Computer Networks: The Web

Ankit Singla

Some slides adapted from Jennifer Rexford, Scott Shenker, Laurent Vanbever
Protocol

Performance

http://

SPEED LIMIT 20
Like request headers, response headers are of variable lengths and human-readable

<table>
<thead>
<tr>
<th>Uses</th>
<th>Location (for redirection)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Allow (list of methods supported)</td>
</tr>
<tr>
<td></td>
<td>Content encoding (e.g., gzip)</td>
</tr>
<tr>
<td></td>
<td>Content-Length</td>
</tr>
<tr>
<td></td>
<td>Content-Type</td>
</tr>
<tr>
<td></td>
<td>Expires (caching)</td>
</tr>
<tr>
<td></td>
<td>Last-Modified (caching)</td>
</tr>
</tbody>
</table>
HTTP is a stateless protocol, meaning each request is treated independently.

<table>
<thead>
<tr>
<th>advantages</th>
<th>disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>server-side scalability</td>
<td>some applications need state!</td>
</tr>
<tr>
<td></td>
<td>(shopping cart, user profiles, tracking)</td>
</tr>
<tr>
<td>failure handling is trivial</td>
<td></td>
</tr>
</tbody>
</table>

How can you maintain state in a stateless protocol?
HTTP makes the client maintain the state. This is what the so-called **cookies** are for!

- Client stores small state on behalf of the server $X$
- Client sends state in all future requests to $X$
- Can provide authentication
telnet google.ch 80

request
GET / HTTP/1.1
Host: www.google.ch

answer
HTTP/1.1 200 OK
Date: Sun, 01 May 2016 14:10:30 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Server: gws

Set-Cookie:
NID=79=g6lgURTq_BG4hSTFhEyIgTVFmSncQVsyTJl260B3xyiXqy2wxD2YeHq1bBlwFyLoJhSc7jmcA6TIFIBY7-dW5lhjiRiQmY1JxT8hGCOtnLjfCL0mYcBBkpk8X4NwAO28; expires=Mon, 31-Oct-2016 14:10:30 GMT; path=/;
domain=.google.ch; HttpOnly
Web pages are far from simple!
Dependencies in a simple page

```html
<html>
  <body onload="done();">
    <link src='1.css'/>
    <script src='d3.js'></script>
    <script src='2.js'></script>
    <div id="main"></div>
  </body>
</html>

[Example adapted from: Speeding up Web Page Loads with Shandian, Wang et al., NSDI 2016]
Dependencies in a simple page

```html
<html>
  <body onload="done();">
    <link src='1.css'/>
    <script src='d3.js'></script>
    <script src='2.js'></script>
    <div id="main"></div>
  </body>
</html>
```
Dependencies in a simple page

<html>
<body onload="done();">
<link src='1.css'/>
<script src='d3.js'></script>
<script src='2.js'></script>
<div id="main"></div>
</body>
</html>
Dependencies in a simple page

```html
<html>
  <body onload="done();">
    <link src='1.css'/>
    <script src='d3.js'></script>
    <script src='2.js'></script>
    <div id="main"></div>
  </body>
</html>
```

Browser has to be conservative, unless it is clear that two resources are independent.
Real pages can be very complex!

[Polaris: Faster Page Loads Using Fine-grained Dependency Tracking, Netravali et. al, NSDI 2016]
And complexity may be increasing ...

(*) see http://httparchive.org/trends.php

Total Transfer Size & Total Requests

KB

#requests

Time
Today, the average webpage size is 2.3 MB as much as the original DOOM game…

(*) see https://mobiforge.com/research-analysis/the-web-is-doom
And complexity may be increasing …

(*) see http://httparchive.org/trends.php
A closer look at the dependencies
What determines load time?
A time-annotated dependency graph

Note: the length of the bars is no longer meaningful.
A time-annotated dependency graph

Note: the length of the bars is no longer meaningful.
Is this the load time?

This path is not a valid execution path — it misses many dependencies.
Is this the load time?

This is the longest path from start to finish, and computes the time to finish.
We want the longest FIN-Start path

How do we compute the longest path between A and start?
What is the longest A-start distance?

\[ \text{max} \ d_1 + 30, \ d_2 + 60 \]
For each edge $u \rightarrow v$, we need $d_v$ to compute $d_u$.

Topological sort: $(u \rightarrow v) \Rightarrow u$ comes before $v$ in the ordering.
Let’s first label the nodes (arbitrarily)

Topological sort: \((u \rightarrow v) \Rightarrow u \text{ comes before } v \text{ in the ordering}\)
G has no incoming edges except from A

Topological sort: \((u \rightarrow v) \Rightarrow u\) comes before \(v\) in the ordering
Adding nodes this way gets us the ordering

Topological sort: $(u \rightarrow v) \Rightarrow u$ comes before $v$ in the ordering
Adding nodes this way gets us the ordering

Topological sort: \((u \rightarrow v) \Rightarrow u\) comes before \(v\) in the ordering
Adding nodes this way gets us the ordering

Topological sort: \((u \rightarrow v) \Rightarrow u \text{ comes before } v \text{ in the ordering}\)
We can obtain longest paths in order

\[ \begin{align*}
    d_B &= 0 \\
    d_C &= d_B + 200 \\
    d_K &= d_C + 40 \\
    d_I &= d_C + 40 \\
    d_J &= d_I + 30 \\
    d_D &= d_I + 30 \\
    d_H &= d_K + 400 \\
    d_F &= d_J + 30 \\
    d_E &= d_H + 120, d_D + 230 \\
    d_G &= d_F + 220, d_E + 40 \\
    d_A &= d_G + 60, d_J + 30 \\
    d_{FI} &= d_A + 50
\end{align*} \]
“Critical path” analysis

Represent a process as a dependency graph
   Nodes are individual tasks
   Edges indicate a “must happen before” relationship

Annotate edges with “costs”
   Typically, the time it takes for the preceding task

Find the longest path
   First, sort the nodes using a topological sort
   Process tasks in reverse-sort order
   Each task’s finish time is max over tasks it depends on
“Critical path” analysis

Represent a process as a dependency graph

Annotate edges with “costs”

Find the longest path

Q: Given a dependency graph \( G(V, E) \), what’s the complexity of finding the critical path?
Using the critical path
Using the critical path

Is this still the critical path? What's the load time now?
We can obtain longest paths in order

\[
\begin{align*}
\text{B} & \xrightarrow{200} \text{C} \\
\text{C} & \xrightarrow{40} \text{K} \\
\text{K} & \xrightarrow{200} \text{I} \\
\text{I} & \xrightarrow{30} \text{J} \\
\text{J} & \xrightarrow{30} \text{D} \\
\text{D} & \xrightarrow{30} \text{H} \\
\text{H} & \xrightarrow{30} \text{F} \\
\text{F} & \xrightarrow{40} \text{E} \\
\text{E} & \xrightarrow{30} \text{G} \\
\text{G} & \xrightarrow{40} \text{A} \\
\text{A} & \xrightarrow{50} \text{FIN}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Path</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_B</td>
<td>0</td>
</tr>
<tr>
<td>d_C</td>
<td>d_B + 200</td>
</tr>
<tr>
<td>d_K</td>
<td>d_C + 40</td>
</tr>
<tr>
<td>d_I</td>
<td>d_C + 40</td>
</tr>
<tr>
<td>d_J</td>
<td>d_I + 30</td>
</tr>
<tr>
<td>d_D</td>
<td>d_I + 30</td>
</tr>
<tr>
<td>d_H</td>
<td>d_K + 200</td>
</tr>
<tr>
<td>d_F</td>
<td>d_J + 30</td>
</tr>
<tr>
<td>d_E</td>
<td>d_H + 120, d_D + 230</td>
</tr>
<tr>
<td>d_G</td>
<td>d_F + 220, d_E + 40</td>
</tr>
<tr>
<td>d_A</td>
<td>d_G + 60, d_J + 30</td>
</tr>
<tr>
<td>d_FI</td>
<td>d_A + 50</td>
</tr>
</tbody>
</table>

Values:
- d_B = 0
- d_C = d_B + 200
- d_K = d_C + 40
- d_I = d_C + 40
- d_J = d_I + 30
- d_D = d_I + 30
- d_H = d_K + 200
- d_F = d_J + 30
- d_E = d_H + 120, d_D + 230
- d_G = d_F + 220, d_E + 40
- d_A = d_G + 60, d_J + 30
- d_FI = d_A + 50
- d_FIN = 50

Total distance: 710
Using the critical path

How about now? Is this still the critical path?
We can obtain longest paths in order

<table>
<thead>
<tr>
<th>Node</th>
<th>Next Node</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>C</td>
<td>200</td>
</tr>
<tr>
<td>C</td>
<td>K</td>
<td>40</td>
</tr>
<tr>
<td>K</td>
<td>I</td>
<td>30</td>
</tr>
<tr>
<td>I</td>
<td>J</td>
<td>30</td>
</tr>
<tr>
<td>J</td>
<td>D</td>
<td>30</td>
</tr>
<tr>
<td>D</td>
<td>H</td>
<td>30</td>
</tr>
<tr>
<td>H</td>
<td>F</td>
<td>40</td>
</tr>
<tr>
<td>F</td>
<td>E</td>
<td>30</td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>40</td>
</tr>
<tr>
<td>G</td>
<td>A</td>
<td>50</td>
</tr>
<tr>
<td>A</td>
<td>FIN</td>
<td>60</td>
</tr>
</tbody>
</table>

- $d_B = 0$
- $d_C = d_B + 200 = 200$
- $d_K = d_C + 40 = 240$
- $d_I = d_C + 40 = 240$
- $d_J = d_I + 30 = 270$
- $d_D = d_I + 30 = 270$
- $d_H = d_K + 20 = 260$
- $d_F = d_J + 30 = 300$
- $d_E = d_H + 120, d_D + 230 = 500$
- $d_G = d_F + 220, d_E + 40 = 540$
- $d_A = d_G + 60, d_J + 30 = 600$
- $d_{FI} = d_A + 50 = 650$
Using the critical path

The critical path has changed now!
Speeding up any task on the critical path will ...

speed up the end-to-end process;

or / and

expose a different critical path.
Many notions of load time

- Last resource loaded?
- Last visual change?
- Last *visible* change?
- Search text box is loaded?
- First visual change?
Even a single request is non-trivial
Recall our data on latency ...

![Diagram showing DNS, "Hi", and Data transfer]

- **DNS**: 20%
- **"Hi"**: 30%
- **Data transfer**: 50%

**Figure 1:** Distribution of connection minimum RTTs for TCP connections.

- **Figure 2:** The 99th percentile indicates that the 99th percentile rebuffer rate for QUIC is 16.7% lower than the 99th percentile latency for TCP.
- **Figure 3:** QUIC includes richer signaling than TCP, which enables QUIC to play a role in decreasing Search latency at higher RTTs.

**Table 1:** Percent reduction in global Search and Video Latency for users

<table>
<thead>
<tr>
<th>Application</th>
<th>Percent Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>99%</td>
</tr>
<tr>
<td>Video</td>
<td>93%</td>
</tr>
</tbody>
</table>

**Figure 4:** CDF of minimum RTT to Google servers (milliseconds).

- **Desktop Search**: 100.0 70.4 60.0 18.5
- **Mobile Search**: 100.0 52.7 8.7

**Figure 5:** CDF of rebuffer rate reduction by percentile.

- **Desktop Video**: 36% of connections over 100ms
- **Mobile Video**: ~1 second for just one request!
Measuring fetch times for small objects

186 PlanetLab nodes as clients

Fetch (using cURL) thousands of Web pages each

- Only HTML of the landing pages
- Typically tens of KB
Measuring fetch times for small objects

c-latency: speed of light along shortest client-server path on Earth’s surface
Measuring fetch times for small objects

c-latency: speed of light along shortest client-server path on Earth’s surface

![CDF diagram showing inflation over c-latency]

- CDF: Cumulative Distribution Function
- Inflation over c-latency: The graph illustrates how the cumulative distribution function (CDF) changes with inflation over the speed of light (c-latency). The point marked '37x' indicates a significant increase in the CDF, suggesting a notable impact on fetch times for small objects.
How to speed this up?
Many possibilities to speed up Web browsing

- Simplify, restructure, redesign Web pages
- Use faster computing devices
- Increase network bandwidth
- Make network RTTs smaller
- Simplify network protocols
- Caching

Obviously, not all of this is strictly “networking”
Many possibilities to speed up Web browsing

- Simplify, restructure, redesign Web pages
- Use faster computing devices
- Increase network bandwidth
- Make network RTTs smaller
- Simplify network protocols
- Caching
Simplify, restructure, redesign Web pages

apple.com

Compress using gzip and more efficient image codecs like WebP

In-line JSS, CSS

Tag “async” resources (explicitly identifying lack of dependencies)
Top web sites have decreased in size because they care about performance

(*) see https://mobiforge.com/research-analysis/the-web-is-doom
Many possibilities to speed up Web browsing

- **Simplify, restructure, redesign Web pages**
- **Use faster computing devices**
- Increase network bandwidth
- Make network RTTs smaller
- Simplify network protocols
- Caching
Use faster computing devices

Sunspider is a popular Javascript / Web benchmark

Caveat:

Many possibilities to speed up Web browsing

- Simplify, restructure, redesign Web pages
- Use faster computing devices
- **Increase network bandwidth**
- Make network RTTs smaller
- Simplify network protocols
- Caching
Increase network bandwidth

Significant gains up to a few Mbps

[Study by Mike Belshe, Google. Figure via hpbn.co, Ilya Grigorik / O'Reilly Media, Inc]
Many possibilities to speed up Web browsing

- Simplify, restructure, redesign Web pages
- Use faster computing devices
- Increase network bandwidth
- **Make network RTTs smaller**
- Simplify network protocols
- Caching
Making network RTTs smaller

Large, linear impact

Reducing RTTs is hard, which is why my group is working on it :)
The Internet at the Speed of Light
Many possibilities to speed up Web browsing

- Simplify, restructure, redesign Web pages
- Use faster computing devices
- Increase network bandwidth
- Make network RTTs smaller
- **Simplify network protocols**
- Caching & replication
Relying on TCP forces an HTTP client to open a connection before exchanging anything.
Most Web pages have multiple objects, naive HTTP opens one TCP connection for each…

Fetching $n$ objects requires $\sim 2n$ RTTs

TCP establishment
HTTP request/response
One solution to that problem is to use multiple TCP connections in parallel.

Server-side burden of concurrent connections

Bandwidth contention among connections
Another solution is to use persistent connections across multiple requests, default in HTTP/1.1

Avoid overhead of connection set-up and teardown
clients or servers can tear down the connection

Allow TCP to learn more accurate RTT estimate
and with it, more precise timeout value

Allow TCP congestion window to increase
and therefore to leverage higher bandwidth
Yet another solution is to pipeline requests & replies asynchronously, on one connection.

- batch requests and responses to reduce the number of packets
- multiple requests can be packed into one TCP segment
Considering the time to retrieve $n$ small objects, pipelining wins

<table>
<thead>
<tr>
<th>Method</th>
<th># RTTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-at-a-time</td>
<td>$\sim 2n$</td>
</tr>
<tr>
<td>$M$ concurrent</td>
<td>$\sim 2n/M$</td>
</tr>
<tr>
<td>persistent</td>
<td>$\sim n+1$</td>
</tr>
<tr>
<td>pipelined</td>
<td>2</td>
</tr>
</tbody>
</table>

Efficient implementation must make sure some requests don’t block others
Google’s ongoing work on QUIC is addressing the handshake
Many possibilities to speed up Web browsing

- Simplify, restructure, redesign Web pages
- Use faster computing devices
- Increase network bandwidth
- Make network RTTs smaller
- Simplify network protocols
- Caching & replication
Caching leverages the fact that highly popular content largely overlaps

Just think of how many times you request the Facebook logo per day vs how often it actually changes

Caching saves time for your browser and decreases network and server load
Caching leverages the fact that highly popular content *largely* overlaps.

Can also cache DNS responses.

* Caching saves time for your browser
* and decreases network and server load
Yet, a significant portion of
the HTTP objects are “uncachable"

| Examples      | dynamic data | stock prices, scores, ...
|               | scripts      | results based on parameters
|               | cookies      | results may be based on passed data
|               | SSL          | cannot cache encrypted data
|               | advertising  | wants to measure # of hits ($$$)
To limit staleness of cached objects, HTTP enables a client to validate cached objects

Server hints when an object expires (kind of TTL) as well as the last modified date of an object

Client conditionally requests a resource using the “if-modified-since” header in the HTTP request

Server compares this against “last modified” time of the resource and returns:

- **Not Modified** if the resource has not changed
- **OK** with the latest version
Caching can and is performed at different locations

<table>
<thead>
<tr>
<th>Location</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>browser cache</td>
</tr>
<tr>
<td>close to the client</td>
<td>forward proxy</td>
</tr>
<tr>
<td></td>
<td>Content Distribution Network (CDN)</td>
</tr>
<tr>
<td>close to the destination</td>
<td>reverse proxy</td>
</tr>
</tbody>
</table>
Many clients request the same information.
This increases servers and network’s load, while clients experience unnecessary delays.
Reverse proxies cache documents close to servers, decreasing their load
Forward proxies cache documents close to clients, decreasing network traffic, server load and latencies.
Many possibilities to speed up Web browsing

- Simplify, restructure, redesign Web pages
- Use faster computing devices
- Increase network bandwidth
- Make network RTTs smaller
- Simplify network protocols

Caching & replication