324, 432, 864 or 1296 octets (bytes). To send to Derry (window size 1728 [= 8 x 216]), we generally chose segment sizes of 216, 432, 576 or 864 octets.

These window sizes allowed the standard TCP protocol implemented in NOS to vary the number of segments sent at one time from between one and twelve in the case of Bedford and Norfolk and between one and eight in the case of messages sent to Derry. (To take advantage of the overhead savings that arise from TCP's ability to ACK several datagrams with a single transmission, we usually chose the MSS to be an integer divisor of the window size.)

Sent messages generally comprised between 10k and 50k bytes of ASCII text, with the most common size 20 or 30k. (Messages 20 to 30k long—five to eight pages of text—take around six minutes to send on average.) This range of sizes allows a nearly optimal tradeoff between a small ratio of overhead to information content in transferred files and time spent sending each file. The table below lists the average throughput in octets per second for uncompressed files with sizes between Nk and (N.1)k octets, N = 10, 20, 30, 40 and 50, transferred by FTP during the daytime. The "SS" values in brackets are the sample sizes on which the average throughputs are based.

The table suggests that file sizes from 20 to 40k produce roughly the same throughput and that the throughput does not rise significantly as the file size increases above that range. This leveling-off of average throughput as file size increases is a general property of HF ARQ protocols and the leveling-off appears to occur at roughly the same size for all modern HF protocols.

File Size (octets)	E(tput) [SS] (octets/s)
5-5.1k	40 [24]
10-10.1k	49 [65]
20-20.1k	54 [46]
30-30.1k	64 [28]
40-40.1k	58 [22]
50-50.1k	64 [33]

Section 5

Description of NOS Operation Over an HF Channel

Most of our testing involved point-to–point (two-station) links on which all files or messages were sent in the datagram mode. In this mode, flow control, frame (datagram) acknowledgments and reaction to collisions are handled only by the TCP layer (the fourth or transport layer in Open Systems Interconnection [OSI] terms) of the NOS stack.

TCP in the standard stack is known as a "sliding-window protocol without negative or selective acknowledgments." A sliding window is the means used by TCP for **flow control**, and it allows the protocol to acknowledge more than one segment at a time. This lowers overhead and generally raises throughput when the channel supports it.

The amount of data that can be sent at a particular time may be viewed as occupying a window whose varying size is "advertised" or "offered" to the sender by the receiver in each of the receiver's acknowledgment packets. (TCP segments are acknowledged by giving the sequence number of the most recently received byte rather than by giving a segment number, but the effect is the same.) This advertised size depends on how much data the receiver has accepted as correct and how much space is available in its buffers. As a transfer proceeds, the left part of the window contains sent but unacknowledged data, and the right part contains data that can be sent. As data are acknowledged, the left edge of the window moves, in effect, to the right, "closing the window." When more data are sent, the right edge of the window moves to the right, "re-opening" the window. The window thus slides (albeit in a jerky way) across the data to be sent until the transfer is done.

The maximum window size in bytes that can be advertised and therefore sent is specified by the WIN parameter in the receiver's autoexec file. WIN should be picked as the greatest amount of data the receiver thinks can be sent *in good channel conditions* with a high probability of being ACKed on the first try. (The largest window size allowable in standard TCP is 65535 bytes.) When conditions are in fact good and the receiver offers WIN bytes, the sender can send as many segments as can fit into those WIN bytes. As mentioned in Section 4, to ensure that an integral number of segments get sent (avoiding inefficient fragmentation of segments into packets smaller than the channel can support), we generally chose our maximum segment sizes (MSSs) at sending stations dynamically (between transfers) from the set {216, 324, 432, 864 or 1296} for the station pair (Bedford and Norfolk) that set their WINs to 2592 and from the set {216, 432, 576 or 864} for stations sending to Derry, where the WIN was normally set to 1728.

Stations in our network that were configured to *receive* FTP or SMTP traffic normally set their MSS at half their window size (that is, an MSS of 1296 at Norfolk and Bedford and an MSS of 864 at Derry). This allowed a sending station to choose its MSS as any size from one of the two sets above, (depending on who was receiving) according to perceived channel conditions. Note that this was done *manually* in our experiments. In a future version of NOS this could and should be done automatically in response to some measure of current channel conditions.

If segments in a window arrive with bit errors the receiver can't correct, the receiver will drop such segments and will ACK only the range of sequence numbers received correctly without gaps. This means that all segments in a window that arrive after a dropped segment will be discarded. This describes the behavior of TCP's "go-back-N" ARQ protocol, which is one of the properties of standard TCP that makes it less well suited to use on HF or other radio channels than protocols that use a "selective acknowledgment" or "negative acknowledgment" ARQ protocol.

Under a go-back-N protocol, if an early segment in a window containing many segments is lost, then all of the segments following the lost segment are thrown away and must be resent, causing throughput to fall drastically. A protocol that asks for repeats of only bad segments is obviously more efficient than this one. A great deal of thought is now being devoted to producing versions of TCP whose ARQ scheme is better suited to use with radio channels than the standard scheme.

Although entire windows of segments can be sent at one time in good conditions, TCP takes a conservative approach to ramping up to such behavior. This approach is called **slow start**. In this technique, the number of segments that is sent at one time is dictated by the rate at which segments have been ACKed. This rate is determined by a parameter called the Congestion Window (*cwind*) that is dynamically adjusted by TCP. The Congestion Window helps determine the amount of data the sender can transmit at any one time, the *allowed window*, according to the formula

allowed_window = min(cwind, advertised_window).

In the absence of congestion, the Congestion Window is the same as the window advertised by the receiver, but when the Congestion Window is lowered far enough by perceived collisions, the formula says that fewer datagrams are transmitted over the connection than otherwise.

Every transfer starts with the *cwind* set to the size of one segment (as specified by the MSS). Every time a segment is ACKed thereafter, the *cwind* is increased by MSS bytes, which leads to a geometric (exponential) increase in the amount of sent data, limited ultimately by the

offered WIN. Thus, at any time the sender can send up to the minimum of the offered window and the Congestion Window. This can be viewed as flow control imposed by the *sender* in reaction to what TCP views as congestion, whereas the offered window reflects flow control imposed by the *receiver* in reaction to the size of its buffer space.

Sooner or later during many transfers over HF channels, one or more packets suffer uncorrectable errors and fail to be acknowledged (a sender timeout) or the receiver times out waiting for more data and re-sends an ACK. Standard TCP interprets these acknowledgment failures or duplications as having been caused by congestion (a collision with packets sent by another user), and deals with them by invoking a **congestion avoidance** procedure.

Under this procedure, the first thing that happens after recognition of a likely acknowledgment failure is that roughly one-half the current *cwind* (but at least 2 x MSS) is saved as a variable called the "slow-start threshold" (*ssthresh*). Then the *cwind* is lowered to MSS octets. After this, a slow start begins again, but instead of exponentially increasing the amount of data sent until WIN is reached, the procedure increases it exponentially only until *ssthresh* octets are reached. Thereafter, in the absence of failed ACKs, the *cwind* increases only *additively* by one MSS worth of octets per round-trip time. The next time a packet is lost the whole procedure is repeated starting with a new, lower *ssthresh* and a new slow-start phase.

Adjustment of the *cwind* and *ssthresh* is in practice even a bit more complicated than we have described it and depends among other things on whether the sender times out waiting for an ACK or the receiver times out waiting for more data. Both parameters can, during the course of a data transmission, increase or decrease at almost any stage, but the description above captures the essence of their adjustment. Appendix C lists pseudo-code for the actual algorithms used by NOS to adjust the congestion window and slow-start threshold in response to missed packets or duplicate ACKs.

The division of the rate of data increase after a lost packet into two phases, the (somewhat confusingly named) slow start exponential-growth phase, and the linear-growth congestion-avoidance phase, make up standard TCP's approach to countering *all* packet loss, whether it's caused by congestion or channel effects. When the loss is really caused by congestion, the adaptive throttling-down of the sending rate through changes in the *cwind* and *ssthresh* makes sense: the more congestion, the less aggressive the rate of segment injection into the channel.

However, when the loss is caused by bit errors, this approach is generally not what should be taken; what is better is to re-inject a segment as soon as its loss is detected. (Some of the recent work to improve TCP's performance on radio links involves fine-tuning of the algorithm that adjusts the slow-start threshold and changing the "default" assumption that all packet loss is caused by collisions.) Coping with the undesirable effects of TCP's default collision avoidance algorithms to the extent allowed by the standard stack requires an understanding of TCP's **retry strategy.**

As a preface to discussion of this strategy we turn first to an important aspect of TCP's operation in lossy (i.e., error-prone) radio channels: its **estimation of round-trip times** (RTTs). The estimated RTT, also known as the smoothed RTT (SRTT), is used in an algorithm that determines the retry strategy when frames fail to be acknowledged. (TCP acknowledges only data-segments and not ACKs themselves.) RTT estimation is important because estimated RTTs are used to decide when a frame has been lost (has failed to be ACKed) and needs to be re-sent. If the estimated RTT is too large, packet loss will be detected late and throughput in a lossy channel will fall. On the other hand, if the estimated RTT is too small, then packets that have not been lost will be re-sent and multiple ACKs will be sent by the receiver. This can have a catastrophic effect on performance. The estimate should therefore never be too low and should not be much higher than the real (but unknown) RTT on any one transmission.

Following much discussion and experimentation, the Internet community (Jacobson, Karn and others) devised an algorithm (now in RFC 793) for estimating RTTs that takes into account previous measurements of RTTs, the most recent RTT measurement and the variability of previous RTT measurements (see References 1 and 2). The algorithm first calculates a smoothed average ($SRTT_{new}$) of the most recent RTT (the MRTT) and the previous average ($SRTT_{old}$) according to the formula (sometimes called a "low-pass filter")

$$SRTT_{new} = \alpha\, SRTT_{old} + (1 - \alpha)\, MRTT,$$

where α, a *decay constant*, is usually set to about 0.9. This estimate is updated with each new RTT measurement. When TCP is first launched, the algorithm is initialized by setting $SRTT_{old}$ = IRTT, where IRTT, the *initial round-trip time*, is specified for NOS in the autoexec.net file with the command "ifconfig dr0 tcp irtt," where irtt is in milliseconds. (We took IRTT = 15000, or 15 s.) If TCP is running and has already estimated the SRTT during previous on-air activity, the initial round-trip time and its initial variability are taken as the most recent estimates. Re-initialization of the SRTT can be performed between transfers from the NOS command line.

The RTT estimate, called the "Retransmission Time-Out" (RTO) value, is then taken as $\beta\, SRTT_{new}$, where $\beta > 1$ is a function of a smoothed estimate of RTT variability. Taking $\beta > 1$ is intended to prevent under-estimation of the RTT, which is generally more serious than (a modest) over-estimation. Poor performance with early versions of TCP caused by excessive variability in, and ambiguity of, round-trip times was removed to a large extent by updating the SRTT only with round-trip times for new (rather than retransmitted) packets. This is called **Karn's algorithm**, after Phil Karn, an amateur radio operator who wrote the first version of NOS.

If a data segment is successfully sent but its ACK is lost, it could happen that both sender and receiver could wait endlessly—the receiver waiting for more data in response to its offered window and the sender waiting for the window offer that allows it to send more data. Standard wire-line versions of TCP deal with this by defining a **persistence timer** that causes the sender

to periodically ask the receiver for a window-size update. Sending a "window-probe segment" if no offer has been received after a persistence time has elapsed does this.

In NOS, an analogous function is performed by the TCP *maxwait* parameter (see below). NOS does have a “persist” parameter for congestion control *at the AX.25 link-layer*. However, this parameter is used for contention control in an algorithm that calculates the probability that a sender will retransmit a packet after a backoff of a single “slot time” if it detects no carrier on the operating frequency (see Section 9). In the absence of contention for channels, link-layer congestion control played no significant role in our NVIS testing. Section 9 discusses some experiments we did run to assess the effects of contention.

If a segment fails to be ACKed after the current RTO estimate has elapsed, standard TCP assumes that the cause of this is a collision with a transmission from another network station. TCP’s method for dealing with the collision is to invoke a **backoff** in an attempt to avoid further collisions. Each backoff lasts for one or more multiples of the current RTO estimate up to some settable limit (see below). In NOS the backoff strategy can be set to be *exponential* (double the backoff time after each missed ACK) or linear (increase the backoff time *arithmetically* [by one backoff time] after each missed ACK). In each case, backing off continues until the settable backoff limit (*blimit*) on the number of backoffs is reached.

Because our point-to-point measurements took place on channels free of congestion from other network stations, we chose a linear backoff to avoid the undesirably wasted overhead that an exponential backoff would cause. Note that congestion is viewed here as being different from interference (QRM), which is treated as noise.

NOS allows control of its backoff protocol through the **backoff limit** (“ifconfig dr0 tcp blimit” in the autoexec.net file) and the **maximum wait-time** (“ifconfig dr0 tcp maxwait”) settings, both of which limit the extent of backing off under TCP’s congestion control strategy. The backoff limit is the maximum *number* of allowed backoffs. The maximum wait-time (*maxwait*, in milliseconds) is the *amount of time* that may elapse “avoiding congestion” before a retry is forced. The RTO, the amount of time that may elapse waiting for an ACK before retransmission of a datagram, is calculated in NOS by the formula

RTO = min[backoff_no*(4*MDEV + SRTT), maxwait],

where backoff_no ≤ *blimit* is the number of times a backoff has been invoked by TCP and SRTT and MDEV are current smoothed estimates of the round-trip time and its deviation (|SRTT – RTT|). The term 4*MDEV + SRTT can be construed as a multiple of the RTT and plays the role of the multiplier β referred to above. Hence, with a linear backoff strategy and a *blimit* of two there will be at most two backoffs. The first backoff lasts

min[4*MDEV + SRTT, maxwait] milliseconds;

the second lasts

min[2*(4*MDEV + SRTT), maxwait] milliseconds, etc.,

and the larger the SRTT and its mean deviation, the longer is each backoff.

Since our point-to-point channels had no congestion in the usual sense, we usually set the *blimit* for "aggressive retrying" (e.g., *blimit* = 2; we found that setting *blimit* = 1 sometimes caused timing problems). We set the *maxwait* to be a few seconds longer than the predicted time (based on the modem baud rate) it would take to send the largest window of data (e.g., *maxwait* = 30 or 40 seconds). Our approach was guided by the idea that since only signal fading or long-distance broadcast interference were causing frame errors, it was generally best to suppress most of the "congestion control" and keep trying to get frames through until they did get through or the number of retries was exceeded and the transfer attempt failed.

It is important to note that a careful watch must be kept on the *maxwait* setting, especially when the channel improves and bigger segment windows begin to be sent. If *maxwait* has been made too small in an attempt to reduce the delay associated with unnecessary collusion-avoidance measures, undesirable retransmissions of data will occur when the retry timer expires before the protocol has finished sending a big window. The *maxwait* value must be raised (between transfers) in this case.

The net effect of TCP's congestion control procedures is to reduce the volume and rate of retransmission exponentially when congestion is perceived but increase the rate and volume of transmission somewhat less aggressively when the perceived congestion goes away. Studies have shown that this approach prevents instabilities in network operation that would otherwise occur if recovery from congestion proceeded as rapidly as the response to it.

As described in the previous section, our measurement strategy involved observation of performance (throughput) and adjustment between transfers of the maximum segment size, modem data rate and occasionally the modem interleaver depth. This strategy was devised to cope best with current channel conditions and thereby offset the undesirable effects of backoffs caused by "misguided" congestion control.

Figures 4 and 5 illustrate changes in the congestion window, slow-start threshold and smoothed round-trip time during an actual FTP 10k-file transfer over a NVIS channel in good channel conditions. The data for the figures were logged at the sending station (MB1) by issuing the "tcp status" command from the NOS command line after every transmission from the file-receiving station (NFK) and writing the results to a file with the NOS "record" command. The recorded status data, which include current *cwind, ssthresh* and *srtt* values, were then extracted and formatted for plotting.

Figure 4 shows changes in the total number of received error-free octets (bytes) and the SRTT during the transfer, which used an MSS of 432 octets, a data rate of 1200 bps and a short

interleaver. The transfer took 170 seconds, for a throughput of 58 octets/s. Note that the total number of octets sent rose steadily during the transfer, with the next-to-last transmission sending a full window of 2592 octets. The final transmission was short and carried only the remaining 64 octets in the file plus the headers.

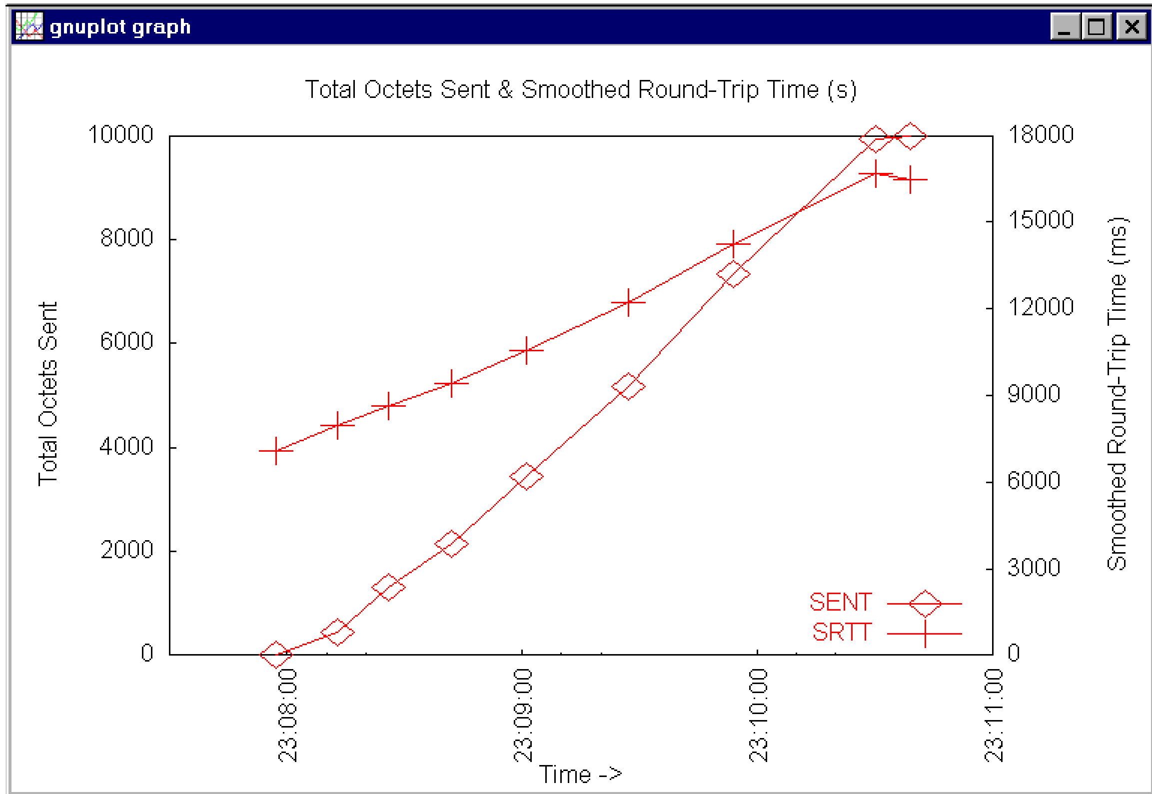

Figure 4. Total Octets Sent and SRTT for an Easy FTP Transfer

The smoothed round-trip time estimates started low from a previously stored value and then rose steadily toward their "current" mean under the conditions of the transfer of around 17 seconds. (Some of the round-trip time is taken up by backoffs and software processing rather than transmitting.) The slight drop in the SRTT at the end of the transfer was caused by the shorter round-trip time that elapsed during the transfer of the last 64-octet part of the file.

Figure 5 shows changes in congestion window and slow-start threshold during the same 10k-transfer. The *cwind* rose steadily during the transfer to the offered window size of

2592 octets and stayed at that size for the last three data transmissions, and since there were no lost packets, the slow-start threshold stayed at the maximum segment size value of 432 octets.

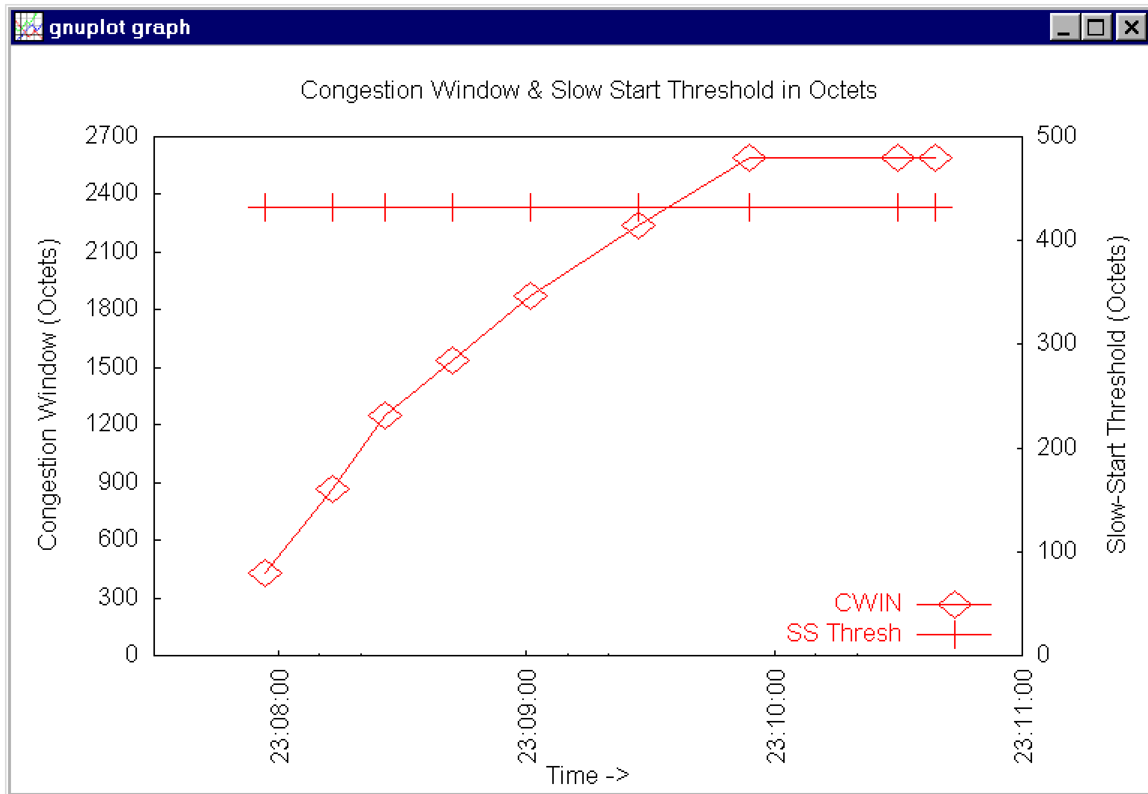

Figure 5. Congestion Window and Slow-start Threshold for an Easy FTP Transfer

Figures 6 and 7 show changes in the congestion window, slow-start threshold and smoothed round-trip time during a 10k FTP file transfer in poor channel conditions (mid-day with high D-layer absorption). In these conditions packets were lost (were received with too many bit errors to be corrected) and had to be resent following failure of the sender to receive ACKs of such packets.

Figure 6 is a plot of changes in the total number of received error-free octets and the SRTT during the transfer, which used an MSS of 216 octets, a data rate of 600 bps and a short interleaver. The transfer took 489 seconds, for a throughput of 20 octets/s. The total number of octets sent rose steadily until about a third of the way into the transfer (about 17:25:22 GMT),

when the *cwind* (see Figure 7) had risen to 1310 octets. At that time a window of 1296 octets failed to be ACKed (possibly because *maxwait* was set too small) and had to be resent.

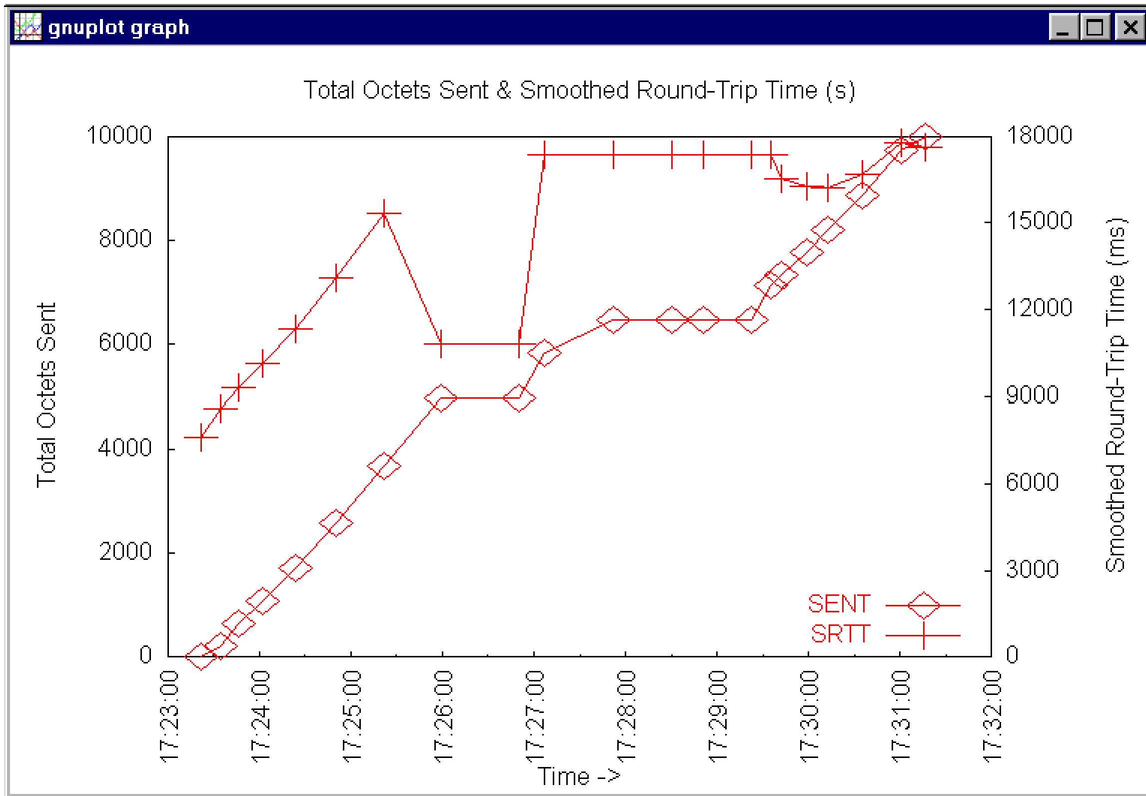

Figure 6. Total Octets Sent and SRTT for a Difficult FTP Transfer

Following the retransmission, the Jacobson "fast recovery" algorithm was invoked (see Appendix B). This raised the *ssthresh* to $cwind_{old}/2 = 1310/2 = 655$ and lowered the *cwind* to 968 (Figure 7). At 17:26:50 a further 216 (= MSS) octets failed to be ACKed, the *ssthresh* was halved again to $cwind_{old}/2 = 968/2 = 484$ and the *cwind* was lowered to its smallest value of 216 (the MSS). At 17:27:07 GMT data began to flow (i.e., be ACKed) again and the *cwind* was raised to 626. Starting at 17:27:52 four transmissions in a row failed to be ACKed and the *cwind* was lowered to 540 and then to 216, where it stayed, throttling down data flow accordingly. After an initial rise during this period to 540/2 = 270 in response to the previous resumption of flow, the *ssthresh* also fell to its minimum value of 216, where it also stayed during the outages.

At 17:29:42 the NVIS channel had improved sufficiently for error-free data flow to resume and the rest of the file (3520 octets) was successfully transferred (Figure 6). During this final period the *cwind* expanded steadily and the *ssthresh* stayed at 216 (Figure 7).

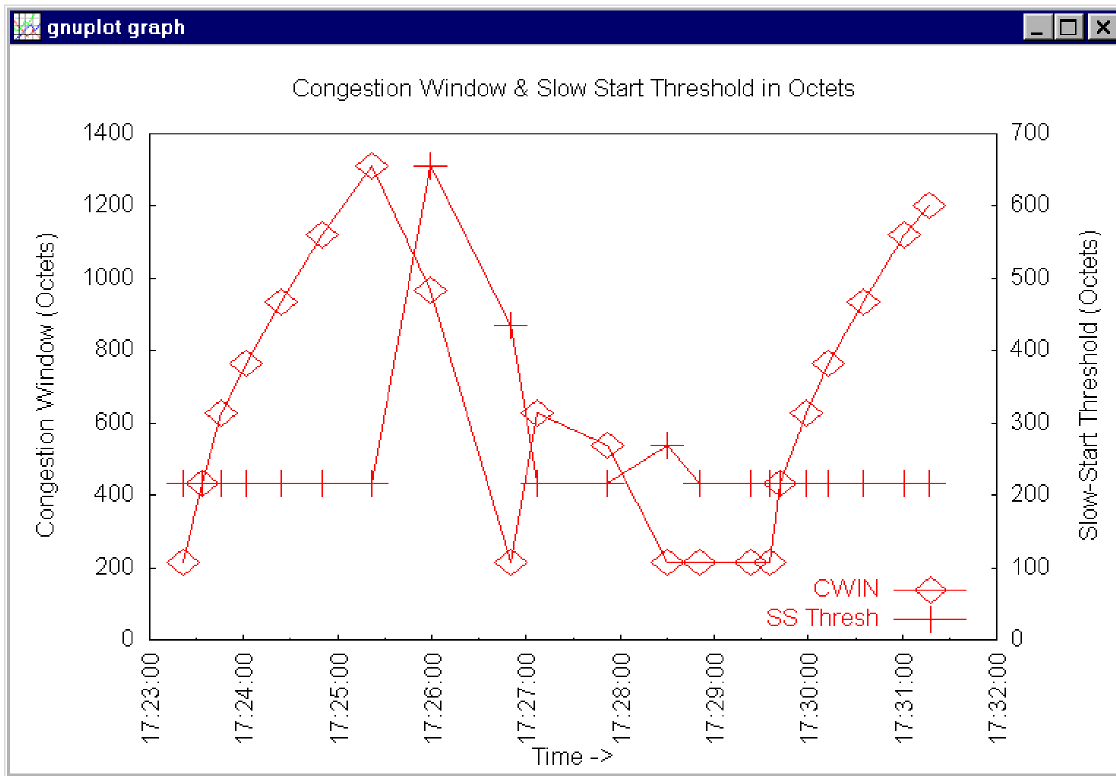

Figure 7. Congestion Window and Slow-start Threshold for a Hard FTP Transfer

The smoothed round-trip time estimates (Figure 6) again started low from a previously stored value and then rose steadily (toward their current mean of about 18 seconds) during the initial period of smooth data flow. After the first lost packet, the amount of sent data was restricted by the reduction in the value of *cwind* and this in turn lowered the round-trip time when data began to flow again. The effect of Karn's algorithm (described above) is seen in the constant value of the SRTT during the periods when data flow stopped, since no *new* round-trip times were measured then. The slight drop in the SRTT at the end of the transfer was caused in this case also by the shorter round-trip time that elapsed during the transfer of the last, small, 280-octet part of the file.

Section 6

Data Collection and Data Format for Point-to-Point Tests

As mentioned above, we manually collected our point-to-point data into a single ASCII archive file called archive.mdm after the modem we used in most of our experiments. Shown below is an excerpt from this file that was recorded in March 1999.

```
18.03.99 23:36:00  NOR  MB1 DG   s   864   2592  2400  S   191 20635    1   20635  Q
18.03.99 23:46:00  NOR  MB1 DG   s   864   2592  1200  S   393 30088    1   30088  Q
22.03.99 15:24:00  NOR  MB1 DG   I   432   2592  1200  S   290 10000    1   10000  Q
```

The entries in each line are

[DATE] [TIME] [TOCALL] [FROMCALL] [MODE] [TYPE] [MSS] [WIN] [RATE] [INTERLEAVER] [TRANSFER TIME] [SIZE] [NO RESENT] [SENTSIZE] [CHANNEL].

All TIMEs are Greenwich Mean (GMT). TOCALL and FROMCALL are the link-layer callsigns used by the receiving and sending stations during the transfer recorded on the line. The MODE can be either DG (datagram) or VC (virtual circuit). All of the tests reported on here used the datagram mode, in which retransmission of unacknowledged datagrams is controlled by the transport layer alone, rather than by both the transport layer and the data link layer, as in the *virtual circuit* mode.

TYPE stands for the transfer mode: I=IMAGE (FTP uncompressed files), A=ASCII (FTP compressed files), s=SMTP uncompressed messages and S=SMTP compressed messages.

MSS is the maximum segment size in bytes, the largest individually addressed packet of data that could be sent during the transfer (a parameter we adjusted between transfers for highest throughput in the perceived channel conditions). WIN is the window size, the largest number of bytes that can be sent in one transmission to the station that set it. (Note that a data window usually comprises several packets of data.) As pointed out in the previous section, under the TCP protocol the window size used by a file-sender is set by the *receiving* station. That is, the window values listed in our archive file were set by the station with the TOCALL address. The operative MSS during a TCP data transfer is the minimum of the MSS values at the sending and receiving stations. We set the MSS values at our receiving stations large enough to allow the sender to set the operative MSS for his transfers, and the sender's MSS is the value listed in our transfer data. The MSS and WIN are discussed further below.

The RATE and INTERLEAVER are the modem information rate in bits per second (75, 150, 300, 600, 1200 or 2400), and the modem interleaver depth (S = short, 0.6 s; L = long, 4.8 s).

The transfer time is the time in seconds between when correct reception of a file (by FTP) or message (by SMTP) was noted at the *sending* station and the time when the keystroke corresponding to start of transmission was hit. In the case of FTP file transfers, the latter time (the start of file transmission) was the time the "put filename" command be sent to the NOS FTP application from the keyboard. Note that the transfer time for FTP transfers does *not* include the time needed to establish the FTP session that has to be opened before files can be put (sent) or got (retrieved). Establishment of the session, and entry and confirmation of the user name and user password take up this time, typically about a minute. (Since the radio and modem are active during FTP sessions only when files are being sent or retrieved, we assumed that a typical FTP file transfer would occur during an already opened session.

To transfer a message file with SMTP, NOS combines session establishment and transfer into a continuous operation. That is, the user is not normally allowed to open an "idle" SMTP session. In this case, therefore, we started our transfer-time reckoning at the time of the keystroke that queued a message for transmission. We sent only one message at a time, so all SMTP transfer times include SMTP link establishment and log-in times.

SIZE is the size in bytes of the file or message transferred and NO RESENT is the total number of bytes resent (because of bit errors occurring on the sending or receiving channel in our case), which is reported to the screen during the course of a transfer by NOS. SENTSIZE is the number of bytes sent over the air during a successful transfer. If the transfer happens in the uncompressed mode, the SENTSIZE is the same as the SIZE. If the transfer happens in the compressed mode, the SENTSIZE is smaller (by about half for text files) than the SIZE.

CHANNEL describes the perceived state of the radio channel during the transfer. Q stands for quiet (little or no noise or interference), M for QRM (interference from other transmissions, usually by shortwave broadcast stations at night), N for QRN (noise from static crashes or manmade sources, especially during the summer storm season), B for QSB (fading caused by rapid changes in ionization levels and multipath at sunrise and sunset, for example) and D for disturbed (often caused by solar flares). Since our data were collected over the course of many months and are summarized in this report to describe average performance over long periods, we have lumped them together irrespective of channel type.

The line with dashes in its TRANSFER TIME and NO RESENT fields records a failed transfer. The number of such lines is used by statistical software described below to calculate the probability of correct transfer for various categories of transfer. To allow comparison with all of our other throughput studies, we recorded as failures only transfers that timed out *after* the corresponding connection and log-on phases.

As examples of the interpretation of lines in the performance-data archive consider the lines

08.03.99 22:15:00 NOR MB1 DG S 432 2592 1200 S 204 20229 1 10561 Q

22.03.99 15:24:00 NOR MB1 DG I 432 2592 1200 S 290 10000 1 10000 Q

The first line summarizes the transfer from Bedford (MB1) to Norfolk (NOR), Mass., of a compressed SMTP message file 20,229 bytes (octets) long that occurred on 8 March 1999 at about 22:15 GMT (5:15 PM local time). The maximum segment and window sizes were 432 and 2,592 bytes and the modem data rate and interleaver depth were 1200 bps and short. The transfer took 204 seconds and only one byte was resent (apparently an artifact of the NOS status reporting since no erroneous datagram containing data had to be resent). Because of the compression the number of bytes actually sent over the air was 10,561 and the channel was quiet. The throughput for this transfer was 20,229/204 ≈ 96 bytes/s. Note that in the case of uncompressed SMTP transfers (none shown above), the number of bytes sent over the air in our tests is slightly larger than the size of the message body—in our point-to-point transfers fewer than 100 bytes larger—because of the added SMTP mail-header.

The second line records similar data from an FTP transfer of a 10k text file in the image (uncompressed binary) mode. Note that the number of bytes sent over the air is the same as the file size.

Section 7

Data Analysis Software and Analysis Output

The results in the point-to-point data archive were analyzed off-line by a program we wrote called summary_mdm.c. This program reads the archive file line by line looking for various strings. As it moves through the file to the end-of-file indicator, the program keeps running totals of throughput and other data corresponding to the strings, from which it calculates statistics such as the average and standard deviation of the throughput. The statistics are written to a summary file after the pass through the archive file. Switches in the summary code are set before each run to pick out specific data (corresponding to various string combinations) for analysis. For example, we select lines with differing actual (SIZE) and compressed file sizes (SENTSIZE) to pick out compressed file transfers, and use the date-time group to distinguish daytime from nighttime transfers. Since the summary program was written to analyze archive files of fixed format but arbitrary length, summaries of the data collected so far can easily be made at any time.

Shown below is the output of the summary program for all TCP/IP NVIS tests run from January 1998 to April 1999 (a subset of all such data). For this output we set the software switches to compute throughput statistics for *nighttime compressed SMTP message transfers*.

Statistics calculated from archive.mdm on 04.06.99 13:24:37

SMTP COMPRESSED TRANSFERS NIGHT

1. NUMBER OF TRANSFERS IN SAMPLE = 118

2. E(MSS) = 535 bytes, sd(MSS) = 251 bytes

3. E(WIN) = 2320 bytes, sd(WIN) = 512 bytes

4. E(BAUD RATE) = 1408 bps, sd(BAUD RATE) = 486 bps

5. E(SIZE) = 20991 bytes, E(SENT) = 11393 bytes

6. E(RESENT) = 3526 bytes, E(FRACT RESENT) = 31%

7. max_RESENTFRACT = 186%, sd(FRACT RESENT) = 33%

8. E(TRANSFER TIME) = 399 s, sd(TRANSFER TIME) = 215 s

9. E(THRUPUT) = 63 cps, sd(THRUPUT) = 24 cps, sd(mean_THRUPUT) = 2.2 cps

10. max_THRUPUT = 127 cps, Median(THRUPUT) = 62.2 cps, E(THRUPUT/Hz) = 0.023 cps/Hz

11. 4 transfer failures; P(transfer success) = 118/122 = 0.97

Line nine of the output shows that the average throughput for 118 compressed-text-file mail transfers was about 63 characters per second (cps). Recall that throughput is equal to SIZE/(TRANSFER TIME) in the notation of Section 9, where SIZE equals the uncompressed size of a message or file. The largest observed throughput in this mode (line ten) was about 127 cps. The sd (THRUPUT) of 24 cps reflects the spread of the throughput measurements about their average, which was fairly large, and typical of the highly variable NVIS channel. Roughly speaking, about two-thirds of a set of measurements will be within one standard deviation of their mean and over 90 percent within two standard deviations of their mean.

We also calculated the "standard deviation of the mean throughput" [sd(mean_THRUPUT)] in characters per second (line 9) and the average throughput per Hz of signaling bandwidth line 10). The standard deviation of the mean throughput (equal to the standard deviation of the throughput divided by the square root of the sample size) is an assessment of the variability of the mean itself (which has its own statistical variability). The sd(mean) here (2.2 cps) suggests that our sample size in this case is big enough to make us confident that if we collected many more throughput measurements under roughly the same conditions, we would not get an average throughput that differed from the one above by more than about 2 cps.

To estimate the average throughput per Hertz [E(THRUPUT/Hz)], which normalizes bandwidth effects, we divide the average throughput by the average signaling bandwidth. For the serial-tone waveform, the signaling bandwidth is about 2700 Hz (see MIL-STD-188-110A) and the throughput per Hz of bandwidth is about 0.02 cps/Hz (line 10).

For the tests analyzed here we also kept track of the number of failed transfers and calculated the percentage of successful transfers. Unsuccessful transfers occurred when, after a successful link, the number of times the modem tried to send a data frame exceeded a programmable limit of 20 (set in autoexec.net with the "tcp retries" command), causing NOS to terminate the transfer. As noted above, we did not include failures to link in our transfer success ratios. For compressed nighttime SMTP transfers, 122 transfer attempts resulted in ARQ links, four of which timed out before the message got through. This led to a transfer success ratio of 118/122 or approximately 97 percent, typical of a properly designed ARQ protocol used with the robust MIL-STD-188-110A modem on HF links.

The second through fourth lines summarize statistically our attempts to adapt the TCP MSS and window sizes, along with the modem data rate to current channel conditions. The second line can be interpreted to mean that we chose a segment size of 432 most of the time and segment sizes of 216 or 864 most of the rest of the time. This is at odds with some recommendations of the "best" segment size to use on HF channels, but it should be kept in mind that we had the freedom to vary our segment sizes between transfers. For most users this is not an option. They will probably use a standard E-mail or FTP client and have no access to, or knowledge of, TCP configuration parameters. For them, a "recommended" segment size around 200 may still be the best choice.

Our average baud rate of around 1400 bps with its relatively small standard deviation shows that we operated most of the time at 1200 bps. Attempts to react to poor channel conditions by dropping the data rate to 300 bps or lower rarely led to good results (throughput was either very low or the attempt failed). This may be a reflection of the observation that when conditions are very poor they are often changing very rapidly and a protocol like FED-STD-1052, NATO STANAG 5066, Clover or Pactor-II that can adjust key protocol parameters *during* rather than just *between* transfers will usually perform better than the standard TCP stack.

Note that we rarely changed the interleaver depth to long (4.8 seconds) since that rarely seemed to raise throughput or lower the transfer failure rate. However, since we made no systematic study of "optimal" interleaver settings in our experiments, this should not necessarily be taken as a recommendation for NOS operation over HF. (Other protocols like FED-STD-1052 frequently change the interleaver, but they adapt their settings during and not just between transfers.)

The fifth through seventh lines show the average message file size (about 21,000 bytes or five pages of text) and average number of bytes that had to be resent during transfers. In this case, about a third of each compressed message had to be resent because of bit errors too numerous to be corrected by the modem's built-in error-decoder. The large standard deviation of the fraction of each message that had to be resent (33 percent) is further evidence of the variability of the NVIS channel. Compressed messages of this mean size took between six and seven minutes to transmit on average.

Section 8
Summary of Point-to-Point Performance Results

The results of our NVIS tests of TCP/IP (as of April 1999) are summarized in Tables 1 and 2 below. They correspond to SMTP and FTP throughput and are divided into day and nighttime performance, each of which is sub-divided into uncompressed and compressed performance. The first column in each table gives the average throughput and its standard deviation, the average throughput per Hz, the standard deviation of the mean throughput and the maximum observed throughput. The second column gives the number of transfers and the probability in percent of successful transfer [P(good xfer)] in each case. The third column gives the mean and standard deviation of the resent fraction for each transfer category. The fourth column gives the mean and standard deviation of the transfer time in seconds and the fifth column the average number of bytes in the original, uncompressed message files.

The average message-file size for all of our transfers was about 20,000 bytes. This is roughly the "optimal" size for throughput comparisons, in the sense that this size involves a large enough ratio of information content to average protocol overhead to produce almost maximal average throughput without wasting test time on larger files that achieve only marginal improvements in throughput (see Section 4).

In the case of compressed transfers (such as ours) that rely on NOS's built-in LZW compression algorithm, DOS memory-allocation methods led to memory overflow for files larger than around 40k bytes, so that is apparently the upper limit on file size for transfers using the applications built into NOS. However, using its FTP application, NOS can transfer binary files compressed *off-line* by other applications, and the size of such files is limited only by the storage capacity of the computers running NOS. Mail clients and servers that allow attachments (Eudora, Netscape, etc.) can also be interfaced with NOS in the manner shown below to allow transfer of large compressed message files.

Table 1 (SMTP) shows that the inherent (no compression used) over-the-air average throughput of NOS TCP/IP for SMTP (E-mail) transfers on our links was about 42 cps during the day and 55 cps at night. The standard deviations of these mean throughputs are about 2 cps, giving us high confidence that additional measurements made under the same conditions would yield nearly the same mean throughputs.

That the nighttime average throughput in this case is somewhat higher than the corresponding daytime throughput is probably caused by channel differences that are in turn a reflection of test scheduling limitations. Many of our uncompressed nighttime SMTP transfers took place in the evening before the arrival of foreign broadcast interference, and during periods when our chosen operating frequency was close to the optimal operating frequency. The corresponding daytime tests covered the day more evenly and were thus often exposed to

D-layer absorption at mid-day. This may have led to the better nighttime performance for SMTP uncompressed transfers. Tests with protocols that more easily allow one to schedule transfers that run all night suggest that average nighttime performance is generally lower than daytime performance.

Average throughput for compressed SMTP (E-mail) transfers on our links was about 67 cps during the day and 63 cps at night. The standard deviations of these mean throughputs are also about 2 cps. The fact that the compressed throughputs are only about 20 to 50 percent larger than the uncompressed ones (one might expect them to be around 100 percent larger) is due to a combination of channel differences, the extra time taken for the exchange of compression information between the sending and receiving stations and time taken by the stations to perform the frame-by-frame message compression and decompression that are built into NOS.

Table 2 (FTP) shows that the uncompressed (using the binary or image mode) average throughput of NOS for FTP was about 55 cps during the day and 53 cps at night. The standard deviations of these mean throughputs are around two cps, about the same as for SMTP transfers.

Average throughput for compressed FTP transfers was about 93 cps during the day and 105 cps at night, with standard deviations of these mean throughputs also about 2 cps. The slightly higher night than day throughput is again probably an artifact of our test schedule, and all-night testing may have produced a different result. Note that the compressed throughputs for FTP are now about twice the uncompressed ones. This is due in part to the fact that our FTP transfer times do not include session-establishment time. It is possibly also due to more efficient exchange of compression information between the sending and receiving stations in the FTP mode than in the SMTP mode.

The probability of correct message reception was above 90 percent in all cases. This is typical of adaptive ARQ protocols for HF, and was achieved by NOS despite the fact that we were adapted TCP/IP and modem parameters to channel conditions only between message transfers. It should be kept in mind that in the case of FTP transfers our data apply only to the transfer phase of TCP operation. Failed attempts to establish an FTP session (which occasionally occur) are not included in our probability-of-success statistics. (SMTP probabilities do include such failures since every one of our SMTP transfers began with establishment of an SMTP session.)

The average fraction of each message that had to be re-sent by TCP's (transport-layer) ARQ protocol during our tests was between 20 and 30 percent. This reflects our adjustment of parameters as channel conditions changed. This fraction would no doubt have been significantly larger if the parameters had remained fixed.

Median throughputs (the 50th percentile of the measured throughput distribution and not listed in the tables) were five to ten percent below average throughputs. This implies that slightly more measurements lie below the means than above them, but the medians are close enough to the mean throughputs to justify use of the latter as a performance index.

Maximum observed throughputs were between 122 and 154 cps for SMTP transfers and between 118 and 219 cps for FTP transfers, depending on transfer mode and time of day. These may be of some value in comparisons with other protocols but are not a good indication of expected performance, since the NVIS channel rarely allows them to be achieved.

Table 1. Statistical Summary of HF NVIS SMTP Throughput Data

Xfer Mode	E(thruput) sd(thruput) E(tput/Hz) sd_mn(tput) max_tput	No. Xfers P(good xfer)	E(Resent Frac.) (RF) sd(RF)	E(xfer_tm) sd(xfer_tm)	E(SIZE)
Uncompr. Text Msg. Day	42 cps 23 cps 0.015 cps/Hz 1.7 cps 122 cps	185 91%	19% 21%	577 s 356 s	19702
Compr. Text Msg Day	67 cps 30 cps 0.025 cps/Hz 2.2 cps 154 cps	179 92%	27% 32%	384 s 191 s	21191
Uncompr. Text Msg Night	55 cps 21 cps 0.020 cps/Hz 1.9 cps 133 cps	128 99%	15% 18%	421 s 198 s	20886
Compr. Text Msg Night	63 cps 24 cps 0.023 cps/Hz 2.2 cps 127 cps	118 97%	31% 33%	399 s 215 s	20991

Table 2. Statistical Summary of HF NVIS FTP Throughput Data

Xfer Mode	E(thruput) sd(thruput) E(tput/Hz) sd_mn(tput) max_tput	No. Xfers P(good xfer)	E(Resent Frac.) (RF) sd(RF)	E(xfer_tm) sd(xfer_tm)	E(SIZE)
Uncompr. Text File Day	55 cps 32 cps 0.020 cps/Hz 2.5 cps 150 cps	161 94%	23% 28%	517 s 427 s	21725
Compr. Text File Day	93 cps 45 cps 0.034 cps/Hz 3.9 cps 219 cps	135 96%	25% 29%	296 s 209 s	21022
Uncompr. Text File Night	53 cps 24 cps 0.020 cps/Hz 2.0 cps 118 cps	141 93%	22% 19%	455 s 333 s	19891
Compr. Text File Night	105 cps 41 cps 0.039 cps/Hz 3.2 cps 218 cps	170 94%	21% 31%	237 s 151 s	21718

Section 9

NOS Performance in a Network

To assess some of the effects on performance of TCP/IP operation in HF networks, we ran a number of tests involving uncompressed one-way FTP file transfers among more than two stations. The main purpose of these tests was to assess the effects of contention and relaying on throughput. Another purpose was to investigate the effects and appropriate settings of various transport-layer (TCP) and link-layer (AX.25) parameters designed to deal with contention, or the longer round-trip times encountered in relaying, or both, and to investigate rules of thumb for setting such parameters in other NOS HF networks. Lack of the time required to carry out network tests forced us to accept small sample sizes for our data (less than ten in some cases). Our findings are therefore based on anecdotal evidence only. Readers will have to decide for themselves whether the observed performance provides useful guidelines for configuration and operation of other TCP/IP-based HF networks.

The tests involved

- Simultaneous FTP transfers from one station to two other stations,
- Simultaneous FTP transfers between two independent pairs of stations and
- File transfers from one station to a second station through a third relay station.

In each case we also ran occasional point-to-point transfers with the same parameter settings as in the corresponding network tests to allow comparisons with performance in non-networked conditions. Note that we did not run any tests of simultaneous FTP transfers *from* two stations to a *single* third station. However, the observations from the second set of tests may shed some light on that kind of operation.

All tests were run with serial-tone modem data rates of either 1200 or 600 bits per second and short interleaver settings. Furthermore, all tests were run in the afternoon or early evening in autumn, when propagation is normally good. TCP Window and maximum segment sizes were set at 2592 and 216 octets at all stations and almost all of the transferred text files had a size of 10k bytes. As observed above, these settings and this file size are not usually ones that lead to highest throughput, but relative throughputs recorded in these experiments are more important than absolute ones.

In the first set of tests, the single sending station opened two FTP sessions, one with each of two recipients. In the second set of tests, each of two sending stations opened a single FTP session to one of the remaining two stations in a four-station network. Here we caused one of the two sending stations to start its transfer slightly after the start of the transfer of the other sending station, since we view it as unlikely that stations in a four-station network would normally start transfers at exactly the same time. Almost simultaneous starts still represent a kind of worst case for contending transfers.

In the relaying tests, the NOS autoexec configuration files at two stations were set up to force them to relay all traffic to each other through a third (relay) station (the commands that accomplished this are listed below). Comparison data in this case came from one of the other pairs of stations that were configured for point-to-point operations.

In network operations that involve contending stations, the link (AX.25)-layer *slottime* and *persist*[ence] parameters play key roles in dealing with contention. The *persist* parameter (set in the autoexec.net file with the command "dr0 persist," and taking values between 0 and 255) is used (in the driver software for the DRSI synchronous I/O card) to calculate the probability that a sender will transmit a packet when it determines that the channel is clear. This persistence probability P_p is calculated as (*persist* + 1)/256.

Any NOS station with data to transmit first checks the HF modem's carrier-detect line for an indication of a serial-tone preamble. As soon as NOS finds there is no carrier (that is, no preamble has been detected), it waits for *slottime* x 10 ms and then begins transmitting data with probability P_p. If the clock-based random number generator used in the DRSI driver determines that NOS will not begin transmitting, then NOS waits another *slottime* x 10 ms and, hearing no carrier, again begins transmitting its waiting data with probability P_p. This process continues until transmission of the data finally occurs.

Making *persist* small at a station causes it to be less likely than otherwise to transmit immediately when it has data to send and detects no carrier. Making *slottime* large causes a longer wait than otherwise before a station with data generates a random number to decide whether to transmit. Lowering *persist* or raising *slottime* at a station gives other stations in a NOS network more channel time. Of course, a station that gives up that time must expect lower average throughput.

The following table gives the throughput in octets (8-bit bytes) per second for four sets of **transfers from one sending station to two receiving stations**, carried out on the same day near 5 MHz. Each set of transfers comprises single puts (transfers) one after the other to each receiving station for comparison, followed a few minutes later by simultaneous puts to both stations starting at almost the same time. The table lists the addresses of the sending and receiving stations, whether the transfer was single or simultaneous and the throughput in bytes/s. In all cases the sender-modem transmission rate was 1200 bps and both receivers

transmitted their ACKs at 600 bps. The *persist* and *slottime* parameters at all three stations were 176 and 180 (the defaults used in some Navy operations) and all stations set their *maxwait* to 30000 (30 s). Note that although the sender does not contend with himself in the simultaneous-transfer case, the two receivers do contend in the sending of their ACKs, so the contention control parameters will play a role in determining performance in this case.

Table 3. FTP Transfers from One Station to Two Other Stations

TO	FROM	Sing/Simul	Tput bytes/s
NFK	MB1	Single	9
DER	MB1	Single	15
NFK	MB1	Simultaneous	5
DER	MB1	Simultaneous	9
NFK	MB1	Single	23
DER	MB1	Single	21
NFK	MB1	Simultaneous	7
DER	MB1	Simultaneous	10
NFK	MB1	Single	16
DER	MB1	Single	19
NFK	MB1	Simultaneous	5
DER	MB1	Simultaneous	Timed Out
NFK	MB1	Single	15
DER	MB1	Single	21
NFK	MB1	Simultaneous	6
DER	MB1	Simultaneous	7

The table shows that the throughput for the simultaneous transfers from MB1 to DER and NFK was between about a third and half that recorded for single transfers. The fact that the simultaneous-transfer throughputs were in some cases significantly smaller than half their single-transfer values might be explainable by differences in the link qualities between the sender and the two receivers, but the sample sizes are of course too small to allow one to make stronger statements. Perhaps the most useful finding in this case is simply the confirmation that NOS can perform simultaneous one-to-many transfers and that the throughput is probably acceptable in some applications.

The next table gives the throughput from several sets of **contending pairs of transfers**. These pairs involved a transfer from one sending station to one receiving station carried out while a second sender independently transferred a file to a second receiving station. All transferred files contained 10k bytes. The data were collected on four different days using the same frequency near 5 MHz. With two exceptions (marked by asterisks) in which the senders worked at 1200 bps, transmission rates at all stations were 600 bps. Both receiving stations had *persists* of 176. The *slottime* at all four stations was 180 (= 1800 ms). Varying the *slottimes* and *maxwait*s at sending stations in our network would have given us more leeway than adjusting only *persist* in coping with congestion, but a study of this must be put off to another time. *Maxwaits* were between 30 and 60 seconds for the four stations.

The table lists the addresses of the sending and receiving stations, the *persist* parameter at the file-sending station, whether the transfer was single or simultaneous and the throughput in bytes/s. (Simultaneous transfers are of course grouped in pairs in the table.) The notation "(1st)" after a sender's address indicates that the sender started his transfer a minute or so before the other sender started.

We observed the following phenomena during these contending transfers: Generally speaking, the station that started transferring first got all or most of its file through before the station that started only slightly later. However, attempts to use the channel by the station held off from transferring still lowered the throughput of the station that finished first. The throughput of the sender that finished first was lowered to about two-thirds of the throughput observed in the absence of contention.

Twice in our data, for reasons perhaps having to do with the vagaries of timing, both stations *shared* the channel until they finished their transfers. However, this resulted in significantly *lower* throughput than in cases when one station effectively seized the channel until it was done.

Three times during simultaneous transfers one of the senders timed out. (The maximum number of retries was set to 20 at all stations.) This almost never happens during single transfers conducted at the time of day of these tests.

Lowering of the *persist* parameter at one of the senders to reduce contention had an inconclusive effect on throughput. More data will have to be collected to make any recommendations on appropriate *persist* and *slottime* settings.

The best advice we can derive from our experiments is to avoid contending transfers. If they can't be avoided, follow the conventional advice drawn from packet radio experience that says to use as much variety in the setting of *persist* and *slottime* at potentially contending stations as fairness requirements allow.

Table 4. FTP Transfers Between Two Independent, Contending Pairs of Stations

TO	FROM	TX Persist	Sing/Simul	Tput bytes/s
NFK	MB2(1st)	255	Simultaneous	21
DER	MB1	50	Simultaneous	20
NFK	MB2*	255	Simultaneous	8
DER	MB1(1st)*	50	Simultaneous	Timed Out
NFK	MB2	127	Simultaneous	6
DER	MB1(1st)	127	Simultaneous	6
NFK	MB2	127	Single	24
NFK	MB2(1st)	127	Simultaneous	21
DER	MB1	127	Simultaneous	Timed Out
NFK	MB2(1st)	20	Simultaneous	5
DER	MB1	20	Simultaneous	8
NFK	MB2	20	Simultaneous	9
DER	MB1(1st)	20	Simultaneous	6
NFK	MB2	20	Simultaneous	10
DER	MB1(1st)	20	Simultaneous	Timed Out
NFK	MB2(1st)	20	Simultaneous	7
DER	MB1	20	Simultaneous	11
NFK	MB2	20	Simultaneous	4
DER	MB1(1st)	20	Simultaneous	3
NFK	MB2	20	Single	21
DER	MB1	20	Single	17
NFK	MB2	176	Simultaneous	10
DER	MB1(1st)	60	Simultaneous	7
NFK	MB2(1st)	176	Simultaneous	17
DER	MB1	176	Simultaneous	22
NFK	MB2	176	Simultaneous	8
DER	MB1(1st)	176	Simultaneous	7
NFK	MB2(1st)	60	Simultaneous	14
DER	MB1	176	Simultaneous	11
DER	MB1	176	Single	32

The last table gives the throughput from several sets of **transfers using a relay**. The relay was set up for communications between MB2 and DER via NFK. It was set up by replacing the normal default routing statements in MB2 and DER's autoexecs ("route add der.mbpr.org dr0" and "route add mb2.mbpr.org dr0") with the statements

route add der.mbpr.org dr0 nfk.mbpr.org [at MB2] and

route add mb2.mbpr.org dr0 nfk.mbpr.org [at DER].

No contention from stations not involved in the relayed transfer took place. All transferred files contained 10k bytes and the data were collected on three different days using the same frequency near 3 MHz. Transmission rates at all stations were 600 bps. The *persists* and *slottimes* of all three stations were 176 and 180 and all three stations had *maxwaits* of 60 or 65s.

Table 5. FTP Transfers Through a Relay Station

TO	FROM	Pt-Pt/Relay	File Size	Tput bytes/s
DER	MB1	Pt-Pt	10k	30
DER	MB2	Relay	10k	13
DER	MB1	Pt-Pt	10k	24
DER	MB2	Relay	10k	10
NFK	MB1	Pt-Pt	10k	27
DER	MB2	Relay	10k	13
DER	MB1	Pt-Pt	10k	20
DER	MB2	Relay	10k	10
DER	MB1	Pt-Pt	10k	30
DER	MB2	Relay	10k	12
DER	MB1	Pt-Pt	10k	22
DER	MB2	Relay	10k	14
DER	MB1	Pt-Pt	20k	31
DER	MB2	Relay	20k	11

The table lists the addresses of the sending and receiving stations, whether the transfer was point to point or via relay, the file size and the throughput in bytes/s. As above, the point-to-point transfers were also run to allow comparisons with non-relayed performance. (With one exception, a transfer from MB1 to NFK, all the point-to-point tests were run from MB1 to DER. MB1 is a few hundred yards from the relay initiator MB2.)

The table shows that relaying reduced throughput to between a third and a half of comparable point-to-point throughput. In cases where the relayed throughput was significantly less than half the point-to-point throughput, the difference may have been due to random differences in link quality on the two legs of the relay path.

Section 10

Setup of Trumpet Winsock and the Pipe and Winpkt TSR Shims for 32-bit Operation via NOS

Our approach to running IP-based, 32-bit Windows 95 applications via the 16-bit, DOS-based NOS TCP/IP stack required installing a second 32-bit stack called Trumpet Winsock (windows socket). Trumpet Winsock communicates with 32-bit applications in the same way as the "native" Window 95 stack (Winsock.dll) does. However, Trumpet Winsock is configurable via an ASCII setup file (*trumpwsk.ini*), whereas the native Winsock.dll appears to be configurable only by editing the Windows 95/98 Registry. (It is not clear that the native Winsock could be configured to communicate with NOS even with an understanding of the Registry parameters that govern operation of the Winsock.dll.)

When IP-based applications are to be run over the internal Serial Line Internet Protocol (SLIP) link between Trumpet Winsock and NOS, a batch file is run that replaces the native Winsock.dll with Trumpet Winsock in the path the applications follow to find a TCP/IP socket. When applications are to be run in the usual way over say the Ethernet, another batch file is run that reverses this process. The applications then find and use the native Winsock. Figure 8 shows the flow of data between applications, Trumpet Winsock, the pipe and winpkt drivers and NOS when such applications are run over a radio network. Trumpet Winsock is available in several versions on the Internet (www.trumpet.com.au) for a small registration fee. We have installed and used the 32-bit Version 3.0d.

Examples of the settings in the trumpwsk.ini file relevant to our network are:

ip=129.83.41.193 [IP "hardware" address of the DTE at this station. (The network ID part of this address is "129," corresponding to one of MITRE's Ethernet groups).]

netmask=255.255.255.0 [Distinguishes the network ID portions of IP addresses on this network (the first three numerals) from the host ID (individual node) parts (the last seven numerals).]

gateway=128.83.66.65 [NOS IP "radio" address of this station (the network ID part of this address is "128," corresponding to a notional radio network we use for our tests)]

dns=128.83.66.65 [IP address of the domain name server for the 128 network]

domain=mbpr.org [domain name for this network: mitre bedford packet radio]

vector=61 [software interrupt vector; a "hook" to a pipe driver]

mtu=576	[MTU in bytes for internal TWinsock-NOS SLIP link]
rwin=2048	[TCP Receive Window in bytes for internal TWinsock-NOS SLIP link]
mss=512	[TCP MSS in bytes for internal TWinsock-NOS SLIP link]

The “gateway” and “dns” entries configure Trumpet Winsock to use JNOS as the gateway and Domain Name Server for TCP/IP radio connections. The MTU, RWIN and MSS settings for the internal SLIP link connecting Trumpet Winsock and NOS were chosen for acceptable performance of IP-based applications over the radio network. The choice of these parameters depends on the values of similar parameters chosen for the radio links controlled by NOS. The choice of TCP parameters for radio links is an art rather than a science and will always be a subject for further study. An annotated copy of one of our HF trumpwsk.ini files is shown in Appendix A.

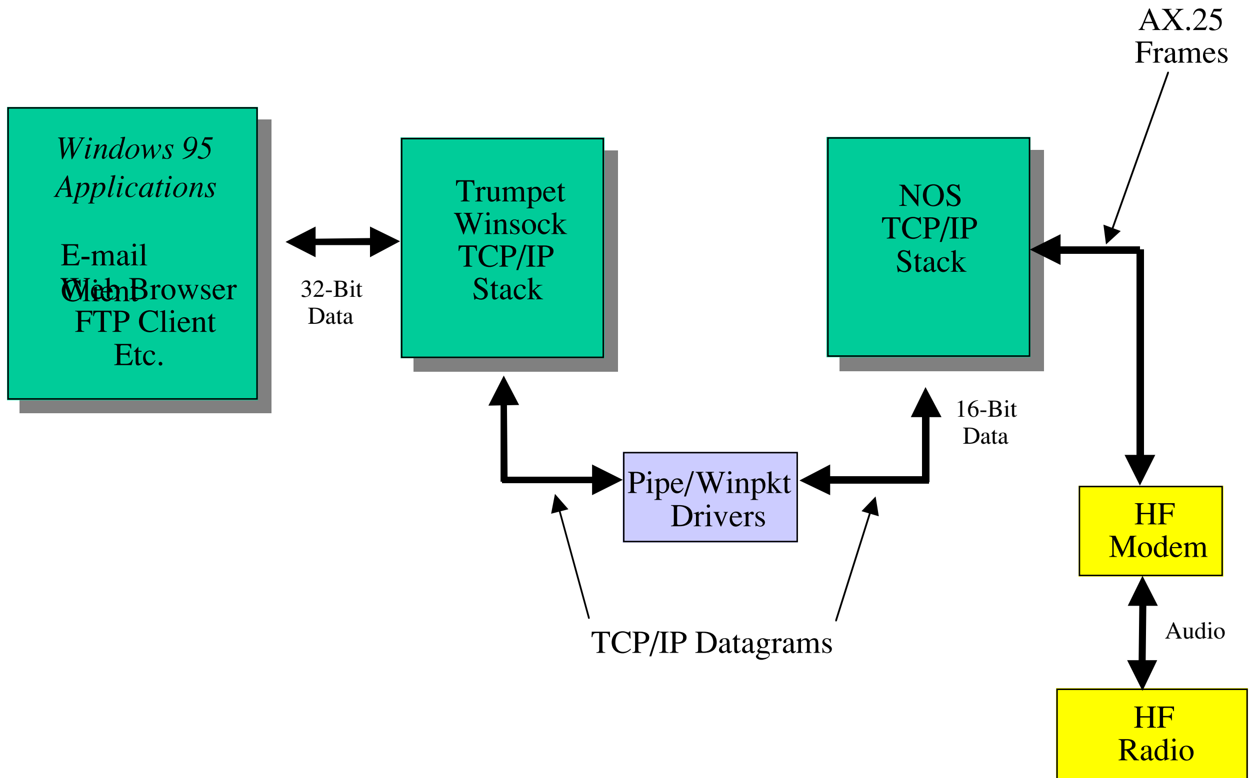

Figure 8. 32- and 16-Bit Data Flow at a Station in an IP-Based Radio Network

To get Trumpet Winsock to talk to NOS, two interface programs ("drivers") are required. These are called *pipe.com* (a "double ended packet driver") and *winpkt.com* (a windows packet driver). These programs are loaded into system memory to run as terminate-and-stay-resident (TSR) applications.

The two-sided pipe.com packet driver provides two data channels (a so-called "wormhole") between two programs (NOS and Trumpet Winsock). It is invoked with the following "attach" command placed in the basic autoexec.net file for NOS:

attach packet 0x60 pipe 3000 6000

This command attaches the pipe driver at software interrupt vector 60, and gives it a buffer size of 3000 octets (bytes) and an MTU of 6000 octets. The parameters of the pipe interface are set up below the attach command in the autoexec.net file. These are ("ifc" = ifconfig specifies the command as one that configures an interface)

ifc pipe tcp mss 216

ifc pipe tcp win 2592

ifc pipe tcp maxwait 60000

The *mss* and *win* have been described above. *Maxwait* is the time in milliseconds NOS will wait for an acknowledgment of a packet over this interface before re-sending it.

The winpkt.com driver is loaded into memory as a TSR program. Commands in the trumpwsk.ini and NOS autoexec.net configuration files that call out the same memory address allow the Trumpet Winsock and NOS stacks to "see" the areas of memory that are occupied by the winpkt and pipe drivers. This allows the two stacks to use these drivers for the internal SLIP link over which they talk to each other.

Section 11

Addressing and Routing in an HF Network

Each station in an AX.25 network such as our HF NVIS network requires an AX.25 callsign/hardware address. These callsigns are used for identification during AX.25 connections. (NOS IP datagrams are encapsulated in AX.25 frames by NOS itself.) Every data packet that goes out over the air is labeled with the AX.25 callsigns of the sending and receiving stations and their IP addresses. In the syntax of the NOS autoexec file, the AX.25 callsign "mb1" is entered with the command

```
ax25 mycall mb1
```

The use of two stacks in our implementation of the interface between 32-bit applications and NOS requires that two IP addresses be used by each HF station: a NOS ("radio") address and a "Windows address" associated with the Windows applications that communicate directly with Trumpet Winsock. An example of a NOS IP address and its associated "host name" (taken from the NOS autoexec file at "mb1") is

```
ip address    128.83.66.45
hostname      mb1.mbpr.org
```

This IP address is a member of the 128-subnet. The host name is prefixed by the AX.25 callsign and has the domain name associated with the subnet as its suffix. The domain name in our closed-network tests was *mbpr.org*, the domain name used for a mitre bedford packet radio network. Hostnames are used in E-mail addressing and to look up IP addresses in the "host files" that domain name servers consult to associate aliases (host names) with IP addresses.

In most of our tests we route all traffic directly to the synchronous (dr0) interface card that communicates with the serial-tone modem. This is done by putting the command

```
route add default dr0
```

in all of our NOS autoexecs. If we wanted to use a particular node as a relay for all traffic—the one with callsign DER in Derry, N.H., for example—we would replace this command with the command "route add default dr0 der.mbpr.org" in their autoexecs. The two commands above are sufficient to address a TCP/IP station that uses only NOS and related DOS applications. However, since we wish to run Windows 95/98 applications using a two-stack approach, we require more complicated addressing.

The following autoexec (or directly entered) command assigns the "Windows address" 129.83.41.193 to the corresponding DTE:

```
route add 129.83.41.193 pipe
```

(This address belongs to the set of those used on the firewall-protected MITRE Ethernet. This notional example would thus correspond to a radio network that—if authorized—could allow an HF net member to be connected to the internal MITRE desktop network.) The assigned address is the one that will be used by IP-based Windows applications installed in the local DTE when they communicate with their counterparts running on DTEs at other stations. The command says further that the pipe TSR program (pipe.com) is the interface NOS is to use when routing incoming datagrams to the corresponding Windows application(s).

The command below is the last one needed for basic connectivity for Windows applications in a simple HF network. It is part of TCP/IP's Address Resolution Protocol (ARP) and is required to establish how stations that need to reach Windows applications at other stations should do so.

```
arp publish 129.83.41.193 ax25 mb1 dr0
```

The address resolution process maps addresses of different forms (for example, those from Ethernet and IP networks) to each other. In the case of TCP/IP, the ARP dynamically maps "hardware addresses" used by various networks (e.g., Ethernet or AX.25 addresses) to 32-bit IP-addresses. The command above maps the IP address 129.83.41.193 to the AX.25 link layer hardware address mb1. Stations that want to get application data to the address 129.83.41.193 need only send it to the AX.25 address mb1, which knows how to get it to 129.83.41.193 (namely, through the pipe). Resolution can be local or in response to requests. For example, the ARP publish command

```
arp publish 129.83.46.68 ether 00:10:4b:7b:9a:98 pk0
```

would allow responses to Ethernet members' requests for information on how to get datagrams to an HF net member with *Windows IP address* 129.83.46.68 ("use the Ethernet hardware address 00:10:4b:7b:9a:98 via the pk0 port"). The command

```
arp publish 44.56.8.103 ether 00:10:4b:7b:9a:98 pk0
```

would allow responses to Ethernet members' requests for information on how get to an HF net member with *ham-radio-net IP address* 44.56.8.103 ("use the Ethernet hardware address 00:10:4b:7b:9a:98 via the pk0 port").

In our Trumpet-Winsock-NOS configurations, NOS resolves hardware addresses by ARP-publishing hardware-to-IP-address correspondences at HF stations. The command

```
arp publish 129.83.41.193 ax25 mb1 dr0
```

referred to above is in the autoexec at the HF station with AX.25 callsign mb1. This command announces to other interfaces that applications using the Windows IP address 129.83.41.193 are associated with the AX.25 (radio) callsign mb1 via the dr0 (synchronous DRSI-card) port.

Section 12

Running 32-bit Applications over the HF NVIS Network

In this section we describe the main events that occur when an application running on a DTE on a platform communicates with a corresponding application running on a DTE on another platform. The second DTE will generally be at another HF station (where it hosts another user application like an E-mail client) or it may be on a non-radio platform acting as a DNS server or a gateway/router. Our approach will be to discuss the actions taken by the user(s) and those performed automatically by servers or routers between them. Excerpts from NOS "trace files" will be shown to illustrate what's happening at various OSI layers as the DTEs, servers and routers interact with each other.

12.1 A PING Example

We start with perhaps the simplest stack operation, a "ping" sent from one HF station to another without intervening relay. In this example, NOS itself (rather than a separate application) will do the pinging, so no discussion of 32-bit applications is needed.

Pinging, whose operation is governed by the Internet Control Message Protocol (ICMP), involves sending a packet of user-specified length to another user-stack with a request that the packet be acknowledged with a packet of the same size. If the acknowledgment is received, the protocol calculates and displays the ping's round-trip time. An option of displaying and recording statistics from scheduled pings may also be exercised from the NOS command prompt.

Pinging is especially useful in radio applications when channel or congestion conditions make links unreliable and one wants to check if signals are propagating on a link and how long they take to be received ("echoed") when they are propagating. (This time includes both transmission and processing delays.) We have also found it to be crucial in understanding how various HF and other stations use the Address Resolution Protocol to discover routes to stations in other networks.

In our HF network pings can be sent from the command line of NOS or from various software packages designed to monitor network performance. We do most of our pinging from NOS itself.

As an illustration, consider pings sent using NOS from one HF station to another when the sender knows the hardware address of the recipient. To monitor the flow of ICMP datagrams

that occur in this situation we turn on the “trace” function of NOS. Tracing allows us to view (and if desired, record to a file) all or part of each AX.25 packet sent and received on most of the interfaces NOS has been configured (with “attach” commands) to use. The traced packets are labeled with the sender’s and receiver’s AX.25 and IP addresses and with the acronyms of the protocols being employed in the exchange (AX.25, ARP, ICMP, TCP, etc.).

The following traced packets show AX.25 callsign MB1 (Bedford, Mass.) PINGing callsign NFK (Norfolk, Mass.) over an HF link. MB1 was sending at a modem information rate of 1200 bps with a short (0. 6 second) interleaver and NFK was sending at 600 bps with a short interleaver. Notable parts of the exchange have been highlighted in boldface. Before pinging, MB1 has entered the command “arp add 128.83.66.46 ax25 nfk dr0” into his autoexec.net. This tells his version of NOS that the AX.25 hardware address NFK corresponds to the IP “radio” address 128.83.66.46.

Fri Sep 17 19:55:26 1999 - dr0 sent:
AX25: **MB1->NFK** UI pid=IP
IP: len 32 128.83.66.45->128.83.66.46 ihl 20 ttl 254 **prot ICMP**
ICMP: type **Echo Request** id 65535 seq 0
0000 O\..

Fri Sep 17 19:55:33 1999 - dr0 recv:
AX25: **NFK->MB1** UI pid=IP
IP: len 32 128.83.66.46->128.83.66.45 ihl 20 ttl 254 **prot ICMP**
ICMP: type **Echo Reply** id 65535 seq 0
0000 O\..

Note that the ping has been returned in about seven seconds. The next four traced packets show what happens when MB1 pings NFK *without* knowing the hardware address (NFK) that corresponds to the IP address 128.83.66.46.

Fri Sep 17 19:54:48 1999 - dr0 sent:
AX25: **MB1->QST** UI pid=**ARP**
ARP: len 30 hwtype AX.25 **prot IP op REQUEST**
sender IPaddr 128.83.66.45 hwaddr **MB1**
target IPaddr 128.83.66.46 hwaddr

Fri Sep 17 19:54:54 1999 - dr0 recv:
AX25: **NFK->MB1** UI pid=**ARP**
ARP: len 30 hwtype AX.25 prot IP op **REPLY**
sender IPaddr 128.83.66.46 hwaddr NFK
target IPaddr 128.83.66.45 hwaddr MB1

Fri Sep 17 19:54:54 1999 - dr0 sent:
AX25: **MB1->NFK** UI pid=IP
IP: len 32 128.83.66.45->128.83.66.46 ihl 20 ttl 253 prot **ICMP**
ICMP: type **Echo Request** id 65535 seq 0
0000 ;...

Fri Sep 17 19:55:01 1999 - dr0 recv:
AX25: **NFK->MB1** UI pid=IP
IP: len 32 128.83.66.46->128.83.66.45 ihl 20 ttl 254 prot **ICMP**
ICMP: type **Echo Reply** id 65535 seq 0
0000 ;...

Note that in this case, before he can ping NFK, MB1 must first send an ARP request (a "QST" or broadcast call) to it to find out the hardware-to-IP-address correspondence there. Since ARP packets are about as long as the default ping size, the need for an ARP request roughly doubles the roundtrip time (now about 13 seconds) of the first ping to NFK. (Once NFK has been "ARPed," MB1's NOS remembers the ARP information obtained with the ARP request and subsequent pings will take about half the time of the first one so long as NOS is running.)

12.2 An FTP Example

Our second example is more complicated than pinging: a file transfer from a Windows 95 FTP client (WS_FTP.exe) running at MB1 to the NOS FTP server at NFK. This will involve use of the pipe TSR between the 32-bit WS_FTP application and the 16-bit NOS stack at MB1 and the synchronous dr0 ports for the AX.25 packet radio link. Since the associated trace record in this case is several pages long, we will give only a narrative of the significant events recorded in it. This will leave out most of the activity on the pipe utility that connects the Trumpet Winsock and JNOS stack via the internal SLIP link, since pipe activity generally mirrors radio-link activity. Notable parts of the exchange have again been highlighted in boldface. Numbers that preface certain announcements, like the "230" in "230 w1imm logged in" are standard labels for FTP events and are generated by the FTP protocol.

The HF user at Bedford (hardware address 129.83.41.193, radio address MB1) initiates the transfer by launching his WS_FTP client application and requesting a connection to NFK (IP address 128.83.66.46). MB1 advertises a window of 2048 octets and an MSS of 512 octets. (Data on the server have already been entered into the user's WS_FTP client application).

Fri Sep 17 21:05:43 1999 - dr0 sent:
AX25: MB1->NFK UI pid=IP
IP: len 44 129.83.41.193->128.83.66.46 ihl 20 ttl 59 prot TCP
TCP: 1052->21 Seq x70560000 **SYN Wnd 2048 MSS 512**

NFK responds, advertising a window of 2592 and an MSS of 1296:

Fri Sep 17 21:05:58 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 44 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 21->1052 Seq x512c5000 **Ack x70560001 ACK SYN Wnd 2592 MSS 1296**

Soon the pipe passes NFK's FTP log-on notification to MB1's FTP client:

Fri Sep 17 21:06:00 1999 - pipe sent:
serial line IP: len: 136
IP: len 136 128.83.66.46->129.83.41.193 ihl 20 ttl 253 prot TCP
TCP: 21->1052 Seq x512c5001 Ack x70560001 ACK PSH Wnd 2592 Data 96
0000 220- nfk.mbpr.org, JNOS FTP version 1.10m/JNOSC (8088)..220 **Rea**
0040 **dy** on Fri Sep 17 21:05:32 1999..

The user at MB1 (*w1imm*) sends his user name:

Fri Sep 17 21:06:00 1999 - dr0 sent:
AX25: MB1->NFK UI pid=IP
IP: len 52 129.83.41.193->128.83.66.46 ihl 20 ttl 59 prot TCP
TCP: 1052->21 Seq x70560001 Ack x512c5061 ACK PSH Wnd 2048 Data 12
0000 **USER w1imm**..

and then his password in response to the corresponding prompt from NFK passed up by the pipe:

Fri Sep 17 21:06:18 1999 - pipe recv:
serial line IP: len: 53
IP: len 53 129.83.41.193->128.83.66.46 ihl 20 ttl 60 prot TCP
TCP: 1052->21 SeK->MB1 UI pid=IP
IP: len 64 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 21->1052 Seq x512c5061 Ack x7056000d ACK PSH Wnd 2592 Data 24
331 **Enter PASS command**..

NFK compares w1imm's user name and password with a table of authorized users and their passwords and tells MB1 that w1imm is logged onto the NFK FTP server:

Fri Sep 17 21:06:26 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 61 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 21->1052 Seq x512c5079 Ack x7056001a ACK PSH Wnd 2592 Data 21
3. **230 w1imm logged in**..

w1imm asks NFK for the name of its current directory (PWD = "print working directory"):

Fri Sep 17 21:06:36 1999 - dr0 sent:
AX25: MB1->NFK UI pid=IP
IP: len 45 129.83.41.193->128.83.66.46 ihl 20 ttl 59 prot TCP
TCP: 1052->21 Seq x70560027 Ack x512c50a5 ACK PSH Wnd 2048 Data 5
00 **PWD**..

NFK responds:

Fri Sep 17 21:06:45 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 70 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 21->1052 Seq x512c50a5 Ack x7056002c ACK PSH Wnd 2592 Data 30
3. **257 "/" is current directory**..

w1imm's WS_FTP application automatically—and wastefully on a narrowband radio channel—asks for a listing of the current FTP directory:

Fri Sep 17 21:06:54 1999 - dr0 sent:
AX25: MB1->NFK UI pid=IP
IP: len 46 129.83.41.193->128.83.66.46 ihl 20 ttl 59 prot TCP
TCP: 1052->21 Seq x70560045 Ack x512c50da ACK PSH Wnd 2048 Data 6
00 **LIST**..

NFK begins sending the 4626 bytes of the listing:

Fri Sep 17 21:07: NFK->MB1 UI pid=IP
IP: len 93 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 21->1052 Seq x512c50da Ack x7056004b ACK PSH Wnd 2592 Data 53
0000 150 **Opening data connection for LIST / (4626 bytes)**..

The data begin to arrive in segments of 512 octets and are passed up the pipe to WS_FTP:

Fri Sep 17 21:07:32 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 552 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 20->1053 Seq x62bf9001 Ack x9fd70001 ACK Wnd 2592 Data 512
0000 **3comdr~1/ 12:05 8/09/98 acrobat3/**
0040 **8:19 8/13/97..atlas5/ 11:54 8/22/98 aurora**
0080 **.zip/ 7:58 9/18/97..autoexec.nav al line** IP: len: 44
IP: len 44 129.83.41.193->128.83.66.46 ihl 20 ttl 60 prot TCP
TCP: 1053->20 Seq x9fd70000 Ack x62bf9001 ACK SYN Wnd 2048 MSS 512

About five minutes later (in channel conditions that caused some repeats) the listing has arrived:

Fri Sep 17 21:12:21 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 58 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 21->1052 Seq x512c510f Ack x7056004b ACK PSH Wnd 2592 Data 18
0000 **226 File sent OK**..

w1imm now asks NFK to switch to its FTP current working directory (c:\temp) and list its contents (865 octets):

Fri Sep 17 21:19:30 1999 - dr0 sent:
AX25: MB1->NFK UI pid=IP
IP: len 50 129.83.41.193->128.83.66.46 ihl 20 ttl 59 prot TCP
TCP: 1052->21 Seq x7056004b Ack x512c5121 ACK PSH Wnd 2048 Data 10
00 **CWD temp**..

Fri Sep 17 21:20:02 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 96 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 21->1052 Seq x512c517c Ack x70560079 ACK PSH Wnd 2592 Data 56
00 150 Opening data connection for LIST /temp (865 bytes)..

The listing gets sent in one transmitted window and is ACKed 18 seconds later:
Fri Sep 17 21:20:20 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 58 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 21->1052 Seq x512c51b4 Ack x70560079 ACK PSH Wnd 2592 Data 18
0000 **226 File sent OK..**

After receiving the listing, w1imm decides to send the text file "file.2k" to NFK. (The command to do this is sent from the GUI to the local WS_FTP application and does not appear as such in the traced data.)

WS_FTP will send this file in binary ("Image") mode and the stations negotiate this:

Fri Sep 17 21:29:39 1999 - dr0 sent:
AX25: MB1->NFK UI pid=IP
IP: len 48 129.83.41.193->128.83.66.46 ihl 20 ttl 59 prot TCP
TCP: 1052->21 Seq x70560079 Ack x512c51c6 ACK PSH Wnd 2048 Data 8
0000 **TYPE I..**

Fri Sep 17 21:29:46 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 55 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 21->1052 Seq x512c51c6 Ack x70560081 ACK PSH Wnd 2592 Data 15
00 200 Type I OK..

The transfer begins:

Fri Sep 17 21:30:02 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 93 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 21->1052 Seq x512c51ec Ack x705600a8 ACK PSH Wnd 2592 Data 53
00 150 Opening data connection for STOR /temp/file.2k ..

and the first segment of text data is sent via the pipe and ACKed:

Fri Sep 17 21:30:26 1999 - pipe recv:
serial line IP: len: 552
IP: len 552 129.83.41.193->128.83.66.46 ihl 20 ttl 60 prot TCP
TCP: 1055->20 Seq x60810001 Ack xb3b7f001 ACK Wnd 2048 Data 512
0000 **Agamemnon, Vis-.-vis.[6.3.95]..I have always believed in the lit**
0040 **eral truth of the Trojan War; my strong belief in Homer and trad**
0080 **ition has never been shaken by modern critics, and I thank this**
00c0 **belief for the discovery of Troy . . . . [A similar belief] led**
0100 **me to the discovery of the five graves [at Mykenai] with their**
0140 **outstanding treasures. ....-Heinrich Schlie**
TCP: 1055->20 Seq x60810000 Ack xb3b7f001 ACK SYN Wnd 2048 MSS 512

Fri Sep 17 21:30:26 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 40 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 20->1055 Seq xb3b7f001 **Ack x60810001** ACK Wnd 2592

The file gets sent and w1imm closes the FTP connection with a QUIT command that receives an ACK via the pipe:

Fri Sep 17 21:35:02 1999 - dr0 sent:
AX25: MB1->NFK UI pid=IP
IP: len 46 129.83.41.193->128.83.66.46 ihl 20 ttl 59 prot TCP
TCP: 1052->21 Seq x705600c1 Ack x512c526a ACK PSH Wnd 2048 Data 6
00 QUIT..

Fri Sep 17 21:35:08 1999 - pipe sent:
serial line IP: len: 54
IP: len 54 128.83.66.46->129.83.41.193 ihl 20 ttl 253 prot TCP
TCP: 21->1052 Seq x512c526a Ack x705600c7 ACK PSH Wnd 2592 Data 14
0000 **221 Goodbye!..**

The client and server negotiate closing of the FTP session and the MB1 FTP client resets (closes) the TCP socket connection via the pipe:

Fri Sep 17 21:35:18 1999 - pipe recv:
serial line IP: len: 40
IP: len 40 129.83.41.193->128.83.66.46 ihl 20 ttl 60 prot TCP
TCP: 1052->21 Seq x705600c8 **RST Wnd 0**

12.3 An E-mail Example

We conclude our examples with a complete E-mail exchange between a Windows 95 Eudora POP client at one of our HF stations and a NOS E-mail POP server running at a second station. The associated trace record in this case is also many pages long so we again present only a narrative of what the trace file has recorded. Furthermore, the narrative highlights only the most significant events in the trace, leaving out most of the packets traced on the pipe interface.

The HF user at MB1 starts the exchange by composing and sending at the Eudora GUI a message to anapol@129.83.66.46 (NFK's IP address). This launches via the pipe and NOS a connection request (SYN packet) to NFK in whose header the SMTP (E-mail) protocol is specified. NFK responds via the pipe with confirmation of the request:

Fri Sep 17 20:33:37 1999 - pipe sent:
serial line IP: len: 69
IP: len 69 128.83.66.46->129.83.41.193 ihl 20 ttl 253 prot TCP
TCP: 25->1049 Seq x76faf001 Ack xe2610001 ACK PSH Wnd 2592 Data 29
0000 **220 nfk.mbpr.org SMTP ready..**

MB1 sends Eudora log-on information:

Fri Sep 17 20:34:34 1999 - pipe recv:
serial line IP: len: 71
IP: len 71 129.83.41.193->128.83.66.46 ihl 20 ttl 60 prot TCP
TCP: 1049->25 Seq xe261003e Ack x76faf05b ACK PSH Wnd 2048 Data 31
00 RCPT TO:<anapol@129.83.66.46>..

and NFK ACKs it:

Fri Sep 17 20:34:44 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 48 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 25->1049 Seq x76faf05b Ack xe261005d ACK PSH Wnd 2592 Data 8
00 **250 Ok**..

MB1 sends via the pipe and NOS a Eudora-generated header containing the addressee anapol@129.83.66.46, the message subject ("Autoexec"), and the encoding type (MIME):

Fri Sep 17 20:34:52 1999 - pipe recv:
serial line IP: len: 552
IP: len 552 129.83.41.193->128.83.66.46 ihl 20 ttl 60 prot TCP
TCP: 1049->25 Seq xe2610063 Ack x76faf07f ACK Wnd 2048 Data 512
0000 Message-Id: <3.0.5.32.19990917203320.007d2e60@128.83.66.46>..X-S
0040 ender: anapol@128.83.66.46 (Unverified)..Disposition-Notificatio
0080 n-To: <anapol@128.83.66.46>..X-Mailer: QUALCOMM Windows Eudora P
00c0 ro Version 3.0.5 (32)..Date: Fri, 17 Sep 1999 20:33:20 -0500..To
0100 : anapol@129.83.66.46..From: Weeper <anapol@128.83.66.46>..**Subje**
0140 **ct: Autoexec..Mime-Version: 1.0..**Content-T4:52 1999 - dr0 sent:
AX25: MB1->NFK UI pid=IP
IP: len 40 129.83.41.193->128.83.66.46 ihl 20 ttl 59 prot TCP
TCP: 1049->25 Seq xe2610063 Ack x76faf07f ACK Wnd 2048

MB1 now sends the first data segment of the attached autoexec.net file (the numbered lines contain message data and the "====" line has been manufactured by Eudora):

Fri Sep 17 20:34:52 1999 - dr0 sent:
AX25: MB1->NFK UI pid=IP
IP: len 552 129.83.41.193->128.83.66.46 ihl 20 ttl 59 prot TCP
TCP: 1049->25 Seq xe2610263 Ack x76faf07f ACK Wnd 2048 Data 512
0000 ii"....See attached file..--======================_937636400==_..
0040 Content-Type: text/plain; charset="us-ascii"....# **AUTOEXEC**
0080 .**NET setup file for HF NVIS at MB1 4.8.99 ..#..# Miscellaneous**
00c0 **setup **..# 286/386 clo**
0100 **ck..isat yes..# dump/session.log..log 1**
0140 **..watch off..memory thresh 14336..memory ibufsizeup file for HF NVIS at MB1 4.8.99 ..#..# Miscellaneous**
00c0 **setup **..# 286/386 clo**
0100 **ck..isat yes..# dump/session.log..log 1**
0140 **..watch off..memory thresh 14336..memory ibufsize**
0180 **3000..#..# Set up domain defaults ********************************
01c0 ************..domain cache size 50..domain suffix none..domain ret**
.
.
.

About two minutes later NFK ACKs receipt of the message:

Fri Sep 17 20:37:09 1999 - dr0 recv:
AX25: NFK->MB1 UI pid=IP
IP: len 50 128.83.66.46->129.83.41.193 ihl 20 ttl 254 prot TCP
TCP: 25->1049 Seq x76faf07f Ack xe26112df ACK PSH Wnd 2592 Data 10
00 **250 Sent**..

and closes the connection (SMTP session):

Fri Sep 17 20:37:17 1999 - pipe sent:
serial line IPpid=IP
IP: len 46 129.83.41.193->128.83.66.46 ihl 20 ttl 59 prot TCP
TCP: 1049->25 Seq xe26112df Ack x76faf089 ACK PSH Wnd 2048 Data 6
00 QUIT..

which it then resets:

Fri Sep 17 20:37:18 1999 - pipe recv:
serial line IP: len: 40
IP: len 40 129.83.41.193->128.83.66.46 ihl 20 ttl 60 prot TCP
TCP: 1049->25 Seq xe26112e5 **RST Wnd 0**

Section 13

Interfacing a NOS HF Network to a LAN

To configure our HF network for operation with another network requires appropriate hardware and driver software for the interface to the other network. If the other network is an Ethernet, for example, this would typically be done by installing an Ethernet card and driver in a DTE at at least one of the HF stations, which would then serve as a gateway to the Ethernet for the rest of the HF network.

Operating such a card with NOS requires a DOS-based Ethernet driver. DOS-format drivers for many, but not all, hardware interfaces to NOS are available on the Internet. We normally configure a NOS batch file (*nosstrt.bat*) to activate a DOS driver on a DTE as a TSR with a specified software interrupt.

It should be noted that since NOS will invoke the driver, setup of the latter requires that its software be in a format suitable for DOS drivers, rather than Windows 95/98 plug-and-play ("have disk") drivers.

13.1 Connecting the HF Network to the Ethernet

To attach an interface in NOS at a station that can also communicate with an Ethernet LAN, a command like the following is entered into the corresponding NOS autoexec.net file:

```
attach packet 0x65 pk0 10 1500
```

This command attaches a packet driver at memory address 65, gives it the label "pk0," allows up to 10 packets in the receive buffer and sets the MTU on the corresponding port to 1500 octets (the standard Ethernet packet value). Note that the capacity assigned to the receive buffer on an Ethernet interface is measured in packets, not bytes.

The Ethernet interface (for example, at an HF Ethernet gateway station) is configured with commands like the following:

```
ifconfig pk0 ipaddress 129.83.46.108
ifconfig pk0 tcp mss 1460
ifconfig pk0 tcp win 14600
ifconfig pk0 tcp retries 3
ifconfig pk0 tcp blimit 10
ifconfig pk0 tcp maxwait 100
ifconfig pk0 tcp irtt 1000
```

Most of these are analogous to the parameters for other IP-carrying interfaces and have the same meanings. Note, however, that the Ethernet interface gets its own IP address since IP addresses for each "network adapter" (NOS I/O port) must be distinct. (Since IP addresses identify "hardware devices" on a network, each device needs its own IP address.)

This means that an HF-to-Ethernet gateway running 32-bit applications must be assigned *three* IP addresses: a "radio-network" address (like the 128.83.66.45 in the NOS-setup example above), a Trumpet Winsock "software" address (like the 129.83.41.193 in the example trumpwsk.ini file discussed in Appendix A) and an additional IP address (like the sample address 129.83.46.108 listed above) assigned to the pk0 Ethernet port. (HF net members that aren't gateways require only the first two types of IP addresses if they are not running 32-bit applications.) Note that we have set the MSS above equal to 40 less than the MTU to account for the 20 bytes of IP and 20 bytes of TCP header in each segment.

The following commands resolve ("ARP") the gateway's Windows address (129.83.41.193) *and* the gateway's Ethernet port address (129.83.46.108) into the Ethernet *hardware* address assigned to the gateway's Ethernet card (e.g., 00:10:4b:7b:9a:98):

```
arp publish 129.83.41.193 ether 00:10:4b:7b:9a:98 pk0
arp publish 129.83.46.108 ether 00:10:4b:7b:9a:98 pk0
```

These commands configure NOS to tell other DTEs on the Ethernet how to reach the two gateway stacks when those DTEs send an ARP request asking for that information.

Connecting to another network also requires addition of appropriate addressing and routing commands to the NOS autoexec files of the HF stations that want to talk to that network. Normally, all stations would gain access to the other network through an HF gateway/router. In the case of the Ethernet, NOS itself can serve at one or more stations as such a gateway/router for an HF network.

The following examples of routing commands for the HF gateway's NOS autoexec show various ways of setting up the DTE that could function as a gateway to the MITRE Ethernet:

Send all 129.x.x.x IP traffic to the MITRE Ethernet gateway at 129.83.46.254:

```
route add 129/8 pk0 129.83.46.254
```

Send *all* IP traffic to the MITRE gateway at 129.83.46.254:

```
route add default pk0 129.83.46.254
```

Send all 129.x.x.x IP traffic out the Ethernet port (pk0) to a private Ethernet that requires no gateway:

```
route add 129/8 pk0
```

Send *all* IP traffic to a private Ethernet that requires no gateway:

```
route add default pk0
```

The following gateway commands ARP publish a MITRE Domain Name Server (DNS) at IP address 129.83.20.100 and a MITRE Web (WWW) Server at IP address 129.83.11.40.

```
arp publish 129.83.20.100 ax25 mb1 dr0
arp publish 129.83.11.40 ax25 mb1 dr0
```

The first command would allow HF stations to use an HF gateway to get to a MITRE domain name system (DNS) for sending E-mail to MITRE accounts. The second would allow HF stations to log onto a MITRE internal website.

For further details and some diagrams of data flow in an IP-based radio network connected to an Ethernet or other LANs, see our report on experimentation with an IP-based communications appliqué for the ARC-164 UHF HAVE QUICK radio (Reference 5).

Section 14

Findings

In this report we have described the installation, setup, operation and performance of the standard NOS TCP/IP protocol stack over HF near-vertical-incidence-skywave radio channels. We have also described the installation and setup of software that allows 32-bit Windows 95 applications such as E-mail clients to run over HF links via NOS. Both types of implementation use inexpensive COTS hardware, standard COTS application software (Eudora, WS_FTP, etc.) and the DOS-based shareware NOS stack that manages routing and interfacing to other networks. NOS is configurable via an ASCII autoexec.net file and various versions of its C source code are widely available.

In our on-air point-to-point testing we emphasized suitable choices of radio or protocol parameters (including operating frequency), and good manual adjustment of TCP and modem parameters between message transfers, in an attempt to achieve high throughput in the face of varying HF channel conditions. Such parameter adjustments are often desirable on radio channels to offset the effects of TCP's response to channel-induced packet corruption as if it were channel congestion. The most important parameters to be adjusted dynamically for HF NVIS channels appear to be the TCP maximum segment (data-frame) size and the HF modem data rate.

Under the conditions of our point-to-point tests the average throughput of the NOS TCP suite lay between about 42 and 67 characters per second for SMTP transfers, depending on whether a file was compressed or not and whether transfers were run during the day or night. FTP transfers achieved average throughputs between 53 and 105 characters per second under the same stipulations. The probability of successful message-file transfer was above 90 percent in all cases.

These throughputs are roughly one-half to two-thirds the corresponding throughputs of the high-performance HAL CLOVER-2000 and Harris RF-6710/6750 (Federal-Standard-1052) HF data-transmission systems, which employ proprietary software and use link-layer protocols specifically tailored (unlike those in NOS) to the properties of half-duplex HF radio channels. All three systems have comparable probabilities of successful message-file transfer (90 percent or higher).

The link-layer protocols of the CLOVER-2000 and the Harris RF-6710/6750 systems adapt automatically to changing channel conditions. CLOVER-2000 adapts rapidly—about every four seconds, and the Harris RF-6710/6750 less rapidly—roughly once a minute. Rapid adaptation follows the channel more accurately at the expense of overhead, whereas slower adaptation uses less overhead but leads to less agile channel-following. Which is the better approach is probably still open to debate.

NOS does not adapt TCP segment size or a modem's data rate automatically. Instead, we adjusted the segment size and the serial-tone modem's data rate by hand between point-to-point message-file transfers (NOS does not allow manual segment-size adjustment during transfers). As a general rule, we lowered the segment size and data rate in point-to-point tests when conditions were poor and raised them when they were good. This often had a profound effect on throughput.

Although this strategy led to what many users might consider acceptable or even good throughput, NOS performance over HF channels could be improved significantly by modifying it to adjust the maximum segment size and perhaps the maxwait in response to current assessments of the channel as reflected by "performance." NOS could be further improved for HF use by letting it adjust the serial-tone modem's data rate in response to current performance. Of particular interest in such an improvement would be determining the performance measurements that might be used to guide this automatic adjustment and the frequency with which the adjustments should be made (during transfers, between transfers, etc.).

What such performance measurements should be is a recommended subject of study. Some candidate measurements are the

- Throughput of the last transfer, weighted by how long ago it occurred,
- Number of resent octets (bytes) during the last or current transfer,
- Average signal-to-noise ratio (SNR) during the last or current transfer (taken from the modem),
- Current (smoothed) SNR (for possible parameter adjustments *during* a transfer) and
- Current Congestion Window (for possible parameter adjustments *during* a transfer).

Another modification of NOS that would improve its HF radio performance is to change the NOS transport-layer ARQ protocol to one that uses *selective acknowledgments or selective non-acknowledgments* (NACKs) of missed packets. This would remove the wasteful effects of the standard go-back-N protocol, which frequently rejects correct data packets. Selective ACKs or NACKs have been added to a number of recent upgrades of the TCP/IP stack (including the one used by Windows 98).

A further improvement would be introduction of algorithms that can discriminate between channel errors and collisions, reacting aggressively to the former and backing off (as is currently done by NOS) in reaction to the latter. The SCPS modification of the standard stack includes such algorithms, including one variant that turns off the standard congestion control altogether.

It should be noted that modifications like those suggested here will generally make the modified stack incompatible with standard stacks. This has the disadvantage that unless there is a way to revert to a standard stack for sockets used for non-radio communications (e.g., for connections to Ethernet LANs), the “HF radio stack” will be usable only with stacks that have been similarly modified.

A small number of NVIS network tests were run to investigate the effects on performance of contention and relaying. Although the sample sizes of the tests were too small to allow us to arrive at definitive conclusions about network operations with NOS, some tentative guidelines were suggested. These were that NOS is easy to configure for relaying, and that file transfers probably work best when stations with TCP/IP data to send either schedule their transfers so as to avoid contention altogether or set their contention control parameters (persist and slottime) so as to give different stations widely differing access to channels.

List of References

1. D. C. Lynch and Rose, M. T., eds., *Internet System Handbook*, Addison-Wesley, Reading, Mass. 1993.

2. W. R. Stevens, *TCP/IP Illustrated*, Vol. 1, Addison-Wesley, Reading, Mass. 1994.

3. R. Fahnestock and Dugal, James P., *JNOS Commands Manual*, No date. Tailored to JNOS version 1.11e, updated in Dec. 1999. Can be found at the FTP site pc.usl.edu/pub/ham/jnos.

4. I. Wade, *NOS Intro*, ARRL Publications, Newington, Ct. 1992.

5. R. P. Levreault, Lokuta, R. S., and Wickwire, K. H., *An IP-based Data-Communications Applique for the ARC-164 (HAVE QUICK) Radio*, MITRE Working Note 99B0000030, May 1999.

Appendix A

Example NOS Autoexec File

This appendix lists an annotated copy of the HF autoexec file used at our NVIS station MB1 in Bedford, Mass ("#" marks comment lines). Although Section 3 discusses key aspects of this file in overview, understanding of its entries and their proper settings requires documentation and some experience. A good set of documentation on the meaning of most of the commands in this autoexec can be found in Reference 4 and at the FTP site listed in Reference 3. Note that although these references define all the commands, they do not constitute a complete guide to how they should be set. One of the purposes of this note is to advance the state of this guidance as far as the use of NOS over HF radio is concerned.

The use of TCP/IP is normally associated with use of a domain-name-to-IP-address lookup table, which makes avoids memorization of IP addresses. However, the table is not strictly necessary for simple networks like ours.

```
#   AUTOEXEC.NET setup file for the HF NVIS at Station MB1 4.8.99
#
# Miscellaneous setup ****************************************
# 286/386 clock
isat            yes
# dump/session.log
log             1
watch           off
memory thresh    14336
# Make sure this is not < RX buf in attach command:
memory ibufsize  3000
#
# Set up domain defaults ****************************************
domain cache size 50
domain suffix none
domain retries 2
domain cache clean off
#
```

```
# Station Identification ****************************************
ip address        128.83.66.45
hostname          mb1.mbpr.org
# This MUST precede the "attach" command.
# Note.  The callsign used in an ax.25 header is used much like an
# ethernet address in an ethernet datagram.
# “Radio” callsign
ax25 mycall mb1
#
# Set up an Ethernet interface ***********************************
# Format:      address  label  #pkts in RX buf     MTU
attach packet 0x65     pk0    10                    256
#
# Attach pipe interface between Trumpet Winsock & JNOS ***************
# Format:       address   type   rcv buffer  MTU
attach packet   0x60      pipe   3000        6000
# Set up TCP parameters for the pipe shim
ifc pipe tcp mss         216
ifc pipe tcp win         2592
ifc pipe tcp maxwait 60000
#
# Attach Synchronous interface to DRSI card ************************
# Format:   addr    irq proto  IFace  rxbuf MTU   rate1  rate2
attach drsi 0x310 5    ax25  dr0    2700 2600 2400  2400
# Alternative address & IRQ
#attach drsi 0x300 11  ax25  dr0    2048  1792 2400  2400
# AX25 parameters for DRSI interface
param dr0 txdelay   45
param dr0 enddelay 45
param dr0 txtail      25
param dr0 slottime 180
param dr0 persist   176
# Turnaround time.  Use 70 for USQ-122 modem/20 for MDM-3001 modem
param dr0 turn 20
#
# Configure DRSI Interface ************************************
#Backoff timer mode for "congestion control" on DRSI interface
ifconfig dr0 tcp timertype linear
ifconfig dr0 tcp retries 20
ifconfig dr0 tcp blimit  2
# MAXWAIT in milliseconds
```

```
ifconfig dr0 tcp maxwait 30000
# MAXWAIT for networked operation.
# dr0 maxwait in milliseconds, make all nodes in net a few
# tens of seconds different (600000-1000000)
# dr0 irtt in milliseconds, make all nodes in nets a few
# seconds different (10000-17000)
# IRTT in ms
# Startup parameters:
ifconfig dr0 tcp irtt      15000
ifconfig dr0 tcp mss       216
ifconfig dr0 tcp window 2592
#
# Set up TCP/IP defaults (some over-ridden by DRSI values)*************
ip rtimer    60
ip ttl       255
tcp mss      216
tcp irtt     15000
tcp window 216
#
# Start various "network services" ********************************
start pop3
start smtp
start ttylink
start telnet
start ftp
start finger
start remote
remote -s 4643
#
# Mbox setup ***********************************************
mbox nrid         yes
mbox sendquery  off
mbox attend       on
mbox tmsg         "WELCOME TO THE MB1 HF NVIS MAILBOX\n"
attend            on
third             on
#
```

```
# Set up IP routing ********************************************
route add default dr0
route add 128.83.66.46 dr0
route add 128.83.66.47 dr0
route add 129.83.41.193 pipe
# Address Resolution Protocol setup
arp add 128.83.66.46 ax25 nfk dr0
arp add 128.83.66.47 ax25 der dr0
arp publish [129.83.41.193] ax25 mb1 dr0
#
# Set up the mailbox *******************************************
# Timer (if used) in seconds, the longer the better
#smtp timer 21
#Only an SMTP kick will launch messages
smtp timer    off
smtp mode     route
# Use 1 for no tracing, use 3 to trace smtp client exchange
smtp trace    1
# Batch shipment of messages:
#smtp batch   off
#time interval used to notify user/return undelivered mail
#return mail after DTIMEOUT hours as undelivered
smtp dtimeout 72
# Turn on extended memory for SMTP.
smtp usemx    on
strace        off
# Set degree of LZW message compression (16 = max)
lzw bits 16
lzw mode compact
domain dns off
# >>>>>>> LAST COMMAND IN FILE <<<<<<<<
# THE END
```

Appendix B

Example *trumpwsk.ini* File

Shown below is a trumpwsk.ini initiation file for the Trumpet Winsock v. 3.0d TCP/IP stack that communicates with Windows 95 applications, and over an internal SLIP link with NOS, in our HF network configurations. The parameter settings we used in our configurations are at the top of the file and are annotated. We left the indented entries below the annotated entries at their indicated default values.

[Trumpet Winsock]	[a label for this file]
ip=129.83.41.193	[IP "hardware" address of the DTE/PC at this station]
netmask=255.255.255.0	[distinguishes assignable from fixed parts of addresses in this network]
gateway=128.83.66.65	[NOS IP "radio" address of this station: "send outgoing here"]
dns=128.83.66.65	[IP address of domain name server for this network (the NOS "domain.txt" file)]
vector=61	[software interrupt vector; a "hook" to a pipe driver]
mtu=576	[Maximum Transmission Unit in bytes for internal TWinsock-NOS SLIP link]
rwin=2048	[TCP Receive Window in bytes for internal TWinsock-NOS SLIP link; the corresponding parameter in NOS is called "win."]
mss=512	[TCP Maximum Segment Size in bytes for internal TWinsock-NOS SLIP link. Another stack can send up to 4 (rwin/mss) unacknowledged segments at a time to this station]

```
    time=
    domain=
    rtomax=60
    ip-buffers=32
    pkt-buffers=16
    slip-enabled=0
    slip-port=4
    slip-baudrate=38400
    slip-handshake=1
    slip-rtsflow=0
    slip-compressed=0
    dial-option=0
    online-check=1
    inactivity-timeout=5
    slip-timeout=0
    slip-redial=0
    slip-logging=0
```

```
slip-rcvbuf=8192
slip-sndbuf=8192
dial-parity=0
font=Courier,9
use-socks=0
socks-host=0.0.0.0
socks-port=1080
socks-id=
socks-local1=0.0.0.0 0.0.0.0
socks-local2=0.0.0.0 0.0.0.0
socks-local3=0.0.0.0 0.0.0.0
socks-local4=0.0.0.0 0.0.0.0
ppp-enabled=0
ppp-usepap=0
ppp-usechap=0
ppp-username=""
ppp-password=""
no-close-message=0
use-waitmessage=0
clock-period=100
seen-license=1
ip-routing=0
registration-check="j_\G3Sm!GTZn(92>A+beiJFY"
trace-options=1706
next-port=1515
win-posn=15 19 546 328 -1 -1 -1 -1 1
registration-name="#0%/I+:Gz-7<"
registration-password="h]NE6Xiu"
```

Appendix C

NOS *cwind* and *ssthresh* Adaptation Algorithms

This appendix lists pseudo-code for the algorithms used by NOS to adjust the slow-start threshold (*ssthresh*) and the congestion window (*cwind*) in response to reception of duplicate ACKs by a sender (The Jacobson Algorithm) or a timeout waiting for more expected data by a receiver (includes the Nagle Algorithm). The listings were derived from the NOS C-code modules tcpin.c and tcpout.c in the NOS source distribution. (“//” inaugurates a comment.)

Van Jacobson "Quick Recovery" by Sender Algorithm (C++-style pseudo-code from tcpin.c). Activated if sender receives DUPACK or more duplicate ACKs.

```
 if(dupacks == DUPACKS)                      //dupacks = no. counted ACKs. DUPACKS = 3 is default
                                             //limit.
  {                                          //Duplicate ACKs rcvd; packet probably lost.
                                             //Resend to avoid timeout. MSS = rcvd. MSS.
  ssthresh = max(cwind/2, MSS);              //Reduce threshold as if timeout has occurred,
                                             //since there is congestion.
  tcp_output(tcb);                           //Retransmit lost packet from TCP-layer.
                                             //Inflate congestion window, as if dup ACKs
                                             //had really ACKed packets beyond lost one.
  cwind = ssthresh + DUPACKS*MSS;
  }                                          //End if(dupacks == DUPACKS).

else if(dupacks > DUPACKS)                   //Continue to inflate congestion window until ACKs get "unstuck."
  {
  cwind = cwind + MSS;
  }
                                             //ACKs finally "unstuck." Deflate congestion window
                                             //to where it would have been at end of slow start.
if( (dupacks >= DUPACKS) && (cwind > ssthresh) )
  {
  cwind = ssthresh;
  }

dupacks = 0;                                 //Restart ACK count; ACK must have ACKed a packet.
                                             //Expand cwind if not at limit & rcvd. packet is new.
if( (cwind < send_window) && (no_retran) )
  {                                          //send_window is the offered window.
```

```
  if(cwind < ssthresh)                    //Still doing SS; expand window by number of octets
                                          //ACKed.
    {
  expand = min(acked, MSS);               //acked = number of octets just ACKed.
    }
  else
    {
    expand = (MSS*MSS)/cwind;
    }
                                          //Don't expand beyond offered window.
  if(cwind + expand > send_window)
    {
    expand = send_window - cwind;
    }

  if(expand != 0)                         //Expansion is possible.
    {
    cwind = cwind + expand;               //Expand congestion window.
    }

 }                                        //End if( (cwind < snd.wnd) && (!tcb->flags.retran) ).
```

React to Timeout at Sender Algorithm (C++-style pseudo-code from tcpout.c).
(Activated if sender times out waiting for ACK.)

```
                                          //Transmitter has been idle for longer than an SRTT.
                                          //Reduce cwind to one packet.
if(current_clock() - time_lastactive) > srtt)
  {
  cwind = MSS;
  }

                                          //Calculate usable send window as smaller of offered
                                          //window and cwind, minus amount of data "in flight."
usable = min(send_window, cwind);

if(usable > sent)
  {
  usable = usable - sent;                 //Most common case.
  }
```

```
else if( (usable == 0) && (sent == 0) )        //No data sent & no usable window is apparent.
  {
  usable = 1;                                  //Prepare "closed window probe."
  }

else
  {
  usable = 0;                                  //Window has closed or shrunk.
  }
                                               //Calculate size of window that could be sent.
                                               //This is the smallest of the usable window, the received MSS
                                               //and the amount of data on hand.

segment_size = min(data_on_hand, usable, MSS);

                                               //Apply Nagle's "single outstanding segment" rule:
                                               //If data are in pipeline, don't send more unless
                                               //data are MSS-sized or make up the last packet.
if( (sent != 0) && (segment_size < MSS) && (not_last_segment) )
  {
  segment_size = 0;
  }
ip_send(segment_size + headers);               //(Re-)send data segment via IP-layer if appropriate.
```

Glossary

ACK	**Acknowledgment**
ALE	**Amplitude modulation**
ARP	**Address resolution protocol**
ARQ	**Automatic repeat request**
ASCII	**American standard code for information interchange**
BLIMIT	**Back-off limit**
COTS	**Commercial off-the-shelf**
CRC	**Cyclic redundancy check**
CWIND	**Congestion window**
DCE	**Data communications equipment (e.g., HF modem)**
DNS	**Domain name system**
DTE	**Data terminal equipment (e.g., PC controller)**
FTP	**File transfer protocol**
GATM	**Global Air Traffic Management**
GUI	**Graphical user interface**
HF	**High Frequency**
HTTP	**Hyper-text transfer protocol**
ICMP	**Internet control message protocol**
IP	**Internet protocol**
IRC	**Internet request for comments**
IRTT	**Initial round trip time**
ISO	**International Standardization Organization**
JNOS	**The version of NOS we used**
LAN	**Local-area network**
LZW	**Lempel-Ziv-Welch**
MHz	**Megahertz**

MS	**Military Standard**
ms	**millisecond**
MSS	**Maximum segment size**
MTU	**Maximum transmission unit**
MUF	**Maximum usable frequency**
NACK	**Non-acknowledgment**
NOS	**Net operating system**
NVIS	**Near-vertical-incidence skywave**
PAD	**Packet assembler-de-assembler**
PID	**Protocol ID**
POP	**Post office protocol**
PSK	**Phase-shift keying**
RFC	**Request for comment (Internet protocol specification)**
RTO	**Retransmission time-out**
RTT	**Round trip-times**
SLIP	**Serial line internet protocol**
SMTP	**Simple mail transfer protocol**
SCPS	**Space Communications Protocol Specification**
SNR	**Signal-to-noise ratio**
SRTT	**Smothed retransmission time-out**
SSTHRESH	**Slow-start threshold**
TCP	**Transmission control protocol**
TSR	**Terminate-and-stay-resident**
UDP	**User datagram protocol**
WIN	**Window size**
WWW	**Worldwide Web**