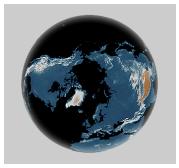


HTTP

Reading: Section 9.1.2 and 9.4.3

COS 461: Computer Networks
Spring 2012

Outline

- HTTP overview
- Proxies
- Persistent HTTP
- HTTP caching

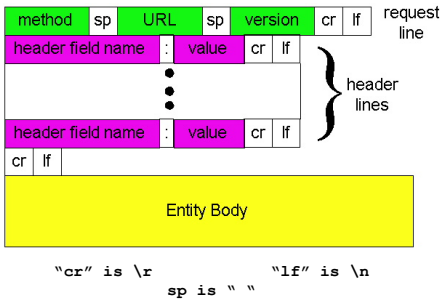
Application Layer Protocols

- Variable Headers vs. Fixed Headers
  - App headers handled by program rather than hardware
  - Variable headers allow for incrementally adding features
- Human Readable
  - Easy for programmers to reason about
  - Parsed by humans / programs rather than hardware
- More later on, but useful for understanding HTTP's design

HTTP Basics (Overview)

- HTTP layered over bidirectional byte stream
  - Almost always 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
  - Server maintains no info about past client requests

HTTP Request



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

7

HTTP Request Example

GET /index.html HTTP/1.1
Host: www.example.com

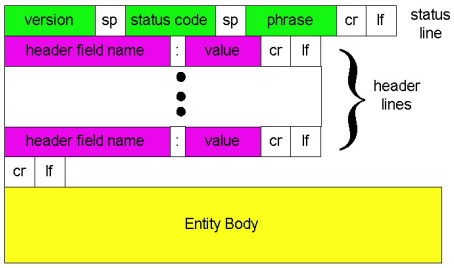
8

HTTP Request Example

GET /index.html HTTP/1.1
Host: www.example.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Connection: Keep-Alive

9

HTTP Response



10

HTTP Response

- Status-line
  - HTTP version
  - 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

11

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

12

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?

- Content-Length
  - Must know size of transfer in advance
- Close connection
  - Only server can do this
- Implied length
  - 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 <CRLF>
Transfer-Encoding: chunked <CRLF>
<CRLF>
25 <CRLF>
This is the data in the first chunk <CRLF>
1A <CRLF>
and this is the second one <CRLF>
0 <CRLF>
```

- 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

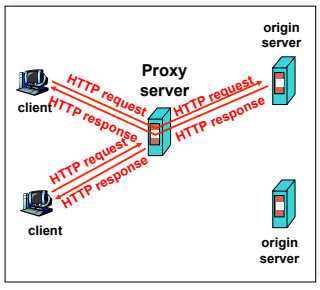
- HTTP overview
- Proxies
- Persistent HTTP
- HTTP caching

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

- Non-caching, HTTP 1.0 proxy
  - Support only GET requests
- 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

19

Assignment 1 (Cont.)

- 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

20

Assignment 1 (Cont.)

- Non-GET request?
  - return "Not Implemented" (code 501)
- Unparseable request?
  - return "Bad Request" (code 400)
- Use provided parsing library
- Postel's law
  - Be liberal in what you accept, and conservative in what you send
  - convert HTTP 1.1 request to HTTP 1.0
  - convert \r to \r\n
  - etc

21

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

22

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

23

Outline

- HTTP overview
- Proxies
- Persistent HTTP
- HTTP caching

24

Problems with simple model

- Simple model
  - request one object a time, sequentially
- Multiple connection setups
  - Connection setup for each item (imgs, js, etc)
- Short transfers are hard on TCP
- Lots of extra connections
  - Increases server state/processing

25

Persistent HTTP

- Reuse connection
  - Request header "Connection: Keep-Alive"
  - Reduces # of connection setups
- Benefits
  - Reduces server overhead
  - Reduces latency (i.e., faster page loads)
  - Allows pipelining

26

Pipelining

- Issue multiple requests at a time
  - Don't have to wait for previous response
  - More efficient use of link
- Use carefully
  - POST requests should not be pipelined (changes server state)
  - GET/HEAD requests are usually okay

27

Persistent HTTP

Non-persistent HTTP issues:

- Connection setup for each request
- But browsers often open parallel connections

Persistent HTTP:

- Server leaves connection open after sending response
- Subsequent HTTP messages between same client/server are sent over connection

Persistent without pipelining:

- Client issues new request only when previous response has been received
- One RTT for each object

Persistent with pipelining:

- Default in HTTP/1.1 spec
- Client sends multiple requests
- As little as one RTT for all the referenced objects
- Server must handle responses in same order as requests

28

"Persistent without pipelining" most common

- When does pipelining work best?
  - Small objects, equal time to serve each object
  - Small because pipelining simply removes additional 1 RTT delay to request new content
- Alternative design?
  - Multiple parallel connections (typically 2-4). Also allows parallelism at server
  - Doesn't have problem of head-of-line blocking like pipelining
    - Dynamic content makes HOL blocking possibility worse
- In practice, many servers don't support, and many browsers do not default to pipelining

29

Outline

- HTTP overview
- Proxies
- Persistent HTTP
- HTTP caching

30

HTTP Caching

- Why cache?
  - Lot of objects don't change (images, js, css)
  - Reduce # of client connections
  - Reduce server load
  - Reduce overall network traffic; save $$$

31

Caching is Hard

- Significant fraction (>50%?) of HTTP objects uncachable
  - Dynamic data: Stock prices, scores, web cams
  - CGI scripts: results based on passed parameters
  - Cookies: results may be based on passed data
  - SSL: encrypted data is not cacheable
  - Advertising / analytics: owner wants to measure # hits
    - Random strings in content to ensure unique counting
- Want to limit staleness of cached objects

32

Validating Cached Objects

- Timestamps
  - Server hints when an object "expires" (Expires: xxx)
  - Server provides last modified date, client can check if that's still valid
  - Why the server's timestamp?
- Problems
  - Server replicas won't agree on time
  - Objects may go back to previous value, and using time will have you redownload the object
- There are other ways (look up ETags)

33

Example Cache Check Request

GET / HTTP/1.1
Accept-Language: en-us
If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT
Host: www.example.com
Connection: Keep-Alive

34

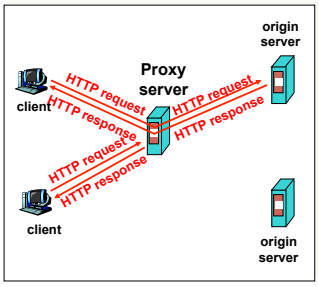
Example Cache Check Response

HTTP/1.1 304 Not Modified
Date: Tue, 27 Mar 2001 03:50:51 GMT
Connection: Keep-Alive

35

Web Proxy Caches

- User configures browser: Web accesses via cache
- Browser sends all HTTP requests to cache
  - Object in cache: cache returns object
  - Else: cache requests object from origin, then returns to client



36

Summary

- HTTP: Simple text-based file exchange protocol
  - Support for status/error responses, authentication, client-side state maintenance, cache maintenance
- How to improve performance
  - Persistent connections
  - Pipelining
  - Proxies
  - Caching

37