Network Applications: HTTP/1.0/1.1/2

Qiao Xiang, Congming Gao, Qiang Su

https://sngroup.org.cn/courses/cnns-xmuf25/index.shtml

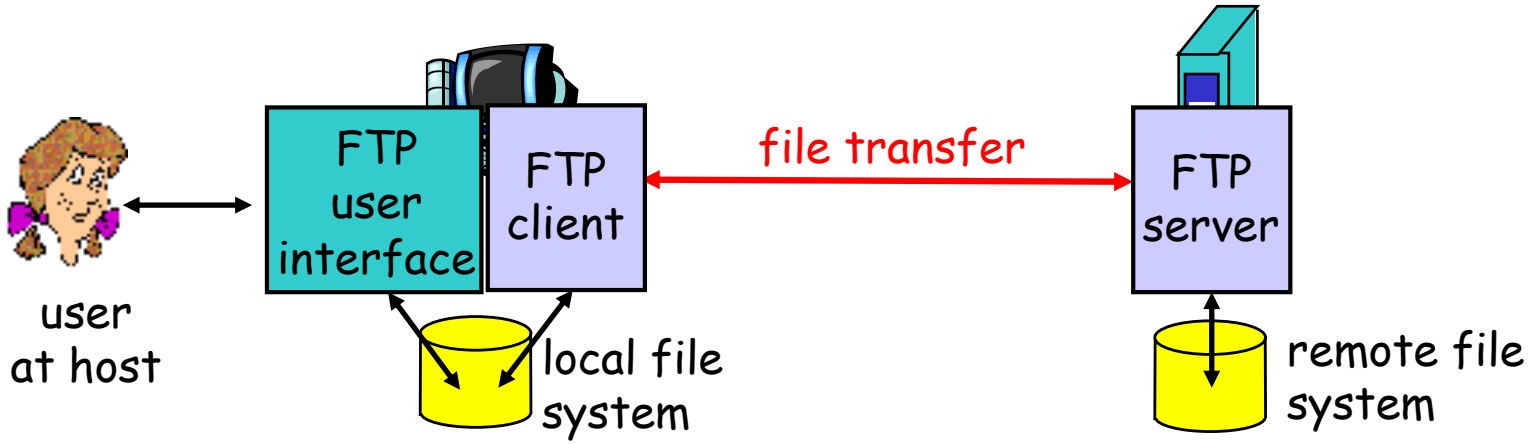
09/30/2025

This deck of slides are heavily based on CPSC 433/533 at Yale University, by courtesy of Dr. Y. Richard Yang.

Outline

- Admin. and recap
- Network applications (continue)
  - File transfer (FTP) and extension
  - *HTTP*

Recap: FTP



- 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)

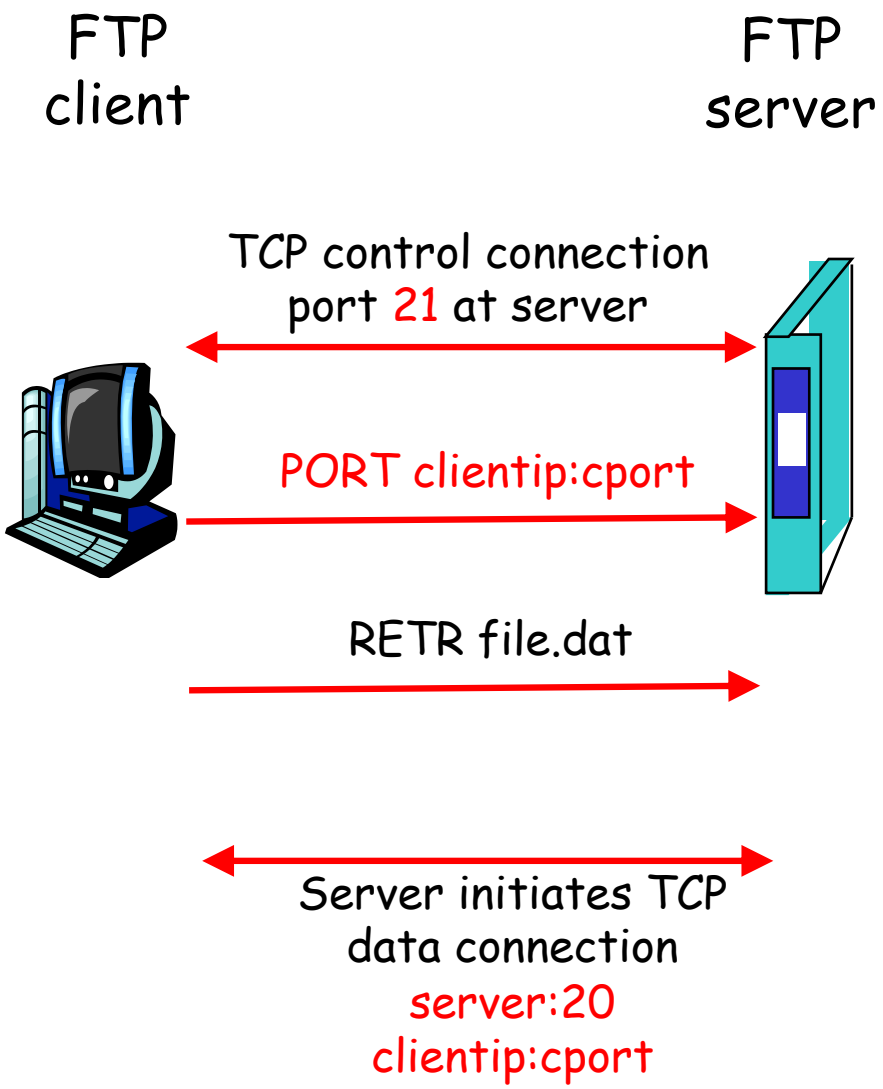
FTP: A Client-Server Application with Separate Control, Data Connections

- Two types of TCP connections opened:
  - A control connection: exchange commands, responses between client, server.
    "out of band control"
  - Data connections: each for file data to/from server

Discussion: why does FTP separate control/data connections?

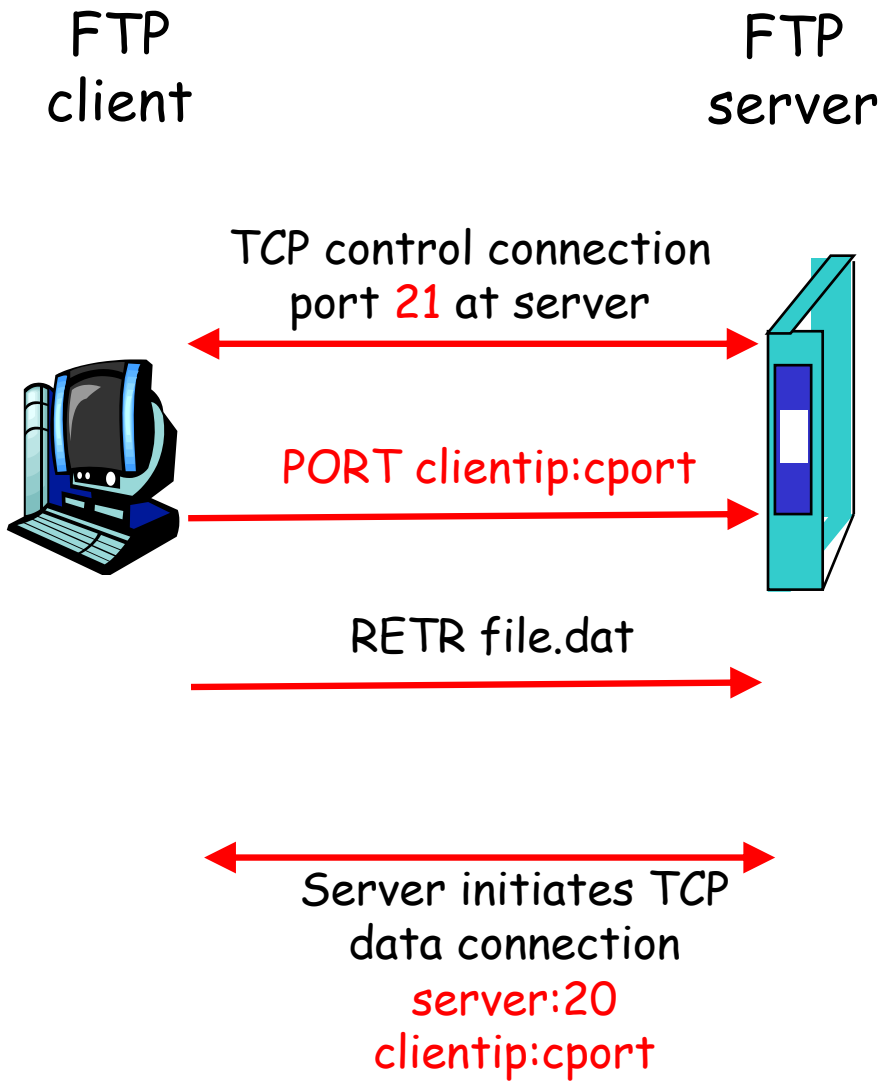
Q: How to create a new data connection?

Traditional FTP: Client Specifies Port for Data Connection

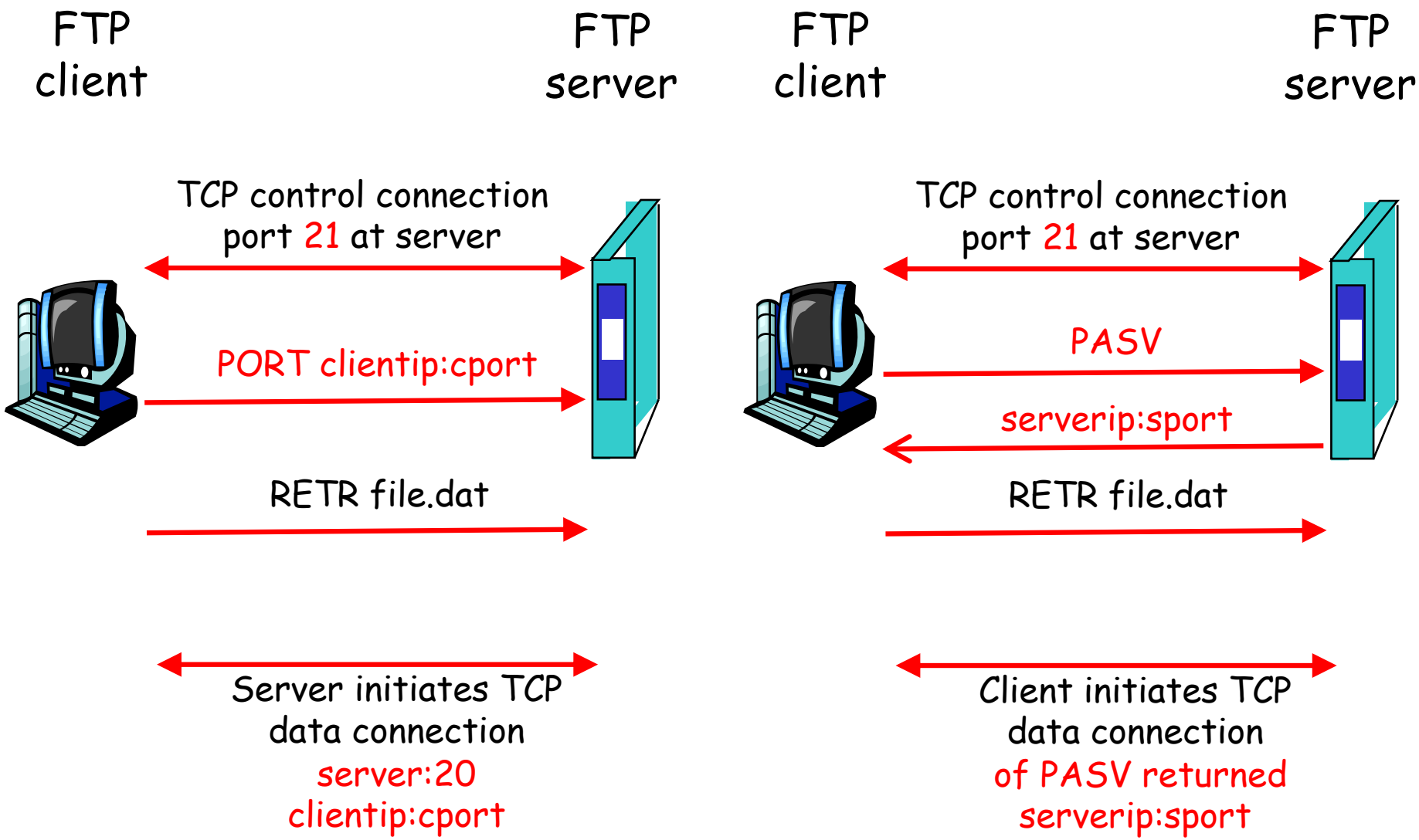


Problem of the Client PORT Approach

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



FTP PASV: Server Specifies Data Port, Client Initiates Connection



Outline

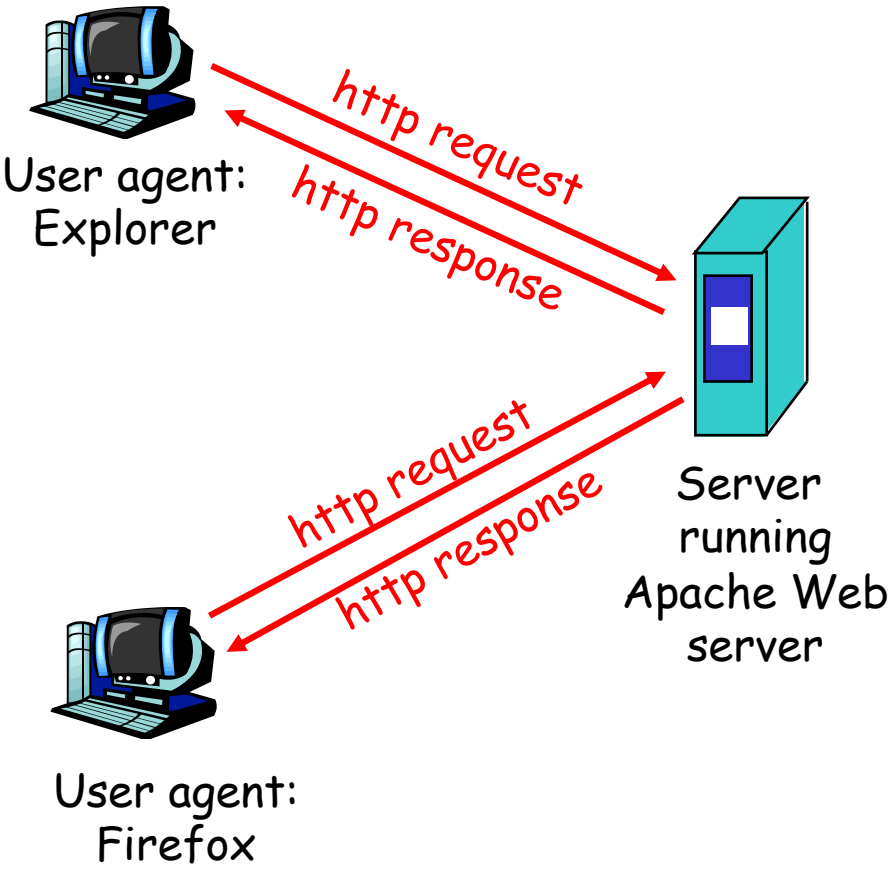
- Admin. and recap
- Network application programming
  - UDP sockets
  - TCP sockets
- Network applications (continue)
  - File transfer (FTP) and extension
  - *HTTP*

From Opaque Files to Web Pages

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

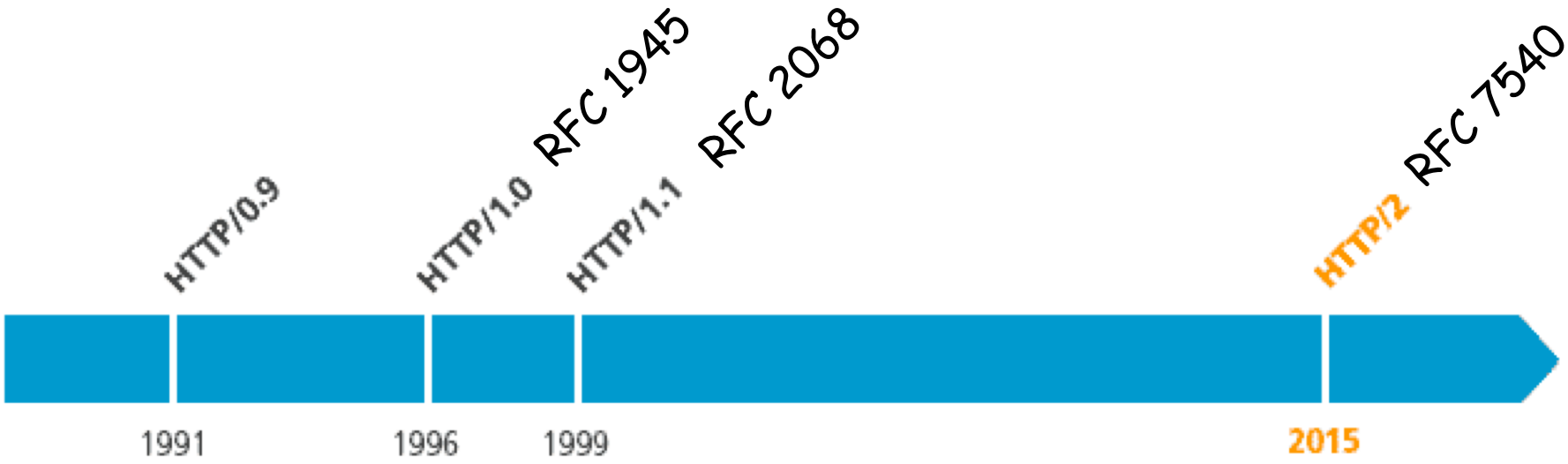
`http://qiaoxiang.me:80/index.html`

- Most Web pages consist of:
  - base HTML page, and
  - several referenced objects
    - E.g., image



The Web pages are requested through HTTP: hypertext transfer protocol

HTTP is Still Evolving



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
qiaoxiang.me/index.html

0. http server at host qiaoxiang.me waiting for TCP connection at port 80.

1a. http client initiates TCP connection to http server (process) at qiaoxiang.me. Port 80 is default for http server.

1b. server "accepts" connection, ack. client

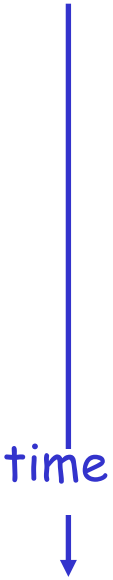
2. http client sends http *request message* (containing URL) into TCP connection socket

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)

time

HTTP 1.0 Message Flow (cont.)

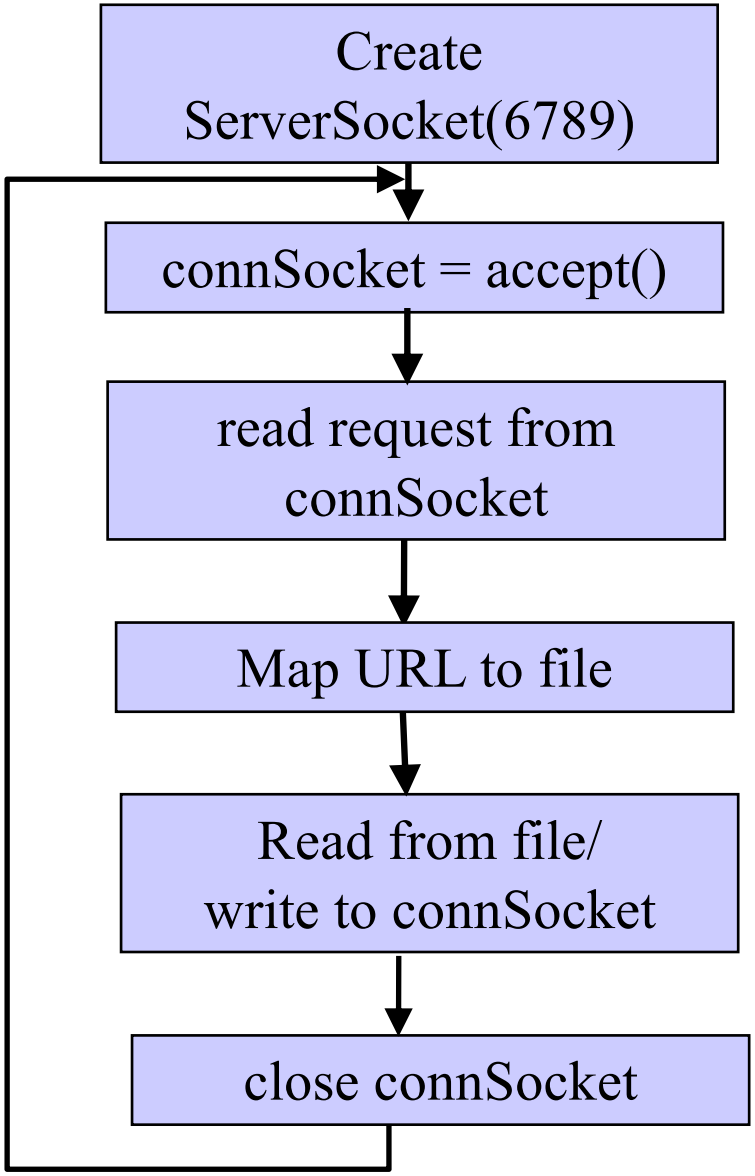
4. http server closes TCP connection.

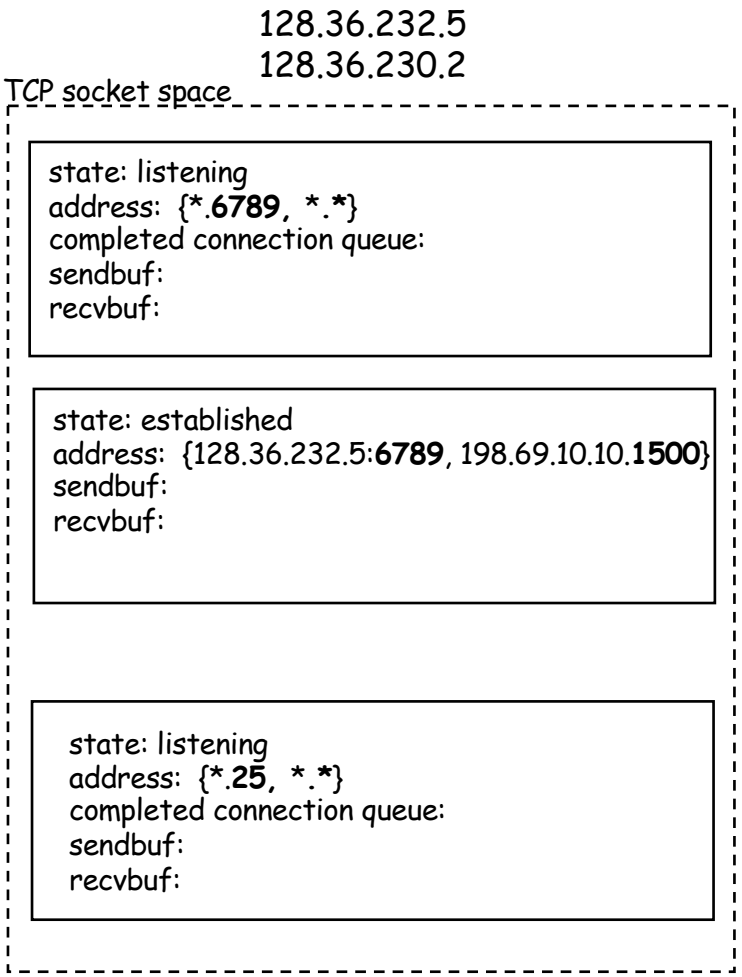


5. http client receives response message containing html file, parses html file, finds embedded image

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

Basic HTTP Server Workflow





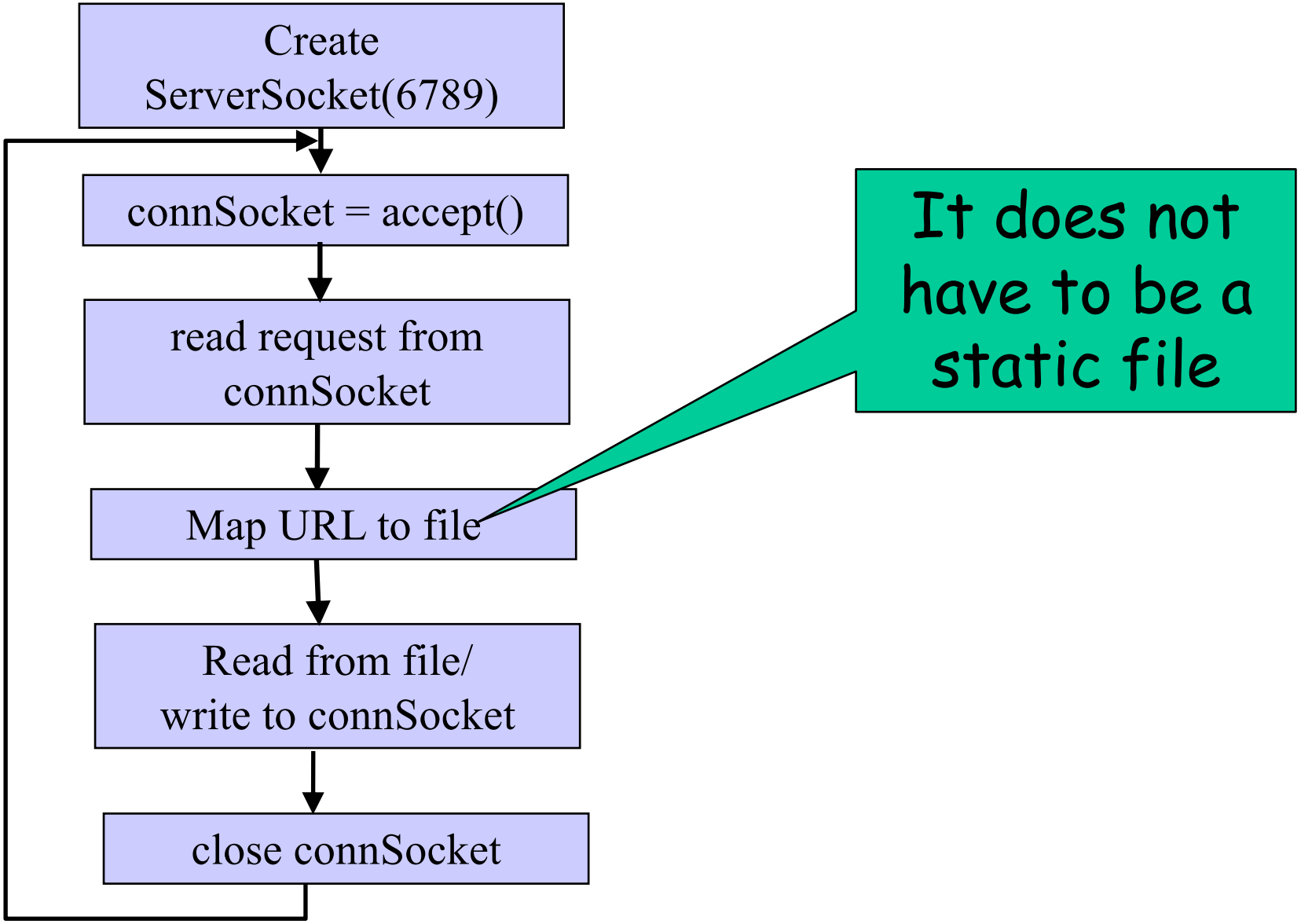
Example Code

- See BasicWebServer.java

- Try using telnet and real browser, and fetch
  - file1.html
  - index.html

  what difference in behavior?

Static -> Dynamic Content



Outline

- Admin and recap
- HTTP/1.0
  - Basic design
  - *Dynamic content*

Dynamic Content Pages

- There are multiple approaches to make dynamic web pages:
  - Embed 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)
    - E.g., Common Gateway Interface (CGI)

`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`

Example SSI

- See programming/examples-java-socket/BasicWebServer/ssi/index.shtml, header.shtml, ...

Example SSI

- See programming/examples-java-socket/BasicWebServer/ssi/index.shtml, header.shtml, …

- To enable ssi, need configuration to tell the web server (see conf/apache-htaccess)
  - https://httpd.apache.org/docs/2.2/howto/htaccess.html (Server Side Includes example)

CGI: Invoking External Programs

- Two issues
  - Input: Pass HTTP request parameters to the external program
  - Output: Redirect external program output to socket

Example: Typical CGI Implementation

- Starts the executable as a child process
  - Passes HTTP request as environment variables
    - `http://httpd.apache.org/docs/2.2/env.html`
    - CGI standard: http://www.ietf.org/rfc/rfc3875
  - Redirects input/output of the child process to the socket

Example: CGI

- Example:
  - `GET /search?q=Yale&sourceid=chrome HTTP/1.0`
  - setup environment variables, in particular $QUERY_STRING=`q=Yale&sourceid=chrome`
  - start `search` and redirect its input/output

https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html

https://httpd.apache.org/docs/2.2/howto/htaccess.html
(CGI Example)

Example

- http://172.28.229.215/BasicWebServer/cgi/price.cgi?appl

```
#!/usr/bin/perl -w

$company = $ENV{'QUERY_STRING'};
print "Content-Type: text/html\r\n";
print "\r\n";

print "<html>";
print "<h1>Hello! The price is ";

if ($company =~ /appl/) {
  my $var_rand = rand();
  print 450 + 10 * $var_rand;
} else {
  print "150";
}

print "</h1>";
print "</html>";
```

Client Using Dynamic Pages

- See ajax.html and wireshark for client code example

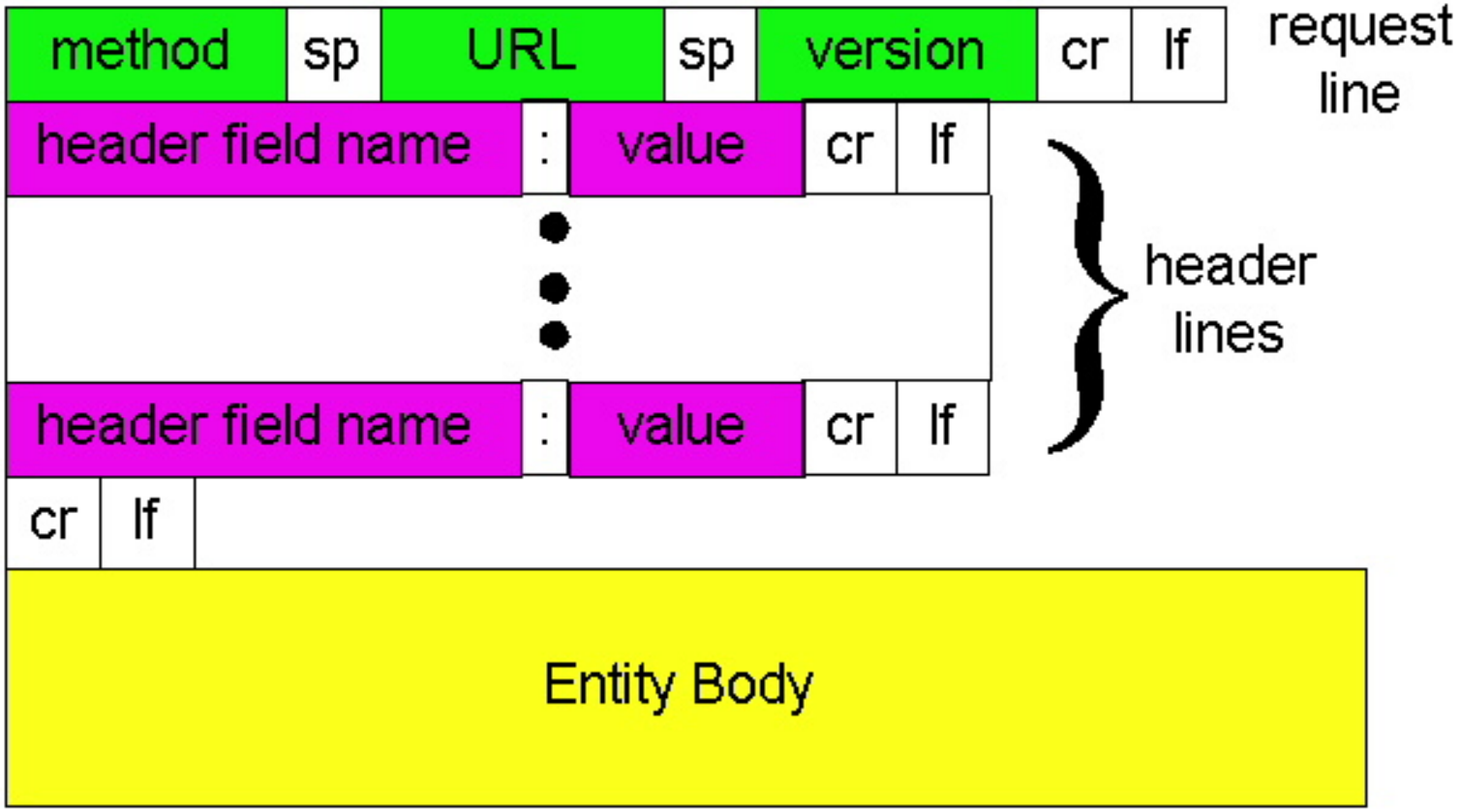
  http://172.28.229.215/BasicWebServer/cgi/ajax.html

Discussions

- What features are missing in HTTP that we have covered so far?

HTTP: POST

- If an HTML page contains forms or parameter too large, they are sent using POST and encoded in message body



HTTP: POST Example

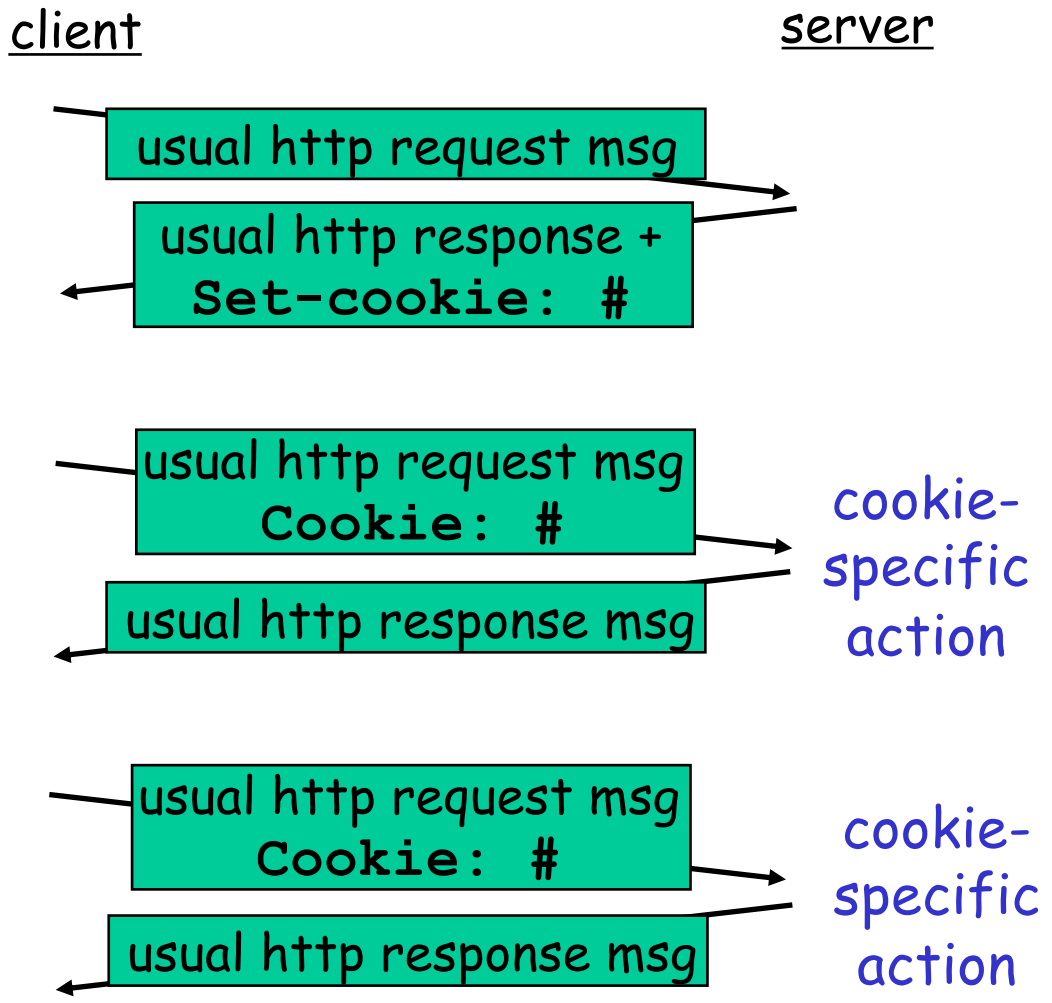
```
POST /path/script.cgi HTTP/1.0
User-Agent: MyAgent
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

item1=A&item2=B
```

Example using nc:
programming/examples-java-socket/BasicWebServer/nc/

Stateful User-server Interaction: Cookies

Goal: no explicit application level session

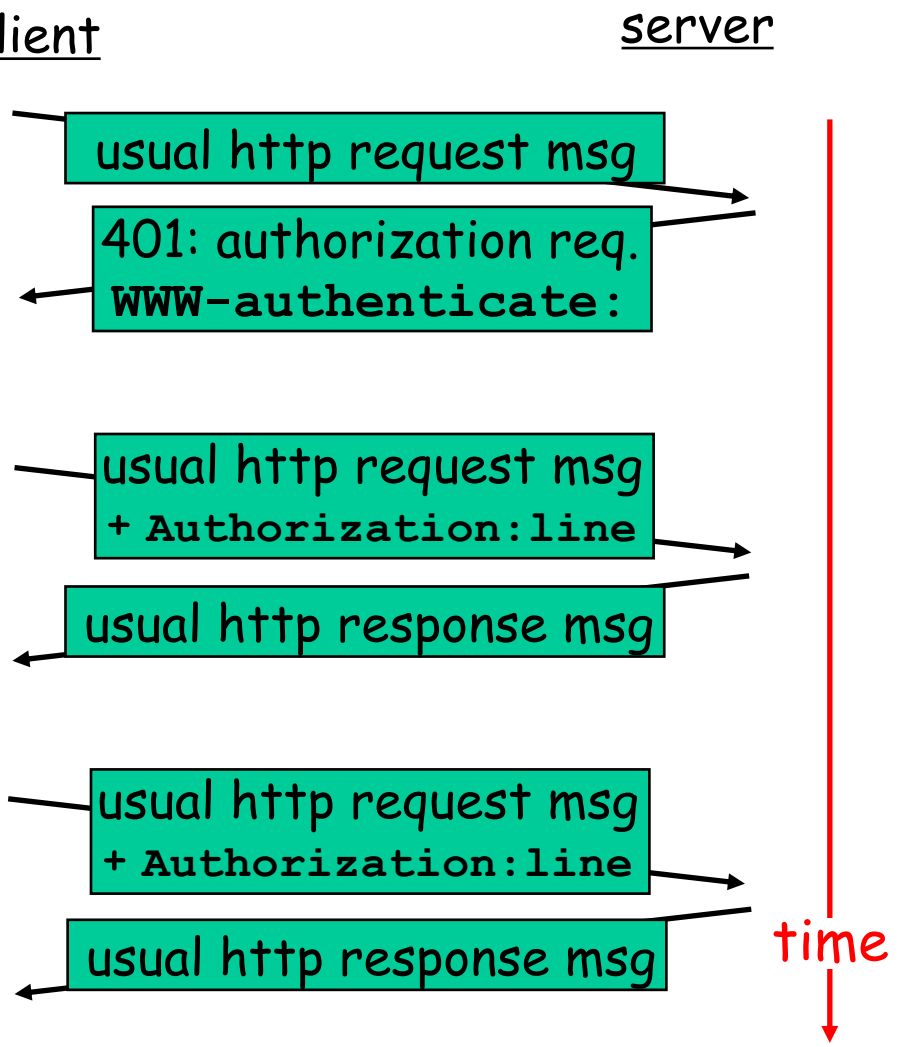
- Server sends "cookie" to client in response msg
  - `Set-cookie: 1678453`
- Client presents cookie in later requests
  - `Cookie: 1678453`
- Server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices



Authentication of Client Request

Authentication goal: control access to server documents

- stateless: client must present authorization in each request
- authorization: typically name, password
  - `Authorization:` header line in request
  - if no authorization presented, server refuses access, sends `WWW-authenticate:` header line in response

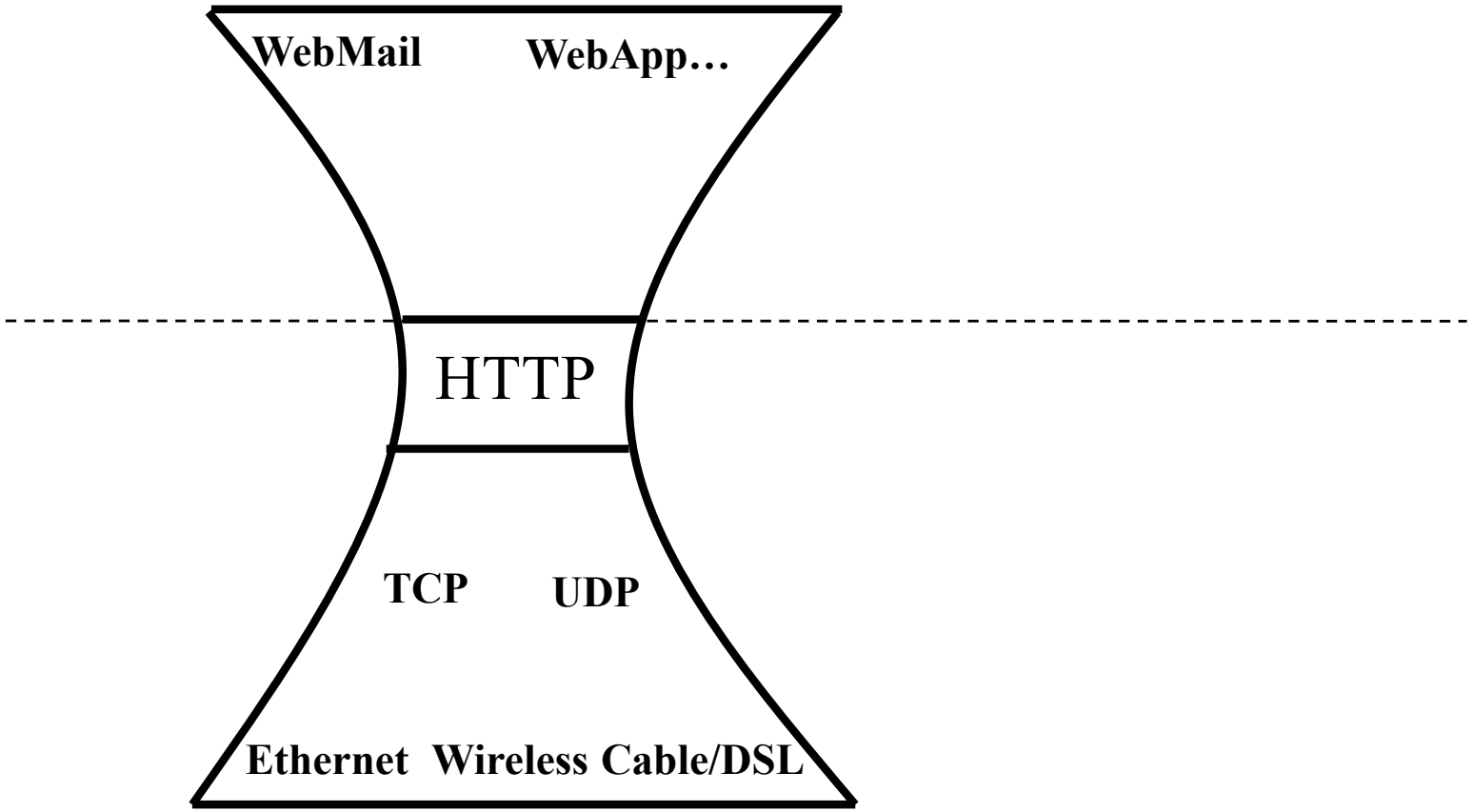


Browser caches name & password so that user does not have to repeatedly enter it.

Example: Amazon S3

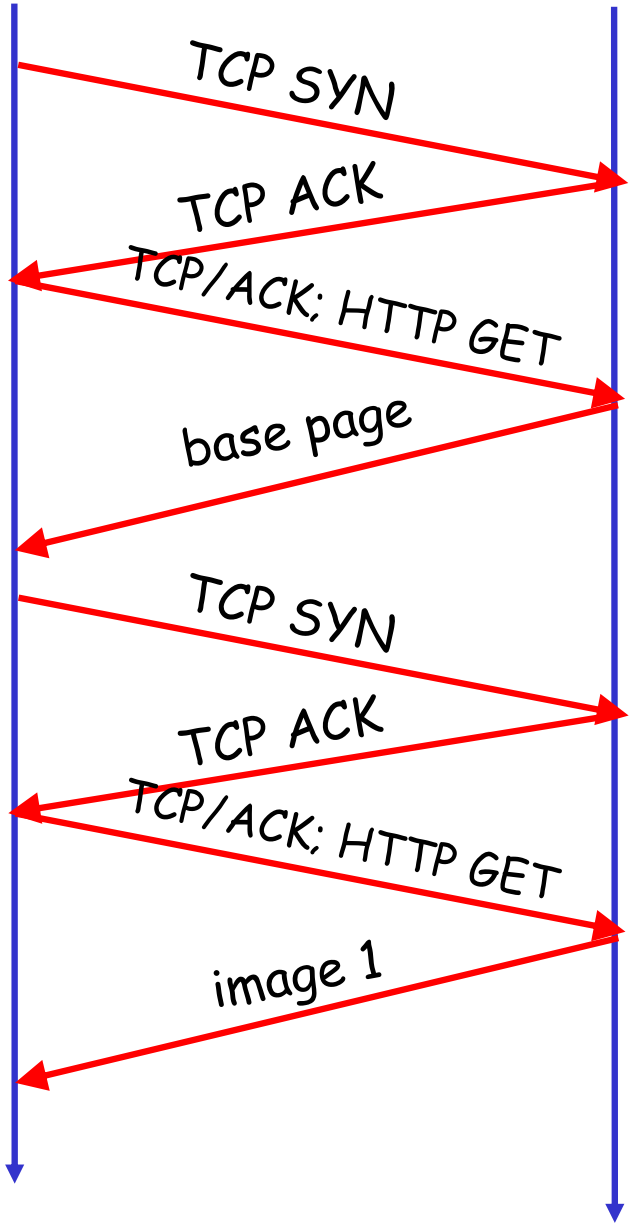
- Amazon S3 API
  - http://docs.aws.amazon.com/AmazonS3/latest/API/APIRest.html

HTTP as the Thin Waist



Protocol Flow of Basic HTTP/1.0

- >= 2 RTTs per object:
  - TCP handshake --- 1 RTT
  - client request and server responds --- at least 1 RTT (if object can be contained in one packet)



Outline

- Admin and recap
- HTTP/1.0
- *HTTP "acceleration"*

Substantial Efforts to Speedup Basic HTTP/1.0

- Reduce the number of objects fetched [Browser cache]

- Reduce data volume [Compression of data]
- Header compression [HTTP/2]

- Reduce the latency to the server to fetch the content [Proxy cache]
- Remove the extra RTTs to fetch an object [Persistent HTTP, aka HTTP/1.1]

- Increase concurrency [Multiple TCP connections]
- Asynchronous fetch (multiple streams) using a single TCP [HTTP/2]

- Server push [HTTP/2]

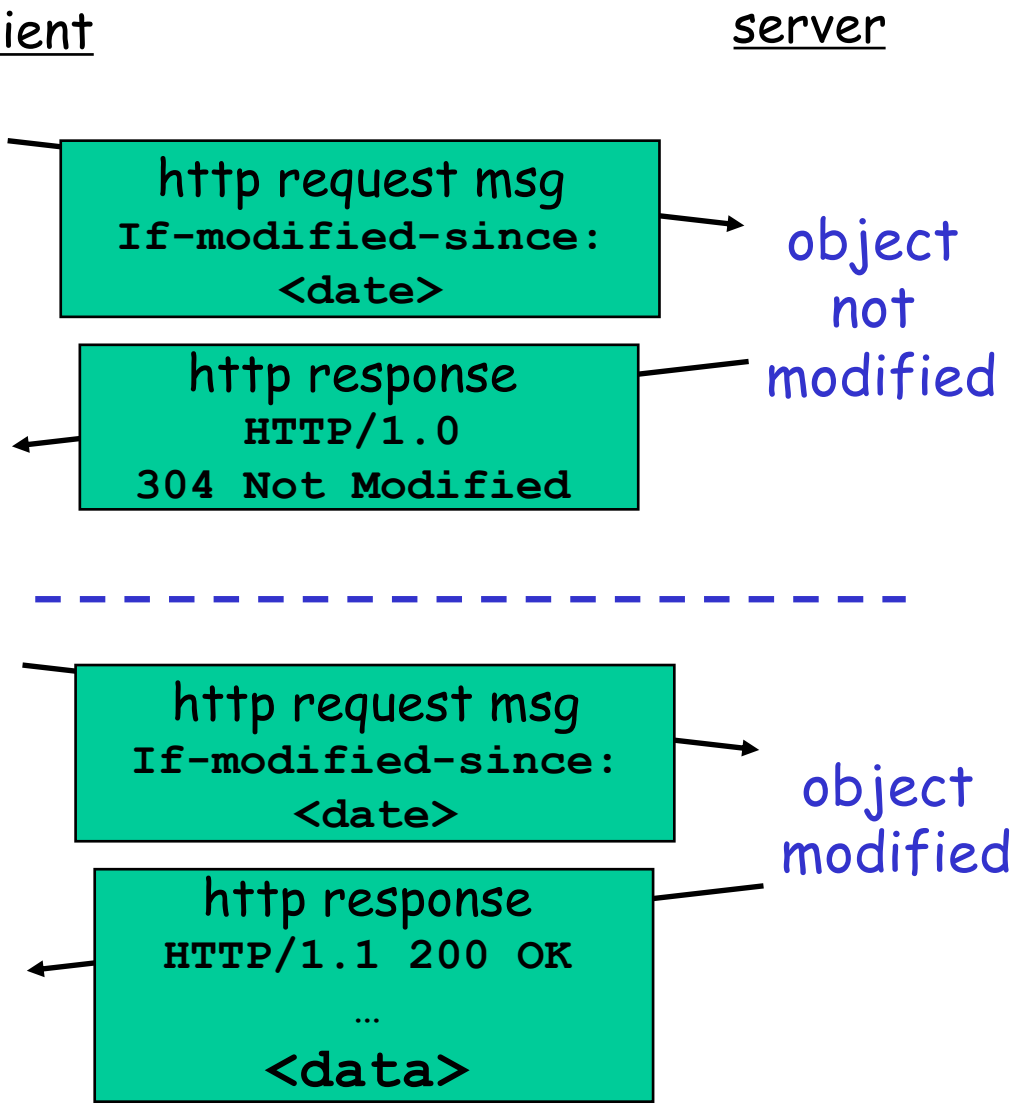
Browser Cache and Conditional GET

- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request

```
If-modified-since:
   <date>
```

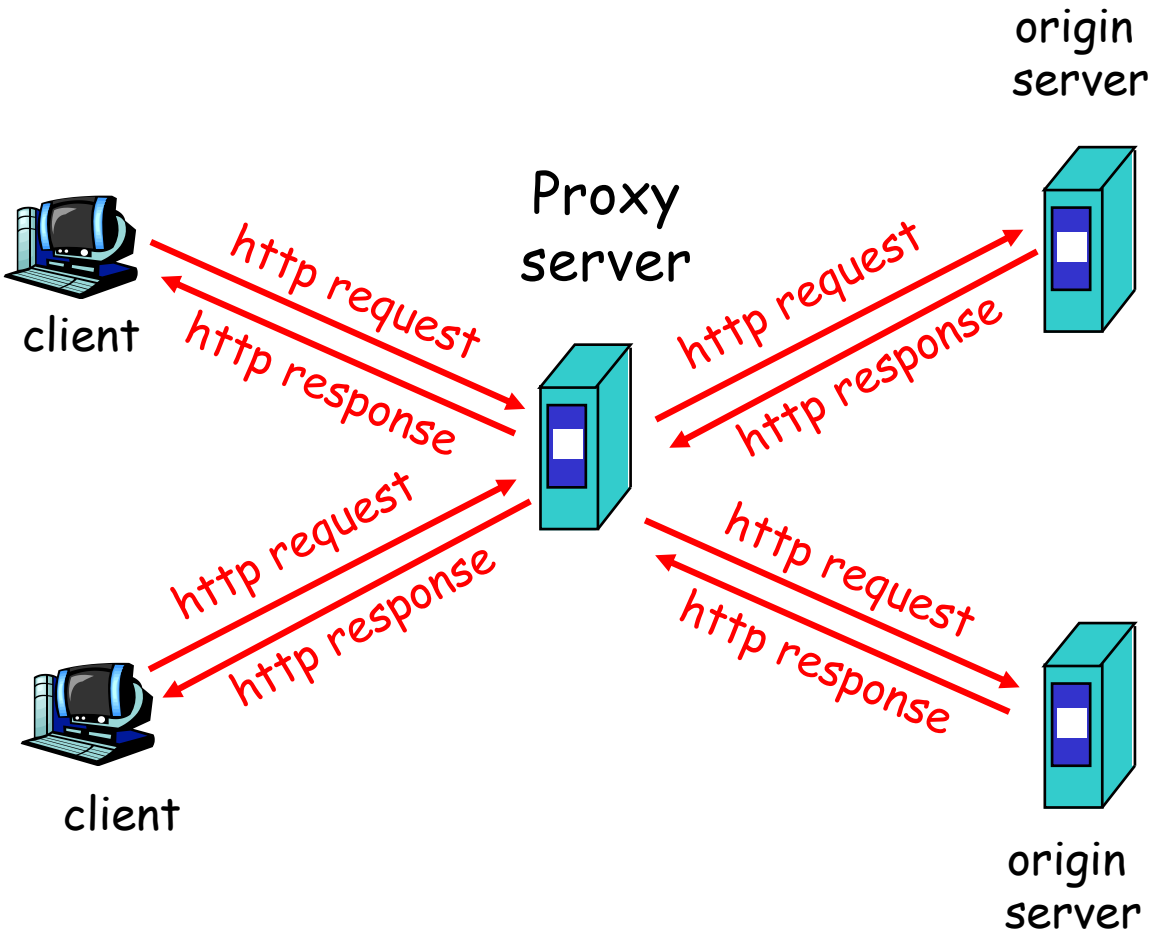
- server: response contains no object if cached copy up-to-date:

```
HTTP/1.0 304 Not
   Modified
```



Web Caches (Proxy)

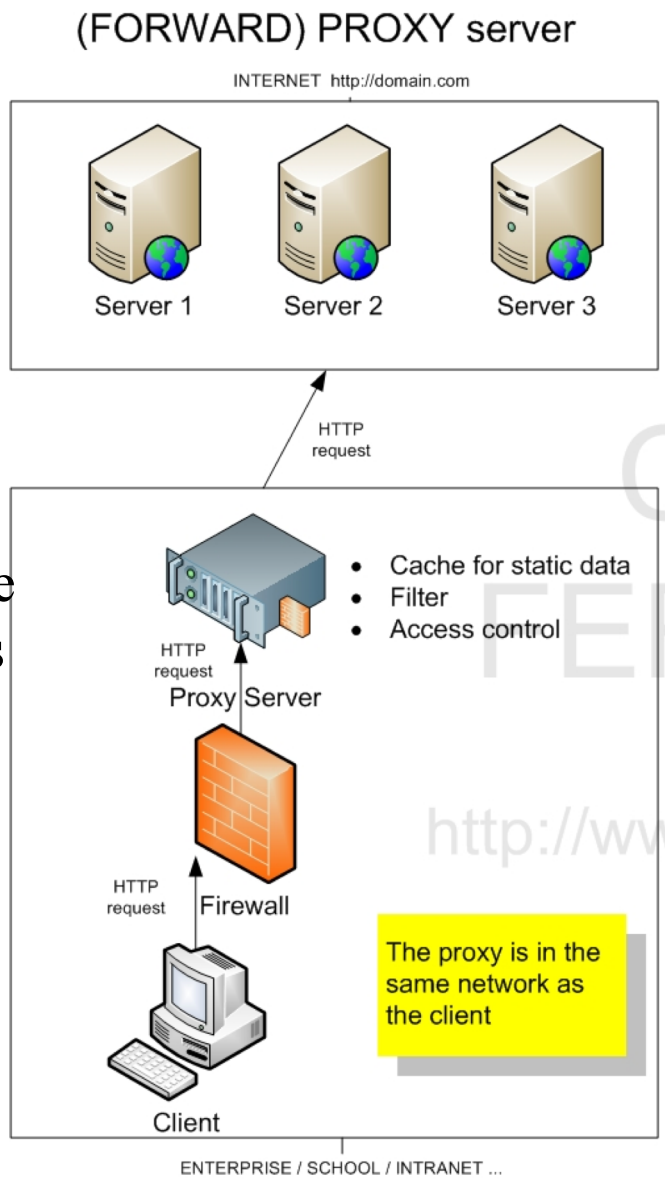
Goal: satisfy client request without involving origin server

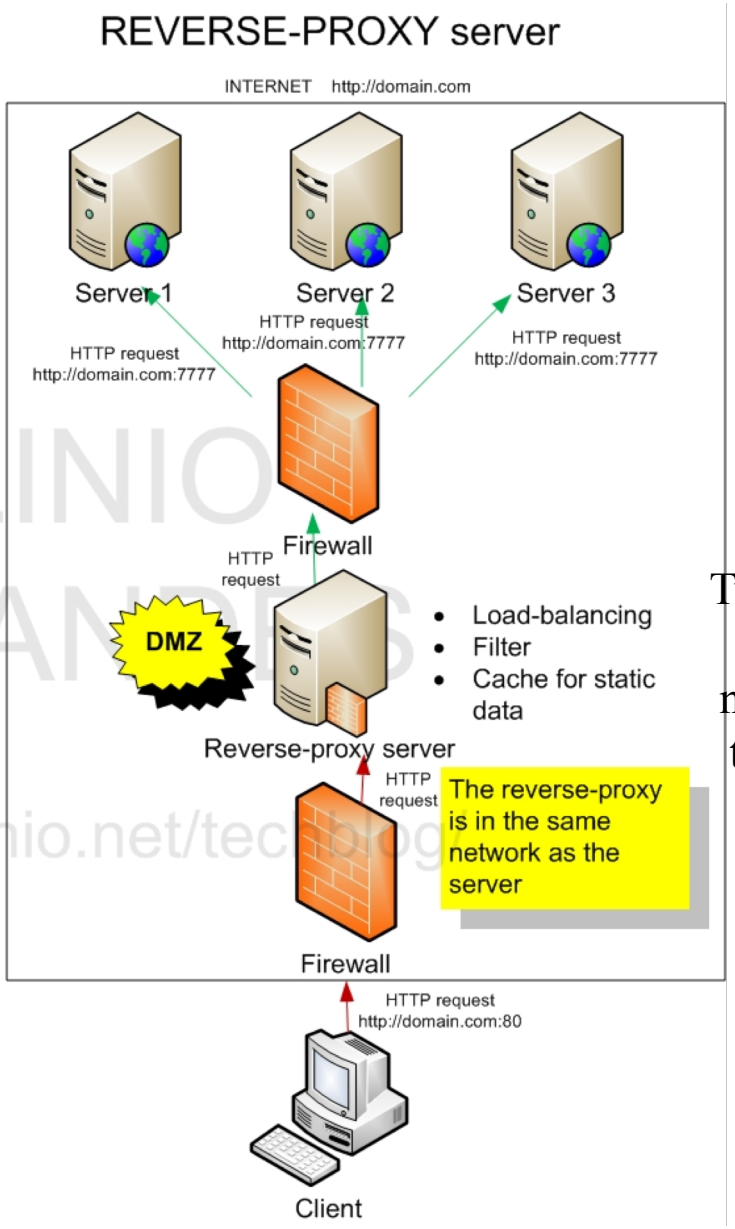


http://www.celinio.net/techblog/?p=1027

Two Types of Proxies

Typically in the same network as the client



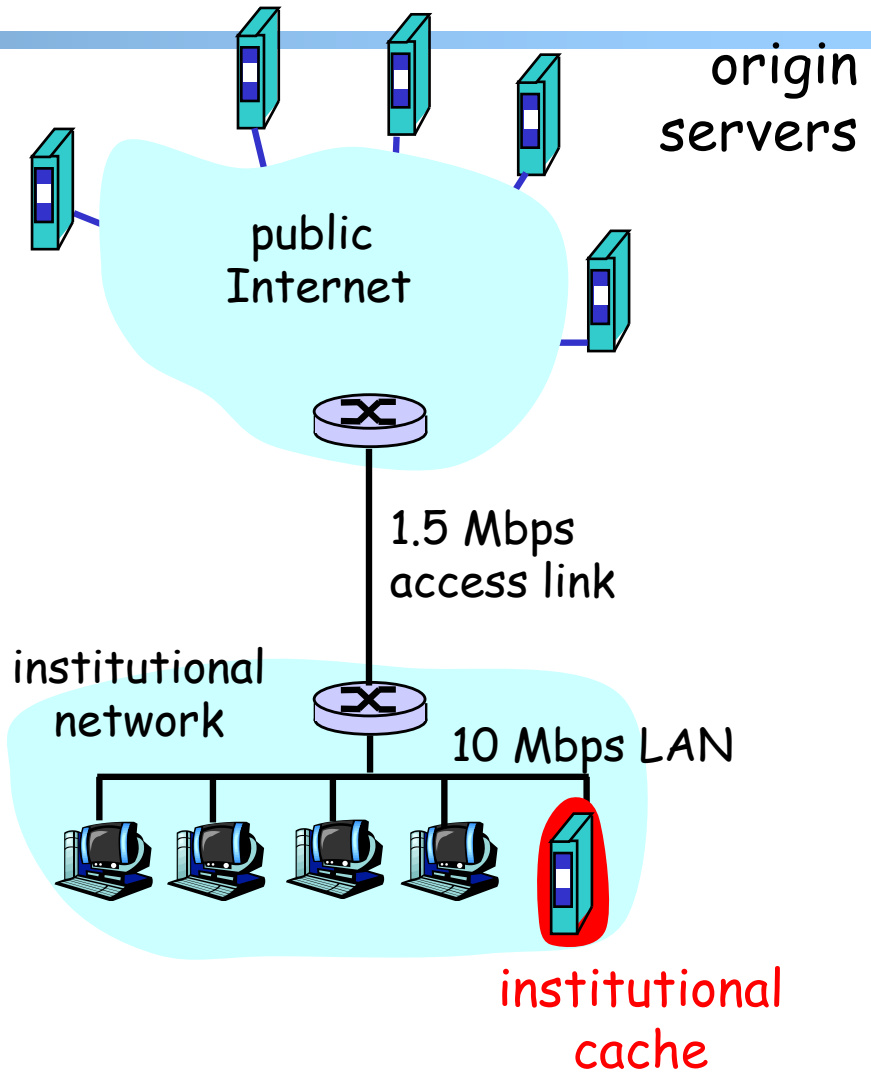


Typically in the same network as the server

Benefits of Forward Proxy

Assume: cache is "close" to client (e.g., in same network)

- smaller response time: cache "closer" to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often is bottleneck



No Free Lunch: Problems of Web Caching

- The major issue of web caching is how to maintain consistency
- Two ways
  - pull
    - Web caches periodically pull the web server to see if a document is modified
  - push
    - whenever a server gives a copy of a web page to a web cache, they sign a lease with an expiration time; if the web page is modified before the lease, the server notifies the cache