Network Working Group
Request for Comments: 1986
Category: Experimental

W. Polites
W. Wollman
D. Woo
The MITRE Corporation
R. Langan
U.S. ARMY CECOM

August 1996

Experiments with a Simple File Transfer Protocol for Radio Links using Enhanced Trivial File Transfer Protocol (ETFTP)

Status of this Memo

This memo defines an Experimental Protocol for the Internet community. This memo does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

1. INTRODUCTION SECTION

This document is a description of the Enhanced Trivial File Transfer Protocol (ETFTP). This protocol is an experimental implementation of the NETwork BLock Transfer Protocol (NETBLT), RFC 998 [1], as a file transfer application program. It uses the User Datagram Protocol (UDP), RFC 768 [2], as its transport layer. The two protocols are layered to create the ETFTP client server application. The ETFTP program is named after Trivial File Transfer Protocol (TFTP), RFC 1350 [3], because the source code from TFTP is used as the building blocks for the ETFTP program. This implementation also builds on but differs from the work done by the National Imagery Transmission Format Standard [4].

This document is published for discussion and comment on improving the throughput performance of data transfer utilities over Internet Protocol (IP) compliant, half duplex, radio networks.

There are many file transfer programs available for computer networks. Many of these programs are designed for operations through high-speed, low bit error rate (BER) cabled networks. In tactical radio networks, traditional file transfer protocols, such as File Transfer Protocol (FTP) and TFTP, do not always perform well. This is primarily because tactical half duplex radio networks typically provide slow-speed, long delay, and high BER communication links. ETFTP is designed to allow a user to control transmission parameters to optimize file transfer rates through half-duplex radio links.

The tactical radio network used to test this application was developed by the Survivable Adaptive Systems (SAS) Advanced Technology Demonstration (ATD). Part of the SAS ATD program was to address the problems associated with extending IP networks across tactical radios. Several tactical radios, such as, SINgle Channel Ground and Airborne Radio Systems (SINCGARS), Enhanced Position Location Reporting Systems (EPLRS), Motorola LST-5C, and High Frequency (HF) radios have been interfaced to the system. This document will discuss results obtained from using ETFTP across a point-to-point LST-5C tactical SATellite COMmunications (SATCOM) link. The network includes a 25 Mhz 486 Personal Computer (PC) called the Army Lightweight Computer Unit (LCU), Cisco 2500 routers, Gracilis PackeTen Network switches, Motorola Sunburst Cryptographic processors, a prototype forward error correction (FEC) device, and Motorola LST-5C tactical Ultra High Frequency (UHF) satellite communications (SATC! OM) radio. Table 1, "Network Trans fer Rates," describes the equipment network connections and the bandwidth of the physical media interconnecting the devices.

Table 1: Network Transfer Rates

Equipment	Rate (bits per second)
Host Computer (486 PC)	10,000,000 Ethernet
Cisco Router	10,000,000 Ethernet to 19,200 Serial Line Internet Protocol (SLIP)
Gracilis PackeTen	19,200 SLIP to 16,000 Amateur Radio (AX.25)
FEC	half rate or quarter rate
Sunburst Crypto	16,000
LST-5C Radio	16,000

During 1993, the MITRE team collected data for network configurations that were stationary and on-the-move. This network configuration did not include any Forward Error Correction (FEC) at the link layer. Several commercially available implementations of FTP were used to transfer files through a 16 kbps satellite link. FTP relies upon the Transmission Control Protocol (TCP) for reliable communications. For a variety of file sizes, throughput measurements ranged between 80 and 400 bps. At times, TCP connections could be opened, however, data

transfers would be unsuccessful. This was most likely due to the smaller TCP connection synchronization packets, as compared to the TCP data packets. Because of the high bit error rate of the link, the smaller packets were much more likely to be received without error. In most cases, satellite channel utilization was less than 20 percent. Very often a file transfer would fail because FTP implementations would curtail the transfer due t! o the poor conditions of the commu nication link.

The current focus is to increase the throughput and channel utilization over a point to point, half duplex link. Follow on experiments will evaluate ETFTP's ability to work with multiple hosts in a multicast scenario. Evaluation of the data collected helped to determine that several factors limited data throughput. A brief description of those limiting factors, as well as, solutions that can reduce these networking limitations is provided below.

Link Quality

The channel quality of a typical narrow-band UHF satellite link does not sufficiently support data communications without the addition of a forward error correction (FEC) capability. From the data collected, it was determined that the UHF satellite link supports, on average, a 10e-3 bit error rate.

Solution: A narrow-band UHF satellite radio FEC prototype was developed that improves data reliability, without excessively increasing synchronization requirements. The prototype FEC increased synchronization requirements by less than 50 milliseconds (ms). The FEC implementation will improve an average 10e-3 BER channel to an average 10e-5 BER channel.

Delays

Including satellite propagation delays, the tactical satellite radios require approximately 1.25 seconds for radio synchronization prior to transmitting any data across the communication channel. Therefore, limiting the number of channel accesses required will permit data throughput to increase. This can be achieved by minimizing the number of acknowledgments required during the file transfer. FTP generates many acknowledgments which decreases throughput by increasing the number of satellite channel accesses required.

To clarify, when a FTP connection request is generated, it is sent via Ethernet to the router and then forwarded to the radio network controller (RNC). The elapsed time is less than 30 ms. The RNC keys the crypto unit and 950 ms later modem/crypto synchronization occurs. After synchronization is achieved, the FTP connection request is

transmitted. The transmitting terminal then drops the channel and the modem/crypto synchronization is lost. Assuming that the request was received successfully, the receiving host processes the request and sends an acknowledgment. Again the modem/crypto have to synchronize prior to transmitting the acknowledgment. Propagation delays over a UHF satellite also adds roughly 500 ms to the total round trip delay.

Solution: When compared to FTP, NETBLT significantly reduces the number of acknowledgments required to complete a file transfer. Therefore, leveraging the features available within an implementation of NETBLT will significantly improve throughput across the narrow-band UHF satellite communication link.

To reduce the number of channel accesses required, a number of AX.25 parameters were modified. These included the value of p for use within the p-persistence link layer protocol, the slot time, the transmit tail time, and the transmit delay time. The p-persistence is a random number threshold between 0 and 255. The slot time is the time to wait prior to attempting to access the channel. The transmit tail increases the amount of time the radio carrier is held on, prior to dropping the channel. Transmit delay is normally equal to the value of the radio synchronization time. By adjusting these parameters to adapt to the tactical satellite environment, improved communication performance can be achieved.

First, in ETFTP, several packets within a buffer are transmitted within one burst. If the buffer is partitioned into ten packets, each of 1024 bytes, then 10,240 bytes of data is transmitted with each channel access. It is possible to configure ETFTP’s burstsize to equal the number of packets per buffer. Second, the transmit tail time was increased to hold the key down on the transmitter long enough to insure all of the packets within the buffer are sent in a single channel access. These two features, together, allow the system to transmit an entire file (example, 100,000 bytes) with only a single channel access by adjusting buffer size. Thirdly, the ETFTP protocol only acknowledges each buffer, not each packet. Thus, a single acknowledgment is sent from the receiving terminal containing a request for the missing packets within each buffer, reducing the number of acknowledgment packets sent. Which in turn, reduced the number of times the channel has to be turned around.

To reduce channel access time, p-persistence was set to the maximum value and slot time to a minimum value. These settings support operations for a point-to-point communication link between two users. This value of p would not be used if more users were sharing the satellite channel.

Backoffs

TCP's slow start and backoff algorithms implemented in most TCP packages assume that packet loss is due to network congestion. When operating across a tactical half duplex communication channel dedicated to two users, packet loss is primarily due to poor channel quality, not network congestion. A linear backoff at the transport layer is recommended. In a tactical radio network there are numerous cases where a single host is connected to multiple radios. In a tactical radio network, layer two will handle channel access. Channel access will be adjusted through parameters like p-persistence and slot time. The aggregate effect of the exponential backoff from the transport layer added to the random backoff of the data link layer, will in most cases, cause the radio network to miss many network access opportunities. A linear backoff will reduce the number missed data link network access opportunities

Solution: Tunable parameters and timers have been modified to resemble those suggested by NETBLT.

Packet Size

In a tactical environment, channel conditions change rapidly. Continuously transmitting large packets under 10e-3 BER conditions reduces effective throughput.

Solution: Packet sizes are dynamically adjusted based upon the success of the buffer transfers. If 99 percent of all packets within a buffer are received successfully, packet size can be increased to a negotiated value. If 50 percent or more of all packets within a buffer are not successfully delivered, the packet size can be decreased to a negotiated value.

2. PROTOCOL DESCRIPTION

Throughout this document the term packet is used to describe a datagram that includes all network overhead. A block is used to describe information, without any network encapsulation.

The original source files for TFTP, as downloaded from ftp.uu.net, were modified to implement the ETFTP/NETBLT protocol. These same files are listed in "UNIX Network Programming" [5].

ETFTP was implemented for operations under the Santa Cruz Operations (SCO) UNIX. In the service file, "/etc/services", an addition was made to support "etftp" at a temporary well known port of "1818" using "UDP" protocol. The file, "/etc/inetd.conf", was modified so the "inetd" program could autostart the "etftpd" server when a

connection request came in on the well known port.

As stated earlier, the transport layer for ETFTP is UDP, which will not be discussed further here. This client server application layer protocol is NETBLT, with four notable differences.

The first change is that this NETBLT protocol is implemented on top of the UDP layer. This allowed the NETBLT concepts to be tested without modifying the operating system's transport or network layers. Table 2, "Four Layer Protocol Model," shows the protocol stack for FTP, TFTP and ETFTP.

Table 2: Four Layer Protocol Model

PROTOCOL STACK			
APPLICATION	FTP	TFTP	ETFTP/NETBLT
TRANSPORT	TCP	UDP	UDP
NETWORK	IP		
LINK	Ethernet, SLIP, AX.25		

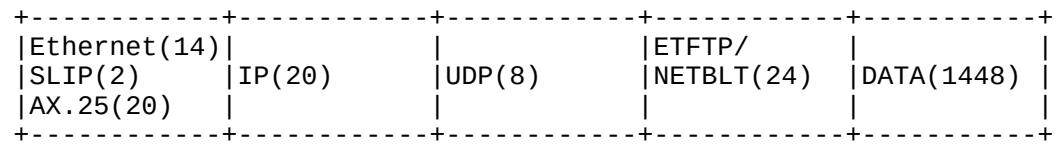
The second change is a carryover from TFTP, which allows files to be transferred in netascii or binary modes. A new T bit flag is assigned to the reserved field of the OPEN message type.

The third change is to re-negotiate the DATA packet size. This change affects the OPEN, NULL-ACK, and CONTROL_OK message types. A new R bit is assigned to the reserved field of the OPEN message type.

The fourth change is the addition of two new fields to the OPEN message type. The one field is a two byte integer for radio delay in seconds, and the next field is two bytes of padding.

The ETFTP data encapsulation is shown in Table 3, "ETFTP Data Encapsulation,". The Ethernet, SLIP, and AX.25 headers are mutually exclusive. They are stripped off and added by the appropriate hardware layer.

Table 3: ETFTP Data Encapsulation



2.1 MESSAGE TYPES AND FORMATS

Here are the ETFTP/NETBLT message types and formats.

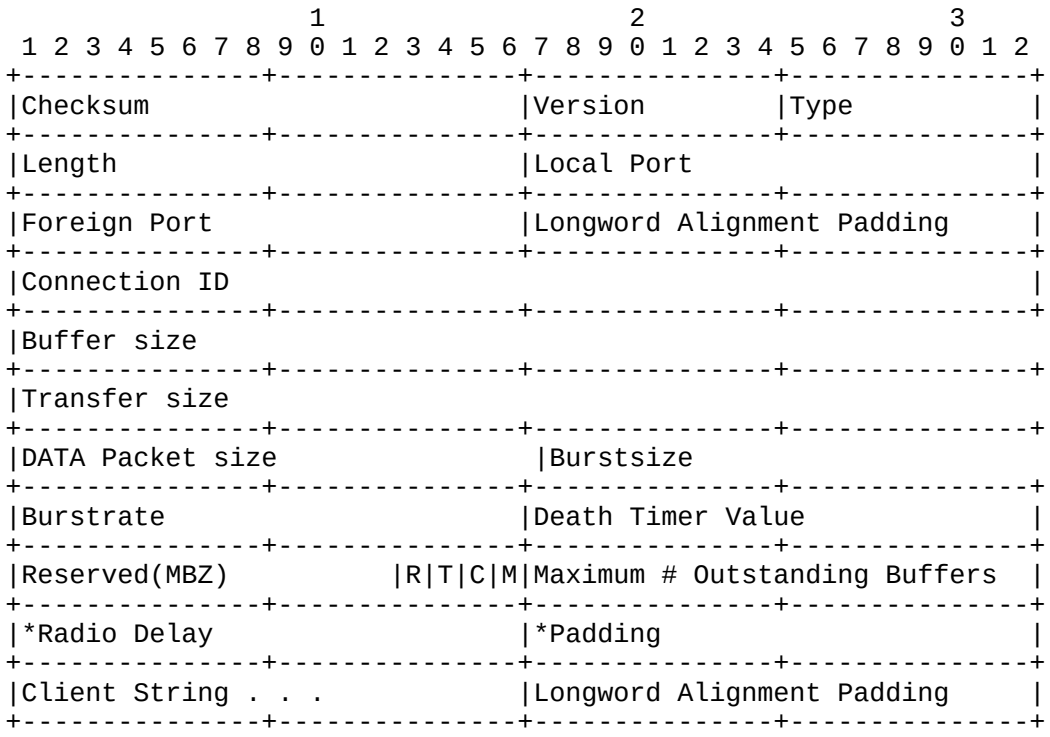
```
MESSAGES        VALUES
OPEN    0  Client request to open a new connection
RESPONSE        1  Server positive acknowledgment for OPEN
KEEPALIVE       2  Reset the timer
QUIT    3  Sender normal Close request
QUITACK 4  Receiver acknowledgment of QUIT
ABORT   5  Abnormal close
DATA    6  Sender packet containing data
LDATA   7  Sender last data block of a buffer
NULL-ACK        8  Sender confirmation of CONTROL_OK changes
CONTROL 9  Receiver request to
        GO      0 Start transmit of next buffer
        OK      1 Acknowledge complete buffer
        RESEND  2 Retransmit request
REFUSED 10 Server negative acknowledgment of OPEN
DONE    11 Receiver acknowledgment of QUIT.
```

Packets are "longword-aligned", at four byte word boundaries. Variable length strings are NULL terminated, and padded to the four byte boundary. Fields are listed in network byte order. All the message types share a common 12 byte header. The common fields are:

```
Checksum        IP compliant checksum
Version Current version ID
Type    NETBLT message type
Length  Total byte length of packet
Local Port      My port ID
Foreign Port    Remote port ID
Padding Pad as necessary to 4 byte boundary
```

The OPEN and RESPONSE messages are similar and shown in Table 4, "OPEN and RESPONSE Message Types,". The client string field is used to carry the filename to be transferred.

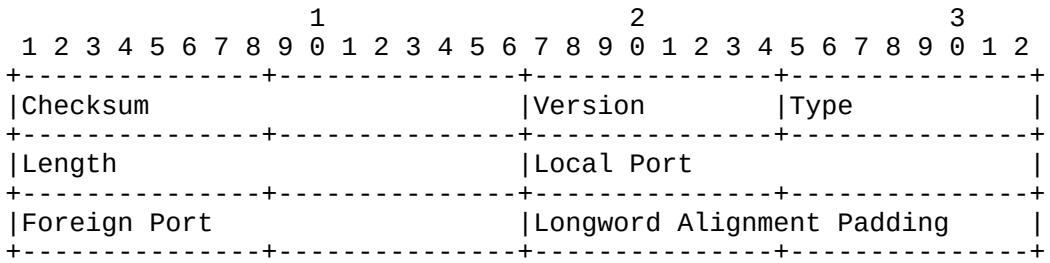
Table 4: OPEN and RESPONSE Message Types



Connection ID The unique connection number
Buffer size Bytes per buffer
Transfer size The length of the file in bytes
DATA Packet size Bytes per ETFTP block
Burstsize Concatenated packets per burst
Burstrate Milliseconds per burst
Death Timer Seconds before closing idle links
Reserved M bit is mode: 0=read/put, 1=write/get
C bit is checksum: 0=header, 1=all
*T bit is transfer: 0=netascii, 1=binary
*R bit is re-negotiate: 0=off, 1=on
Max # Out Buffs Maximum allowed un-acknowledged buffers
Radio Delay *Seconds of delay from send to receive
Padding *Unused
Client String Filename.

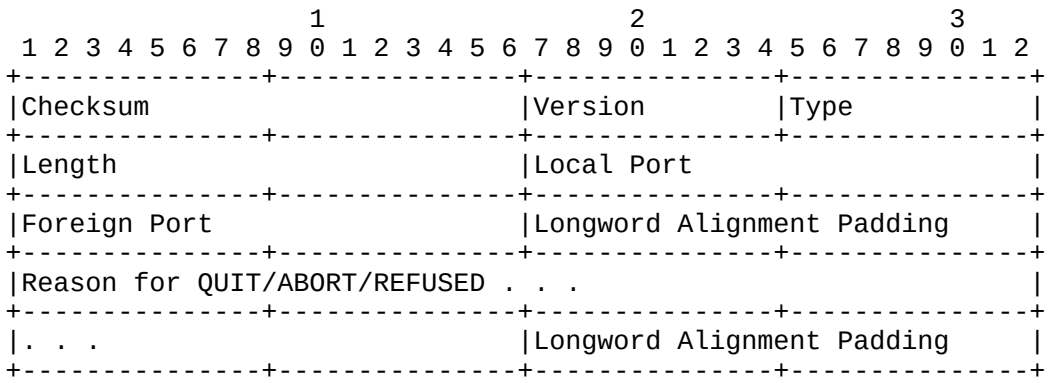
The KEEPALIVE, QUITACK, and DONE messages are identical to the common header, except for the message type values. See Table 5, "KEEPALIVE, QUITACK, and DONE Message Types,".

Table 5: KEEPALIVE, QUITACK, and DONE Message Types



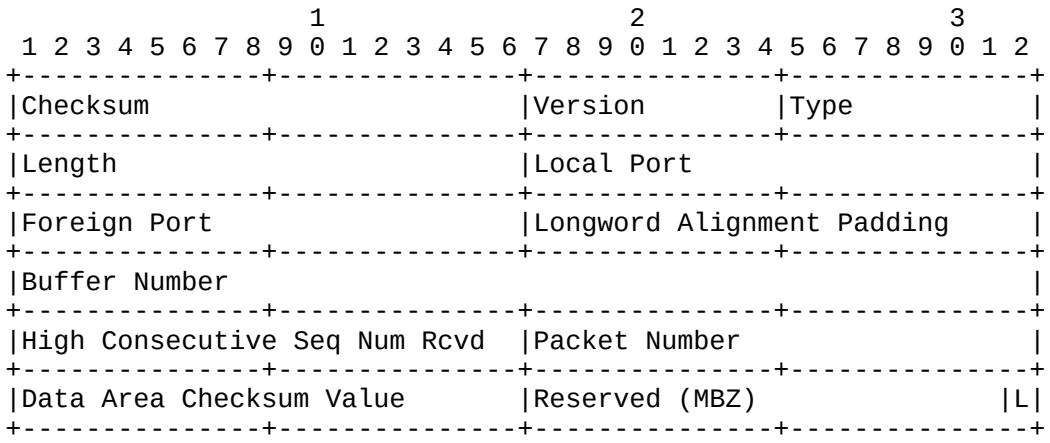
The QUIT, ABORT, and REFUSED messages allow a string field to carry the reason for the message. See Table 6, "QUIT, ABORT, and REFUSED Message Types,".

Table 6: QUIT, ABORT, and REFUSED Message Types



The DATA and LDATA messages make up the bulk of the messages transferred. The last packet of each buffer is flagged as an LDATA message. Each and every packet of the last buffer has the reserved L bit set. The highest consecutive sequence number is used for the acknowledgment of CONTROL messages. It should contain the ID number of the current CONTROL message being processed. Table 7, "DATA and LDATA Message Types,", shows the DATA and LDATA formats.

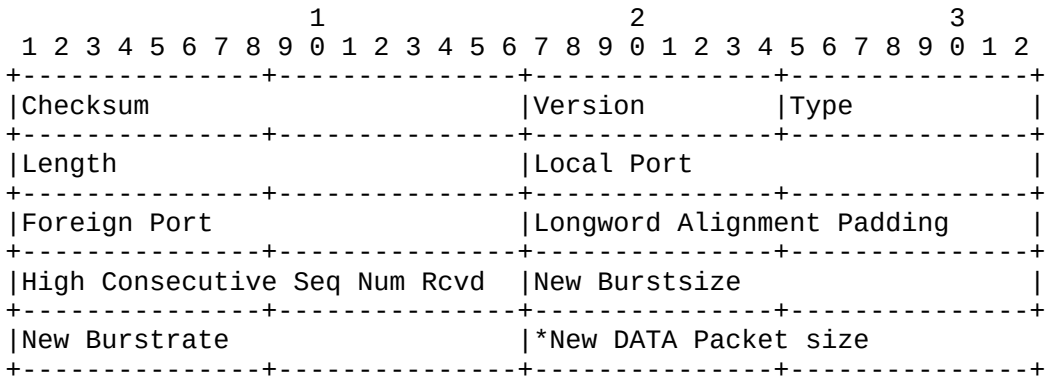
Table 7: DATA and LDATA Message Types



Buffer Number	The first buffer number starts at 0
Hi Con Seq Num	The acknowledgment for CONTROL messages
Packet Number	The first packet number starts at 0
Data Checksum	Checksum for data area only
Reserved	L: the last buffer bit: 0=false, 1=true

The NULL-ACK message type is sent as a response to a CONTROL_OK message that modifies the current packet size, burstsize, or burstrate. In acknowledging the CONTROL_OK message, the sender is confirming the change request to the new packet size, burstsize, or burstrate. If no modifications are requested, a NULL-ACK message is unnecessary. See Table 8, "NULL-ACK Message Type," for further details.

Table 8: NULL-ACK Message Type



The CONTROL messages have three subtypes: GO, OK, and RESEND as shown in Tables 9-12. The CONTROL message common header may be followed by any number of longword aligned subtype messages.

Table 9: CONTROL Message Common Header

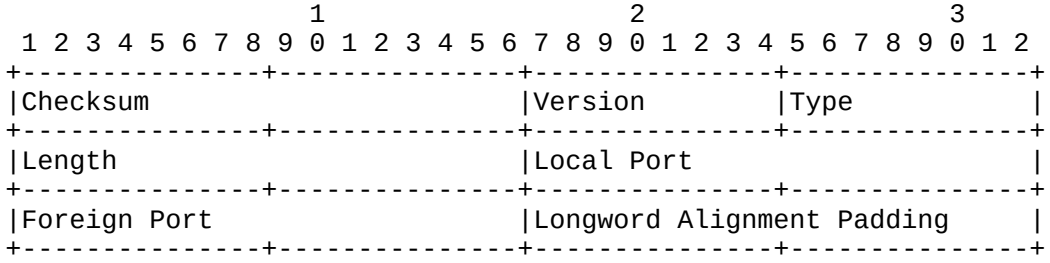


Table 10: CONTROL_GO Message Subtype

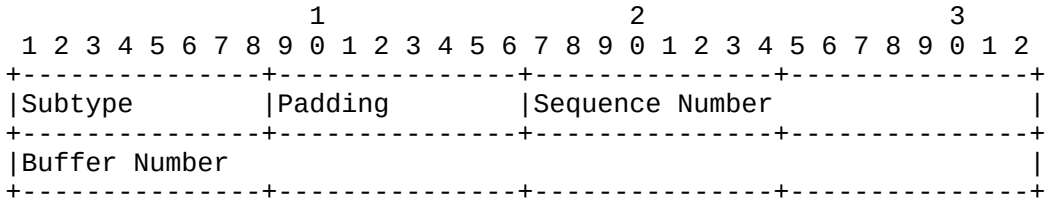


Table 11: CONTROL_OK Message Subtype

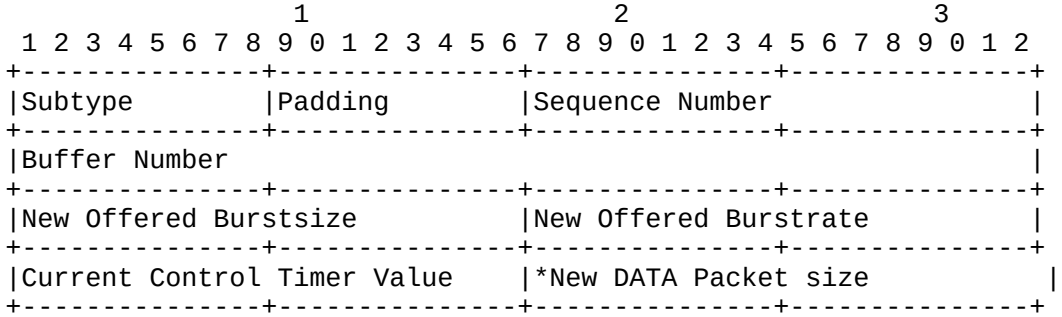
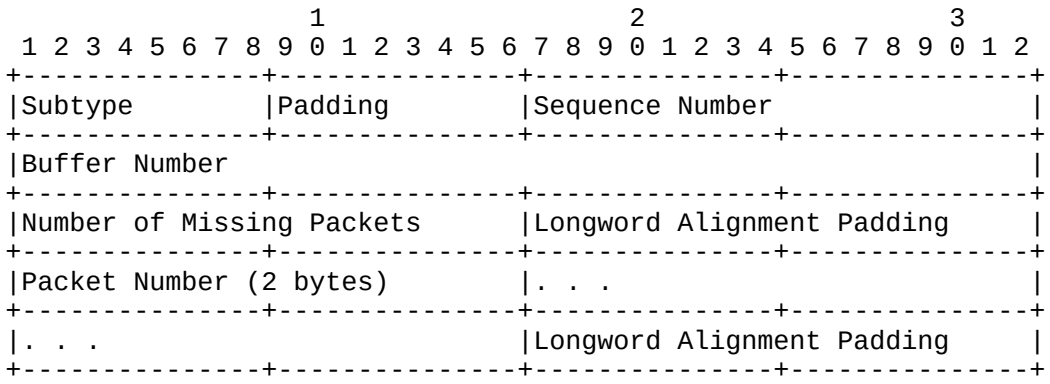


Table 12: CONTROL_RESEND Message Subtype



2.2 ETFTP COMMAND SET

Being built from TFTP source code, ETFTP shares a significant portion of TFTP's design. Like TFTP, ETFTP does NOT support user password validation. The program does not support changing directories (i.e. cd), neither can it list directories, (i.e. ls). All filenames must be given in full paths, as relative paths are not supported. The internal finite state machine was modified to support NETBLT message types.

The NETBLT protocol is implemented as closely as possible to what is described in RFC 998, with a few exceptions. The client string field in the OPEN message type is used to carry the filename of the file to be transferred. Netascii or binary transfers are both supported. If enabled, new packet sizes, burstsizes, and burstrates are re-negotiated downwards when half or more of the blocks in a buffer require retransmission. If 99% of the packets in a buffer is successfully transferred without any retransmissions, packet size is re-negotiated upwards.

The interactive commands supported by the client process are similar to TFTP. Here is the ETFTP command set. Optional parameters are in square brackets. Presets are in parentheses.

```
?       help, displays command list
ascii   mode ascii, appends CR-LF per line
autoadapt       toggles backoff function (on)
baudrate baud   baud rate (16000 bits/sec)
binary  mode binary, image transfer
blocksize bytes packet size in bytes (512 bytes/block)
bufferblock blks        buffer size in blocks (128 blocks/buff)
burstsize packets       burst size in packets (8 blocks/burst)
```

```
connect host [p]        establish connection with host at port p
exit    ends program
get rfile lfile copy remote file to local file
help    same as ?
mode choice     set transfer mode (binary)
multibuff num   number of buffers (2 buffers)
put lfile rfile copy local file to remote file
quit    same as exit
radiodelay sec  transmission delay in seconds (2 sec)
status  display network parameters
trace   toggles debug display (off).
```

2.3 DATA TRANSFER AND FLOW CONTROL

This is the scenario between client and server transfers:

Client sends OPEN for connection, blocksize, buffersize, burstsize, burstrate, transfer mode, and get or put. See M bit of reserved field.

Server sends a RESPONSE with the agreed parameters.

Receiver sends a CONTROL_GO request sending of first buffer.

Sender starts transfer by reading the file into multiple memory buffers. See Figure 1, "File Segmentation,". Each buffer is divided according to the number of bytes/block. Each block becomes a DATA packet, which is concatenated according to the blocks/burst. Bursts are transmitted according to the burstrate. Last data block is flagged as LDATA type.

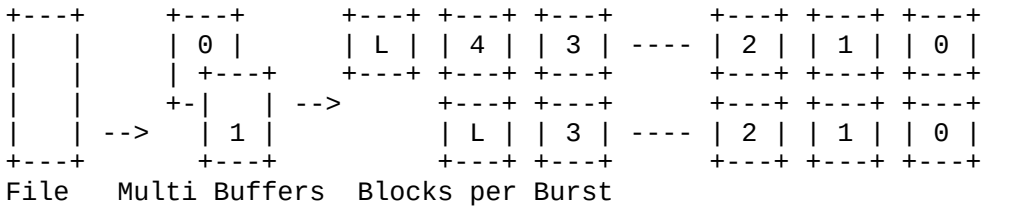


Figure 1. File Segmentation

Receiver acknowledges buffer as CONTROL_OK or CONTROL_RESEND.

If blocks are missing, a CONTROL_RESEND packet is transmitted. If half or more of the blocks in a buffer are missing, an adaptive algorithm is used for the next buffer transfer. If no blocks are missing, a CONTROL_OK packet is transmitted.

Sender re-transmits blocks until receipt of a CONTROL_OK. If the adaptive algorithm is set, then new parameters are offered, in the CONTROL_OK message. The priority of the adaptive algorithm is:

- Reduce packetsize by half (MIN = 16 bytes/packet)
- Reduce burstsize by one (MIN = 1 packet/burst)
- Reduce burstrate to actual tighttimer rate

If new parameters are valid, the sender transmits a NULL-ACK packet, to confirm the changes.

Receiver sends a CONTROL_GO to request sending next buffer.

At end of transfer, sender sends a QUIT to close the connection.

Receiver acknowledges the close request with a DONE packet.

2.4 TUNABLE PARAMETERS

These parameters directly affect the throughput rate of ETFTP.

Packetsize The packetsize is the number of 8 bit bytes per packet. This number refers to the user data bytes in a block, (frame), exclusive of any network overhead. The packet size has a valid range from 16 to 1,448 bytes. The Maximum Transfer Unit (MTU) implemented in most commercial network devices is 1,500 bytes. The de-facto industry standard is 576 byte packets.

Bufferblock The bufferblock is the number of blocks per buffer. Each implementation may have restrictions on available memory, so the buffersize is calculated by multiplying the packetsize times the bufferblocks.

Baudrate The baudrate is the bits per second transfer rate of the slowest link (i.e., the radios). The baudrate sets the speed of the sending process. The sending process cannot detect the actual speed of the network, so the user must set the correct baudrate.

Burstsize The burstsize in packets per burst sets how many packets are concatenated and burst for transmission efficiency. The burstsize times the packetsize must not exceed the available memory of any intervening network devices. On the Ethernet portion of the network, all the packets are sent almost instantaneously. It is necessary to wait for the network to drain down its memory buffers, before the next burst is sent. The sending process needs to regulate the rate used to place packets into the network.

Radiodelay The radiodelay is the time in seconds per burst it takes to synchronize with the radio controllers. Any additional hardware delays should be set here. It is the aggregate delay of the link layer, such as transmitter key-up, FEC, crypto synchronization, and propagation delays.

These parameters above are used to calculate a burstrate, which is the length of time it takes to transmit one burst. The ov is the overhead of 72 bytes per packet of network encapsulation. A byte is defined as 8 bits. The burstrate value is:

```
burstrate = (packetsize+ov)*burstsize*8/baudrate
```

In a effort to calculate the round trip time, when data is flowing in one direction for most of the transfer, the OPEN and RESPONSE message types are timed, and the tactical radio delays are estimated. Using only one packet in each direction to estimate the rate of the link is statistically inaccurate. It was decided that the radio delay should be a constant provided by the user interface. However, a default value of 2 seconds is used. The granularity of this value is in seconds because of two reasons. The first reason is that the UNIX supports a sleep function in seconds only. The second reason is that in certain applications, such as deep space probes, a 16-bit integer maximum of 32,767 seconds would suffice.

2.5 DELAYS AND TIMERS

From these parameters, several timers are derived. The control timer is responsible for measuring the per buffer rate of transfer. The SENDER copy is nicknamed the loosetimer.

```
loosetimer = (burstrate+radiodelay)*bufferblock/burstsize
```

The RECEIVER copy of the timer is nicknamed the tighttimer, which measures the elapsed time between CONTROL_GO and CONTROL_OK packets. The tighttimer is returned to the SENDER to allow the protocol to adjust for the speed of the network.

The retransmit timer is responsible for measuring the network receive data function. It is used to set an alarm signal (SIGALRM) to interrupt the network read. The retransmit timer (wait) is initially set to be the greater of twice the round trip or 4 times the radiodelay, plus a constant 5 seconds.

wait = MAX (2*roundtriptime, 4*radiodelay) + 5 seconds

and

alarm timeout = wait.

Each time the same read times out, a five second backoff is added to the next wait. The backoff is necessary because the initial user supplied radiodelay, or the initial measured round trip time may be incorrect.

The retransmit timer is set differently for the RECEIVER during a buffer transfer. Before the arrival of the first DATA packet, the original alarm time out is used. Once the DATA packets start arriving, and for the duration of each buffer transfer, the RECEIVER alarm time out is reset to the expected arrival time of the last DATA packet (blockstogo) plus the delay (wait). As each DATA packet is received, the alarm is decremented by one packet interval. This same algorithm is used for receiving missing packets, during a RESEND.

alarmtimeout = blockstogo*burstrate/burstsize + wait

The death timer is responsible for measuring the idle time of a connection. In the ETFTP program, the death timer is set to be equal to the accumulated time of ten re-transmissions plus their associated backoffs. As such, the death timer value in the OPEN and RESPONSE message types is un-necessary. In the ETFTP program, this field could be used to transfer the radio delay value instead of creating the two new fields.

The keepalive timer is responsible for resetting the death timer. This timer will trigger the sending of a KEEPALIVE packet to prevent the remote host from closing a connection due to the expiration of its death timer. Due to the nature of the ETFTP server process, a keepalive timer was not necessary, although it is implemented.

2.6 TEST RESULTS

The NETBLT protocol has been tested on other high speed networks before, see RFC 1030 [6]. These test results in Tables 13 and 14, "ETFTP Performance," were gathered from files transferred across the network and LST-5C TACSAT radios. The radios were connected together via a coaxial cable to provide a "clean" link. A clean link is defined to a BER of 10e-5. The throughput rates are defined to be the file size divided by the elapsed time resulting in bits per second (bps). The elapsed time is measured from the time of the "get" or "put" command to the completion of the transfer. This is an all inclusive time measurement based on user perspective. It includes the

connection time, transfer time, error recovery time, and disconnect time. The user concept of elapsed time is the length of time it takes to copy a file from disk to disk. These results show only the average performances, including the occasional packet re-transmissions. The network configuration was set as:

ETFTP Parameters:

```
Filesize                101,306 bytes
Radiodelay      2 seconds
Buffersize      16,384-131,072 bytes
Packetsize      512-2048 bytes
Burstsize               8-16 packets/burst
```

Gracilis PackeTen Parameters:

```
0 TX Delay      400 milliseconds
1 P Persist     255 [range 1-255]
2 Slot Time     30 milliseconds
3 TX Tail               300 milliseconds
4 Rcv Buffers   8 2048 bytes/buffer
5 Idle Code     Flag
```

Radio Parameters:

```
Baudrate                16,000 bps
Encryption      on
```

Table 13: ETFTP Performance at 8 Packets/Burst in Bits/Second

buffersize (bytes)	packetsize 2,048 bytes	packetsize 1,448 bytes	packetsize 1,024 bytes	packetsize 512 bytes
16,384	7,153	6,952	6,648	5,248
32,768	7,652	7,438	7,152	4,926
65,536	8,072	8,752	8,416	5,368
131,072	8,828	9,112	7,888	5,728

Table 14: ETFTP Performance at 16 Packets/Burst in Bits/Second

buffersize (bytes)	packetsize 2,048 bytes	packetsize 1,448 bytes	packetsize 1,024 bytes	packetsize 512 bytes
16,384	5,544	5,045	4,801	4,570
32,768	8,861	8,230	8,016	7,645
65,536	9,672	9,424	9,376	8,920
131,072	10,432	10,168	9,578	9,124

2.7 PERFORMANCE CONSIDERATIONS

These tests were performed across a tactical radio link with a maximum data rate of 16000 bps. In testing ETFTP, it was found that the delay associated with the half duplex channel turnaround time was the biggest factor in throughput performance. Therefore, every attempt was made to minimize the number of times the channel needed to be turned around. Obviously, the easiest thing to do is to use as big a buffer as necessary to read in a file, as acknowledgments occurred only at the buffer boundaries. This is not always feasible, as available storage on disk could easily exceed available memory. However, the current ETFTP buffersize is set at a maximum of 524,288 bytes.

The larger packetsizes also improved performance. The limit on packetsize is based on the 1500 byte MTU of network store and forward devices. In a high BER environment, a large packetsize could be detrimental to success. By reducing the packetsize, even though it negatively impacts performance, reliability is sustained. When used in conjunction with FEC, both performance and reliability can be maintained at an acceptable level.

The burstsize translates into how long the radio transmitters are keyed to transmit. In ETFTP, the ideal situation is to have the first packet of a burst arrive in the radio transmit buffer, as the last packet of the previous burst is just finished being sent. In this way, the radio transmitter would never be dropped for the duration of one buffer. In a multi-user radio network, a full buffer transmission would be inconsiderate, as the transmit cycle could last for several minutes, instead of seconds. In measuring voice communications, typical transmit durations are on the order of five to twenty seconds. This means that the buffersize and burstsize could be adjusted to have similar transmission durations.

3. REFERENCE SECTION

[1] Clark, D., Lambert, M., and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol", RFC 998, MIT, March 1987.

[2] Postel, J., "User Datagram Protocol" STD 6, RFC 768, USC/Information Sciences Institute, August 1980.

[3] Sollins, K., "Trivial File Transfer Protocol", STD 33, RFC 1350, MIT, July 1992.

[4] MIL-STD-2045-44500, 18 June 1993, "Military Standard Tactical Communications Protocol 2 (TACO 2) fot the National Imagery Transmission Format Standard", Ft. Monmouth, New Jersey.

[5] Stevens, W. Richard, 1990, "UNIX Network Programming", Prentice-Hall Inc., Englewood, New Jersey, Chapter 12.

[6] Lambert, M., "On Testing the NETBLT Protocol over Divers Networks", RFC 1030, MIT, November 1987.

4. SECURITY CONSIDERATIONS

The ETFTP program is a security loophole in any UNIX environment. There is no user/password validation. All the problems associated to TFTP are repeated in ETFTP. The server program must be owned by root and setuid to root in order to work. As an experimental prototype program, the security issue was overlooked. Since this protocol has proven too be a viable solution in tactical radio networks, the security issues will have to be addressed, and corrected.

5. AUTHORS' ADDRESSES

William J. Polites
The Mitre Corporation
145 Wyckoff Rd.
Eatontown, NJ 07724

Phone: (908) 544-1414
EMail:wpolites@mitre.org

William Wollman
The Mitre Corporation
145 Wyckoff Rd.
Eatontown, NJ 07724

Phone: (908) 544-1414
EMail:wwollman@mitre.org

David Woo
The Mitre Corporation
145 Wyckoff Rd.
Eatontown, NJ 07724

Phone: (908) 544-1414
EMail: dwoo@mitre.org

Russ Langan
U.S. Army Communications Electronics Command (CECOM)
AMSEL-RD-ST-SP
ATTN: Russell Langan
Fort Monmouth, NJ 07703

Phone: (908) 427-2064
Fax: (908) 427-2822
EMail: langanr@doim6.monmouth.army.mil

6. GLOSSARY

ATD	Advanced Technology Demonstration
AX.25	Amateur Radio X.25 Protocol
BER	Bit Error Rate
EPLRS	Enhanced Position Location Reporting Systems
ETFTP	Enhanced Trivial File Transfer Protocol
FEC	Forward Error Correction
FTP	File Transfer Protocol
HF	High Frequency
LCU	Lightweight Computer Unit
ms	milliseconds
MTU	Maximum Transfer Unit
NETBLT	NETwork Block Transfer protocol
NITFS	National Imagery Transmission Format Standard
PC	Personal Computer
RNC	Radio Network Controller
SAS	Survivable Adaptive Systems
SATCOM	SATellite COMmunications
SCO	Santa Cruz Operations
SINCGARS	SINgle Channel Ground and Airborne Radio Systems
SLIP	Serial Line Internet Protocol
TACO2	Tactical Communications Protocol 2
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
UHF	Ultra High Frequency

* Modification from NETBLT RFC 998.
* The new packet size is a modification to the NETBLT RFC 998.
* The new packet size is a modification to the NETBLT RFC 998.