CS144: HTTP

Basic interaction

Example: Q: http://www.youtube.com interaction. What is going on behind the scene?

- Q: What entities are involved in this interaction?

- Q: What is the role of each entity?

- Q: There are many Web servers on the Internet. How can the Web browser reach and communicate with the YouTube server?

- Q: Many things are exchanged over Internet. Email, instant messaging, file transfer, etc. How does the server know that this client wants a “Web page”?

- Q: Only bytes are transferred. How do they communicate pages that contain text and rich, dynamic multimedia content?
Basic Internet Standards

- TCP/IP (transmission control protocol and internet protocol)
  - internet routing and transportation protocol
- DNS (domain name service)
  - internet protocol to map domain names to IPs
  - ICANN manages TLD (top-level domains)
- HTTP (hypertext transportation protocol)
  - communication protocol between web servers and web clients
- MIME (multipurpose internet mail extensions)
  - internet standard to specify the type of data being exchanged
- Encoding
  - standard to represent different types of data as a sequence of bytes
  - Text: ASCII, Unicode
  - Multimedia: JPEG, MP3, H.264, ...
- HTML (hypertext markup language)
  - markup standard for Web pages

HTTP

- HTTP/2 is most recent, but HTTP/1.1 is most popular
- Request & response paradigm
  - all interactions start with a client’s request
  
  
  \[
  \text{client} \rightarrow \text{server}
  \text{< - response -} \]

- Stateless: every request is handled independently from others
  - Q: what are pros/cons of stateless protocol?

- Example:
CS144: Web Applications

- Real request example: http://oak.cs.ucla.edu/classes/cs144/examples/show_request/
- Simple request and response example: telnet to http://oak.cs.ucla.edu/classes/cs144

- HTTP message = request/status line + header + body
- HTTP request
  - the bare minimum HTTP request (can be issued through telnet):

```
GET / HTTP/1.0
```

- More realistic example

```
GET /cs144/examples/form.html HTTP/1.1 /* request line */
Host: oak.cs.ucla.edu /* beginning of header */
User-Agent: Mozilla/5.0 ...
Referer: http://oak.cs.ucla.edu/cs144/
Accept:text/xml,text/html;q=0.9,text/plain;q=0.8,image/png
   ,*/*;q=0.5
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
Cookie: __utmz=125574670.1174236576.14.14...

/* end of header */
```

- request line: the actual request

* METHOD PATH PROTOCOL_VERSION
* method
  ▶ GET: “retrieve” data
    ◦ Most common method
    ◦ IMPORTANT: GET should not leave NO significant side effect at server
  ▶ POST: “post” data at the specified URL
    ◦ May leave a side-effect on the server
  ▶ PUT: “place” the data at the URL (~ replace the data)
DELETE: “delete” the data at the URL

less common:
- HEAD: the same but the header only
- OPTIONS: requests for information on available options at the server
- TRACE: the final recipient returns the whole request message in the response body
- Q: When will TRACE be useful?

- header: additional information for the request
  - Host: the name of the web server
    - Q: why do we need the “Host:” field? Aren’t we already contacting it?

- User-Agent: information on the client software
- Referer: The page linking to the requested page.
  - Q: where can Referer be used?

- Accept . . . : media/content type the client can accept q=... specifies the degree of preference of a particular type
- Keep-Alive, Connection: in case we want to make multiple requests through one connection
  - Q: why do we want to make multiple requests per connection?

- Cookie: more on this later

• HTTP response
  - e.g.

```
HTTP/1.1 200 OK  /* status line */
```
Status line:

* 2xx: Success - The action was successfully received, understood, and accepted
* 3xx: Redirection - Further action must be taken in order to complete the request
* 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
* 5xx: Server Error - The server failed to fulfill an apparently valid request

- Vary: Do not use cached content if the specified HTTP header(s) are different
- Content-Length: length of the body
- Content-Type: the type of the content html, flash, pdf, etc.

- Looking at request and response using Chrome Developer console
- HTTP/2
  - Current standard (approved on Feb 17, 2015), but not yet widely deployed
  - Design rationale
    * Many objects need to be fetched to display a single page
      * ~ 100 objects, ~ 2MB
    * Web is often accessed through high-latency mobile connections
  - Makes it possible to
    * Send multiple objects through a single TCP connection
    * Reduces latency and overload
HTTP2
- Uses binary format (not text)
- Works with TLS (encryption) in most implementations
- "Multiplexed streams" with priority specification
  - requests/responses are split into small frames
  - able to specify which stream to prioritize if resource constrained
- Enables HTTP header compression
- Enables "server push" (allows predictive cache "push" by server)
- But its wide-scale adoption is still uncertain
- More detail at https://daniel.haxx.se/http2/

References
- HTTP/1.1: RFC 7230 – RFC 7237
- HTTP/2: RFC 7540