Network Working Group   R. Troth
Request for Comments: 1440   Rice University
                                                               July 1993

SIFT/UFT: Sender-Initiated/Unsolicited File Transfer

Status of this Memo

This memo defines an Experimental Protocol for the Internet community. It does not specify an Internet standard. Discussion and suggestions for improvement are requested. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

1. Introduction

This document describes a Sender-Initiated File Transfer (SIFT) protocol, also commonly called Unsolicited File Transfer (UFT) protocol. The acronyms SIFT and UFT are synonymous throughout this document. The term "unsolicited" does not imply that the file is unwanted, but that the receiver did not initiate the transaction.

Sender-Initiated File Transfer contrasts with other file transfer methods in that the sender need not have an account or any registration on the target host system, and the receiving user may have less steps to take to retrieve the file(s) sent. Unlike traditional file transfer, UFT lends itself handily to background or deferred operation, though it may be carried out immediately, even interactively.

2. Rationale

In certain non-IP networks, notably NJE based networks such as BITNET, it is possible to send a file to another user outside of the realm of "mail". The effect is that the file sent is not perceived as correspondence and not processed by a mail user agent. This convenient service is missed in the standard TCP/IP suite. The author maintains that traditional electronic mail is not suited to non-correspondence file transfer. There should be a means of sending non-mail, analogous to the sending of parcels rather than surface mail. Several groups and individuals have shown an interest in this type of service.

3. Specification

We define sender-initiated file transfer for IP as a TCP service as follows: a receiver program (the server or "daemon") listens on port 608 for inbound connections. Client programs connect to this port and send a sequence of commands followed by a stream of data. The entire job stream may be thought of as the concatenation of two files, 1) a control file, and 2) a data file, where the control file is plain text and the data file may be any of several formats, but is stored and sent as binary. After each command, the receiver either ACKs (signals positive acknowledgement) or NAKs (signals negative acknowledgement). The target host may reject a file for various reasons, most obvious being 1) that there is no local user matching the intended user, or 2) that there is not enough space to hold the incoming file.

Most UFT commands are parametric. That is, they don’t necessarily invoke an action as much as change parameters of the one action, transfer of the file(s) being sent. This means that UFT is suitable for encapsulation in some higher-level "envelope", such as mail. However, the obvious prefered medium for UFT is TCP.

When files arrive at the destination host, they are kept in a public area, say /usr/spool/uft, until accepted or rejected by the recipient user or discarded for age by the system. This staging area is public in the sense of shared space, not unrestricted access. Exactly how long files may remain unprocessed and exactly how large these transient files may be is a local administrative or implementation decision.

But not all hosts have IP connectivity; not all hosts will want to put up yet another server; not all hosts will be on the unrestricted side of a "fire wall" that only passes mail. In such cases, UFT may be transported via MIME (Multipurpose Internet Mail Extensions) as Content-Type: application/octet-stream. UFT commands then become parameters to the Content-Type field and the data file is carried as the mail body. While the data file is carried in raw (binary) form over TCP, it is encoded in BASE64 when carried by mail.

UFT supports several representation types. The receiving host should accept any file type sent. If the representation type is not meaningful to the target host system, then it should be treated as "binary" (image). The data file (body) should be processed as little as possible until the target user (recipient) acts to accept (receive) it. The commands from the client may be stored in the form of a plain-text file so that processing otherwise foreign to the receiver may be off-loaded from the TCP listener. So there are actually two files: the command sequence and the file body.

Job Entry capability:

The target "user" may actually be no user at all, but may be the name of some software service engine. An example of this is the job entry queue available as a pseudo-user on many NJE networked hosts.

4. Essential commands and Syntax:

```
        FILE    size    sender    [auth]
        USER    recipient

        TYPE    type   [parm]

        Representation Types:

        TYPE         A           ASCII, CR/LF (0D/0A)
                     B           binary (image; octet stream)

                     C           ASCII, CC, CR/LF (ASA print)

                     U           unformatted (binary; image)
                     V           var-length records (16 bit)
                     W           wide var-len records (32 bit)
                     X           extra-wide var-length (64 bit)

                     I           image (binary; octet stream)
                     E           EBCDIC, NL (15)
                     F  reclen   fixed-length records (binary)

                     N           NETDATA
                     M           ASCII, mail

        Additional Parameters:

        NAME    filename
        DATE    date    time    [time-zone]

        CLASS   class
        FORM    paper-form-code  or  print-stock-code
        DEST    destination

        DIST | BIN | BOX        distribution-code  or  mail-stop
        FCB | CTAPE             forms-control-buffer  or  carriage-tape
        UCS | CHARSET | TRAIN   print-train  or  character-set

        LRECL           logical-record-length
        RECFM           record-format
        BLKSIZE         block-size

        MODE            file access permissions

        File disposition commands:

        DATA  [burst-size]

        EOF
        ABORT

        QUIT
```

5. Details:

Commands consist of command words, possibly followed by tokens delimited by white space. Command lines are ASCII terminated by CR/LF. White space may be composed of any mixture of blanks or tab characters, but use of ordinary blank space (ASCII 0x20) is strongly recommended.

One connection (one socket) is used for both commands and data. While a data burst is being received, command interpretation is suspended. Command lines are read until CR/LF; data bursts are read until burst-size number of octets are received, at which point command interpretation is resumed. After data transmission has begun, the only commands valid are DATA, EOF, ABORT and QUIT. EOF causes the server to close the file at the receiving end and return to normal command processing. ABORT signals that the client wishes to discard a file partially transmitted. QUIT closes any open file, closes the connection, and can appear anywhere in the job.

For the daring, a "fast" mode is available. If the burst-size token is omitted from the DATA command, processing switches to data mode and the stream is read until the client closes the connection. In this case there is no EOF or QUIT command sent. NOTE: with the former mode of operation, the connection may remain open indefinitely passing multiple files, while in this latter case the connection must close to terminate the transaction.

Acknowledgement is by simple "NULL ACK". A server accepts a command by sending a single packet back to the client that starts with a NULL character, decimal 0. Anything else may be considered negative acknowledgement, and the client should close the connection. Any characters following the NULL may be ignored. An ACK response packet may signal only one acknowledgement.

When a client first connects to a server, the server immediately sends a herald of the form:

```
xxx hostname UFT 1.0 server-version xxx
```

where "xxx" represents arbitrary data. The first "xxx" must be a single blank delimited token. 1.0 is the protocol version. Hostname is the IP name of the host where this server is running. Server-version is the name and level of UFT server code on this host.

A US English server might send:

```
100 ricevm1.rice.edu UFT 1.0 VM/CMS-0.9.2 ready.
```

The purpose of this herald is partly for client/server synchronization, but mainly for protocol agreement. There may be future versions of UFT beyond 1.0 which support more features than are outlined here. The herald indicates what level of UFT the server will accept.

The FILE Command:

```
FILE    size    from    [auth]
```

The size is in bytes and may be followed by an ’M’, ’K’, or ’G’, indicating Mega, Kilo, or Giga. Size may be an inexact value (the data file will be read until one of the above end-of-file indications is received). The size specified is used to answer the question, "is there room for it?"

The from token is the login name of the user sending this file.

The auth token is an unimplemented authentication ticket. Authentication is not ensured in the protocol as described. There are several ways that it might be added to UFT over TCP, but this author will wait for authentication developments by others to come to fruition before implementing any. When UFT is piggy-backed on mail, authentication is left to the mail transfer system.

The FILE command is required in any transaction.

The USER Command:

```
USER    recipient
```

The recipient is a valid local user or service name.

The USER command is required in any transaction. Without it, the destination of the file is unknown.

The TYPE Command:

```
            TYPE    type   [parm]
```

Some representation types need additional specification. As an example, the type "F" (fixed length, record oriented) obviously needs more qualification. How long are these fixed length records? A record length in ASCII decimal should follow the "F" resulting in a command like "TYPE F 80".

UFT types V, W, X use a tape model for file transfer. Files in transit consist of blocks that vary in size based on the range of sizes specifiable with 16, 32, or 64 bits, respectively. Whether the blocking is significant to the recipient is the decision of the recipient, but if the file originally had some kind of blocking, it is preserved without additional processing. In the stream, the 16, 32, or 64-bit block length is prepended to each record in TCP/IP network order.

Type N (NETDATA) is an IBM representation common on NJE networks.

The TYPE command is required in any transaction.

The NAME Command:

```
            NAME    filename
```

A name should typically be associated with the file being sent, although this is not mandatory. This is a mixed case token delimitted by white space. If the filename contains blanks or white space, it must be quoted. Quotation is not valid within the filename. ASCII control characters (hex 00 thru 1F and 80 thru 9F) are not valid as part of the filename. Some characters may have special meaning to the receiving operating system and their effect is not guaranteed.

The NAME command is optional.

The DATE Command:

```
            DATE    date    time    [time-zone]
```

The time stamp on the file as it appears at the sending site may be sent and applied to the copy at the receiving site. The form is US mm/dd/yy and hh:mm:ss. A time zone is optional. If the time zone is omitted, local time is assumed. If the DATE command is omitted, time and date of arrival are assumed.

The DATE command is optional.

The DATA Command:

```
            DATA  [burst-size]
```

If no data bursts have yet been received since the connection was opened or since an EOF or ABORT was received, the server opens a new file on the receiving end and writes this burst of data to it. The file may have already been created by a prior DATA command. There can be any number of DATA commands; most files will be sent using many data bursts. If burst-size is supplied, then burst-size number of octets are read and appended to the open file on the receiving end and the server returns to the command state. If no burst-size parameter is given, then the TCP stream is read until it is closed. (this is the "fast" mode mentioned above)

The DATA command must come after FILE, USER, TYPE, and any other parametric commands and must come before any EOF or ABORT command. The file need not be complete before an ABORT can be received and carried out, but the DATA command must have completed (burst-size number of octets must have been read), thus ABORT is not possible in "fast" mode.

The EOF Command:

```
            EOF
```

This signals the server that the entire file has been sent. The server then closes the file and ensures that it is disposed of appropriately, usually just placing it where a user-level application can retrieve it later.

The ABORT Command:

```
            ABORT
```

This signals the server that the client is unable or unwilling to finish the job. The file should be discarded and the server should return to normal command processing.

The QUIT Command:

```
            QUIT
```

This signals the server that all work is complete. Any open file should be closed and delivered. The TCP stream will be closed.

Other commands:

CLASS	class
FORM	paper-form-code or print-stock-code
DEST	destination
DIST	distribution-code or mail-stop
FCB	forms-control-buffer or carriage-tape
CHARSET	print-train or character-set

The above are relevant to print jobs sent to a print server.

LRECL	logical-record-length
RECFM	record-format
BLKSIZE	block-size
MODE	file access permissions

6. References

NJE	--	Network Job Entry; IBM publication SC23-0070, "Network Job Entry; Formats and Protocols"
NETDATA	--	see IBM publication aann-nnnn (SC24-5461); VM/ESA: CMS Application Development Reference for Assembler
BITNET	--	"Because It's Time"; academic network based on NJE protocol
MIME	--	RFC 1341; Multipurpose Internet Mail Extensions; Borenstein & Freed
FTP	--	File Transfer Protocol; STD 9, RFC 959; Postel & Reynolds
SMTP	--	STD 10, RFC 821; Simple Mail Transfer Protocol; Postel
LPR	--	UNIX Programmer's Manual, LPD(8); 4.2BSD Line Printer Spooler Manual

7. Security Considerations

Security issues are not discussed in this memo.

8. Author's Address

Rick Troth
Rice University
Information Systems
Houston, Texas 77251

Phone: (713) 285-5148
Fax: (713) 527-6099
EMail: troth@rice.edu