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

- **Location** (for redirection)
- **Allow** (list of methods supported)
- **Content encoding** (e.g., gzip)
- **Content-Length**
- **Content-Type**
- **Expires** (caching)
- **Last-Modified** (caching)
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>failure handling is trivial</td>
<td>(shopping cart, user profiles, tracking)</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_BG4hSTFhEy1gTVFmSncQVsyTJl260B3xyiXqy2wxD2YeHq1bBlwFyLoJhSc7jmcA6TIFIBY7-dW5ljhiRiQmY1JxT8hGC0tnLjfCL0mYcBBkpk8X4NwAO28; expires=Mon, 31-Oct-2016 14:10:30 GMT; path=/; domain=.google.ch; HttpOnly

**browser will relay this value in following requests**
Protocol

Performance

http://

SPEED LIMIT 80
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>
```
<html>
<body onload="done();">
<link src='1.css'/>
<script src='d3.js'></script>
<script src='2.js'></script>
<div id="main"></div>
</body>
</html>

Time

dependencies

1.css
2.js
d3.js
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!

80.8% of pages have altered critical paths.

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

\[ \text{max } d + 30, \ d + 60 \]
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: $(u \rightarrow v) \Rightarrow u$ comes before $v$ in the ordering
G has no incoming edges except from A

Topological sort: \((u \rightarrow v) \Rightarrow u \text{ comes before } v \text{ in the ordering}\)
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

\[
\begin{align*}
&d_B \quad 0, \\
&d_C \quad d_B + 200, \\
&d_K \quad d_C + 40, \\
&d_I \quad d_C + 40, \\
&d_J \quad d_I + 30, \\
&d_D \quad d_I + 30, \\
&d_H \quad d_K + 400, \\
&d_F \quad d_J + 30, \\
&d_E \quad d_H + 120, d_D + 230, \\
&d_G \quad d_F + 220, d_E + 40, \\
&d_A \quad d_G + 60, d_J + 30, \\
&d_{FI} \quad d_A + 50
\end{align*}
\]
“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
We can obtain longest paths in order

\[
\begin{align*}
    d_B & \quad 0 \\
    d_C & \quad d_B + 200 \\
    d_K & \quad d_C + 40 \\
    d_I & \quad d_C + 40 \\
    d_J & \quad d_I + 30 \\
    d_D & \quad d_I + 30 \\
    d_H & \quad d_K + 200 \\
    d_F & \quad d_I + 30 \\
    d_E & \quad d_H + 120, d_D + 230 \\
    d_G & \quad d_F + 220, d_E + 40 \\
    d_A & \quad d_G + 60, d_J + 30 \\
    d_{FI} & \quad d_A + 50
\end{align*}
\]
Using the critical path

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

\[
\begin{align*}
    d_B & \quad 0 & d_H & \quad d_K + 20 & 260 \\
    d_C & \quad d_B + 200 & 200 & d_F & \quad d_J + 30 & 300 \\
    d_K & \quad d_C + 40 & 240 & d_E & \quad d_H + 120, d_D + 230 & 500 \\
    d_I & \quad d_C + 40 & 240 & d_G & \quad d_F + 220, d_E + 40 & 540 \\
    d_J & \quad d_I + 30 & 270 & d_A & \quad d_G + 60, d_J + 30 & 600 \\
    d_D & \quad d_I + 30 & 270 & d_{FI} & \quad d_A + 50 & 650 
\end{align*}
\]
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

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

DNS  “Hi”  Data transfer

dns  html  "Hi"

Time

html

1.css

d3.js

2.js

Data transfer
we describe QUIC’s impact on Google Search and YouTube. These represent delivery latency for small, delay-sensitive, images and embedded content. On average, an individual search performed by a user results in a total response load of 100 KB delivered to the client by Google Search, including all corresponding dynamically-generated payloads. Search Latency. The percentile data shows that QUIC represents a heavy-load bandwidth-sensitive application. YouTube represents a low-load latency-sensitive application, and YouTube dynamical applications. For these highly optimized applications has direct revenue impact. Second, they represent diverse transport use-cases: Search is discussed in the rest of this section, but we offer two insights about user-relevant metrics are a measure of the usefulness of the change. First, networking remains just one constituent of end-to-end applications. For instance, handshake latency contributes to the impact of networking changes on applications. Of course, since this impact is of- potentionally ~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

![Graph showing CDF of inflation over c-latency]
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

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

Use faster computing devices
Sunspider is a popular Javascript / Web benchmark

Caveat:

They're (Almost) All Dirty: The State of Cheating in Android Benchmarks

by Anand Lal Shimpi & Brian Klug on October 2, 2013 12:30 PM EST
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 :)
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></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 \frac{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 state in a "transport cookie"
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic data</td>
<td>stock prices, scores, ...</td>
</tr>
<tr>
<td>scripts</td>
<td>results based on parameters</td>
</tr>
<tr>
<td>cookies</td>
<td>results may be based on passed data</td>
</tr>
<tr>
<td>SSL</td>
<td>cannot cache encrypted data</td>
</tr>
<tr>
<td>advertising</td>
<td>wants to measure # of hits ($$$)</td>
</tr>
</tbody>
</table>
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

client

browser cache

close to the client

forward proxy

Content Distribution Network (CDN)

close to the destination

reverse proxy
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.

This is typically done by content provider.
Forward proxies cache documents close to clients, decreasing network traffic, server load and latencies.

This is typically done by ISPs or enterprises.
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
The idea behind replication is to duplicate popular content all around the globe

Spreads load on server
e.g., across multiple data-centers

Places content closer to clients
only way to beat the “speed-of-light”

Also helps speed up some uncacheable content

CDN = Content distribution network
I. Spread the content servers globally
2. Network the sites and the origin
3. Direct clients to appropriate servers
1. Spread the content servers globally
2. Network the sites and the origin
How to get your own way on the Internet
Internet routing can be circuitous
Internet routing can be circuitous
Internet routing can be circuitous
Internet routing can be circuitous

<table>
<thead>
<tr>
<th></th>
<th>IP Address</th>
<th>RTT</th>
<th>RTT</th>
<th>RTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140.109.17.1</td>
<td>1.857 ms</td>
<td>1.830 ms</td>
<td>1.819 ms</td>
</tr>
<tr>
<td>2</td>
<td>140.109.254.65</td>
<td>1.024 ms</td>
<td>0.964 ms</td>
<td>0.965 ms</td>
</tr>
<tr>
<td>3</td>
<td>140.109.254.29</td>
<td>1.702 ms</td>
<td>1.851 ms</td>
<td>2.036 ms</td>
</tr>
<tr>
<td>4</td>
<td>140.109.254.5</td>
<td>0.862 ms</td>
<td>0.895 ms</td>
<td>0.942 ms</td>
</tr>
<tr>
<td>5</td>
<td>202.169.174.1</td>
<td>1.320 ms</td>
<td>1.269 ms</td>
<td>1.433 ms</td>
</tr>
<tr>
<td>6</td>
<td>202.169.174.226</td>
<td>126.510 ms</td>
<td>126.498 ms</td>
<td>126.936 ms</td>
</tr>
<tr>
<td>7</td>
<td>4.59.4.1</td>
<td>276.592 ms</td>
<td>276.389 ms</td>
<td>276.527 ms</td>
</tr>
<tr>
<td>8</td>
<td>4.69.152.145</td>
<td>127.712 ms</td>
<td>127.741 ms</td>
<td>127.682 ms</td>
</tr>
<tr>
<td>10</td>
<td>202.97.50.69</td>
<td>131.517 ms</td>
<td>131.466 ms</td>
<td>131.446 ms</td>
</tr>
<tr>
<td>11</td>
<td>202.97.50.117</td>
<td>305.707 ms</td>
<td>305.464 ms</td>
<td>305.652 ms</td>
</tr>
<tr>
<td>12</td>
<td>202.97.34.49</td>
<td>270.524 ms</td>
<td>270.410 ms</td>
<td>270.370 ms</td>
</tr>
<tr>
<td>14</td>
<td>***</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>218.86.44.170</td>
<td>316.692 ms</td>
<td>316.784 ms</td>
<td>316.585 ms</td>
</tr>
<tr>
<td>16</td>
<td>218.61.10.242</td>
<td>291.273 ms</td>
<td>218.61.10.182</td>
<td>290.361 ms</td>
</tr>
<tr>
<td>17</td>
<td>125.78.249.22</td>
<td>346.717 ms</td>
<td>218.61.10.138</td>
<td>296.387 ms</td>
</tr>
<tr>
<td>18</td>
<td>125.78.249.22</td>
<td>335.653 ms</td>
<td>218.61.23.37</td>
<td>290.276 ms</td>
</tr>
</tbody>
</table>
Internet routing can be circuitous.

Taipei, Taiwan

San Jose, USA

Putian, China
Internet routing can be circuitous

\[ l_{ab} > l_{ax} + l_{xb} \]
Internet routing can be circuitous

This is not free: you’re now paying twice!
How well does overlay routing work?

Old study from 1999

“The end-to-end effects of Internet path selection”

4 university-based measurement nodes

Evaluate direct Internet path to several targets

Evaluate indirect path through an overlay node

$$\text{RTT}_{\text{Internet}} - \text{RTT}_{\text{overlay}} \ (\text{ms}) > 0 \Rightarrow \text{overlay wins}$$
Overlay routing: lower latency, loss

The End-to-End Effects of Internet Path Selection

Stefan Savage, Andy Collins, Eric Hoffman
John Snell, and Thomas Anderson

ACM SIGCOMM, 1999

The CDF of the difference between the mean round-trip time for an alternate path.

Figure 1: CDF of the difference between the mean round-trip time for an alternate path.

Figure 2: CDF of the relative round-trip latency.
Monitor latency, loss on routes!
Tweaking transport for speed
Even bigger advantage for SSL!
3. Direct clients to appropriate servers
The problem of CDNs is to direct and serve your requests from a close, non-overloaded replica.

- **DNS-based**
  - Returns ≠ IP addresses based on:
    - Client geo-localization
    - Server load

- **BGP Anycast**
  - Advertise the same IP prefix from different locations
  - Less flexibility, control
The problem of CDNs is to direct and serve your requests from a close, non-overloaded replica.

- **DNS-based**
  - returns ≠ IP addresses based on
    - client geo-localization
    - server load

- **BGP Anycast**
  - advertise the same IP prefix from different locations
  - less flexibility, control
URL rewriting

<html>
  
  <img src="http://www.abc.com/bigpicture.jpg">

  cdnurl.abc.com

</html>

Manipulate DNS for the rest
DNS customized by location

What if the client isn’t using the local DNS resolver?

DNS response depends on Local DNS resolver’s guessed location
Let’s see this in action!
The problem of CDNs is to direct and serve your requests from a close, non-overloaded replica.

**DNS-based**
- returns ≠ IP addresses based on
  - client geo-localization
  - server load

**BGP Anycast**
- advertise the same IP prefix from different locations
- less flexibility, control
What black magic is this?

Why Google public DNS(8.8.8.8)'s ping latency so low?

I found that DNS's latency is low around the world. Many cities are far from each other, but they got the same low latency in ping (about 5ms).

Adapted from http://stackoverflow.com/
“Sixty-two percent of all Internet traffic will cross CDNs by 2019 globally, up from 39 percent in 2014.”

— Cisco
Akamai is one of the largest CDNs in the world, boasting 240,000+ servers in more than 1600 networks

http://wwwnui.akamai.com/gnet/globe/index.html
Akamai uses a combination of

- *pull* caching
  - direct result of clients requests

- *push* replication
  - when expecting high access rate

*together with some dynamic processing*

dynamic Web pages, transcoding,…