Network Applications: FTP and the Web

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

9/19/2013

Outline

- Admin and recap
  - FTP
  - Web/HTTP
Assignment two (programming) will be posted tonight

Recap: UDP Sockets

server
Public address: 128.36.59.2
Local address: 127.0.0.1

InetAddress sIP1 = InetAddress.getByName("localhost");
DatagramSocket ssock1 = new DatagramSocket(9876, sIP1);

InetAddress sIP2 = InetAddress.getByName("128.36.59.2");
DatagramSocket ssock2 = new DatagramSocket(9876, sIP2);

DatagramSocket serverSocket = new DatagramSocket(6789);
UDP Demultiplexing

Server
Public address: 128.36.59.2
Local address: 127.0.0.1

Public address: 128.36.59.2
Local address: 127.0.0.1

UDP demultiplexing is based on matching (dst address, dst port)

Recap: UDP Demultiplexing

Server
Public address: 128.36.59.2
Local address: 127.0.0.1

UDP demultiplexing is based on matching (dst address, dst port)
Recap: Data Encoding/Decoding

- Pay attention to encoding/decoding of data: transport layer handles only a sequence of bytes

Recap: TCP Socket Demultiplexing
Recap: TCP Demultiplexing

Packet demultiplexing is based on (dst addr, dst port, src addr, src port)
Packet sent to the socket with the best match!

Example: Client Initiates Connection
Example: TCP Handshake Done

server
128.36.232.5
128.36.230.2
TCP socket space
state: listening
address: (*, 6789)
completed connection queue:
sendbuf:
recvbuf:

client
198.69.10.10
TCP socket space
state: connected
address: (198.69.10.10, 128.36.232.5:6789)
sendbuf:
recvbuf:

Recap: TCP Handshake Done

server
128.36.232.5
128.36.230.2
TCP socket space
state: listening
address: (*, 6789)
completed connection queue:
sendbuf:
recvbuf:

state: established
address: (128.36.232.5:6789, 198.69.10.10)
sendbuf:
recvbuf:

state: listening
address: (*, *, *)
completed connection queue:
sendbuf:
recvbuf:

client
198.69.10.10
TCP socket space
state: connected
address: (198.69.10.10, 128.36.232.5:6789)
sendbuf:
recvbuf:

state: listening
address: (*, *, *)
completed connection queue:
sendbuf:
recvbuf:
FTP: the File Transfer Protocol

- Transfer files to/from remote host
- Client/server model
  - client: side that initiates transfer (either to/from remote)
  - server: remote host
- ftp: RFC 959
- ftp server: port 21/20 (smtp 25, http 80)

FTP Commands, Responses

Sample commands:
- sent as ASCII text over control channel
- USER *username*
- PASS *password*
- PWD returns current dir
- STAT shows server status
- LIST returns list of file in current directory
- RETR *filename* retrieves (gets) file
- STOR *filename* stores file

Sample return codes:
- status code and phrase
- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can’t open data connection
- 452 Error writing file
FTP Protocol Design

- What is the simplest design of data transfer?

- See FTP.pdf

FTP: A Client-Server Application with Separate Control, Data Connections

- Two parallel TCP connections opened:
  - Control connection: exchange commands, responses between client, server. “out of band control”
  - Data connections: file data to/from server

Discussion: why does FTP separate control/data connections?

Q: How to create a new data connection?
**Traditional FTP: Client Specifies Port for Data Connection**

FTP client communicates with FTP server through a TCP control connection on port 21. The `PORT` command is used to explicitly specify the client port for the data connection. The server initiates the TCP data connection.

**Example using nc**

- Use `telnet` for the control channel
  - `telnet ftp.freebsd.org 21`

- Use `nc` (NetCat) to receive/send data with server
  - `nc -v -l 1024`
Problem of the Client PORT Approach

Many Internet hosts are behind NAT/firewalls that block connections initiated from outside.

FTP PASV: Server Specifies Data Port, Client Initiates Connection

Server initiates TCP data connection
serverip:sport

Client initiates TCP data connection of PASV returned serverip:sport
Example

- Use Wireshark to capture traffic
  - Using chrome to visit
    ftp://ftp.freebsd.org/pub/FreeBSD/README.TXT

Summary

- What are some interesting design features of the FTP protocol?
From Opaque Files to the Web

Web page:
- authored in HTML
- addressed by a URL
  - URL has two components:
    - host name, port number
    - path name

Most Web pages consist of:
- base HTML page, and
- several referenced objects

User agent: Explorer

User agent: Firefox

Server running Apache Web server

http://www.cs.yale.edu:80/index.html

The Web uses HTTP: hypertext transfer protocol
  - http1.0: RFC 1945
  - http1.1: RFC 2068

HTTP 1.0 Message Flow

Server waits for requests from clients

Client initiates TCP connection (creates socket) to server, port 80

Client sends request for a document

Web server sends back the document

TCP connection closed

Client parses the document to find embedded objects (images)
  - repeat above for each image
HTTP 1.0 Message Flow (more detail)

Suppose user enters URL www.cs.yale.edu/index.html


2. http client sends http request message (containing URL) into TCP connection socket

0. http server at host www.cs.yale.edu waiting for TCP connection at port 80.

1b. server “accepts” connection, ack. client

3. http server receives request message, forms response message containing requested object (index.html), sends message into socket (the sending speed increases slowly, which is called slow-start)


5. http client receives response message containing html file, parses html file, finds embedded image

6. Steps 1-5 repeated for each of the embedded images

HTTP 1.0 Message Flow (cont.)
Discussion

- How about we use FTP as HTTP?

HTTP Message Flow

- HTTP servers are stateless servers: each request is self-contained
### HTTP Request Message: General Format

- **ASCII (human-readable format)**

```
<table>
<thead>
<tr>
<th>method</th>
<th>sp</th>
<th>URL</th>
<th>sp</th>
<th>version</th>
<th>cr if</th>
</tr>
</thead>
<tbody>
<tr>
<td>request line</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>header field name : value</td>
<td>cr if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>header field name : value</td>
<td>cr if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cr if</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
Entity Body
```

### HTTP Request Message Example: GET

```
GET /somedir/page.html HTTP/1.0
Host: www.somechool.edu
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: en
(extra carriage return, line feed)
```

**Virtual host multiplexing**

**Connection management**

**Content negotiation**

**Carriage return, line feed indicates end of message**

**Header lines**
HTTP Response Message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
html file

HTTP/1.0 200 OK
Date: Wed, 23 Jan 2008 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ......
Content-Length: 6821
Content-Type: text/html
data data data data data ...

HTTP Response Status Codes

In the first line of the server->client response message. A few sample codes:

200 OK
  ❍ request succeeded, requested object later in this message

301 Moved Permanently
  ❍ requested object moved, new location specified later in this message (Location:)

400 Bad Request
  ❍ request message not understood by server

404 Not Found
  ❍ requested document not found on this server

505 HTTP Version Not Supported
Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:
   telnet www.yale.edu 80
   Opens TCP connection to port 80 (default http server port) at www.yale.edu.
   Anything typed in sent to port 80 at www.yale.edu.

2. Type in a GET http request:
   GET /index.html HTTP/1.0
   By typing this in (hit carriage return twice), you send this minimal (but complete)
   GET request to http server.

3. Look at response message sent by the http server.

Some interesting web site
- try en.wikipedia.org's main page
Design Exercise

- Workflow of an HTTP server processing a GET request that maps to a file:

```
GET /somedir/page.html HTTP/1.0
Host: www.somechool.edu
```

Basic HTTP Server Workflow

1. Create ServerSocket(6789)
2. `connSocket = accept()`
3. Read request from `connSocket`
4. Map URL to file
5. Read from file/write to `connSocket`
6. Close `connSocket`
Example Code

- See BasicWebServer.java
- Try using telnet and real browser

Dynamic Content Pages

- There are multiple approaches to make dynamic web pages:
  - Embedding code into pages (server side include)
    - HTTP server includes an interpreter for the type of pages
  - Invoke external programs (HTTP server is agnostic to the external program execution)

http://www.cs.yale.edu/index.shtml
http://www.cs.yale.edu/cgi-bin/ureserve.pl
http://www.google.com/search?q=Yale&sourceid=chrome