Introduction to Computer Networks

COSC 4377

Lecture 3

Spring 2012

January 25, 2012

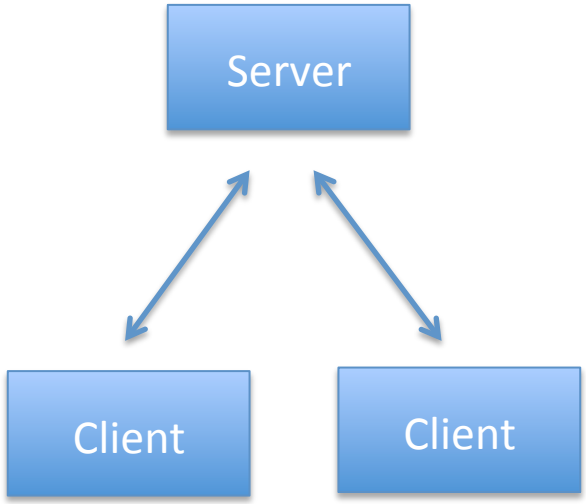
Announcements

- Four HW0 still missing
- HW1 due this week
- Start working on HW2 and HW3
- Re-assess if you found HW0/HW1 challenging
- Lecture recording posted online

HW Discussion Summary

- Give an overview of the most important issues raised and their resolution
- Two slides
- No more than 5 minutes
- Send your (ppt) slides by noon of your assigned day

Network Applications



Client-Server

Hybrid

Peer-to-peer

http://en.wikipedia.org/wiki/Peer-to-peer

1.

2.

3.

1.

2.

3.

Using TCP/IP

- How can applications use the network?
- *Sockets* API.
  - Originally from BSD, widely implemented (*BSD, Linux, Mac OS X, Windows, …)
  - Higher-level APIs build on them
- After basic setup, much like files

Today's Topic

HTTP and the Web

Precursors

- 1945, Vannevar Bush, Memex:
  - *"a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility"*
- Precursors to hypertext
  - *"The human mind [...] operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain*"
- Read his 1945 essay, "As we may think"
  - http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/3881/

Tim Berners-Lee

- Physicist at CERN, trying to solve real problem
  - Distributed access to data
- WWW: distributed database of pages linked through the Hypertext Transfer Protocol
  - First HTTP implementation: 1990
  - HTTP/0.9 – 1991
    - Simple GET commant
  - HTTP/1.0 – 1992
    - Client/server information, simple caching
  - HTTP/1.1 – 1996
    - Extensive caching support
    - Host identification
    - Pipelined, persistent connections, …

Components

- Content
  - Objects (may be static or dynamically generated)
- Clients
  - Send requests / Receive responses
- Servers
  - Receive requests / Send responses
  - Store or generate content
- Proxies
  - Placed between clients and servers
  - Provide extra functions
    - Caching, anonymization, logging, transcoding, filtering access
  - Explicit or transparent

Ingredients

- HTTP
  - Hypertext Transfer Protocol
- HTML
  - Language for description of content
- Names (mostly URLs)

URLs

protocol://[name@]hostname[:port]/directory/resource?k1=v1&k2=v2#tag

- *Name* is for possible client identification
- *Hostname* could be an IP address
- *Port* defaults to protocol default (e.g., 80)
- *Directory* is a path to the resource
- *Resource* is the name of the object
- ?*parameters* are passed to the server for execution
- #*tag* allows jumps to named tags within document

Examples of URL

- http://www2.cs.uh.edu/~gnawali/courses/cosc4377-s12/schedule.html
- http://en.wikipedia.org/wiki/Domain_name#Top-level_domains
- http://www.uh.edu/search/?q=computer+science&x=0&y=0

You will need to parse URLs like this for HW3

HTTP

- Important properties
  - Client-server protocol
  - Protocol (but not data) in ASCII
  - Stateless
  - Extensible (header fields)
- Server typically listens on port 80
- Server sends response, may close connection (client may ask it to say open)
- Currently version 1.1

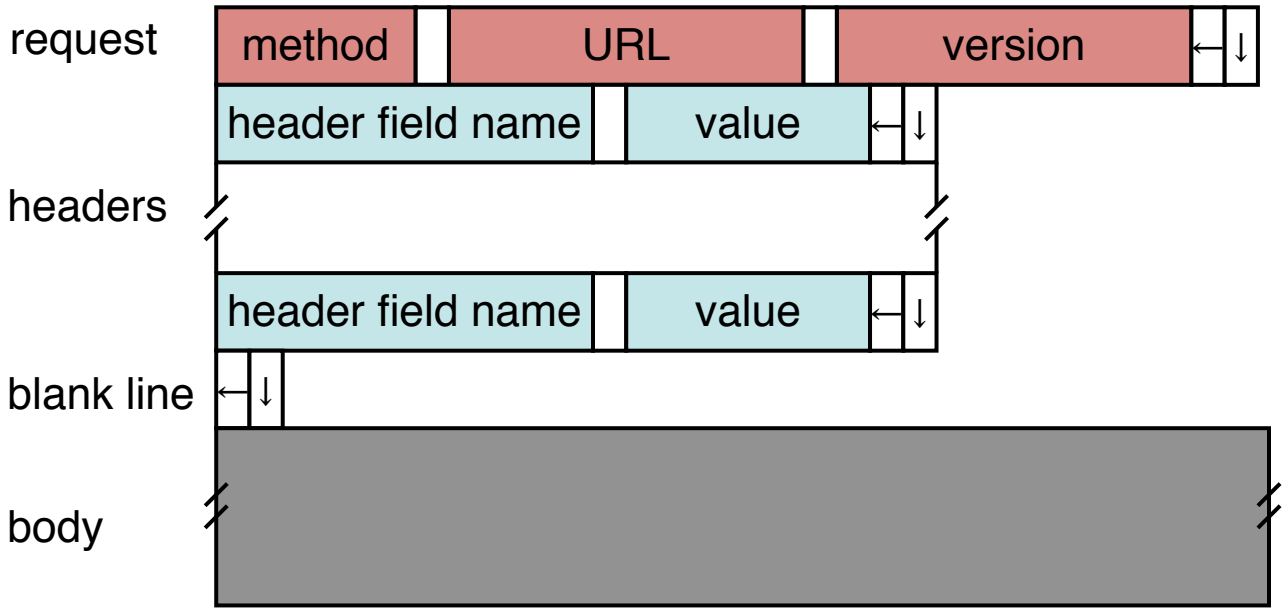
Steps in HTTP Request

- Open TCP connection to server
- Send request
- Receive response
- TCP connection terminates
  - How many RTTs for a single request?
- You may also need to do a DNS lookup first!

Telnet demo

Header inspection

HTTP Request



- Method:
  - GET: current value of resource, run program
  - HEAD: return metadata associated with a resource
  - POST: update a resource, provide input for a program
- Headers: useful info for proxies or the server
  - E.g., desired language

Sample Browser Request

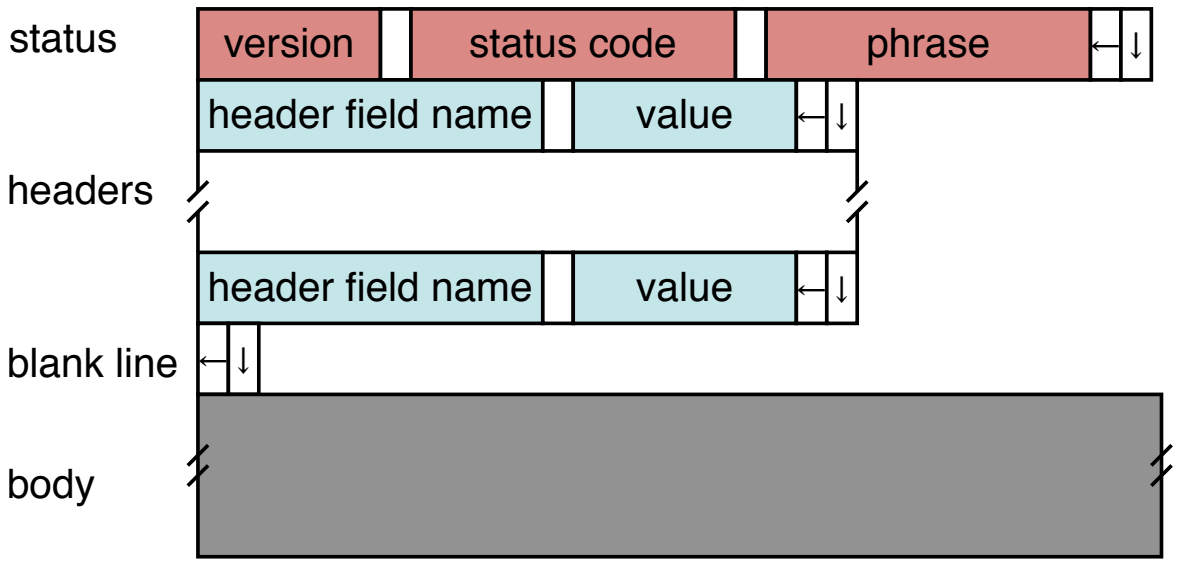
```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
(empty line)
```

Sample HTTP Response

```
HTTP/1.0 200 OK
Date: Wed, 25 Jan 2012 08:11:09 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie: PREF=ID….
 P3P: CP="This is not a P3P policy! See http://
   www.google.com/support/accounts/bin/answer.py?
   hl=en&answer=151657 for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<!doctype html><html><head><meta http-equiv="content-type"
   content="text/html; charset=ISO-8859-1"><meta…
```

HTTP Response



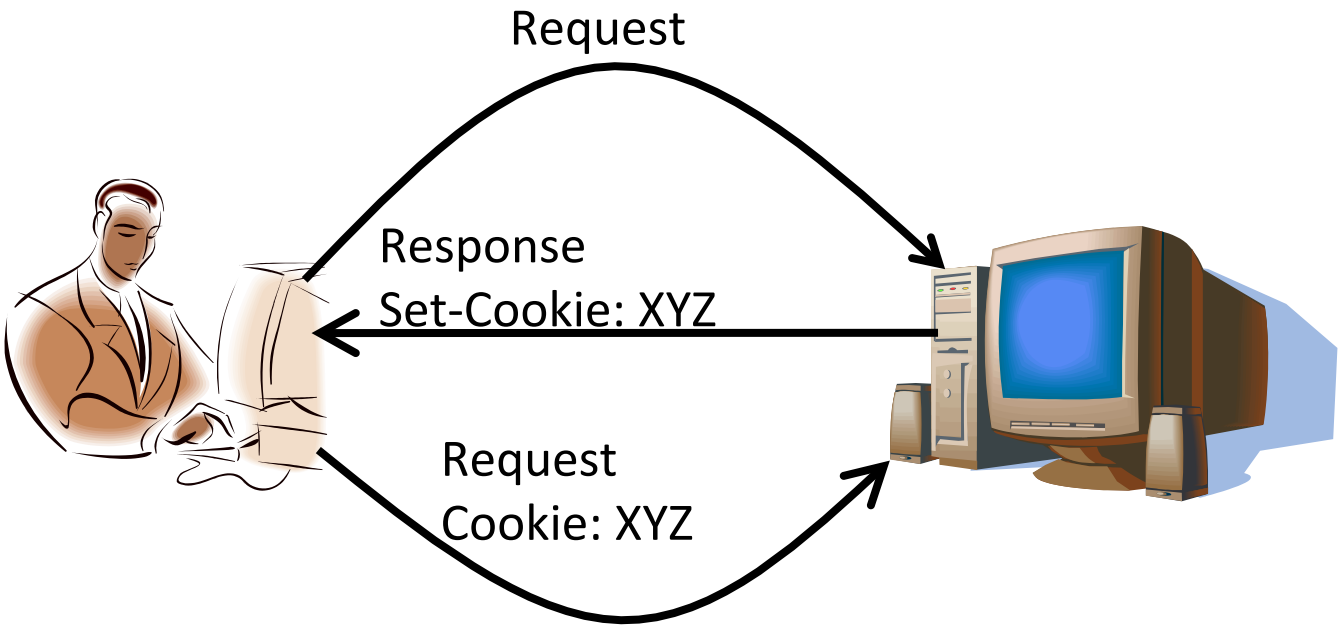
- Status Codes:
  - 1xx: Information e.g, 100 Continue
  - 2xx: Success e.g., 200 OK
  - 3xx: Redirection e.g., 302 Found (elsewhere)
  - 4xx: Client Error e.g., 404 Not Found
  - 5xx: Server Error e.g, 503 Service Unavailable

HTTP is Stateless

- Each request/response treated independently
- Servers not required to maintain state
- This is good!
  - Improves server scalability
- This is also bad…
  - Some applications need persistent state
  - Need to uniquely identify user to customize content
  - E.g., shopping cart, web-mail, usage tracking, (most sites today!)

HTTP Cookies

- Client-side state maintenance
  - Client stores small state on behalf of server
  - Sends request in future requests to the server
  - Cookie value is meaningful to the server (e.g., session id)
- Can provide authentication
- http://en.wikipedia.org/wiki/HTTP_cookie



Where to find official HTTP specification?

www.w3.org

Anatomy of a Web Page

- HTML content
- A number of additional resources
  - Images
  - Scripts
  - Frames
- Browser makes one HTTP request for each object
  - Course web page: 4 objects
  - My facebook page this morning: 100 objects

What about AJAX?

- *Asynchronous Javascript and XML*
- Based on XMLHttpRequest object in browsers, which allow code in the page to:
  - Issue a new, non-blocking request to the server, without leaving the current page
  - Receive the content
  - Process the content
- Used to add interactivity to web pages
  - XML not always used, HTML fragments, JSON, and plain text also popular

HTTP Performance

- What matters for performance?
- Depends on type of request
  - Lots of small requests (objects in a page)
  - Some big requests (large download or video)

Small Requests

- Latency matters
- RTT dominates
- Two major causes:
  - Opening a TCP connection
  - Actually sending the request and receiving response
  - And a third one: DNS lookup!
- Mitigate the first one with persistent connections (HTTP/1.1)
  - Which also means you don't have to "open" the connection each time

Browser Request

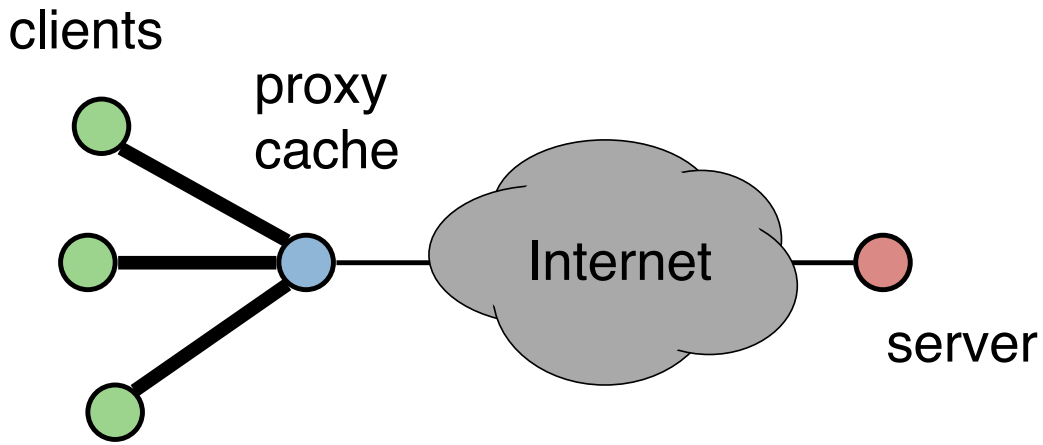
```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Small Requests (cont)

- Second problem is that requests are serialized
  - Similar to stop-and-wait protocols!
- Two solutions
  - Pipelined requests (similar to sliding windows)
  - Parallel Connections
    - HTTP standard says no more than 2 concurrent connections per host name
    - Most browsers use more (up to 8 per host, ~35 total)
  - How are these two approaches different?
  - http://en.wikipedia.org/wiki/HTTP_pipelining

Larger Objects

- Problem is throughput in bottleneck link
- Solution: HTTP Proxy Caching
  - Also improves latency, and reduces server load




?
?
?
?
Client
Proxy
Server