2- Application Level Protocols
HTTP 0.9/1.0/1.1/2

Part A
OVERVIEW
HTTP (HyperText Transfer Protocol)

- Basis for fetching Web pages
- HTTP is the page fetch protocol
  - HTTP is the page contents
Web Protocol Context

• HTTP is a request/response protocol for fetching Web resources
• Runs over TCP
  – typically port 80
    • A more secure version, HTTPS typically on port 443
  – Part of browser/server app
HTML Web Page Content

Page composed of multiple elements fetched as HTTP transfers

HTTP request

HTTP response

Web browser

Web server

youtube.com

www.cs.washington.edu

google-analytics.com
HTTP Protocol

• Today, HTTP transports **typed data**
  – TCP transports bytes

• Like DNS, HTTP is a **request-response protocol**
  – Client sends request message, server sends response message

• HTTP messages have a **header and a payload section**

• Header is **encoded as text**
  – payload is typed: e.g., text vs. images

• Header is **extensible**
  – Can add to it without breaking backward compatibility
Static vs Dynamic Web pages

- The URL is a logical name
  - Doesn’t have to correspond to a file
- Static web page is a file contents, e.g., image
- Dynamic web page is the result of program execution
  - E.g., Javascript on client, PHP on server, or both
Fetching a Web Page

• Browser starts with the page URL:
  http://en.wikipedia.org/wiki/Vegemite

  Protocol  Server  Page on server

• Browser steps:
  – Resolve the server name to an IP address (DNS)
  – Set up a TCP connection to the server (port 80)
  – Send HTTP request for the page
  – Wait for and then read HTTP response
  – (Assuming no errors) Process response data (HTML) and render page
  – Clean up any idle TCP connections
HTTP 0.9

• Intended as allow anonymous fetch of files from remote machines
  – think scp, but you don’t need an account on the remote machine
• Original idea was that anyone could make content available to everyone
• Invented by physicists...
• Request-response protocol
  – Request: “GET /path/to/file\r\n”
  – Response: the file’s contents
HTTP 0.9 Protocol Decisions: Framing

• Q: How does server know when new request starts?
  • when the request ends?
    – How does client know when the response ends?

• A: Transmit HTTP over TCP and misuse TCP events
  – TCP is connection-based
  – There is a “connected” event that occurs when the connection is first established
  – There is a “connection gone” event that occurs when the connection is closed
  – Use connected to tell the server there is a new request
  – Use gone to tell the client that’s the end of the server’s response

• Why is this a bad idea?
HTTP 0.9 Protocol Decisions: Errors

• Q: How does server indicate an error to the client?

• A: The server has only two options
  – Send data back, in which case that data is taken to be the file’s contents
  – Send nothing back

• Getting nothing back could mean:
  – Server crashed
  – Server is up but there is no internet connection between you and it
  – File is empty
  – The file doesn’t exist
  – The request line you sent was bad
  – File exists but I’m having a problem reading it
  – File exists but is larger than the local policy limiting transfer sizes
  – ...

HTTP 0.9: Issues

• The original protocol was very simple, but...

• It quickly had function issues
  – the peer protocol implementations need to be able to talk with each other (independently of the apps using the protocol)
    • e.g., there might be multiple ways to encode data

• The intended application quickly matured
  – Pages became assemblies of components, not just single files
    • An ever growing number of HTTP transfers per page

• The protocol quickly developed performance issues
  – establishing a TCP connection is slow
  – establishing a lot of TCP connections is even slower
HTTP 0.9/1.0/1.1/2.0

- HTTP 0.9: 1991
  - Function
- HTTP 1.0: 1995
  - Function
- HTTP 1.1: 1996
  - Performance
- HTTP 2.0: 2015
  - Performance
FUNCTION: HTTP 1.0 AND BEYOND
You Need a Header

• HTTP 0.9 requests were just a command line
• Responses were just data
• There is no way for the two sides of the communication to talk to each other
• There is no way to indicate an error condition
• Every protocol needs a header
HTTP Message Format

- Header is encoded as text
- Header is a sequence of lines
- Each line ends with \r\n
- Header ends with \r\n\n
- Payload length is given by either:
  - Content-length tag in header
  - Payload is encoded in a format that uses a sentinel (special value that marks the end)
Try It Yourself: View HTTP Request

• $ nc -l 8080
  Opens a TCP socket on port 8080 and waits for an incoming connection

• Point a browser running on the same machine to http://localhost:8080/first/second/third.html

• The output in the nc window is the HTTP request sent by the browser
Example HTTP Request

$ nc -l 8080
GET /first/second/third.html HTTP/1.1
Host: localhost:8080
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
DNT: 1
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.8

What the browser sent
Try It Yourself: HTTP Response

$ nc neverssl.com 80
GET / HTTP/1.0
Host: neverssl.com

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 2536
Connection: keep-alive
Date: Sun, 07 Apr 2019 21:17:55 GMT
Last-Modified: Thu, 14 Jun 2018 00:16:40 GMT
ETag: "e8bb9152091d61caa9d69fed8c4aebc6"
Accept-Ranges: bytes
Server: AmazonS3
Vary: Accept-Encoding
Age: 65787
X-Cache: Hit from cloudfront
Via: 1.1 08f323eee70dda7af34d5feb414ce27.cloudfront.net (CloudFront)
X-Amz-Cf-Id: HQC3zTXMz6vVxWhb9SHJ7M7tHFXQzzwz4OrSYRKKSF7c0G8Y0m2u_e_w==

<html>
<head>
<title>NeverSSL - helping you get online</title>
.... <2536 bytes of data in all>
HTTP Protocol

Commands used in the request

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Read a Web page</td>
</tr>
<tr>
<td>HEAD</td>
<td>Read a Web page's header</td>
</tr>
<tr>
<td>POST</td>
<td>Append to a Web page</td>
</tr>
<tr>
<td>PUT</td>
<td>Store a Web page</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove the Web page</td>
</tr>
<tr>
<td>TRACE</td>
<td>Echo the incoming request</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Connect through a proxy</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Query options for a page</td>
</tr>
</tbody>
</table>

Fetch page → Upload data

Basically defunct
## HTTP Protocol

Result codes returned with the response

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Information</td>
<td>100 = server agrees to handle client's request</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>200 = request succeeded; 204 = no content present</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>301 = page moved; 304 = cached page still valid</td>
</tr>
<tr>
<td>4xx</td>
<td>Client error</td>
<td>403 = forbidden page; 404 = page not found</td>
</tr>
<tr>
<td>5xx</td>
<td>Server error</td>
<td>500 = internal server error; 503 = try again later</td>
</tr>
</tbody>
</table>
PERFORMANCE
Performance Measure: PLT (Page Load Time)

• PLT is the key measure of web performance
  – From click until user sees page

• PLT depends on many factors
  – Structure of page/content
  – HTTP (and TCP!) protocol
  – Network RTT and bandwidth
You Now Have a Shorter Attention Span Than a Goldfish

Kevin McSpadden
May 13, 2015

For more, visit TIME Health.

The average attention span for the notoriously ill-focused goldfish is nine seconds, but according to a new study from Microsoft Corp., people now generally lose concentration after eight seconds, highlighting the affects of an increasingly digitalized lifestyle on the brain.
Page Load Time Impact

From *How One Second Could Cost Amazon $1.6 Billion In Sales*, March 15, 2012
HTTP 1.0 (1996)

- HTTP/1.0 uses one TCP connection to fetch one web resource
  - Made HTTP very easy to build
  - But gave fairly poor PLT ...

- Framing?
  - Length?
  - Sentinel?
HTTP 1.0

- Many reasons why PLT is larger than necessary
  - Sequential request/responses, even when to different servers
    - This is a browser implementation issue, rather than a protocol issue
  - Multiple TCP connection setups to the same server
Reducing PLT: Parallel Connections

• One simple way to reduce PLT
  – Browser runs multiple (8, say) HTTP instances in parallel
  – Server is unchanged; already handled concurrent requests for many clients

• How does this help?
  – Single HTTP wasn’t using network much ...
  – So parallel connections aren’t slowed much
  – Reduces delay to completion of last fetch
HTTP 1.1: PERSISTENT CONNECTIONS
Persistent Connections

• May fetch many resources from one server
  – Fetching in parallel requires opening a new TCP connection for each
  – Opening a TCP connection involves messages back and forth that result in more than a round-trip delay

• Alternative: persistent connections
  – Make a TCP connection to a server
  – Don’t close it when you get the response
  – Instead, use it to send the next request to that server
  – Lower overhead is better for both clients and servers
Connection Use Strategies

One request per connection

Sequential requests per connection

Pipelined requests per connection
Persistent Connections: Framing

• Before persistent connections, a request started with a connection and ended when the connection was closed
  – That’s *framing*

• With persistent connections, how are requests and responses framed?
  – Enforce use of content-length header field?
    • What if content is dynamically generated?
  – If not that, then what?
Can We Further Reduce PLT?

- One request per connection
- Sequential requests per connection
- Pipelined requests per connection
Persistent Connections: Pipelining

- If we pipeline HTTP requests on a single TCP connection, how do we match responses to requests?
- Requests don’t have names
- Can we use ordering
  - I.e., responses must be provided in same order requests were received
- In theory we could
  - In practice, this is a substantial performance hit to the server
- In summary, while some pipelining could in theory be done, it wasn’t