HTTP

Reading: Section 9.1.2 and 9.4.3

COS 461: Computer Networks
Spring 2013
Recap: Client-Server Communication

- **Client “sometimes on”**
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn’t communicate directly with other clients
  - Needs to know server’s address

- **Server is “always on”**
  - Handles services requests from many client hosts
  - E.g., Web server for the www.cnn.com Web site
  - Doesn’t initiate contact with the clients
  - Needs fixed, known address
Outline

• HTTP overview

• Proxies
Two Forms of Header Formats

• **Fixed:** Every field (type, length) defined
  – Fast parsing (good for hardware implementations)
  – Not human readable
  – Fairly static (IPv6 ~20 years to deploy)
  – E.g., Ethernet, IP, TCP headers

• **Variable length headers**
  – Slower parsing (hard to implement in hardware)
  – Human readable
  – Extensible
  – E.g., HTTP (Web), SMTP (Email), XML
HTTP Basics (Overview)

• HTTP over bidirectional byte stream (e.g. TCP)

• Interaction
  – Client looks up host (DNS)
  – Client sends request to server
  – Server responds with data or error
  – Requests/responses are encoded in text

• Stateless
  – HTTP maintains no info about past client requests
  – HTTP “Cookies” allow server to identify client and associate requests into a client session
HTTP Request

```
```

“cr” is \r

“lf” is \n
sp is " "

Entity Body
HTTP Request

• Request line
  – Method
    • GET – return URI
    • HEAD – return headers only of GET response
    • POST – send data to the server (forms, etc.)
  – URL (relative)
    • E.g., /index.html
  – HTTP version
HTTP Request (cont.)

• Request headers
  – Variable length, human-readable
  – Uses:
    • Authorization – authentication info
    • Acceptable document types/encodings
    • From – user email
    • If-Modified-Since
    • Referrer – what caused this page to be requested
    • User-Agent – client software

• Blank-line

• Body
# HTTP Response

## Structure

- **Status Line**
  - `version` (1 byte)
  - `sp` (1 byte)
  - `status code` (3 bytes)
  - `sp` (1 byte)
  - `phrase` (null-terminated string)
  - `cr` (1 byte)
  - `lf` (1 byte)

- **Header Lines**
  - `header field name` (null-terminated string)
  - `:` (1 byte)
  - `value` (null-terminated string)
  - `cr` (1 byte)
  - `lf` (1 byte)

- **Entity Body**
  - (Variable length)

---

**Example:**

```
HTTP/1.1 200 OK
Date: Tue, 19 Jan 2016 12:00:00 GMT
Content-Type: text/html
Content-Length: 131

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Example</title>
</head>
<body>
    <h1>Hello, World!</h1>
</body>
</html>
```
HTTP Response

• **Status-line**
  – HTTP version (now “1.1”)
  – 3 digit response code
    • 1XX – informational
    • 2XX – success
      – 200 OK
    • 3XX – redirection
      – 301 Moved Permanently
      – 303 Moved Temporarily
      – 304 Not Modified
    • 4XX – client error
      – 404 Not Found
    • 5XX – server error
      – 505 HTTP Version Not Supported
  – Reason phrase
HTTP Response (cont.)

• Headers
  – Variable length, human-readable
  – Uses:
    • Location – for redirection
    • Server – server software
    • WWW-Authenticate – request for authentication
    • Allow – list of methods supported (get, head, etc)
    • Content-Encoding – E.g x-gzip
    • Content-Length
    • Content-Type
    • Expires (caching)
    • Last-Modified (caching)

• Blank-line

• Body
HTTP Response Example

HTTP/1.1 200 OK
Date: Tue, 27 Mar 2001 03:49:38 GMT
Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod_ssl/2.7.1 OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24
Last-Modified: Mon, 29 Jan 2001 17:54:18 GMT
Accept-Ranges: bytes
Content-Length: 4333
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

.....
How to Mark End of Message?

• Close connection
  – Only server can do this
  – One request per TCP connection. Hurts performance.

• Content-Length
  – Must know size of transfer in advance

• No body content. Double CRLF marks end
  – E.g., 304 never have body content

• Transfer-Encoding: chunked (HTTP/1.1)
  – After headers, each chunk is content length in hex, CRLF, then body. Final chunk is length 0.
Example: Chunked Encoding

HTTP/1.1 200 OK
Transfer-Encoding: chunked

25
This is the data in the first chunk
1A
and this is the second one
0

• Especially useful for dynamically-generated content, as length is not a priori known
  – Server would otherwise need to cache data until done generating, and then go back and fill-in length header before transmitting
Outline

• HTTP overview

• Proxies
Proxies

• End host that acts a broker between client and server
  – Speaks to server on client’s behalf

• Why?
  – Privacy
  – Content filtering
  – Can use caching (coming up)
Proxies (Cont.)

• Accept requests from multiple clients
• Takes request and reissues it to server
• Takes response and forwards to client
Assignment 1: Requirements

• Non-caching, HTTP 1.0 proxy
  – Support only GET requests
  – No persistent connections: 1 HTTP request per TCP connection

• Multi-process: use fork()

• Simple binary that takes a port number
  – ./proxy 12345 (proxy listens on port 12345)

• Work in Firefox & Chrome
  – Use settings to point browser to your proxy
Assignment 1: Requirements

• What you need from a client request: host, port, and URI path
  - GET http://www.princeton.edu:80/ HTTP/1.0

• What you send to a remote server:
  - GET / HTTP/1.0
    Host: www.princeton.edu:80
    Connection: close

• Check request line and header format

• Forward the response to the client
Why Absolute vs. Relative URLs?

• First there was one domain per server
  – GET /index.html

• Then proxies introduced
  – Need to specify which server

• Then virtual hosting: multiple domains per server
  – GET /index.html
  – Host: www.cs.princeton.edu

• Absolute URL still exists for historical reasons and backward compatibility
Assignment 1: Requirements

• Non-GET request?
  – return “Not Implemented” (code 501)

• Unparseable request?
  – return “Bad Request” (code 400)

• Use provided parsing library
Advice

• Networking is hard
  – Hard to know what’s going on in network layers
  – Start out simple, test often

• Build in steps
  – Incrementally add pieces
  – Make sure they work
  – Will help reduce the effect of “incomplete” information

• Assume teaching staff is non malicious or trying to trick you
Assignment 1 – Getting Started

• Modify Assn 0 to have server respond
  – Simple echo of what client sent
• Modify Assn 0 to handle concurrent clients
  – Use fork()
• Create “proxy” server
  – Simply “repeats” client msg to a server, and “repeats” server msg back
• Client sends HTTP requests, proxy parses
Summary

• **HTTP: Simple text-based file exchange protocol**
  – Support for status/error responses, authentication, client-side state maintenance, cache maintenance

• **How to improve performance**
  – Proxies
  – Caching
  – Persistent connections (more later)
Pop Quiz!

• Advantage of “fast retransmit” over timeouts?

• When are fast retransmits possible?

• When are timeouts particularly expensive?