Jacek WOŁOSZYN
Politechnika Radomska, Polska

Trivial file transfer protocol (TFTP)

1. TFTP overview

FTP is the main protocol used for the majority of general file transfers in TCP/IP internetworks. One of the objectives of the designers of FTP was to keep the protocol relatively simple, but that was possible only to a limited extent. To enable the protocol to be useful in a variety of cases and between many kinds of devices, FTP needed a fairly large set of features and capabilities. As a result, while FTP is not as complex as certain other protocols, it is still fairly complicated in a number of respects.

The complexity of FTP [Komar 2000; Karanjit, Siyan, Parker 2002] is partly due to the protocol itself, with its dozens of commands and reply codes, and partly due to the need of using TCP for connections and data transport. The reliance on TCP means that any device wanting to use FTP needs not only the FTP program but a full TCP implementation as well. It must handle FTP's need for simultaneous data and control channel connections and other requirements.

For a conventional computer, such as a regular PC, Macintosh, or UNIX workstation, none of this is really an issue, especially with today's large hard disks and fast, cheap memory. But remember that FTP was developed more than three decades ago, when hardware was slow and memory was expensive. Furthermore, even today, regular computers are not the only devices used on networks. Some networked devices do not have the capabilities of true computers, but they still need to be able to perform file transfers. For these devices, a full FTP and TCP implementation is a nontrivial matter.

One of the most notable examples of such devices are diskless workstations computers that have no permanent storage, so when they start up, they cannot read a whole TCP/IP implementation from a hard disk like most computers easily do. They start with only a small amount of built-in software and must obtain configuration information from a server and then download the rest of their software from another network device. The same issue arises for certain other hardware devices with no hard disks.

The process of starting up these devices is commonly called bootstrapping and occurs in two phases. First, the workstation is provided with an IP address and other parameters through the use of a host configuration protocol such as the Bootstrap Protocol or the Dynamic Host Control Protocol. Second, the client downloads software, such as an operating system and drivers, that let it function

on the network like any other device. This requires that the device have the ability to transfer files quickly and easily. The instructions to perform this bootstrapping must fit onto a read-only memory (ROM) chip, and this makes the size of the software an important issue again, especially many years ago.

The solution to this need was to create a light version of FTP that would emphasize small program size and simplicity over functionality. This new protocol, TFTP, was initially developed in the late 1970 and first standardized in 1980. The modern version, TFTP version 2, was documented in RFC 783 in 1981, which was revised and published as RFC 1350, The TFTP Protocol in 1992. This is the current version of the standard.

2. Comparing FTP and TFTP

Probably the best way to understand the relationship between TFTP and FTP is to compare it to the relationship between the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) at the transport layer. UDP is a simplified, stripped down alternative to TCP that is used when simplicity is more important than rich functionality. Similarly, TFTP is a greatly simplified version of FTP that allows only basic operations and lacks some of FTP's fancy capabilities in order to keep its implementation easy and its program size small.

Due to its limitations, TFTP is a complement to FTP, not a replacement for it. TFTP is used only when its simplicity is important and its lack of features is not. Its most common application is bootstrapping, though it can be used for other purposes. One specific application that the TFTP standard describes for the protocol is the transport of electronic mail (email). While the protocol supports this explicitly, TFTP is not generally used for this purpose today.

FTP and TFTP have significant differences in at least four significant areas:

– *Transport* The comparison to TCP and UDP is apt not only based on the features/simplicity trade-off, but because FTP uses TCP [Haugdahl Scott 2001] for transport while TFTP uses UDP. Like TFTP, UDP is simple, and this makes the two ideal for embedding together as a hardware program set in a network device.
– *Limited Command Set* FTP includes a rich set of commands to allow files to be sent, received, renamed, deleted, and so forth. TFTP allows files only to be sent and received.
– *Limited Data Representations* TFTP does not include some of FTP's fancy data representation options it allows only simple ASCII or binary file transfers.
– *Lack of Authentication* UDP uses no login mechanism or other means of authentication. This is again a simplification, though it means the operators of TFTP servers must severely restrict the files they make available for access. (It is also part of why TFTP specifically does not allow the client to perform dangerous file operations such as deletion.)

3. Overview of TFTP Operation

Communication and messaging in TFTP is very different from FTP because of the different transport layer protocols used by each. FTP makes use of TCP's rich functionality, including its stream data orientation, to allow it to send bytes of data directly over the FTP data connection. TCP also takes care of reliable delivery of data for FTP, ensuring that files are received correctly. In contrast, since TFTP uses UDP, it must package data into individual messages for both protocol control and data communication. TFTP must also take care of timing transmissions to detect lost datagrams and then retransmitting as needed.

TFTP servers allow connections from TFTP clients to perform file send and receive operations. Many hosts that run FTP Server [Sportack 2004] will also run a separate TFTP server module. TFTP users initiate connections by starting a TFTP client program, which generally uses a command-line interface similar to that of many FTP clients the main difference is the much smaller number of commands in TFTP.

The basic operation of TFTP has not changed since RFC 1350 was published, but a new feature was added to the protocol in 1995. RFC 1782, TFTP Option Extension, defines a mechanism by which a TFTP client and TFTP server can negotiate certain parameters that will control a file transfer prior to the transfer commencing. This allows more flexibility in how TFTP is used, adding a slight amount of complexity to TFTP, but not a great deal.

The option extension is backward-compatible with regular TFTP and is used only if both server and client support it. Two subsequent RFCs define the actual options that can be negotiated: RFC 1783, TFTP Blocksize Option, and RFC 1784, „TFTP Timeout Interval and Transfer Size Options”. This set of three RFCs (1782, 1783, and 1784) was replaced in 1998 by updated versions in RFCs 2347, 2348, and 2349.

TFTP communication is client/server based, as discussed in the overview. The process of transferring a file consists of three main phases:

– Initial Connection The TFTP client establishes the connection by sending an initial request to the server. The server responds back to the client, and the connection **is** effectively opened.

– Data Transfer Once the connection is established, the client and server exchange TFTP messages. One device sends data, and the other sends acknowledgments.

– Connection Termination When the last TFTP message containing data has been sent and acknowledged, the connection is terminated.

Connection Establishment and Identification

The matter of a connection is somewhat different in TFTP than it is with a protocol like FTP that uses TCP. FTP must establish a connection at the TCP level before anything can be done by FTP itself. TFTP, however, uses the connectionless UDP for transport, so there is no connection in the sense that one

exists in TCP. In TFTP, the connection is more in a *logical* sense, meaning that the client and server are participating in the protocol and exchanging TFTP messages.

The TFTP server listens continuously for requests on well-known UDP port number 69, which is reserved for TFTP. The client chooses for its initial communication an ephemeral port number, as is usually the case in TCP/IP. This port number actually identifies the data transfer and is called a *transfer identifier* (*TID*).

What's different about TFTP, however, is that the server also selects a pseudorandom TID that it uses for sending responses back to the client it doesn't send them from port number 69. This is done because by using a unique client port number and source port number, multiple TFTP exchanges can be conducted simultaneously by a server. Each transfer is identified automatically by the source and destination port number, so there is no need to identify in data messages the transfer to which each block data belongs. This keeps the TFTP header size down, allowing more of each UDP message to contain actual data.

For example, suppose the TFTP client selects a TID of 3145 for its initial message. It would send a UDP transmission from its port 3145 to the server's port 69. Say the server selects a TID of 1114. It would send its reply from its port 1114 to the client's port 3145. From then on, the client would send messages back to server port 1114 until the TFTP session was completed.

4. Lock-Step Client/Server Messaging

After the initial exchange, the client and server exchange data and acknowledgment messages in *lock-step* fashion. Each device sends a message for each message it receives one device sends data messages and waits for acknowledgments the other sends acknowledgments and waits for data messages. This form of rigid communication is less efficient than allowing the transmitter to fire away with one data message after another, but it is important because it keeps TFTP simple when it comes to an important issue: retransmissions.

Like all protocols using the unreliable UDP, TFTP has no assurances that any messages sent will actually arrive at their destination, so it must use timers to detect lost transmissions and resend them. What is different about TFTP is that both clients and servers perform retransmission. The device that is sending data messages will resend the data message if it doesn't receive an acknowledgment in a reasonable period of time the device sending the acknowledgments will resend the acknowledgment if it doesn't receive the next data message promptly. The lock-step communication greatly simplifies this process, since each device needs to keep track of only one outstanding message at a time. It also eliminates the need to deal with complications such as reorganizing blocks received out of order. Since TFTP uses UDP rather than TCP, no explicit concept of a connection exists as in FTP. A TFTP session instead uses the concept

of a logical connection, which is opened when a client sends a request to a server to read or write a file. Communication between the client and server is performed in lock-step fashion one device sends data messages and receives acknowledgments so it knows the data messages were received; the other sends acknowledgments and receives data messages so it knows the acknowledgments were received.

5. Difficulties with TFTP's Simplified Messaging Mechanism

One of the most important drawbacks with this technique is that while it simplifies communication, it does so at the cost of performance. Since only one message can be in transit at a time, this limits throughput to a maximum of 512 bytes for exchange of messages between the client and server. In contrast, when using FTP, large amounts of data can be pipelined there is no need to wait for an acknowledgment for the first piece of data before sending the second.

Another complication is that if a data or an acknowledgment message is resent and the original was not lost but rather just delayed, two copies will show up. The original TFTP rules stated that upon receipt of a duplicate datagram, the device receiving it may resend the current datagram. So, receipt of a duplicate block 2 by a client doing a read would result in the client sending a duplicate acknowledgment for block 2. This would result in two acknowledgments being received by the server, which would in turn send block 3 twice. Then there would be two acknowledgments for block 3, and so on.

It's also worth emphasizing that TFTP includes absolutely no security, so no login or authentication process is in place. As mentioned earlier, administrators must use caution in deciding what files to make available via TFTP and in allowing write access to TFTP servers. You saw earlier that TFTP operation consists of three general steps initial connection, data transfer, and connection termination. All operations are performed through the exchange of specific TFTP messages. Let's take a more detailed look now at these three phases of operation and the specifics of TFTP messaging. The first message sent by the client to initiate TFTP is either a read request (RRQ) message or a write request (WRQ) message. This message serves implicitly to establish the logical TFTP connection and to indicate whether the file is to be sent from the server to the client (read request) or the client to the server (write request). The message also specifies the type of file transfer to be performed. TFTP supports two transfer modes *netascii* mode (ASCII text files as used by the Telnet Protocol) and *octet* mode. Originally, a third file type option existed, called mail mode, but TFTP was never really designed for transmitting mail and this option is now obsolete. Assuming no problem occurred with the request (such as a server problem, inability to find the file, and so on), the server will respond with a positive reply. In the case of a read request, the server will immediately send the first data message back to the client. In the case of a write request, the server will send an acknowl-

edgment message to the client, telling it that it may proceed to send the first data message.

After the initial exchange, the client and server exchange data and acknowledgment messages in lock-step fashion as described earlier. For a read, the server sends one data message and waits for an acknowledgment from the client before sending the next one. For a write, the client sends one data message and the server sends an acknowledgment for it, before the client sends the next data message.

Each data message contains a block of between 0 and 512 bytes of data. The blocks are numbered sequentially, starting with 1. The number of each block is placed in the header of the data message carrying that block and then used in the acknowledgment for that block so the original sender knows it was received. The device sending the data will always send 512 bytes of data at a time for as long as it has enough data to fill the message. When it gets to the end of the file and has fewer than 512 bytes to send, it will send only as many bytes as remain.

The receipt of a data message with between 0 and 511 bytes of data signals that this is the last data message. Once this is acknowledged, it automatically signals the end of the data transfer. There is no need to terminate the connection explicidy, just as it was not necessary to establish it explicitly.

TFTP Read Process Steps

Let's look at an example that shows how TFTP messaging works. Suppose the client wants to read a particular file that is 1200 bytes long:

1. The client sends a read request to the server, specifying the name of the file.
2. The server sends back a data message containing block 1, carrying 512 bytes of data.
3. The client receives the data and sends back an acknowledgment for block 1.
4. The server sends block 2, with 512 bytes of data.
5. The client receives block 2 and sends back an acknowledgment for it.
6. The server sends block 3, containing 176 bytes of data. It waits for an acknowledgment before terminating the logical connection.
7. The client receives the data and sends an acknowledgment for block 3. Since this data message had fewer than 512 bytes, it knows the file is complete.
8. The server receives the acknowledgment and knows the file was received successfully.

TFTP Write Process Steps

Here are the steps in the same process, but where the client is writing the file:

1. The client sends a write request to the server, specifying the name of the file.
2. The server sends back an acknowledgment. Since this acknowledgment is prior to the receipt of any data, it uses block 0 in the acknowledgment.
3. The client sends a data message containing block 1, with 512 bytes of data.

4. The server receives the data and sends back an acknowledgment for block 1.
5. The client sends block 2, containing 512 bytes of data.
6. The server receives the data and sends back an acknowledgment for block 2.
7. The client sends block 3, containing 176 bytes of data. It waits for an acknowledgment before terminating the logical connection.
8. The server receives block 3 and sends an acknowledgment for it. Since this data message had fewer than 512 bytes, the transfer is done.
9. The client receives the acknowledgment for block 3 and knows the file write was completed successfully.

A TFTP *read operation* begins with the client sending a read request message to the TFTP server the server then sends the file in 51 2-byte data messages, waiting after each one for the client to acknowledge receipt before sending the next. A TFTP write *operation* starts with a write request sent by the client to the server, which the server acknowledges. The client then sends the file in 512-byte data blocks, waiting after each for the server to acknowledge receipt. In both cases, there is no explicit means by which the end of a transfer is marked the device receiving the file simply knows the transfer is complete when it receives a data message containing fewer than 512 bytes.

If a problem is encountered at any stage of the connection establishment or transfer process, a device may reply with an error message instead of a data or acknowledgment message, as appropriate. An error message normally results in the failure of the data transfer this is one of the prices paid for the simplicity of TFTP.

Each TFTP file transfer proceeds using the process described, which transfers a single file. If another file needs to be sent or received, a new logical communication is established, in a manner analogous to how FTP creates data connections. The main difference is that TFTP has no persistent control connection, as FTP does.

One of the difficulties that designers of simple protocols and applications seem to have is keeping them simple. Many protocols start out small, but over time well-intentioned users suggest improvements that are added slowly but surely.

The reason for adding this capability is that the original TFTP provided no way at all for the client and server to exchange important control information prior to sending a file. This limited the flexibility of the protocol to deal with special cases, such as the transfer of data over unusual network types. The TFTP option negotiation feature allows additional parameters to be exchanged between the client and server that govern how data is transferred. It does this without significantly complicating the protocol and is backward-compatible with normal TFTP. It is used only if both client and server support it, and one device trying to use the feature will not cause problems if the other doesn't support it.

The client begins the negotiation by sending a modified TFTP read request or write request message. In addition to the normal information that appears in this message, list of options may also be included. Each is specified with an option code and an option value. The names and values are expressed as ASCII strings,

terminated by a null character (0 byte). Multiple options may be specified in the request message.

The server receives the request containing the options, and if it supports the option extension, it processes them. It then returns a *special option acknowledgment* (*OACK*) message to the client, where it lists all the options that the client specified that the server recognizes and accepts. Any options that the client requested but the server rejects are not included in the OACK. The client may use only the options that the server accepts. If the client rejects the server's response, it may send back an error message (with error code 8) upon receipt of the unacceptable OACK message.

The server may specify an alternative value in its response for certain options, if it recognizes the option but doesn't like the client's suggested value. Obviously, if the server doesn't support options at all, it will ignore the client's option requests and respond with a data message (for a read) or a regular acknowledgment (for a write) as in normal TFTP.

If the server did send an OACK, the client proceeds to send messages using the regular messaging exchange described in the previous section. In the case of a write, the OACK replaces the regular acknowledgment in the message dialog. In the case of a read, the OACK is the server's first message instead of the first data block that it would normally send. TFTP doesn't allow the same device to send two datagrams in a row, so a reply from the client must be received before that first block can be sent. The client does this by sending a regular acknowledgment with a block number of 0 in it the same form of acknowledgment a server normally sends for a write.

TFTP is supposed to be a small and simple protocol, so it include few extra I features. One that it does support is *option negotiation,* where a TFTP client and server attempt to come to agreement on additional parameters that they will use in transferring a file. The TFTP client includes one or more options in its read request or write request message, the TFTP server then sends an option acknowledgment (OACK) message listing each option the server agrees to use. The use of options when reading a file means that an extra acknowledgment must be sent by the client to acknowledge the OACK before the server sends the first block of the file.

For review, let's take a look at each of the four possible cases: read and write, with and without options.

The initial message exchange for a normal read without option negotiation is as follows:

1. Client sends read request.
2. Server sends data block 1.
3. Client acknowledges data block 1. And so on . . .

With option negotiation, a read is as follows:

1. Client sends read request with options.

2. Server sends OACK.
3. Client sends regular acknowledgment for block 0; that is, it acknowledges the OACK.
4. Server sends data block 1.
5. Client acknowledges data block 1. And so on . . .

The initial message exchange for a normal write (without option negotiation) is as follows:

1. Client sends write request.
2. Server sends acknowledgment.
3. Client sends data block 1.
4. Server acknowledges data block 1. And so on . . .

And here's a write with option negotiation:

1. Client sends write request with options.
2. Server sends option acknowledgment (instead of regular acknowledgment).
3. Client sends data block 1.
4. Server acknowledges data block 1. And so on …

Conclusions

For situations in which the full FTP is either unnecessary or impractical, the simpler Trivial File Transfer Protocol (TFTP) was developed. TFTP is like FTP in that it is used for general file transfer between a client and server device, but it is stripped down in its capabilities. Rather than including a full command set and using TCP for communication, like FTP, TFTP can be used only for reading or writing a single file, and it uses the fast but unreliable UDP for transport. It is often preferred in situations where small files must be transferred quickly and simply, such as for bootstrapping diskless workstations.

Literatura

Haugdahl Scott J. (2001), *Diagnozowanie i utrzymanie sieci. Księga eksperta*, Gliwice.
Karanjit S., Siyan, Parker T. (2002), *TCP/IP Księga eksperta*, Wydanie II, Gliwice.
Komar B. (2000), *Administracja sieci TCP/IP dla każdego*, Gliwice.
Sportack M. (2004), *Sieci komputerowe. Księga eksperta*, Wydanie II, Gliwice 2004.

Abstract

This article provides a description of the operation of TFTP, beginning with an overview description of the protocol, its history and motivation, and the relevant standards that describe it. I discuss its operation in general terms, cover how TFTP clients and servers communicate, and explain TFTP messaging in detail.

I then discuss TFTP options and the TFTP option negotiation mechanism. The article concludes by showing the various TFTP message formats. File Transfer Protocol (FTP) implements a full set of commands and reply functionalities that enables a user to perform a wide range of file movement and manipulation tasks. Although FTP is ideal as a general-purpose protocol for file transfer between computers, on certain types of hardware, it is too complex to implement easily and provides more capabilities than are really needed. In cases where only the most basic file transfer functions are required while simplicity and small program size is of paramount importance, a companion to FTP called the *Trivial File Transfer Protocol* (*TFTP*) can be used.

Key words: file transfer protocol, TFTP, informatyka.

Podstawy technologii TFTP

Streszczenie

W artykule tym przedstawiono opis protokołu TFTP. Należy go uważać za dopełnienie protokołu FTP i używać w przypadku, kiedy istotną rolę odgrywa jego mała złożoność. Ze względu na jego „lekkość" idealne wydaje się być zastosowanie go w systemach o małych mocach obliczeniowych. Opisano mechanizm działania protokołu oraz główne różnice pomiędzy protokołami TFTP oraz FTP.

Słowa kluczowe: protokół transferu plików, TFTP, informatyka.