Protocol Evaluation

- Correctness
- Scaling
- Adaptability

Protocol Correctness

- Does protocol do what it should
  - Any errors?
  - Any cases not handled?
- First
  - What should it do?
  - What is the protocol's purpose?
  - What is NOT its purpose?
- Must understand design aims
  - requirements and constraints

Protocol Scalability

- What happens when
  - Number of users grows
  - Size of network grows
  - Speed of network increases
  - Delays increase
  - Packet loss increases
  - (and more)
- Does protocol still work?
  - Does it still work as well?

Protocol Adaptability

- When changes are needed
  - because of scaling issue
  - or new functionality required
  - or old functions useless
    - or cause problems
- How easily can protocol be updated?
  - How interoperable are new and old?
  - What is upgrade path?
  - Flag days?

Protocol Specification

Text

- English, or ...
- State Machines
- Grammars
- Formal specification methods

TFTP

- Illustrates some of the fundamentals of Internet Protocols
- Defined completely using text
  - and diagrams
  - RFC 783
    - June, 1981

1. Purpose

TFTP is a simple protocol to transfer files, [...]

It is designed to be small and easy to implement. [...]

The only thing it can do is read and write files (or mail) from/to a remote server. It cannot list directories, and currently has no provisions for user authentication.

TFTP Purpose

- Computer Network Bootstrap
  - Load code & configuration
- Goes in BOOT ROM
  - Small
  - Hard/Impossible to alter
- Needs to copy small/medium files
  - In several seconds
    - (say 100KB/second)

TFTP Operational Basics

2. Overview of the Protocol

Any transfer begins with a request to read or write a file, which also serves to request a connection.

If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes.

Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent.

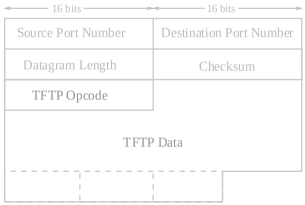
A data packet of less than 512 bytes signals termination of a transfer.

- Very simple protocol
  - Get a block
  - Send ACK
    - Repeat...

TFTP

- The opcode is a two byte integer
  - transmitted in network byte order
  - Most significant octet transmitted first
  - Then next most significant
    - (etc, for as many bytes required)

TFTP in UDP



- Well Known Port for TFTP is 69
  - Initial request from client to server
    - Destination Port 69, Source Port N
  - Replies from server to client
    - Destination Port N, Source Port M
  - Later messages from client to server
    - Destination Port M, Source Port N
      - Note: length not required to be multiple of 4

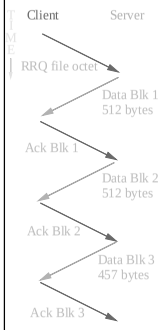
TFTP Opcodes

TFTP Protocol (client read)

- Client sends RRQ to server
  - containing filename and mode
  - mode is netascii or octet
- Server sends DATA block number 1
  - 512 bytes of data
- Client sends ACK of block number 1
- Server sends DATA block number 2
- Client sends ACK of block number 2
  - (etc)
- Server sends DATA block number N
  - last block
  - less than 512 data bytes (can be 0)
- Client sends ACK of block number N

- Transfer finished

TFTP timeline



Complete TFTP exchange for read of 1481 byte file

Lockstep Protocol

TFTP Protocol (client write)

- Client sends WRQ to server
  - containing filename and mode
- Server sends ACK of block number 0
- Client sends DATA block number 1
  - 512 bytes of data
- Server sends ACK of block number 1
- (etc)
- Client sends DATA block number N
  - less then 512 bytes of data
- Server sends ACK of block number N
- Transfer finished

TFTP Errors

- ERR packet used for application errors
  - No Such File
  - Permission Denied
  - (etc)
- Error Number can be used by software
  - (only 8 exist)
- Error message intended for humans
- Network errors handled by
  - timeout and retransmit

NetAscii

- The ASCII character set
  - basically English printable characters
- Lines terminated by
  - <Carriage-Return><Line-Feed> pair
- Server and Client must convert between
  - local filesystem format and NetAscii
    - before sending, and after receiving data
  - Unix uses just <Line-Feed> to terminate lines
  - MacOS uses just <Carriage-Return>
  - DOS (etc) use <CR><LF> pair to terminate lines
- Octet mode is binary transfer
  - no conversions done

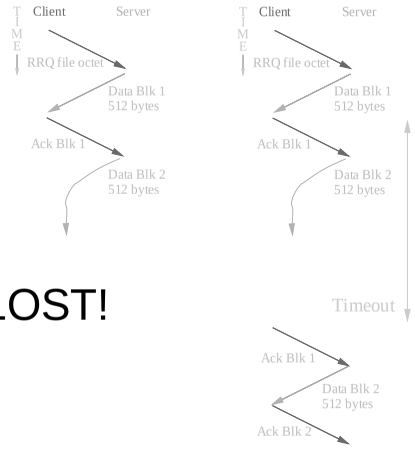
TFTP Evaluation

- Does the protocol work?
  - Lost packets?
  - Corrupted packets?
  - Out of order packets?
  - Duplicated packets?
  - Multiple simultaneous transfers?
  - Large files?

TFTP - lost packets?

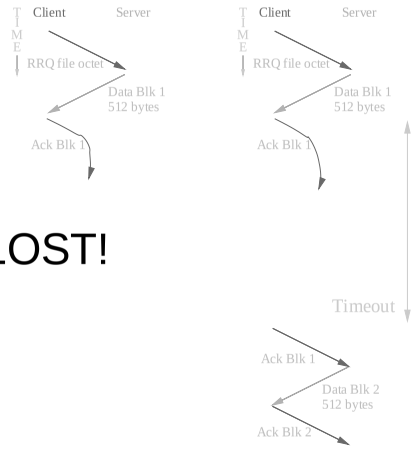
- Packet recovery
- Timeout and retransmit
- Block number in data packets
  - and ACK
- Good enough.

Lost packet retransmit



LOST!

Lost ACK?



LOST!

TFTP - corrupted packets

- Data corruption
- No TFTP checksum
  - Relies on UDP
    - and below
- UDP checksum optional
  - Can have no checks
- Not all that good
  - But just a bootstrap protocol