TFTP

TRIVIAL FILE TRANSFER PROTOCOL

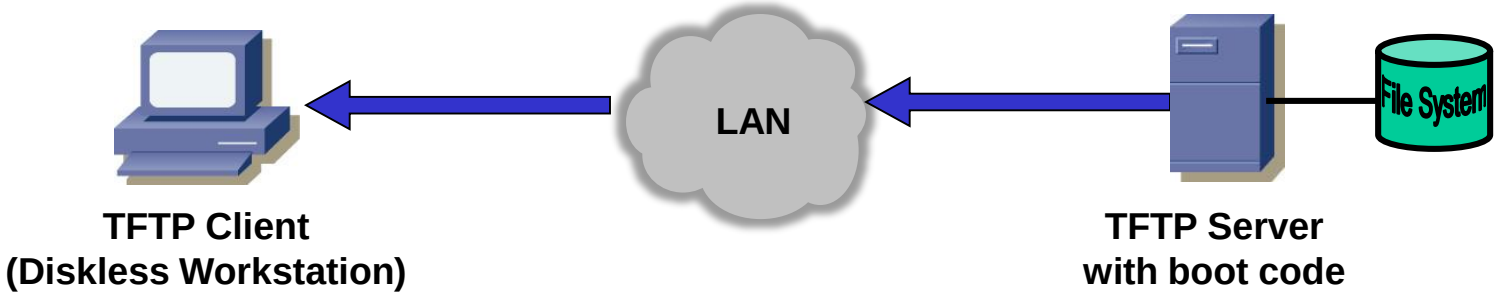
OVERVIEW OF TFTP, A VERY SIMPLE FILE TRANSFER PROTOCOL FOR SIMPLE AND CONSTRAINED DEVICES

Peter R. Egli
peteregli.net

Contents

1. Why TFTP?

In the old days, TFTP was typically used for downloading boot code to diskless workstations. TFTP was simple enough to fit into EEPROMs of diskless workstations (only a few KBytes of code).



Today, TFTP is most often used for downloading new code to Internet appliances (Internet Access Devices, routers, switches, VOIP gateways etc.).

2. TFTP versus FTP

FTP and TFTP both are protocols for transferring files between a client and a server. However, TFTP and FTP are 2 totally different protocols and do not have anything in common.

Value	FTP	TFTP
Authentication	Authentication based on login with username and password.	TFTP does not provide authentication (login).
Connection	FTP uses TCP (reliable transmission). Errors are handled by the underlying TCP layer.	TFTP uses UDP and thus no connections. Errors in the transmission (lost packets, checksum errors) must be handled by the TFTP server.
Protocol algorithm	Transmission of data and control information is handled by the underlying TCP layer. TCP guarantees maximum throughput (flow control, congestion control) and error control.	TFTP uses a simple lock-step protocol (each data packet needs to be acknowledged). Thus the throughput is limited.
Footprint	FTP is more complex than TFTP, thus requires a larger memory footprint. Often FTP is not suited for small device bootloaders which must fit into constrained EEPROM storage.	TFTP is very simple. Because it uses the equally simple UDP transport protocol, TFTP clients or servers have a very small footprint and are thus suited for use in bootloaders.
Control and data channel	FTP separates user data and control information by using 2 separate TCP connections.	TFTP uses only "1 channel", i.e. control packets (commands) flow in one direction while data packets carrying user data flow in the reverse direction over the same UDP sockets.

3. TFTP Protocol RFC1350 (1/5)

The TFTP protocol (1/3):

Request – response protocol:

TFTP is a simple request / acknowledge protocol.
The mode of operation is lock-step because each data packet needs to be acknowledged before the next data packet may be sent.
This makes the implementation very simple (no flow control needed), but limits the throughput because each data packet requires 1 round-trip-time (RTT) for transmission.

Acknowledge of data:

TFTP only uses positive acknowledgements (correctly received packets are acknowledged with an ACK packet).
When the sender does not receive an ACK packet in due time, it re-sends the last DATA packet.

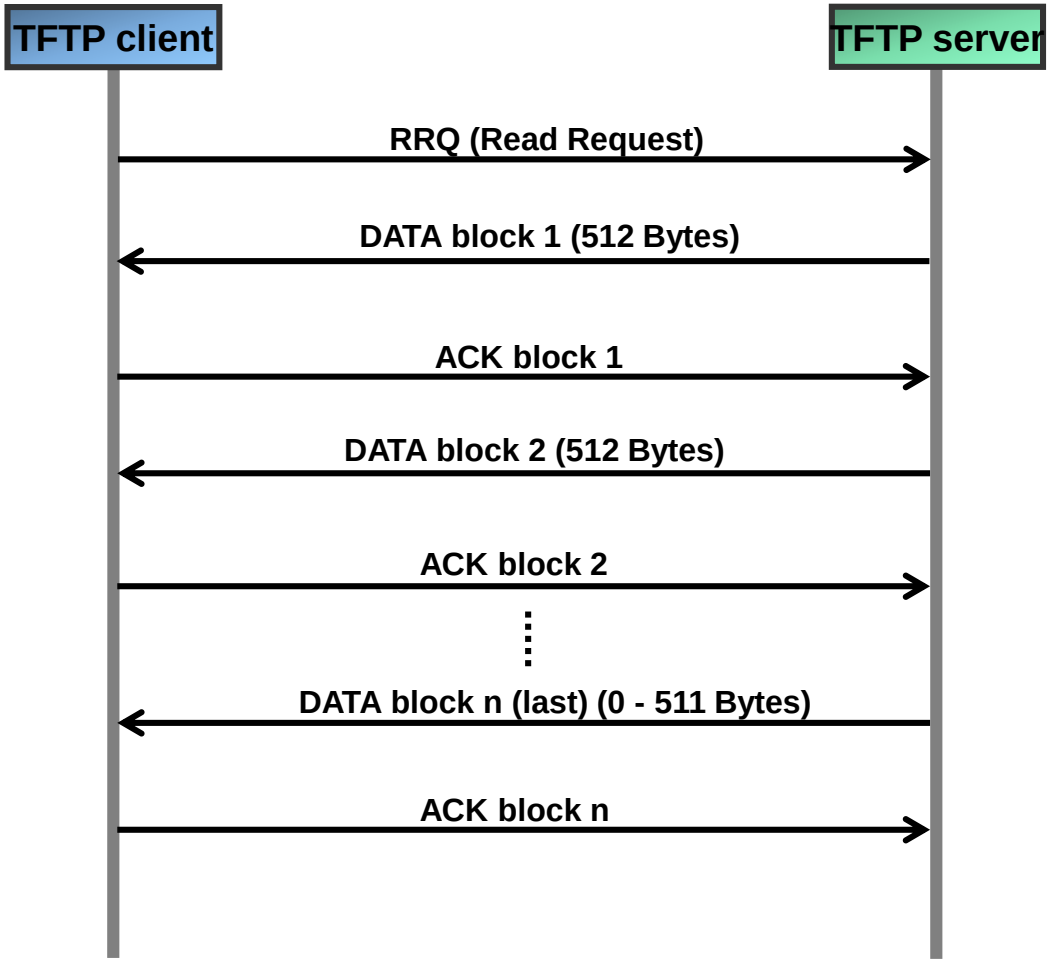
UDP Ports:

The server "listens" on port 69 (TFTP default port), but switches to another port for all replies (DATA, ACK) in order to free the port 69 for other requests (server ‚spawns' a new UDP socket for handling the TFTP request).

3. TFTP Protocol RFC1350 (2/5)

The TFTP protocol (2/3):

TFTP read request (RRQ):



The transfer is initiated by a read request packet (RRQ).

The server responds with the first data block of 512 bytes.

The client acknowledges the reception of the first data block.

The transfer continues with the next data block which is acknowledged by the client.

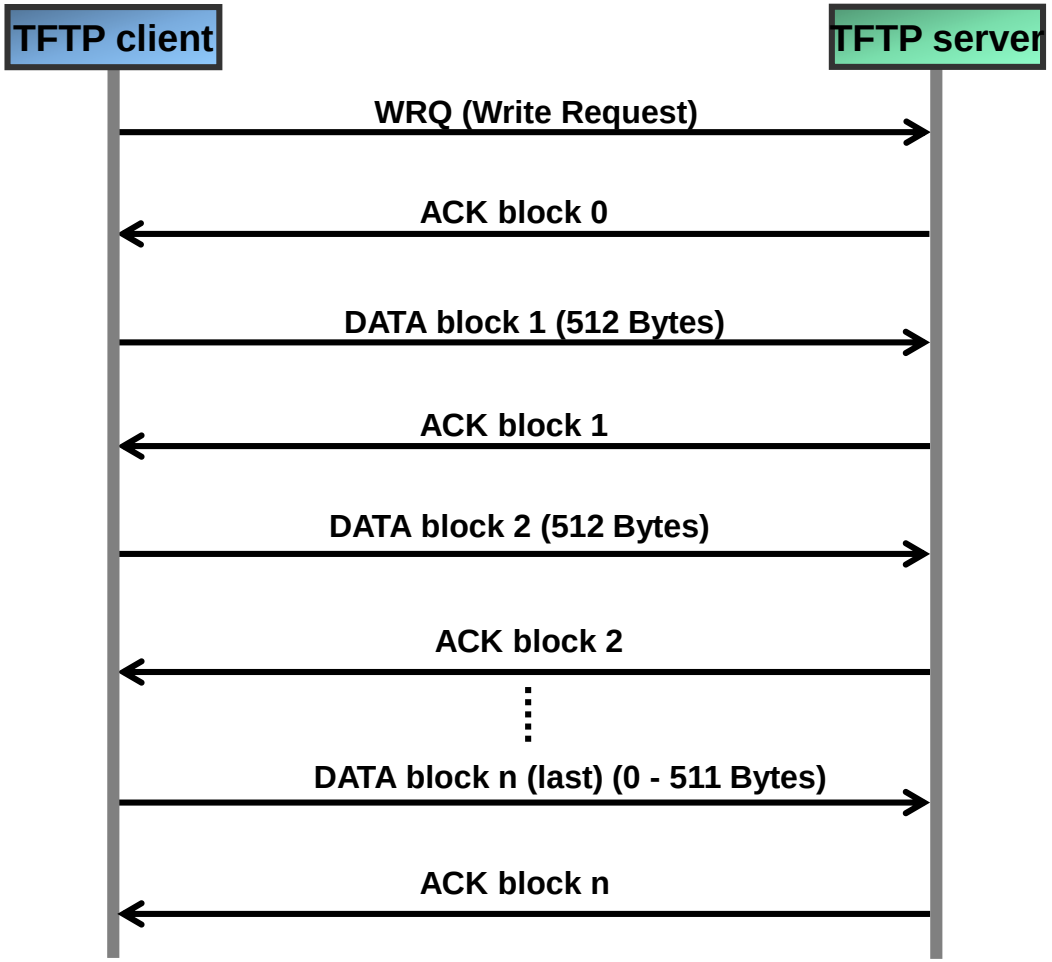
The last data packet contains 0 – 511 bytes. This signals the end of transfer to the client.
If the entire data file to be transferred is dividable by 512, the last packet contains 0 data bytes (an empty packet).

The transfer is completed by the last acknowledge.

3. TFTP Protocol RFC1350 (3/5)

The TFTP protocol (2/3):

TFTP write request (RRQ):



The transfer is initiated by a write request packet (WRQ).

To keep the scheme with acknowledging every packet as is the case in RRQ, the server acknowledges the "virtual" data block 0 which acknowledges the WRQ packet.

The client sends the first data block of 512 bytes.

The server acknowledges the first data block.

The transfer continues with the next data block which is acknowledged by the server.

Again, the last data packet contains 0 – 511 bytes signaling the end of the transfer to the client.

The transfer is completed by the last acknowledge.

3. TFTP Protocol RFC1350 (4/5)

Packet layout:

RRQ (read) WRQ (write)

2 bytes	n bytes	1 byte	mode	1 byte
Opcode 1=RRQ, 2=WRQ	Filename	0x00	“netascii” (=text), “octet” (=binary)	0x00

DATA Data packet

2 bytes	2 bytes	n bytes
Opcode 3=DATA	Block number	Data (0-512 bytes)

All data packets contain 512 bytes except the last one (0-511 bytes).

ACK packet

2 bytes	2 bytes
Opcode 4=ACK	Block number

ERROR packet

2 bytes	2 bytes	n bytes	1 byte
Opcode 5=ERROR	Error number	Error message	0x00

3. TFTP Protocol RFC1350 (5/5)

RRQ (Read Request) and WRQ (Write Request) traces:

Wireshark trace of TFTP read request:

```
C:  1   0.000000 193.5.54.112:44143 -> 193.5.54.29:69   TFTP Read Request, File: text.txt,
                                                        Transfer type: netascii
S:  2   0.001569  193.5.54.29:70 -> 193.5.54.112:44143  TFTP Data Packet, Block: 1
C:  3   0.002043 193.5.54.112:44143 -> 193.5.54.29:70   TFTP Acknowledgement, Block: 1
S:  4   0.002159  193.5.54.29:70 -> 193.5.54.112:44143  TFTP Data Packet, Block: 2 (last)
C.  5   0.009594 193.5.54.112:44143 -> 193.5.54.29:70   TFTP Acknowledgement, Block: 2
```

Wireshark trace of TFTP write request:

```
C:  1   0.000000 193.5.54.112:44547 -> 193.5.54.29:69   TFTP Write Request, File: text.txt,
                                                        Transfer type: netascii
S:  2   0.075201  193.5.54.29:70 -> 193.5.54.112:44547  TFTP Acknowledgement, Block: 0
C:  3   0.075949 193.5.54.112:44547 -> 193.5.54.29:70   TFTP Data Packet, Block: 1
S:  4   0.077696  193.5.54.29:70 -> 193.5.54.112:44547  TFTP Acknowledgement, Block: 1
C:  5   0.077937 193.5.54.112:44547 -> 193.5.54.29:70   TFTP Data Packet, Block: 2 (last)
S:  6   0.084512  193.5.54.29:70 -> 193.5.54.112:44547  TFTP Acknowledgement, Block: 2
```

C: Packet sent by client
S: Packet sent by server

4. TFTP and FTP throughput comparison

Due to the different transmission algorithms, the maximum achievable throughput for TFTP and FTP is different:

TFTP theoretical max. throughput = 512Bytes / RTT	**RTT = Round Trip Time**

FTP theoretical max. throughput = Ws / RTT	**Ws = Window size**

Theoretical throughput example:

Measured RTT = 0.282ms (with ping); Ws = 8192Bytes; assumptions: protocol processing does not consume time; TCP mechanisms (delayed ack etc.) do not increase RTT.
TFTP: 512 Bytes in 0.282ms ➔ 1.8MBytes/s
FTP: 8192 Bytes in 0.282ms ➔ 29MBytes/s
FTP is roughly 16 times faster than TFTP.

Measured throughput example:

TFTP: 143872 Bytes in 0.8s ➔ 180KBytes/s
FTP: 143872 Bytes in 0.11s ➔ 1.3MBytes/s
FTP was roughly 7 times faster than TFTP.

The effective transfer rate is much lower than the theoretical maximum due to processing delays on client and server (increased RTT) and less than ideal usage of protocols (TCP window size is not fully exploited).