File Transfer And Access (FTP, TFTP, NFS)

Chapter 25

By: Sang Oh
Spencer Kam
Atsuya Takagi

History of FTP

- The first proposed file transfer mechanisms were developed for implementation on hosts at M.I.T. (RFC 114) in 1971, even before the TCP/IP was existed.
- FTP general structure was established in 1973.
- The base specification RFC959 was published in 1985.

History of FTP

- The TCP/IP protocol was developed in the late 1970s and early 1980s
- Modern TCP/IP was the result of experimentation and development work that had been underway since the 1960s. This work included both the design and implementation of the protocols that would implement internet, and also the creation of the first networking applications to allow users to perform different tasks.

History of FTP

- The developers of early applications conceptually divided methods of network use into two categories: *direct* and *indirect*.
- Direct network applications let a user access a remote host and use it as if it were local, creating the illusion that the network doesn't even exist (or at least, minimizing the importance of distance).

History of FTP

- Indirect network use meant getting resources from a remote host and using them on the local system, then transferring them back.
- These two methods of use became the models for the first two formalized TCP/IP networking applications: Telnet for direct access and the *FTP* for indirect network use.

History of FTP

- The first FTP standard was RFC 114, published in April 1971, before TCP and IP even existed. This standard defined the basic commands of the protocol and the formal means by which devises communicate using it. At this time the predecessor of TCP (called simply the *Network Control Protocol* or *NCP*) was used for conveying network traffic. There was no Internet back then. Its precursor, the ARPAnet, was tiny, consisting of only a small group of development computers.

History of FTP

- A number of subsequent RFCs refined the operation of this early version of FTP, with revisions published as RFC 172 in June 1971 and RFC 265 in November 1971. The first major revision was RFC 354, July 1972, which for the first time contained a description of the overall communication model used by modern TCP, and details on many of the current features of the protocol. In subsequent months many additional RFCs were published, defining features for FTP or raising issues with it. RFC 542, August 1973, the FTP specification looks remarkably similar to the one we use today, over three decades later, except that it was still defined to run over NCP.

History of FTP

- After a number of subsequent RFCs to define and discuss changes, the formal standard for modern FTP was published in RFC 765, *File Transfer Protocol Specification*, June 1980. This was the first standard to define FTP operation over modern TCP/IP, and was created at around the same time as the other primary defining standards for TCP/IP.

History of FTP

- RFC 959, FTP, was published in October 1985 and made some revisions to RFC 765, including the addition of several new commands, and is now the base specification for FTP. Since that time a number of other standards have been published that define extensions to FTP, better security measures and other features. (Some of these are discussed in the general operation section in the appropriate places.)

The Goal of FTP

- FTP was created with the overall goal of allowing indirect use of computers on a network, by making it easy for users to move files from one place to another. Like most TCP/IP protocols, it is based on a client/server model, with an FTP client on a user machine creating a connection to an FTP server to send and retrieve files to and from the server. The main objectives of FTP were to make file transfer simple, and to shield the user from implementation details of how the files are actually moved from one place to another. To this end, FTP is designed to automatically deal with many of the issues that can potentially arise due to format differences in files stored on differing systems.

Overview of how FTP works

- After a TCP connection is established, an FTP control connection is created. Internal FTP commands are passed over this logical connection based on formatting rules established by the Telnet protocol. Each command sent by the client receives a reply from the server to indicate whether it succeeded or failed. A data connection is established for each individual data transfer to be performed. FTP supports either normal or passive data connections, allowing either the server or client to initiate the data connection. Multiple data types and file types are supported to allow flexibility for various types of transfers.

Overview of how FTP works

- To ensure that files are sent and received without loss of data that could corrupt them, FTP uses the reliable Transmission Control Protocol (TCP) at the transport layer. An authentication system is used to ensure that only authorized clients are allowed to access a server. At the same time, a feature sometimes called *anonymous FTP* allows an organization that wishes it to set up a general information server to provide files to anyone who might want to retrieve them.

Overview of how FTP works

- The interface between an FTP user and the protocol is provided in the form of a set of interactive user commands. After establishing a connection and completing authentication, two basic commands can be used to send or receive files. Additional support commands are provided to manage the FTP connection, as well as to perform support functions such as listing the contents of a directory or deleting or renaming files. In recent years, graphical implementations of FTP have been created to allow users to transfer files using mouse clicks instead of memorizing commands. FTP can also be used directly by other applications to move files from one place to another.

FTP Features

- Interactive Access

  FTP provides an interactive interface to allow humans to interact with remote servers.

- Format Specification

  FTP allows the client to specify the type and representation of stored data.

  - The user can specify whether a file contains text or binary data.

- Authentication Control

  FTP requires clients to authorize themselves by sending a login name and password to the server before requesting file transfers.

  - The server refuses access to clients that cannot provide a valid login and password.

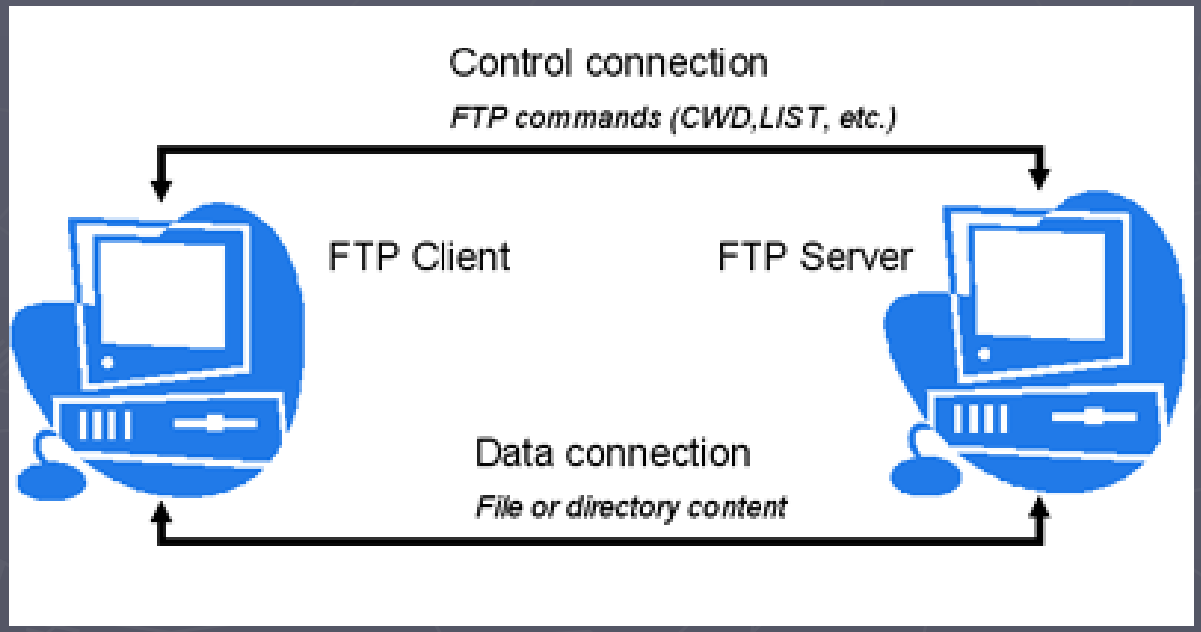
FTP Process Model

- FTP server implementations allow simultaneous access by multiple clients
- Clients use TCP to connect to a server.
- The FTP server process awaits connections and creates a slave process to handle each connection.
- The slave process accepts and handles a *control connection* from the client.
  - The control connection carries commands that tell the server which file to transfer.

FTP Process Model

- An additional TCP connection and process is created to handle each *data transfer* operation.
  - The new TCP connection and process on both the client and server is created for each data transfer operation.
- The control connection is kept alive as long as the client keeps the FTP session active.
- The data transfer connection is kept alive for the duration of one file transfer.
  - For each file that is being transferred, a new data transfer connection is created.

FTP Process Model



TCP Port Numbers and Data Connections

- When the client forms an initial connection to a server, the client uses a random protocol port number.
- The client contacts the server at a common port number (port 21).
- When a file transfer is initiated, the client obtains an unused port number on its machine.

TCP Port Numbers and Data Connections

- The client sends that port number across the control connection to the server.
- The client waits for the server to form a TCP connection to that specified port.
  - The server uses port 20 for the FTP data transfer.

FTP Commands

- ABOR - abort a file transfer
- CWD - change working directory
- DELE - delete a remote file
- LIST - list remote files
- MDTM - return the modification time of a file
- MKD - make a remote directory
- NLST - name list of remote directory
- PASS - send password
- PASV - enter passive mode
- PORT - open a data port
- PWD - print working directory

FTP Commands

- QUIT - terminate the connection
- RETR - retrieve a remote file
- RMD - remove a remote directory
- RNFR - rename from
- RNTO - rename to
- SITE - site-specific commands
- SIZE - return the size of a file
- STOR - store a file on the remote host
- TYPE - set transfer type
- USER - send username

A Simple FTP Transaction

- Client connects to ftp server.
- Server returns code 220 to specify that it is ready for the new user.
- Client sends command “USER <username>”
- Server returns code 331 if a password is required to access the account.
- Client sends command “PASS <password>”.
- Server returns code 230 if authentication is successful.
- Client sends a PORT command to specify the port number that it wants to transfer data over.
- Server returns code 200 if the PORT command was successful.
- Client sends a LIST command.
- The server sends a list of files in the current directory and returns code 226, which closes the connection.

A Simple FTP Transaction

- Client sends another PORT command to open another data connection.
- Server returns code 200 if the PORT command is successful.
- The client sends a command of RETR <file name> in order to initiate a transfer of that file.
- Server returns code 150 is the file status is okay and the file will be transferred.
- When the transfer is complete, the server returns code 226 to tell the client that the transfer is complete and the data connection will be clssed.
- Client sends QUIT command
- Server returns code 221 and closes the control connection.

Problems with FTP

The original FTP design does not work well with security firewalls and NAT systems.

- An extension, passive FTP, was created allows the client to initiate each data transfer connection.
  - This way, FTP can be used across a firewall or NAT system without being a special case.

Problems with FTP

- In the initial FTP specification, data was transferred in an unencrypted fashion.
  - This means that data can be read through the use of a packet sniffer on the network.
    - This includes user names, passwords, FTP commands, and transferred files.

FTP and Web browsers

- Most recent web browsers provide support for FTP.
- This allows for manipulation of remote files through an interface similar to that used for local files.
- FTP servers can be access through the web browser in the format

  ftp(s)://<ftpserveraddress>

  ftp(s)://<login>:<password>@<ftpserveraddress>:<port>

Anonymous FTP

- Requiring a login and password prohibits users from accessing public files.
- FTP servers can allow anonymous users to access to public files, using login name *anonymous* and a password equal to the user's email address or the password *guest*.
- For most of FTP serves, anonymous FTP is disabled by default for security reason.

Need for Secure File Transfer

- All data, including login name and password, are sent unencrypted.
- If someone wiretaps a connection, it is possible to obtain either a copy of the data being transferred or a user's login name and password.

SSL-FTP

- *Secure Sockets Layer* FTP (SSL-FTP) is one of FTP extension uses SSL technology.
- When using SSL-FTP, all transfers are confidential, including password as well as the data being transferred.
- Since basic SSL follows the socket API, only minor modifications are required for FTP client and server to use the extension.

SFTP

- *Secure File Transfer Program* (sftp) is built to use an SSH tunnel.
- Thus, sftp transfers are multiplexed over the single SSH connection, whereas each of FTP connections might use different ports.

SCP

- *Secure Copy* (scp) also uses an SSH channel for transfers and relies on the SSH server for file access on the remote computer.

FTP Is Complex

- Although FTP is the most general file transfer protocol in the TCP/IP suite, it is also the most complex and difficult to program.
- Many applications do not need the full functionalities that FTP offers.

FTP Is Complex

For example, FTP requires clients and servers to manage multiple concurrent TCP connections, however, it may be difficult or impossible on embedded computers that do not have sophisticated OS.

TFTP

Trivial File Transfer Protocol (TFTP) is intended to be simple.
TFTP restricts operations to simple file transfers and does not provide authentication.
Since it is simple, TFTP software tends to be much smaller in size than FTP software.

TFTP

Small in size is big advantage for embedded systems.
Embedded system can have TFTP in its ROM and use it to obtain an initial memory image when the system is powered on.
Thus, when update for initial memory image is needed, updating the image is enough and does not require any modification to the system itself.

TFTP

Unlike FTP, TFTP runs on top of UDP (port 69).
Since UDP is unreliable, TFTP uses timeout and retransmission to ensure that data arrives.
The sending side transmits a file in fixed size (512 byte) blocks and awaits an acknowledgement for each block before sending the next.
The receiver acknowledges each block upon receipt.

TFTP

The first packet sent requests a file transfer and establishes the interaction between client and server.

The first packet also specifies a file name and whether the file will be read, transferred to the client, or written, transferred to the server.

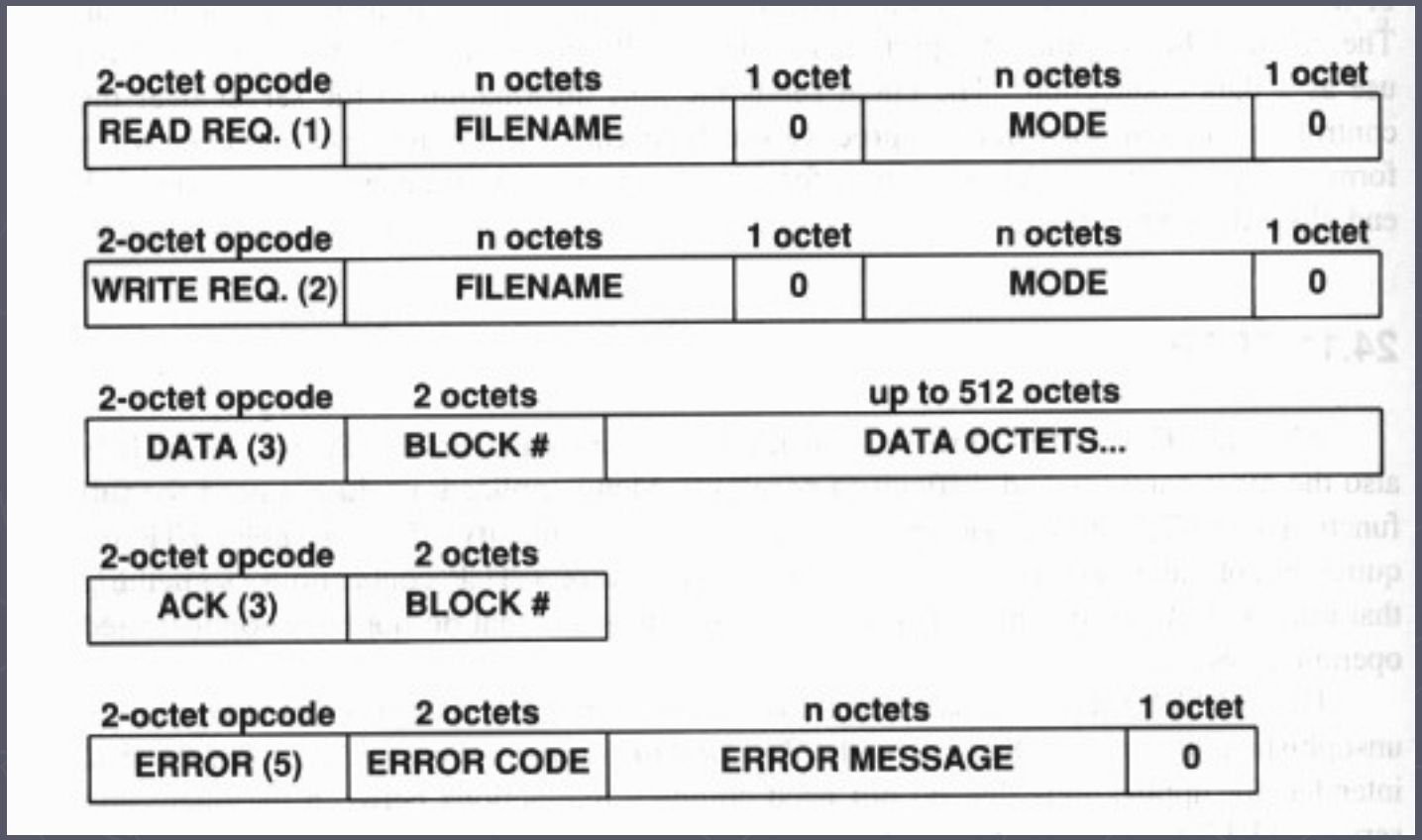
TFTP

Blocks of the file are numbered consecutively starting at 1.
Each data packet contains a header that specifies the number of the block it carries, and each acknowledgement contains the number of the block being acknowledged.
A block of less than 512 bytes signals the end of file.

TFTP

It is possible to send an error message either in the place of data or an acknowledgement.
Upon errors, the transfer will be terminated.

TFTP



2-octet opcode	n octets	1 octet	n octets	1 octet
READ REQ. (1)	FILENAME	0	MODE	0

2-octet opcode	n octets	1 octet	n octets	1 octet
WRITE REQ. (2)	FILENAME	0	MODE	0

2-octet opcode	2 octets	up to 512 octets
DATA (3)	BLOCK #	DATA OCTETS...

2-octet opcode	2 octets
ACK (3)	BLOCK #

2-octet opcode	2 octets	n octets	1 octet
ERROR (5)	ERROR CODE	ERROR MESSAGE	0

TFTP

Once a read or write request has been made, the server uses the IP address and UDP port number of the client to identify subsequent operations. Thus, neither of data messages nor ack messages need to specify the file name.

TFTP

Lost messages can be retransmitted after a timeout.
However, most other errors simply cause termination of the interaction, because TFTP is intended to be simple!

TFTP

TFTP ensures its data arrival by requiring each side to implement a timeout and retransmission.
If the side sending data times out, it retransmits the last data block.
If the side responsible for acknowledgements times out, it retransmits the last acknowledgement.

TFTP

The problem, known as the *Sorcerer's Apprentice Bug*, arises when an acknowledgement for data packet *k* is delayed, but not lost.
The sender retransmits the data packet.
Both acknowledgements eventually arrive, and each triggers a transmission of data packet *k* + *1*.
Fixed in latest version of TFTP.

NFS

Network File System (NFS) was initially developed by Sun Microsystems, and has been published as an IETF standard.

NFS

When an application executes, it calls OS to open/read/write a file. The file access mechanism accepts the request, and passes it to either the local file system software or to the NFS client, depending on whether the file is on the local disk or on a remote machine.

When it receives a request, the client software uses NFS protocol to contact the NSF server on a remote machine and perform the request.

Implementation of NFS

Implementation of NFS consists of three parts: NFS protocol, a general-purpose *Remote Procedure Call* (RPC) and a general-purpose *eXternal Data Representation* (XDR). Intention for this was to make RPC and XDR available for other software.

RPC

On the client side, the programmer designates some procedures as *remote*, forcing the compiler to incorporate RPC code into those procedures. On the server side, the programmer implements the desired procedures and uses other RPC facilities to declare them to be part of a server.

RPC

When the executing client program calls one of the remote procedures, RPC automatically collects values for arguments, forms a message, sends the message to the remote server, awaits a response, and stores returned values in the designated arguments.

RPC mechanism hides all the details of protocols, making it easy to write distributed programs.

XDR

XDR provides a way for programmers to pass data among heterogeneous machines without writing procedures to convert among the hardware data representations.
XDR solves the problem by defining a machine-independent representation.

XDR

At one end of communication channel, a program invokes XDR procedures to convert from the local hardware representation to the machine-independent representation.
Once the data has been transferred to another machine, receiving program invokes XDR procedures to convert from the machine-independent representation to the local representation.