Application Layer: HTTP and the Web

Srinidhi Varadarajan

The Web: the http protocol

http: hypertext transfer protocol
- Web’s application layer protocol
- client/server model
  - client: browser that requests, receives, “displays” Web objects
  - server: Web server sends objects in response to requests
- http1.0: RFC 1945
- http1.1: RFC 2068

The http protocol

http: TCP transport service:
- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- http messages (application-layer protocol messages) exchanged between browser (http client) and Web server (http server)
- TCP connection closed

http is “stateless”
- server maintains no information about past client requests

Protocols that maintain “state” are complex!
- past history (state) must be maintained
- if server/client crashes, their views of “state” may be inconsistent, must be reconciled

http example

Suppose user enters URL www.someSchool.edu/someDepartment/home.index (contains text, references to 10 jpeg images)

1. http client initiates TCP connection to http server (process) at www.someSchool.edu. Port 80 is default for http server.
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 (someDepartment/home.index), sends message into socket
6. Steps 1-5 repeated for each of 10 jpeg objects

Non-persistent and persistent connections

Non-persistent
- HTTP/1.0
- server parses request, responds, and closes TCP connection
- 2 RTTs to fetch each object
- Each object transfer suffers from slow start

Persistent
- default for HTTP/1.1
- on same TCP connection: server, parses request, responds, parses new request...
- Client sends requests for all referenced objects as soon as it receives base HTML.
- Fewer RTTs and less slow start.

Most 1.0 browsers use parallel TCP connections. One for each object
Pipelining HTTP Requests

- Two approaches to sending HTTP requests
- Non-pipelined approach: Send a HTTP request. Wait for it to complete before sending the next request.
  - Latency per object: 1 RTT
- Pipelined approach: Send HTTP requests as soon as you know of them, i.e. parse base html file and send HTTP requests as soon as you encounter a referenced object

http message format: request

- two types of http messages: request, response
- http request message:
  - ASCII (human-readable format)

```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr
```

http response status codes

In first line in 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.eurecom.fr 80
```

2. Type in a GET http request:
   ```
GET /~ross/index.html HTTP/1.0
```

3. Look at response message sent by http server!
User-server interaction: authentication

**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 authentication: header line in response

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

User-server interaction: 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

**Goal:** satisfy client request without involving origin server
- **Solution:** Caching. Store a local copy of the data.
- **Two Types of Caches**
  - Local caches e.g. Netscape cache
    - Can exploit locality of reference from a single user
  - Proxy caches
    - Sit between the user and the server. Usually they are located on the user’s network
    - Can exploit reference locality between multiple users

Web Caches (proxy server)

- **user sets browser:** Web accesses via web cache
- **client sends all http requests to web cache**
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client

Why Web Caching?

**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 bottleneck
Reading

Required Reading
- A Case for Persistent-Connection HTTP, J.C. Mogul, ACM SIGCOMM 1995 pp. 299-313

Recommended Reading