CSCE 515: Computer Network Programming
------ TFTP + Errors

Wenyuan Xu

Department of Computer Science and Engineering
University of South Carolina

TFTP Usage and Design

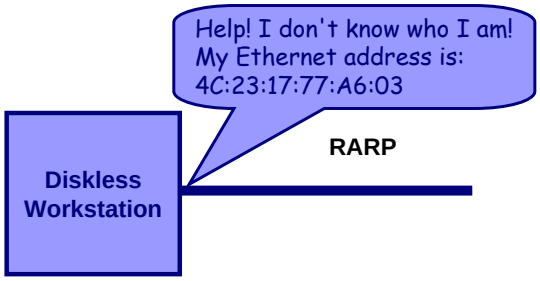
- RFC 783, 1350
- Transfer files between processes.
- Minimal overhead (no security).
- Designed for UDP, although could be used with many transport protocols.

TFTP Usage and Design (cont.)

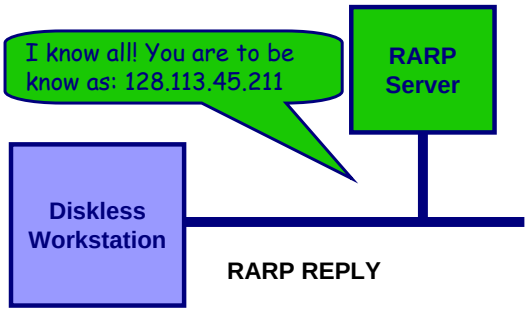
- Easy to implement
- Small - possible to include in firmware
- Used to bootstrap workstations and network devices.

Diskless Workstation Booting 1

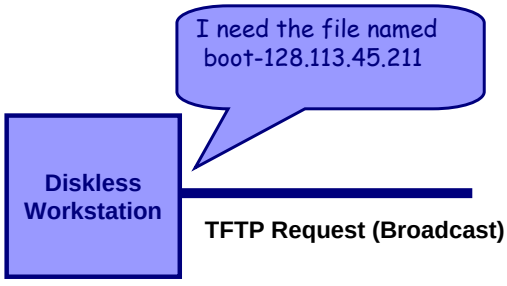
The call for help



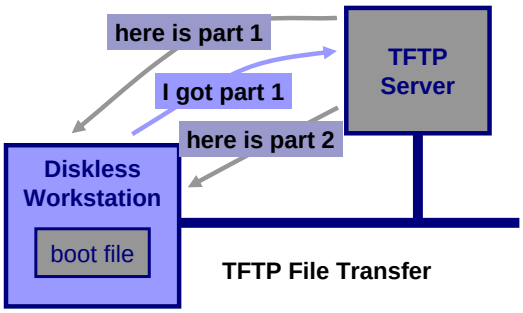
The answer from the all-knowing



The request for instructions



The dialog



TFTP File Transfer

TFTP Protocol

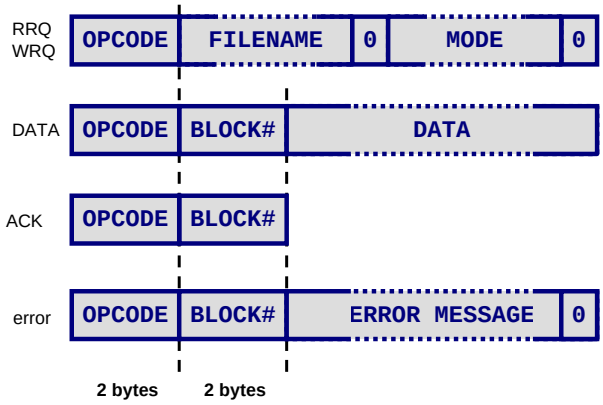
5 message types:

- Read request
- Write request
- Data
- ACK (acknowledgment)
- Error

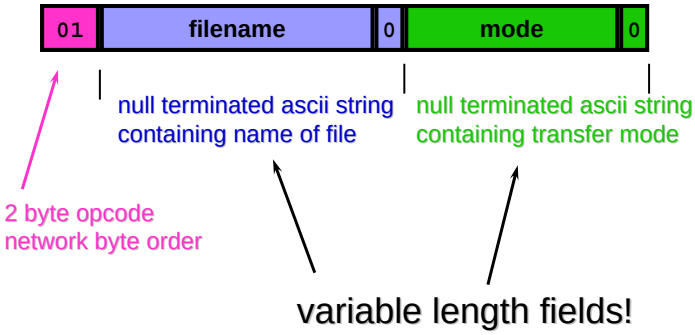
Messages

- Each is an independent UDP Datagram
- Each has a 2 byte opcode (1st 2 bytes)
- The structure of the rest of the datagram depends on the opcode.

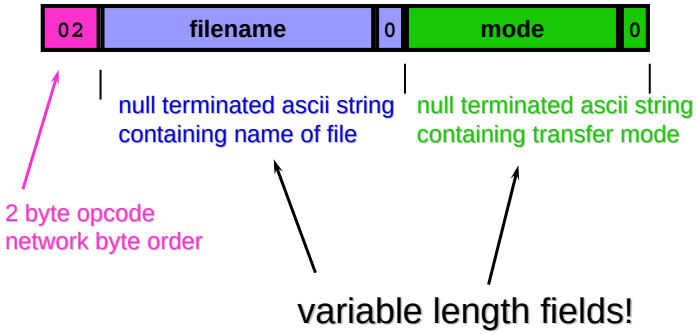
Message Formats



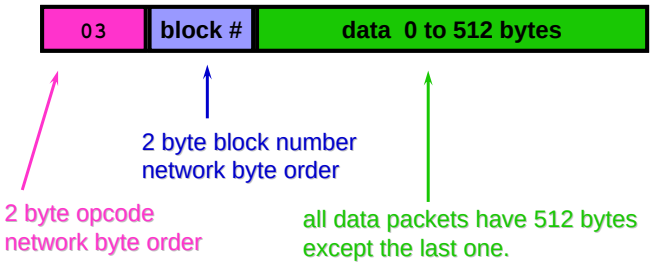
Read Request



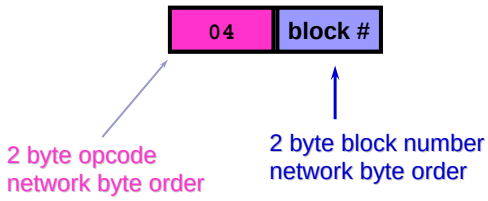
Write Request



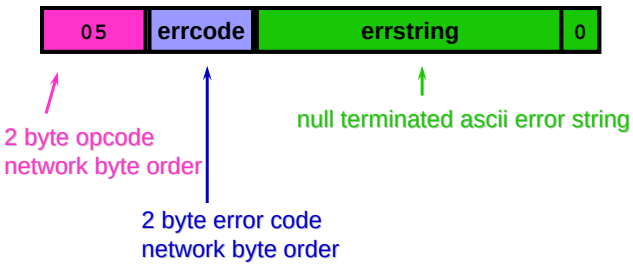
TFTP Data Packet



TFTP Acknowledgment



TFTP Error Packet



TFTP Error Codes (16 bit int)

0 - not defined
1 - File not found
2 - Access violation
3 - Disk full
4 - Illegal TFTP operation
5 - Unknown port
6 - File already exists
7 - No such user

TFTP transfer modes

- "netascii" : for transferring text files.
  - all lines end with \r\n (CR,LF).
  - provides standard format for transferring text files.
  - both ends responsible for converting to/from netascii format.
- "octet" : for transferring binary files.
  - no translation done.

NetAscii Transfer Mode

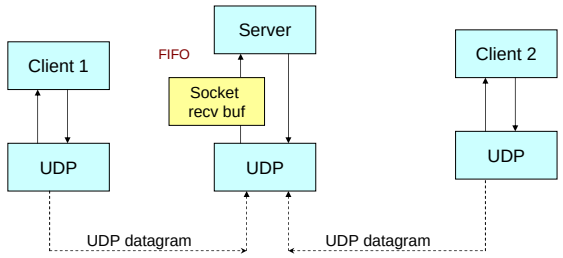
Unix - end of line marker is just '**\n**'

- receiving a file
  - you need to remove '**\r**' before storing data.
- sending a file
  - you need to replace every '**\n**' with "**\r\n**" before sending

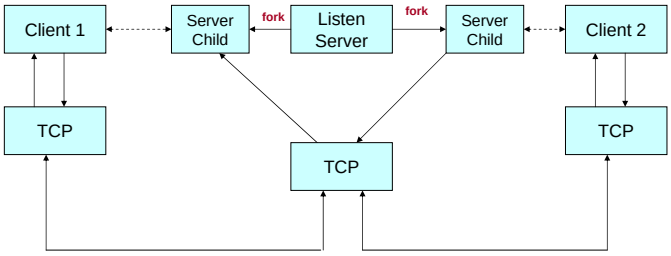
Concurrency

- TFTP servers use a "well known address" (UDP port number).
- How would you implement a concurrent server?
  - forking (alone) may lead to problems!
  - Can provide concurrency without forking, but it requires lots of bookkeeping.

UDP sockets



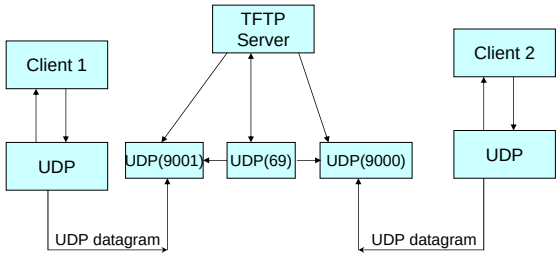
TCP sockets



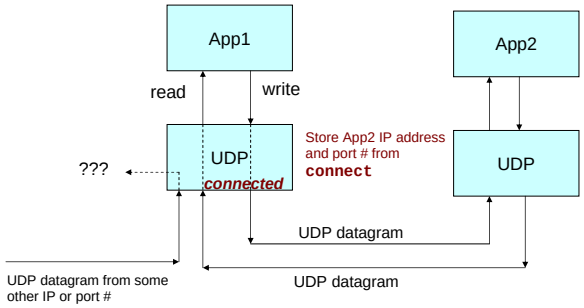
TFTP Concurrency

- According to the protocol, the server may create a *new udp port* and send the initial response from this new port.
- The client should recognize this, and send all subsequent messages to the new port.

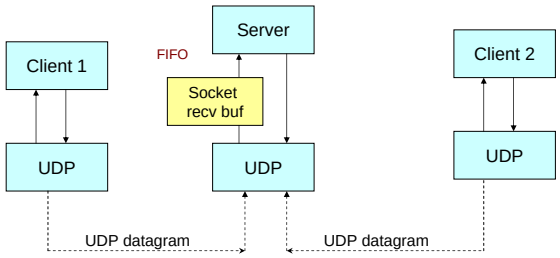
UDP sockets



Connected UDP socket



Who can call `connect`?



Questions

- Can UDP socket connected to a broadcast address?
  - A: yes, a connected UDP socket exchanges datagrams with only one IP address
- Server A is connected to a broadcast address, so….
  - Can this UDP socket send?
  - Can this UDP socket Receive?

RRQ (read request)

- Client sends RRQ
- Server sends back data chunk #1
- Client acks chunk #1
- Server sends data chunk #2
- ...

WRQ (write request)

- Client sends WRQ
- Server sends back ack #0
- Client data chunk #1 (the first chunk!)
- Server acks data chunk #1
- …

there is no data chunk #0!

- *Stop and wait*
  - *What's the advantage?*
  - *Disadvantage?*

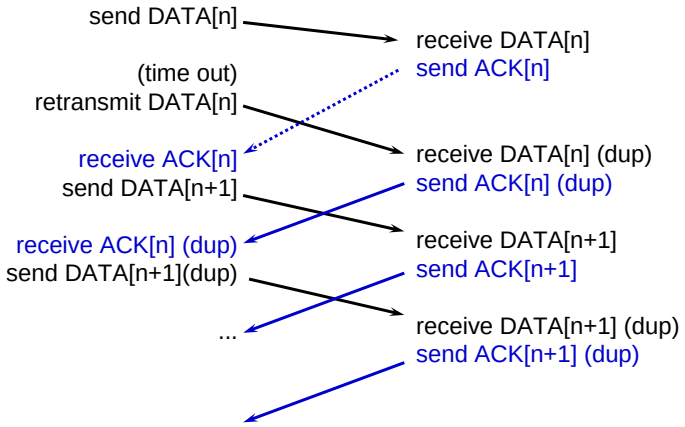
When is it over?

- There is no *length of file* field sent!
- All data messages *except the last one* contain 512 bytes of data.
- The last data message might contain 0 bytes of data!
- When to close the UDP socket?

Lost Data Packets - Original Protocol Specification

- Sender uses a timeout with retransmission.
  - sender could be client or server.
- Duplicate data packets must be recognized and ACK retransmitted.
- This original protocol suffers from the "sorcerer's apprentice syndrome".

Sorcerer's Apprentice Syndrome



The Fix

- Sender should not resend a data packet in response to a duplicate ACK.
- If sender receives ACK[n] - don't send DATA[n+1] if the ACK was a duplicate.

Security

- No username or password
- Obtain copies of Unix password file and then try to guess password
- Solution:
  - Only files in a specific directory can be accessed
  - Give lower access priority

Issues

What if more than 65535 chunks are sent?

- 65536 blocks x 512 bytes/block = 33,554,432 bytes.

- The RFC does not address this issue!
- Remember that the network can duplicate packets!

Error Handling

System Calls and Errors

- In general, systems calls return a negative number to indicate an error.
  - We often want to find out what error.
  - Servers generally add this information to a log.
  - Clients generally provide some information to the user.

`extern int errno;`

- Whenever an error occurs, system calls set the value of the global variable **`errno`**.
  - You can check errno for specific errors.
  - You can use support functions to print out or log an ASCII text error message.

When is **`errno`** valid?

- **`errno`** is valid only after a system call has returned an error.
  - System calls don't *clear* **`errno`** on success.
  - If you make another system call you may lose the previous value of **`errno`**.
    - **`printf`** makes a call to **`write`**!

Error codes

```
#include <errno.h>
```

Error codes are defined in errno.h

EAGAIN	**EBADF**	**EACCESS**
EBUSY	**EINTR**	**EINVAL**
EIO	**ENODEV**	**EPIPE**

…

Support Routines

```
void perror(const char *string);
```

In **`stdio.h`**

```
char *strerror(int errnum);
```

In **`string.h`**

General Strategies

- Include code to check for errors after every system call.
- Develop "wrapper functions" that do the checking for you.
- Develop layers of functions, each hides some of the error-handling details.

Example wrapper

```
int Socket( int f,int t,int p) {
  int n;
  if ( (n=socket(f,t,p)) < 0 )) {
     perror("Fatal Error");
     exit(1);
   }
  return(n);
}
```

What is fatal?

- How do you know what should be a fatal error (program exits)?
  - Common sense.
  - If the program can continue – it should.
  - Example – if a server can't create a socket, or can't bind to it's port - there is no sense continuing…

Wrappers are great!

- Wrappers like those used in the text can make code much more readable.
- There are always situations in which you cannot use the wrappers
  - Sometimes system calls are "interrupted" (EINTR) – this is not always a fatal error !

Word of Caution

- If you use the code from the book for your projects, you must understand it!
- The library of code used in the text is extensive:
  - Wrappers call custom error handing code.
  - Custom error handling code make assumptions about having other custom library functions.
  - …

Another approach

- Instead of simple wrapper functions, you might develop a *layered system*.
- The idea is to "hide" the **`sockaddr`** and error handling details behind a few custom functions:

```
int tcp_client(char *server, int port);
int tcp_server(int port);
```


Layers and Code Re-use

- Developing general functions that might be re-used in other programs is obviously "a good thing".
- Layering is beneficial even if the code is not intended to be re-used:
  - hide error-handling from "high-level" code.
  - hide other details.
  - often makes debugging easier.

The Best Approach to handling errors

- There is no *best approach*.
- Do what works for you.
- Make sure you check *all* system calls for errors!!!!
  - Not checking can lead to security problems!
  - Not checking can lead to bad grades on homework projects!

Assignment & Next time

- Reading:
  - TI 15 **
  - RFC 1350 TFTP*
- Next Lecture:
  - I/O Multiplexing and select()