Application Layer: SPDY, HTTP/2

CS204: Advanced Computer Networking

Oct 9, 2023

Adapted from Jiasi's CS 204 slides for Spring 23

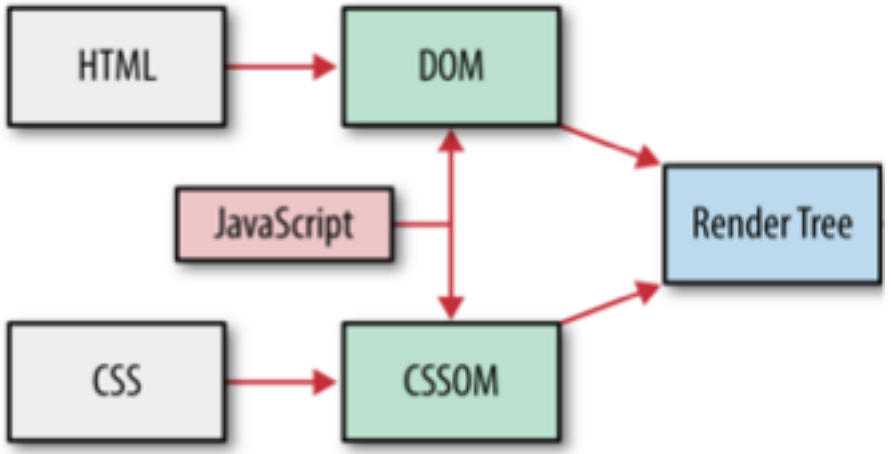
Overview

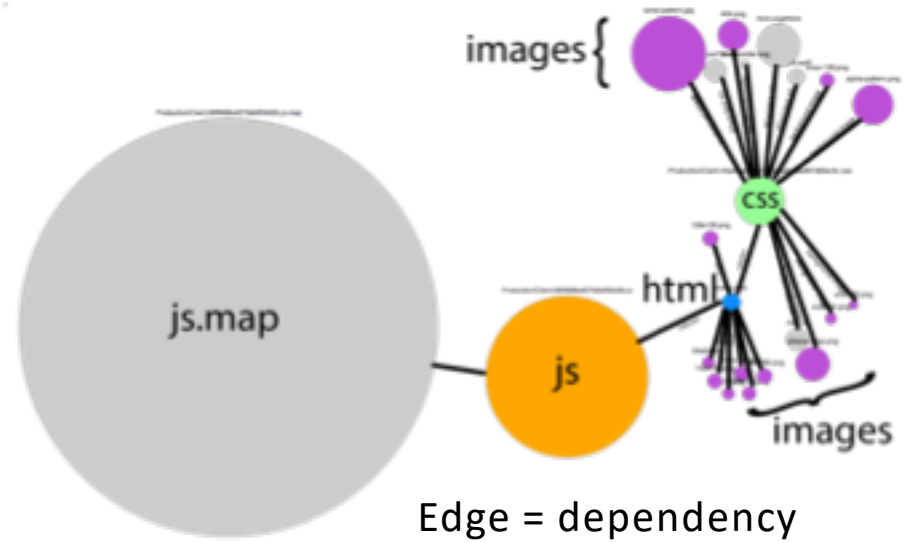
- Measuring webpage latency
- Strategies to reduce latency
- SPDY and HTTP/2
- SPDY over cellular networks

Q: How to reduce page load times for webpages?

What's a Webpage?

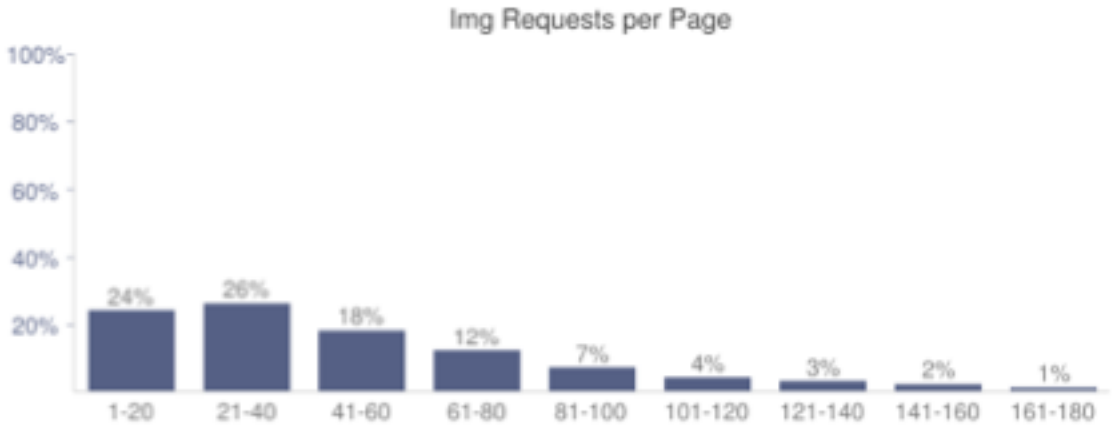
- Hyper-text document (HTTP/0.9)
  - Plain text
  - Metric: document load time
- Website (HTTP/1.0, HTTP/1.1)
  - Images, audio
  - Non-interactive
  - Metric: page load time (PLT)
- Web "application" (HTTP/1.1, HTTP/2)
  - Javascript, stylesheets
  - Dependencies between objects on page

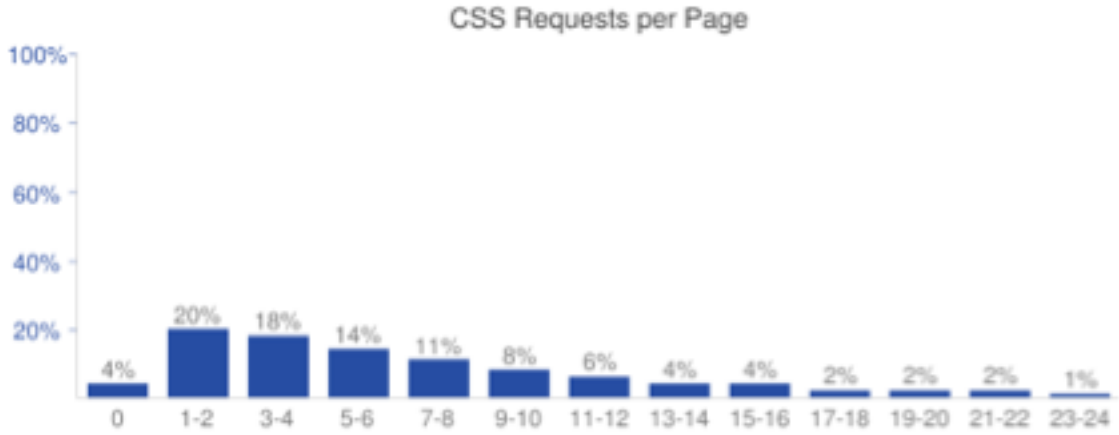


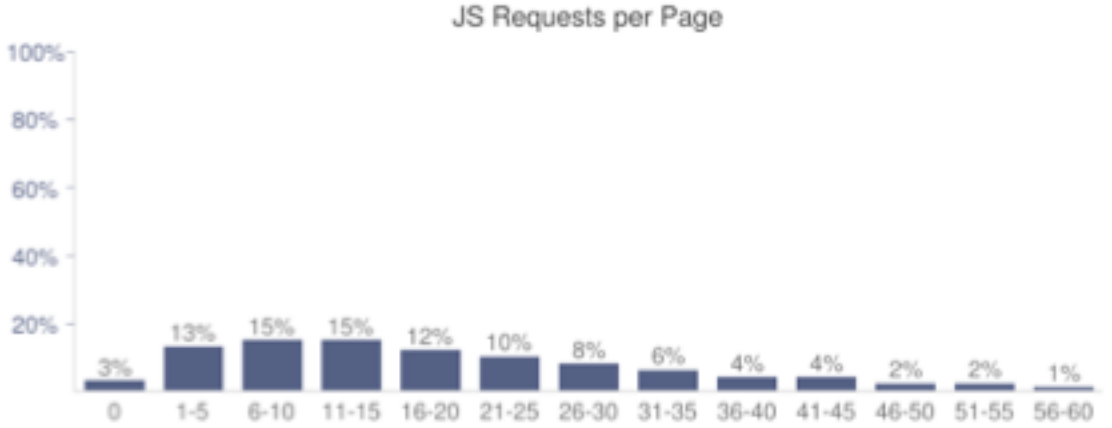


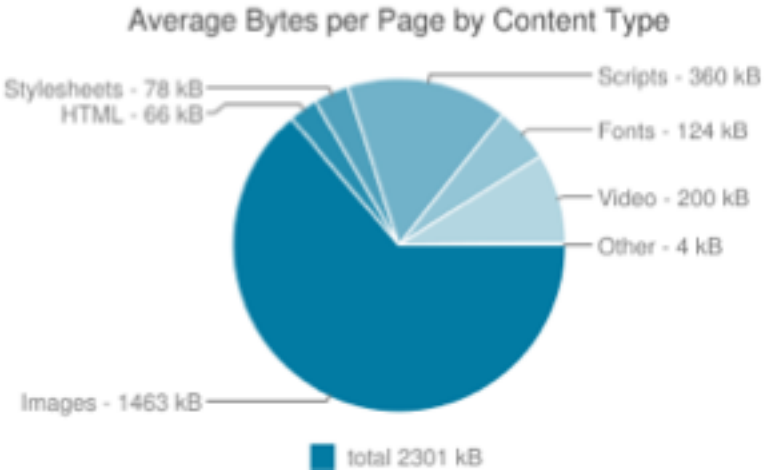
Edge = dependency
Node size = file size

What's in a Webpage?



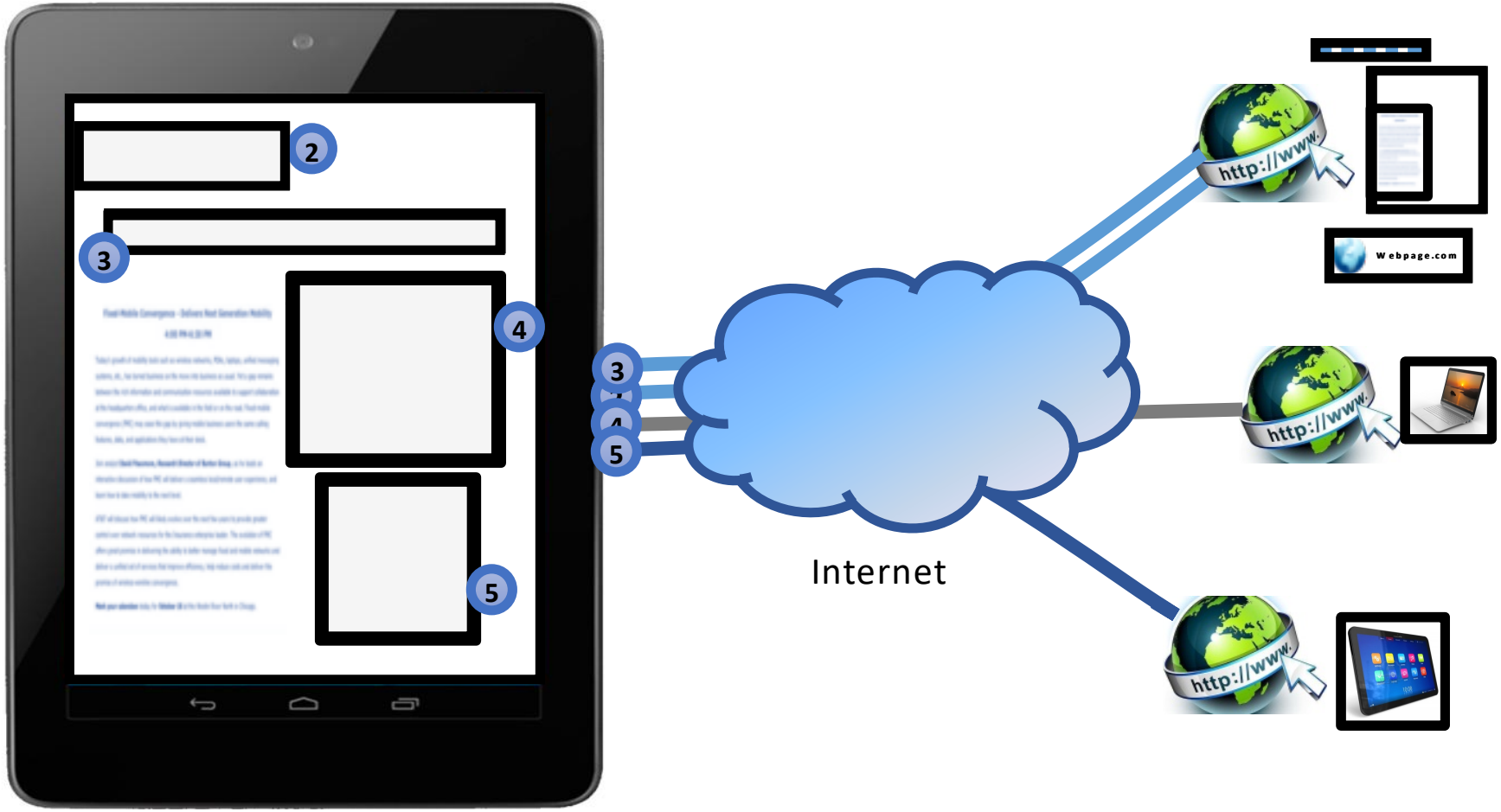






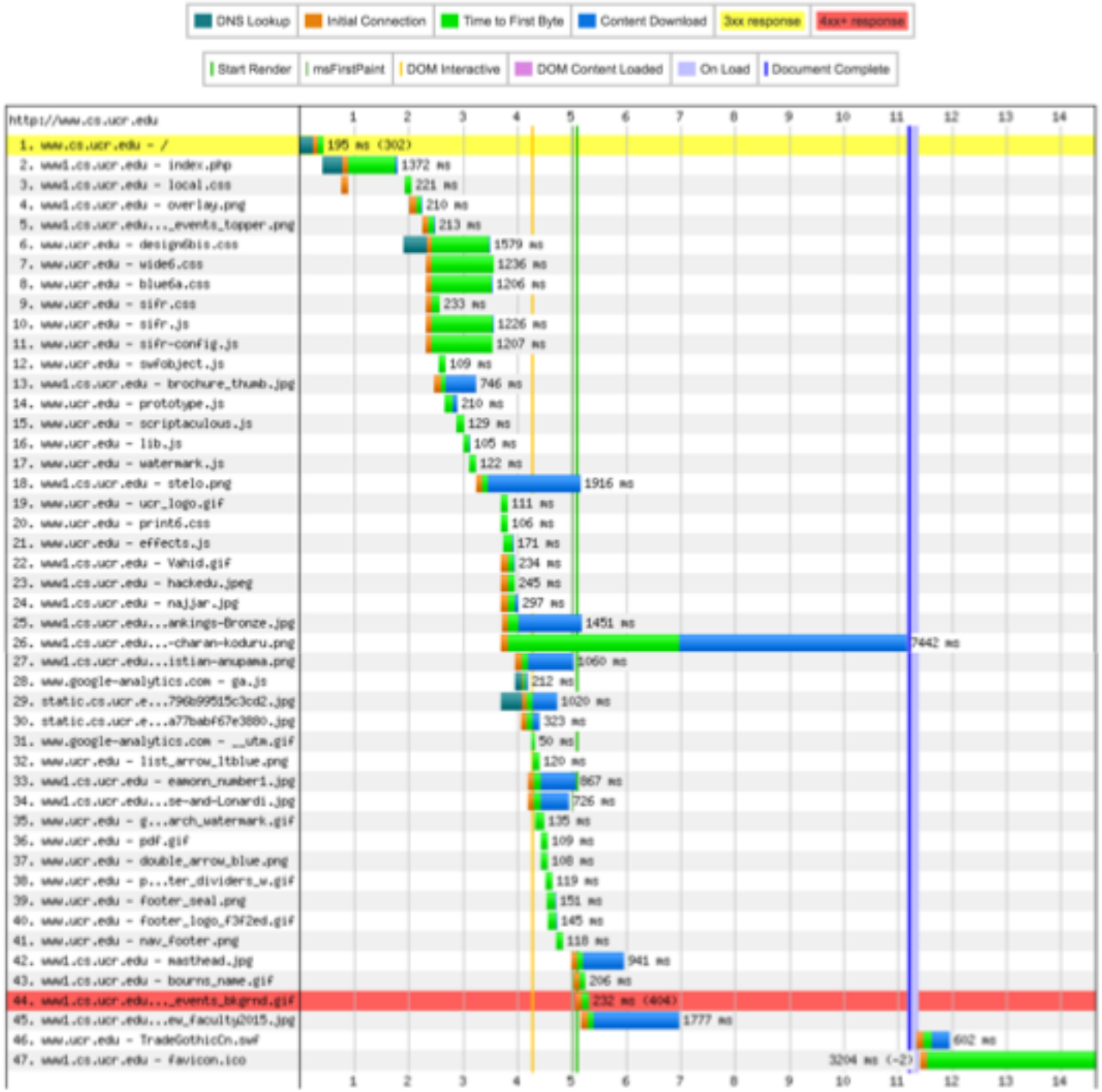
Source: httparchive.org

Web Page delivery using HTTP



Resource Waterfall

- Analyzed UCR CS website
  - www.webpagetest.org
- Different metrics
  - Start render
  - Document complete

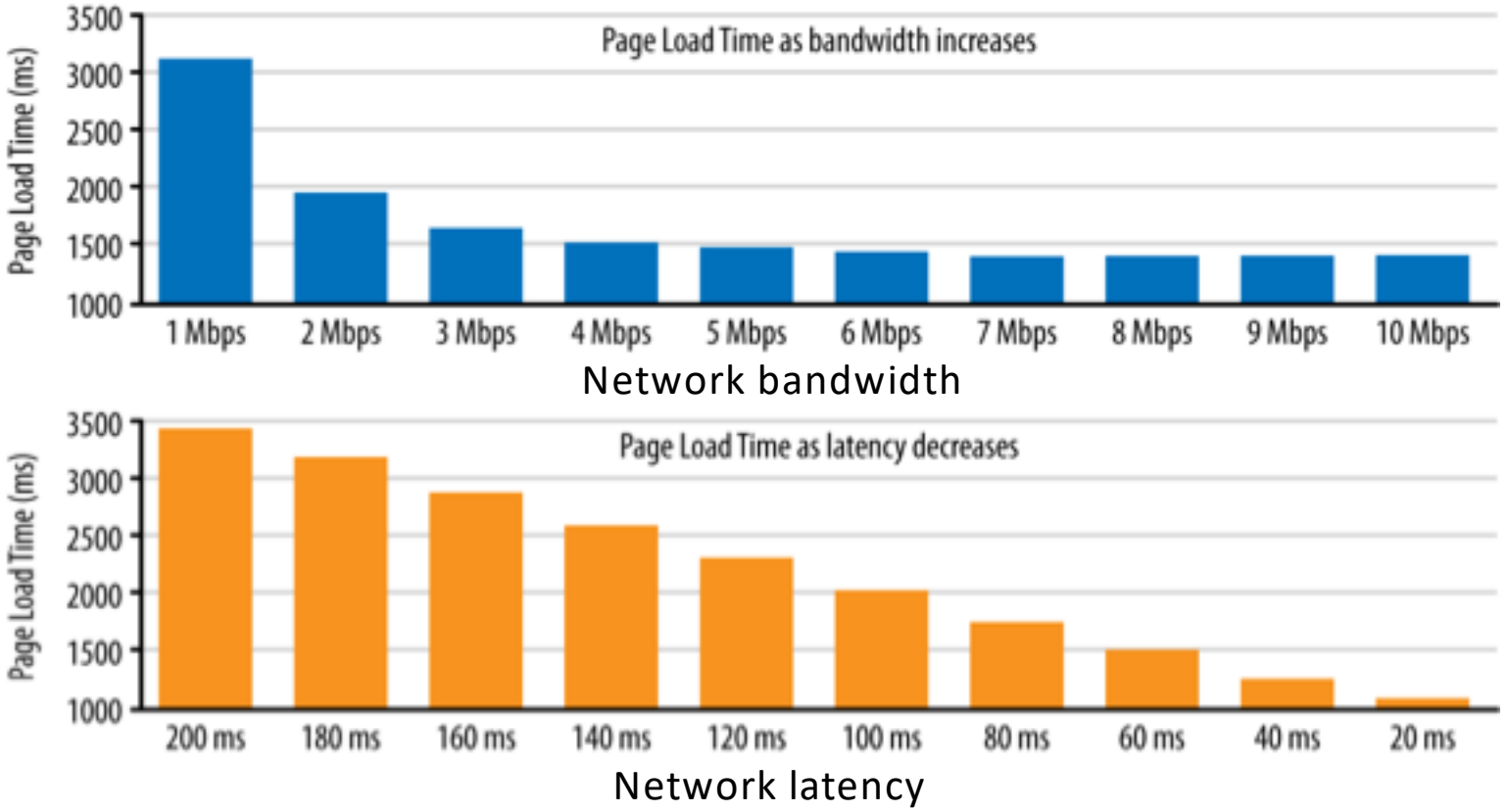


Page Load Time (PLT)

- Matters to companies
  - 2 s delay on Bing decreases per-user revenue by 4.3%

Delay	User perception
0–100 ms	Instant
100–300 ms	Small perceptible delay
300–1000 ms	Machine is working
1,000+ ms	Likely mental context switch
10,000+ ms	Task is abandoned

Latency, not bandwidth



You can buy higher bandwidth… can you buy lower latency?

Source: Mike Belshe

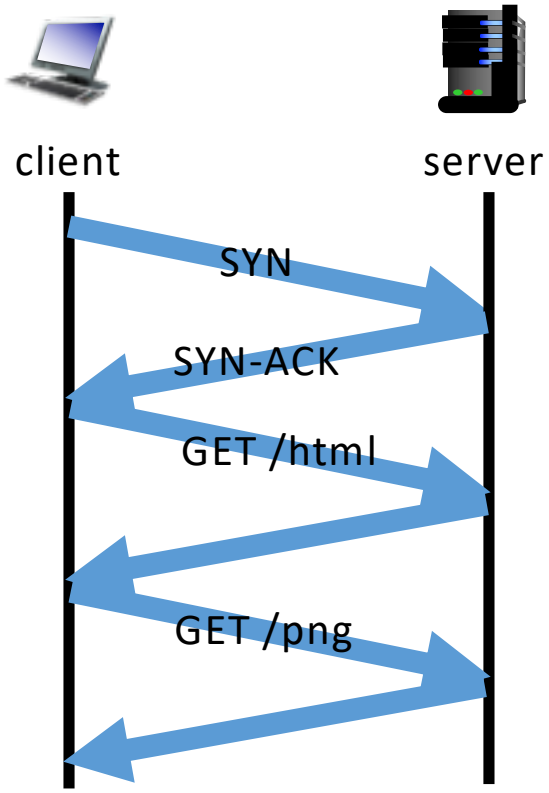
Overview

- Measuring webpage latency
- Strategies to reduce latency
- SPDY and HTTP/2
- SPDY over cellular networks

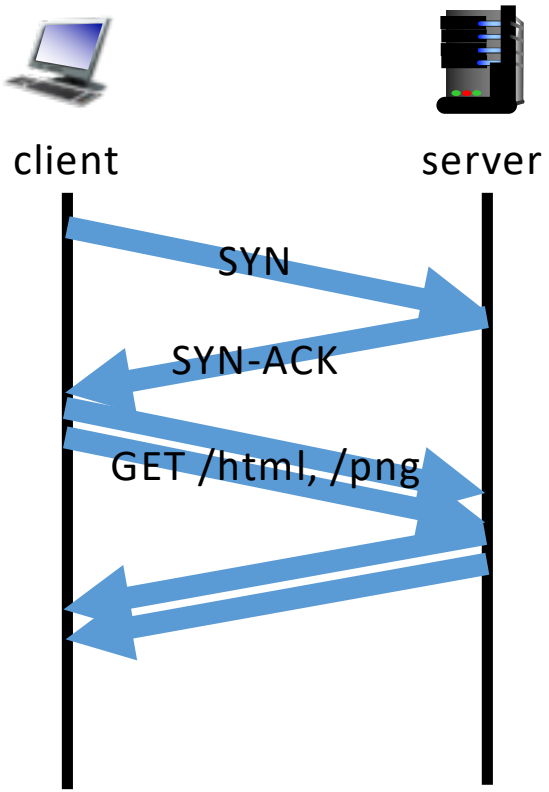
Q: How to reduce page load times for webpages?

Reducing latency: HTTP-based strategies

Persistent connections



HTTP pipelining



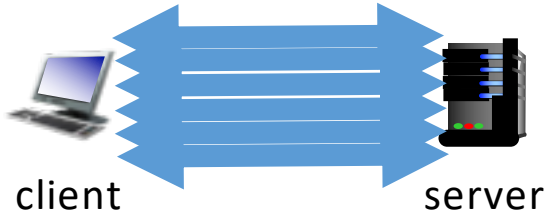
Send all requests on a single TCP connection

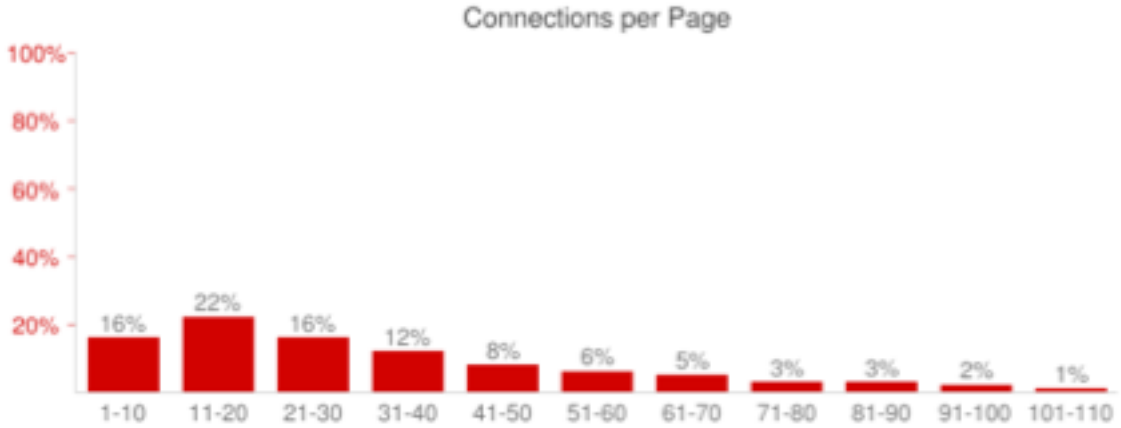
HTTP pipelining not typically used

- Responses must come back in order
- Head-of-line blocking
  - Delay in retrieving 1st content delays subsequent content
- Middleboxes may not understand pipelining
- If server processes requests in parallel, need to maintain large buffers

Reducing latency: Connection-based strategies

- Multiple TCP connections
  - Open up to 6 connections per server
    - Handled by browser
    - cwnd x6, effectively



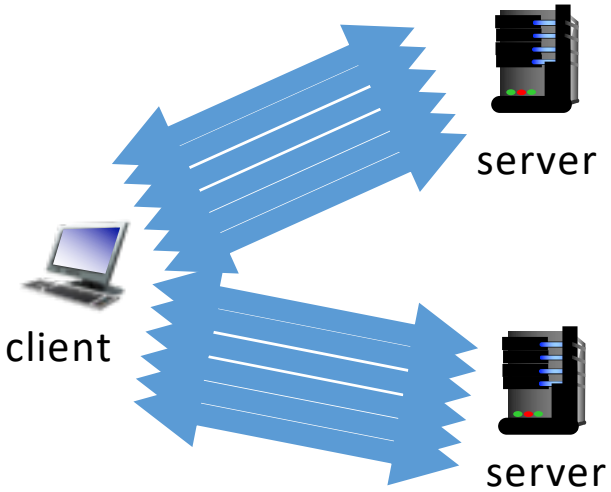


  - Challenges
    - Extra overhead to maintain all those TCP connections

Reducing latency: Content-based strategies

- Concatenating files
  - JavaScript, CSS
  - Less modular, large bundles
- Spriting images
  - Combine multiple images into one
  - What a pain...

- Domain sharding
  - Store files on multiple domains (shards) so that many TCP connections can be opened
  - Congestion control who? 30+ parallel requests --- Yeehaw!!!



- Resource inlining
  - TCP connections are expensive!

```
<head>
  <link rel="stylesheet" href="small.css">
</head>
```

vs

```
<head>
 <style>
  .yellow {background-color: yellow;}
  .blue {color: blue;}
  .big { font-size: 8em; }
  .bold { font-weight: bold; }
 </style>
 </head>
```

Overview

- Measuring webpage latency
- Strategies to reduce latency
- SPDY and HTTP/2
- SPDY over cellular networks

Q: How to reduce page load times for webpages?

HTTP/0.9

$> telnet google.com 80
Connected to 74.125.xxx.xxx

GET /about/

(hypertext response)

(connection closed)

Simplest mode: single request, single response, then connection close

HTTP/1.0

$> telnet website.org 80
Connected to xxx.xxx.xxx.xxx

GET /rfc/rfc1945.txt HTTP/1.0
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
Accept: */*

HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 137582
Expires: Thu, 01 Dec 1997 16:00:00 GMT
Last-Modified: Wed, 1 May 1996 12:45:26 GMT
Server: Apache 0.84

(plain-text response)

(connection closed)

Multi-line requests

Response header, status code

Response data not limited to plain-text

HTTP/1.1

```
$> telnet website.org 80
Connected to xxx.xxx.xxx.xxx

GET /index.html HTTP/1.1
Host: website.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4)...
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.
8
Accept-Encoding: gzip,deflate,sdchAccept-Language: en-
US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: __qca=P0-800083390...

HTTP/1.1 200 OK
Server: nginx/1.0.11
Connection: keep-alive
Content-Type: text/html; charset=utf-8Via: HTTP/1.1 GWA
Date: Wed, 25 Jul 2012 20:23:35 GMT
Expires: Wed, 25 Jul 2012 20:23:35 GMT
Cache-Control: max-age=0, no-cache
Transfer-Encoding: chunked

(html data)
```

request

response header

```
GET /favicon.ico HTTP/1.1
Host: www.website.orgUser-Agent: Mozilla/5.0 (Macintosh; Intel
Mac OS X 10_7_4)...
Accept: */*Referer: http://website.org/
Connection: close
Accept-Encoding: gzip,deflate,sdchAccept-Language: en-
US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: __qca=P0-800083390...

HTTP/1.1 200 OK
Server: nginx/1.0.11
Content-Type: image/x-icon
Content-Length: 3638
Connection: close
Last-Modified: Thu, 19 Jul 2012 17:51:44 GMT
Cache-Control: max-age=315360000
Accept-Ranges: bytes
Via: HTTP/1.1 GWA
Date: Sat, 21 Jul 2012 21:35:22 GMT
Expires: Thu, 31 Dec 2037 23:55:55 GMT
Etag: W/PSA-GAu26oXbDi

(icon data)
(connection closed)
```

request

response header

Additional features in header (e.g. cookies, caching, character set)

Connection NOT closed after every request

Drawbacks of HTTP

- TCP works best if a session is long lived and/or exchanges a lot of data
  - HTTP connections are typically short and exchange small objects
  - TCP cwnd takes time to adjust to the available network capacity
  - TCP does not have sufficient time to utilize the full network capacity
- Client is the only one to initiate request for an object
  - Server has to let client know to request object
- Request and response headers are uncompressed
  - Redundant Headers (User Agent, Host, Accept etc.)
  - Headers can be large – 200 bytes to 2K bytes

HTTP/2.1

- Reduce end-user perceived latency over HTTP/1.1 using TCP
- Address the "head of line blocking" problem in HTTP
- Not require multiple connections to a server to enable parallelism, thus improving its use of TCP

- Retain the semantics of HTTP/1.1, including (but not limited to)
  - HTTP methods
  - Status Codes
  - URIs
  - Header fields

- ~50% adoption as of May 2021

- SPDY (pronounced “speedy”) = precursor of HTTP/2
  - Many small differences

Thanks to Ilya Grigorik for SPDY slides

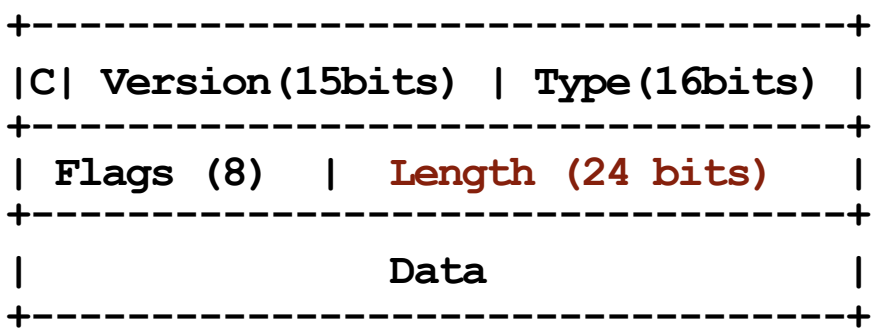
SPDY in a Nutshell

- One TCP connection
- Request = Stream (bi-directional)

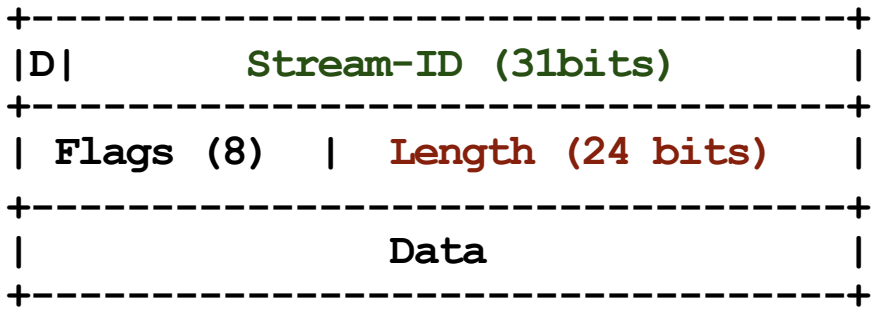
- Streams are multiplexed
- Streams are prioritized

- Binary framing
  - Encode control, data frames as binary
  - No longer human readable!
- Length-prefixed
  - Can quickly skip to next frame
  - As opposed to newlines in HTTP 1.x

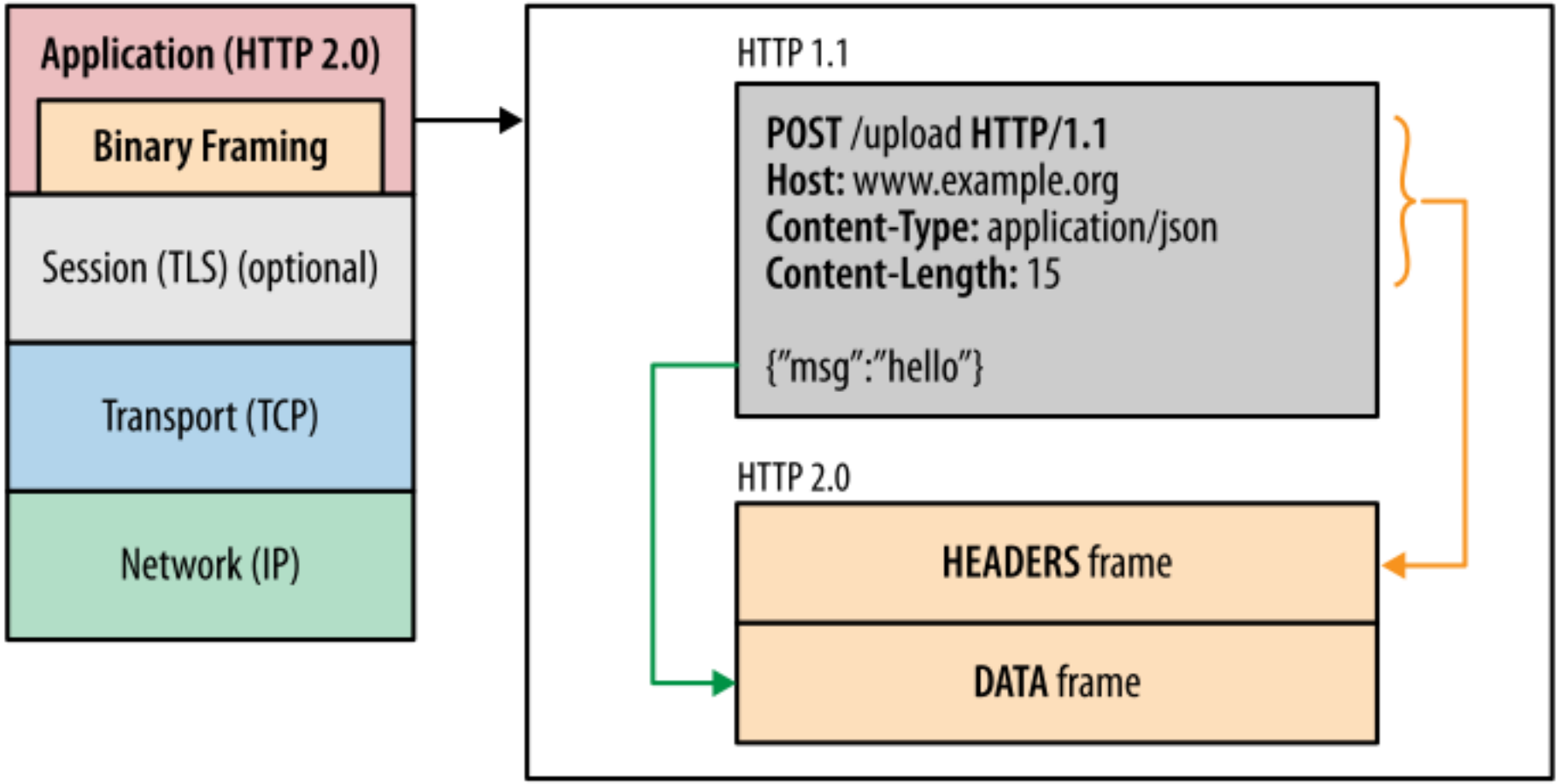
Control Frame:



Data Frame:

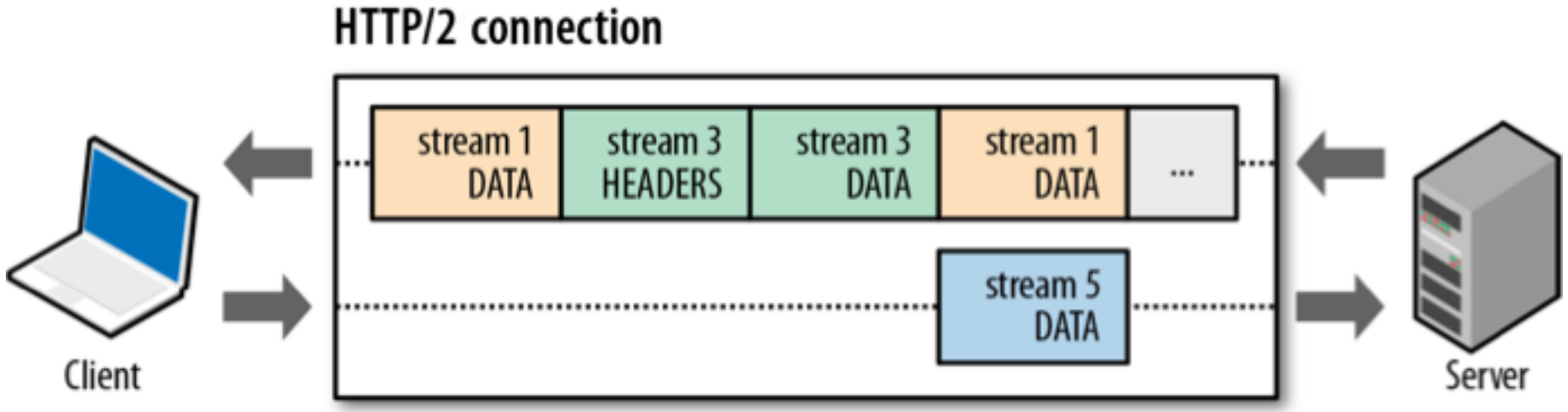


HTTP 1.1 vs HTTP 2 headers

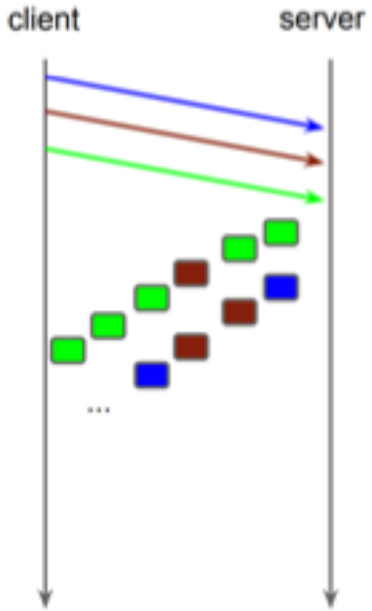


Source: https://developers.google.com/web/fundamentals/performance/http2/

SPDY in action



- Full request & response multiplexing
  - Assign a stream id # to each request
- Mechanism for request prioritization



- Many small files? No problem
- Higher TCP window size
- More efficient use of server resources
- TCP Fast-retransmit for faster recovery

Speaking of HTTP Headers...

```
curl -vv -d'{"msg":"oh hai"}' http://www.igvita.com/api

> POST /api HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0)
libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
> Host: www.igvita.com
> Accept: */*
> Content-Length: 16
> Content-Type: application/x-www-form-urlencoded

< HTTP/1.1 204
< Server: nginx/1.0.11
< Content-Type: text/html; charset=utf-8
< Via: HTTP/1.1 GWA
< Date: Thu, 20 Sep 2012 05:41:30 GMT
< Expires: Thu, 20 Sep 2012 05:41:30 GMT
< Cache-Control: max-age=0, no-cache
....
```

- Average request / response header overhead: **800 bytes**

- No compression for headers in HTTP!
- Huge overhead

- **Solution:** compress the headers!
  - gzip all the headers
  - SPDY and HTTP/2 differ in their compression schemes

SPDY Server Push

Premise: server can push resources to client

Concern: but I don't want the data! Stop it!

Client can cancel SYN_STREAM if it doesn't want the resource

Resource goes into browser's cache (no client API)

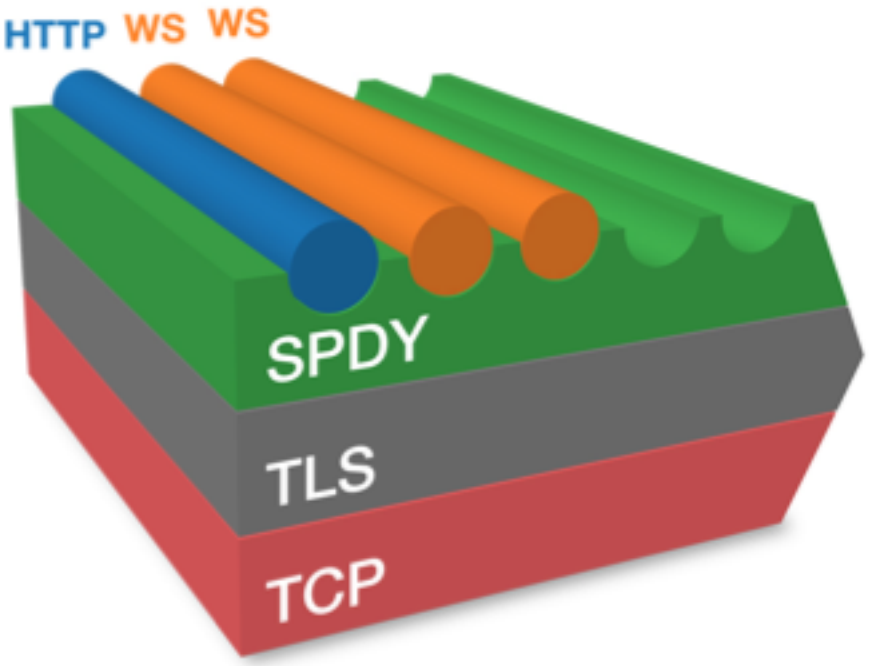
Newsflash: we are already using "server push"

- Today, we call it "inlining"
- Inlining works for unique resources, bloats pages otherwise

Advanced use case: forward proxy (e.g., Amazon's Silk can render remotely and push content to client)

Proxy has full knowledge of your cache, can intelligently push data to the client

Encrypt all the things!!!



SPDY runs over TLS

- Philosophical reasons
- Political reasons
- **Pragmatic + deployment reasons - Bing!**

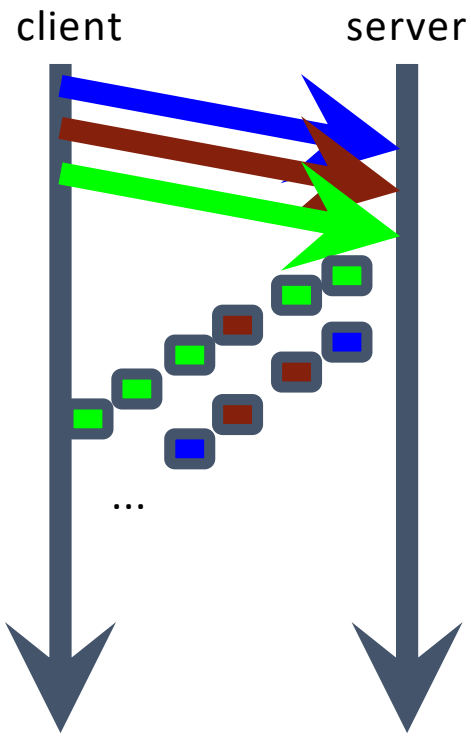
Not required for HTTP/2, but often seen in practice

Observation: intermediate proxies get in the way

Some do it intentionally, many unintentionally

Ex: Antivirus / Packet Inspection / QoS / ...

~~HTTP Head of line blocking...~~ TCP Head of line blocking

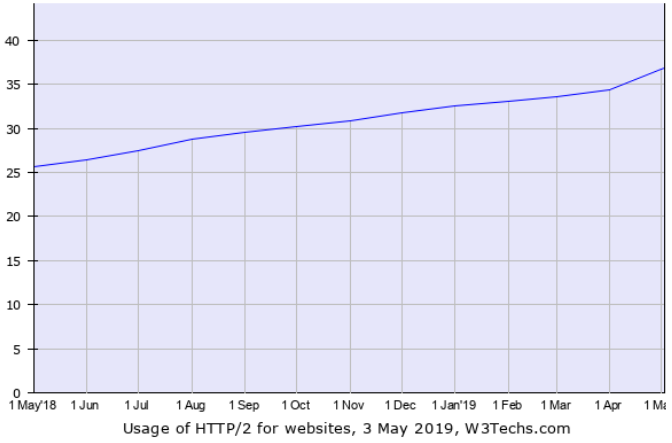


- TCP: in-order, reliable delivery...
  - *what if a packet is lost?*
- **~1~2% packet loss rate**
  - CWND's get chopped
  - Fast-retransmit helps, but..
  - SPDY stalls

HTTP/2 Adoption (Top 50 million websites)

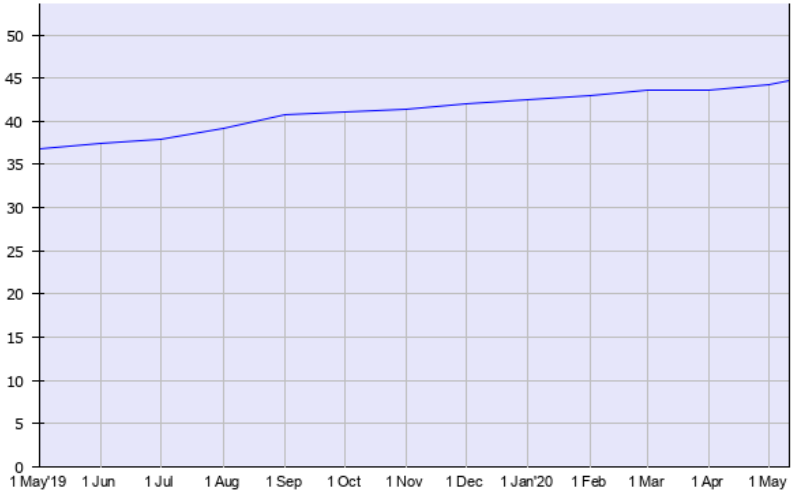
2017

2019



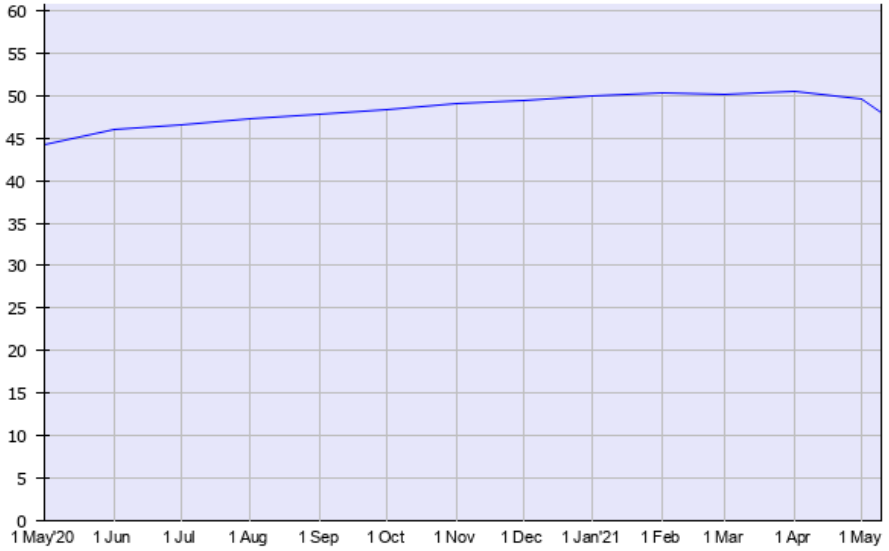
Usage of HTTP/2 for websites, 3 May 2019, W3Techs.com

2020



Usage of HTTP/2 for websites, 11 May 2020, W3Techs.com

2021



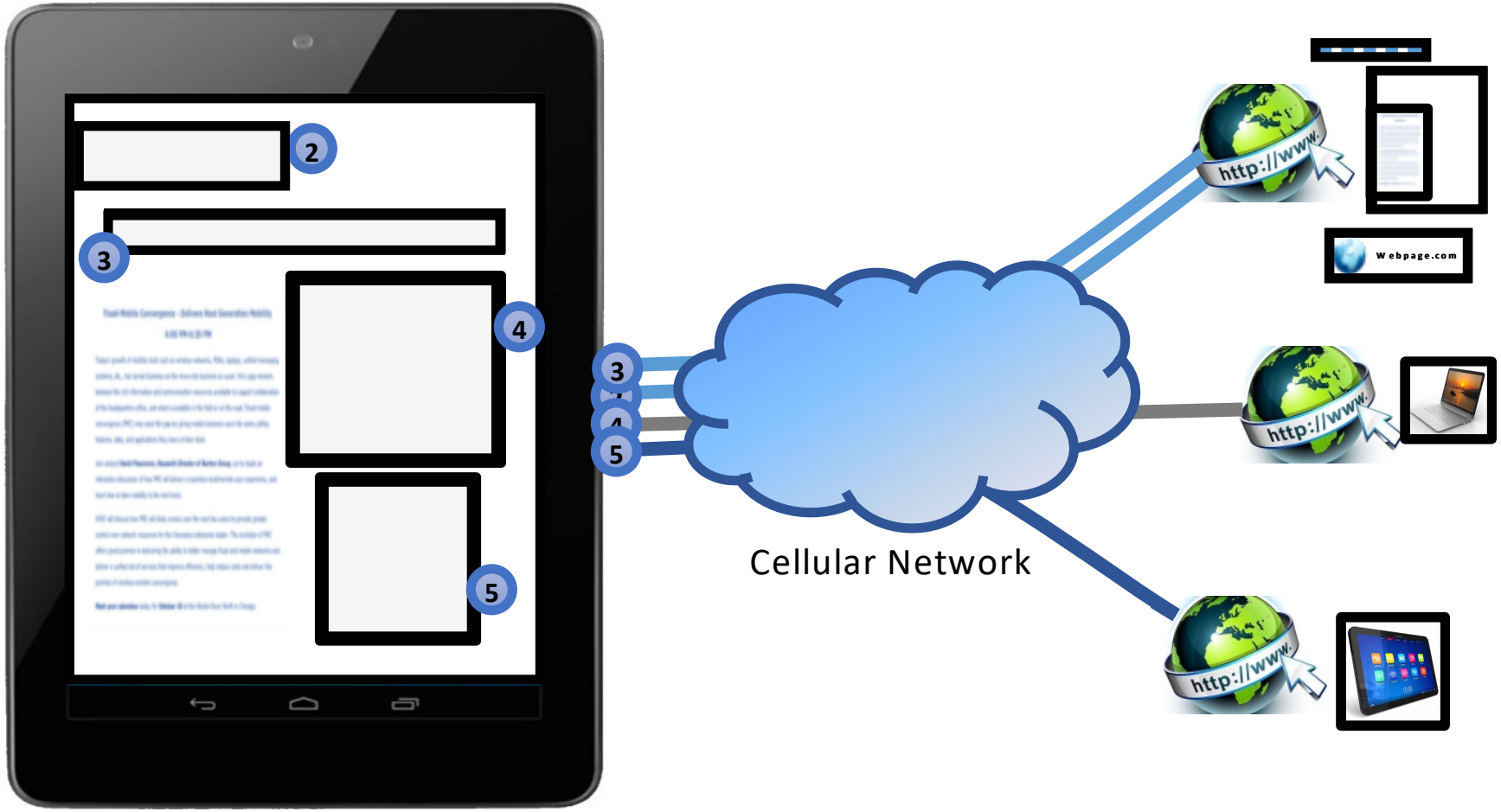
Usage of HTTP/2 for websites, 10 May 2021, W3Techs.com

Overview

- Measuring latency
- Current strategies for reducing latency
- SPDY and HTTP/2
- Mobile SPDY

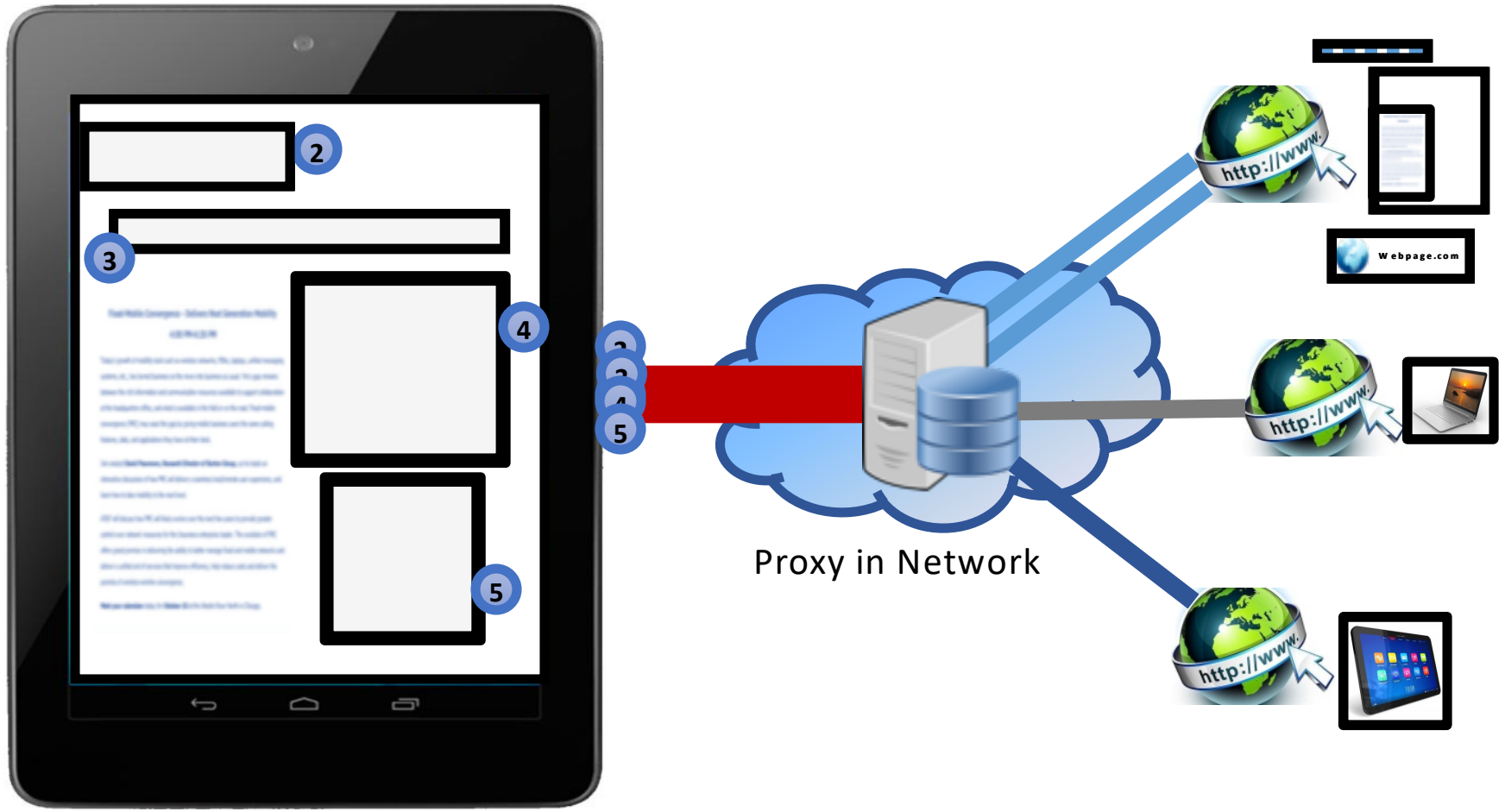
Q: How to reduce page load times for webpages?

Web Page delivery using HTTP



Steps are similar even when a proxy is employed in the cellular network

Web Page delivery using SPDY Proxy

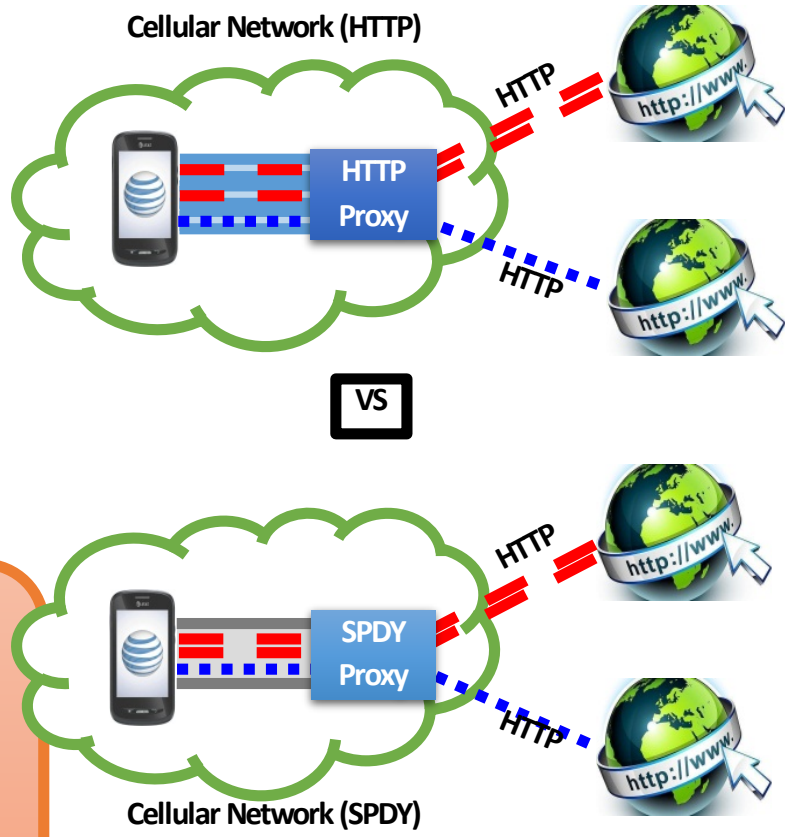


Interactions between SPDY and Transport – Especially Cellular Networks

- **SPDY seemingly solves important shortcomings with HTTP**
  - One long-lived TCP Connection instead of multiple short-lived connections
  - Saves on TCP connection establishment time (important in high latency networks)
  - Connection reuse allows TCP congestion window (`cwnd`) to grow

Does using SPDY help improve cellular web experience in practice?

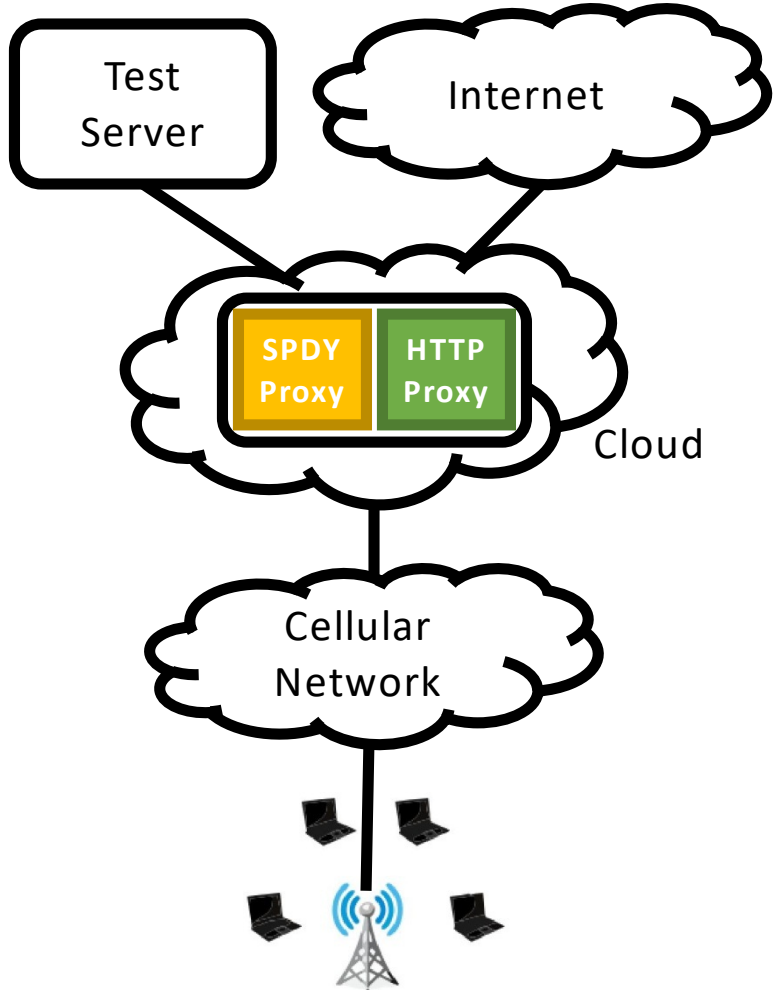
Should operators deploy a SPDY proxy in the cellular network?



Thanks to KK Ramakrishnan for mobile SPDY slides

Test Setup

- Client used Windows7 laptop with 3G USB dongle
- Server in cloud running SPDY proxy (from Google) and HTTP proxy (Squid)
  - Mimics typical cellular deployment
- Placed client in carefully chosen location
  - To prevent handover effects, we avoided placing client near cell edge
  - Relatively strong RSSI signal (-47 to -52 dBm)
  - Experiments conducted during nights (12AM – 6AM) during light cell load conditions
  - Sufficient backhaul capacity; eliminate backhaul bottleneck
- Used a test server for controlled experiments with web page content

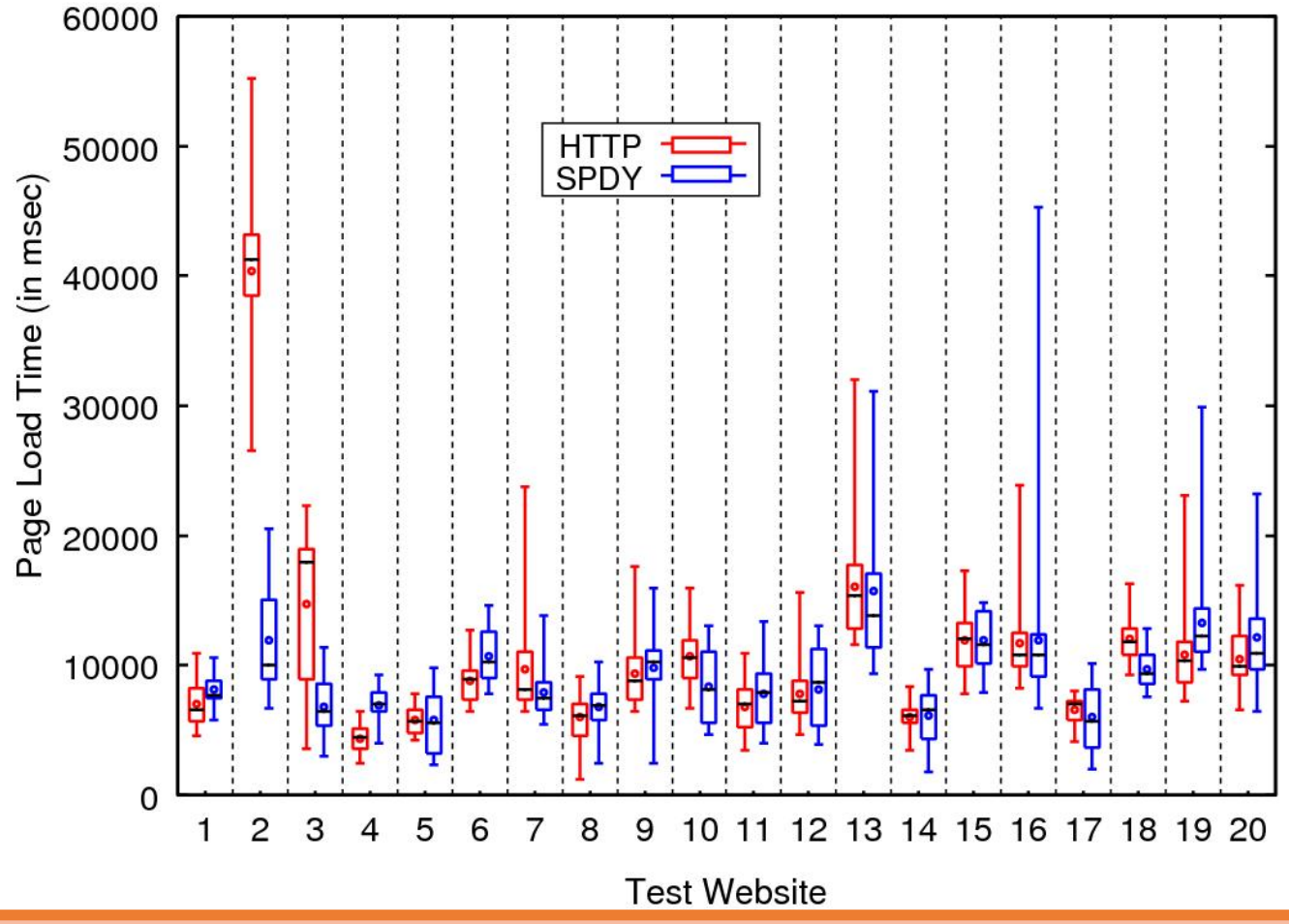


Test Execution and Methodology

- Visited top websites used by mobile users
  - Found in top 50 Alexa web pages also; eliminate landing pages (e.g., Facebook login page)
- Use “full” web site (as opposed to mobile pages)
  - Mimics tablets and cellular-equipped laptops
  - Existing work shows mobile page content similar to full page
- Test execution via automated client (written in Ruby)
  - Uses Chrome remote debugger capability
  - Executes one run with HTTP followed by SPDY;
  - Request new page every minute to account for page load and “think” time
  - Record detailed network statistics and page load time
- Packet capture (using tcpdump) and TCP statistics (using tcpprobe) on proxy server

	Value
Num. of websites	20
Num. of objects/site	5 – 323
Num. of domains	3 – 84
Time between pages	60 seconds

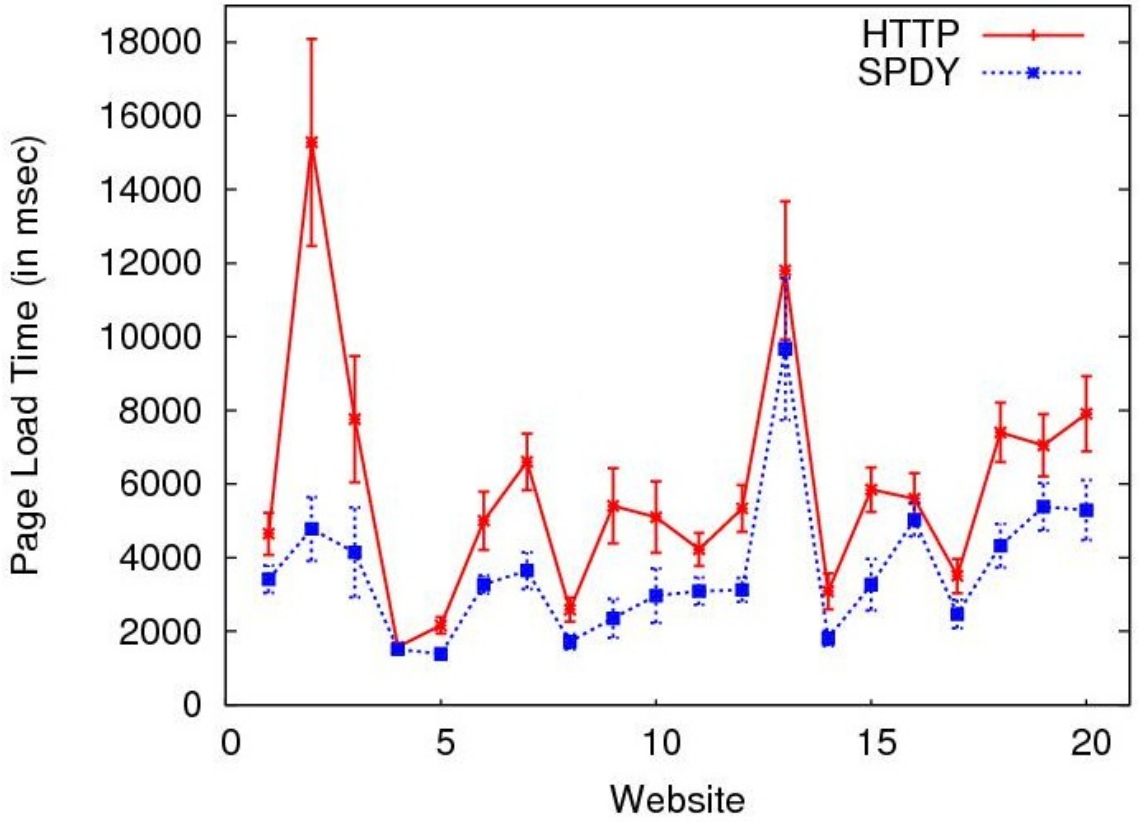
Page Load Time in 3G cellular network



No convincing winner between HTTP and SPDY; Sharp contrast to existing results on SPDY

Page Load Time with WiFi Network

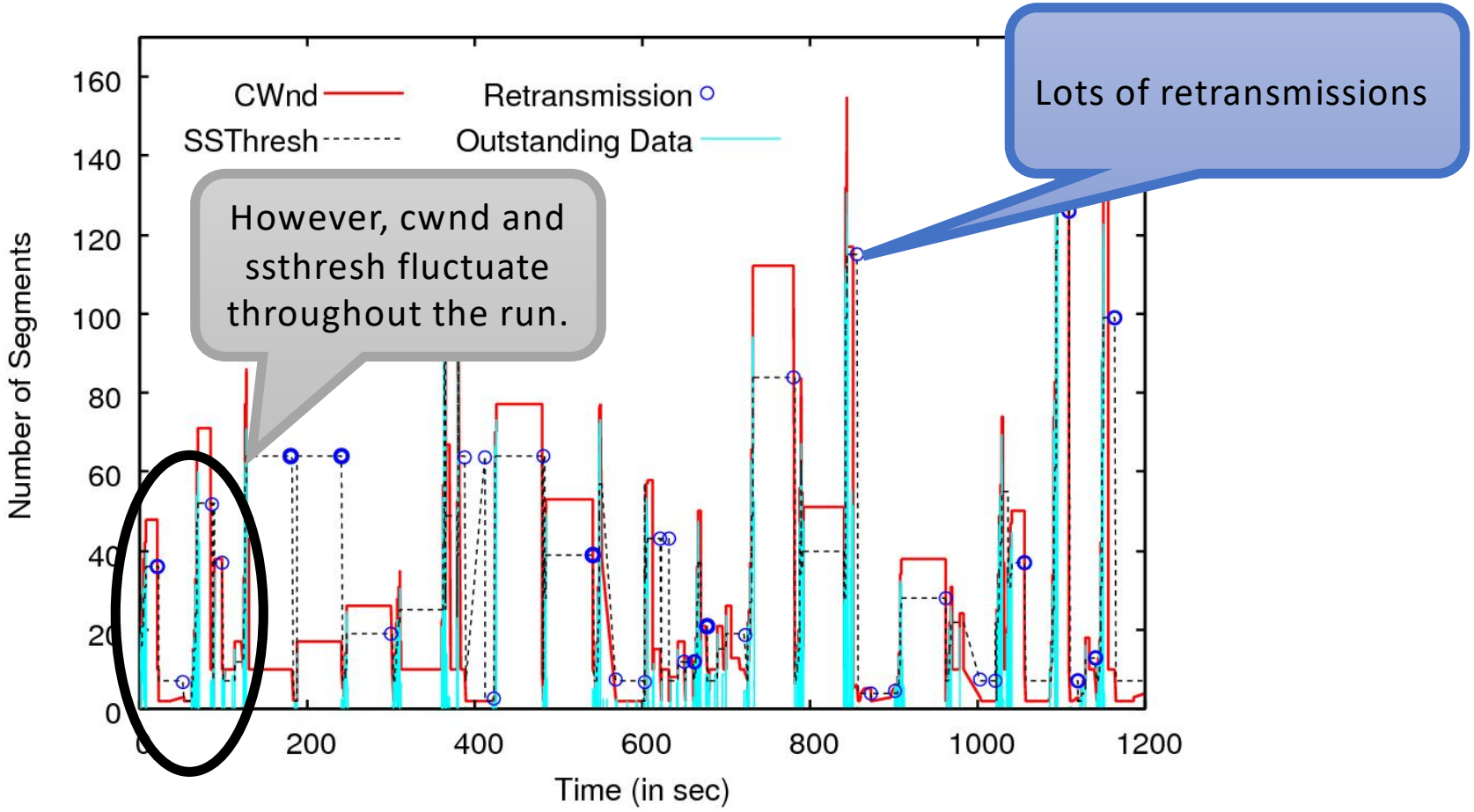
- Check if result was a consequence of setup
- Ran tests with laptop connecting via WiFi
  - WiFi router connects to Internet via broadband
- Same test procedure and sites



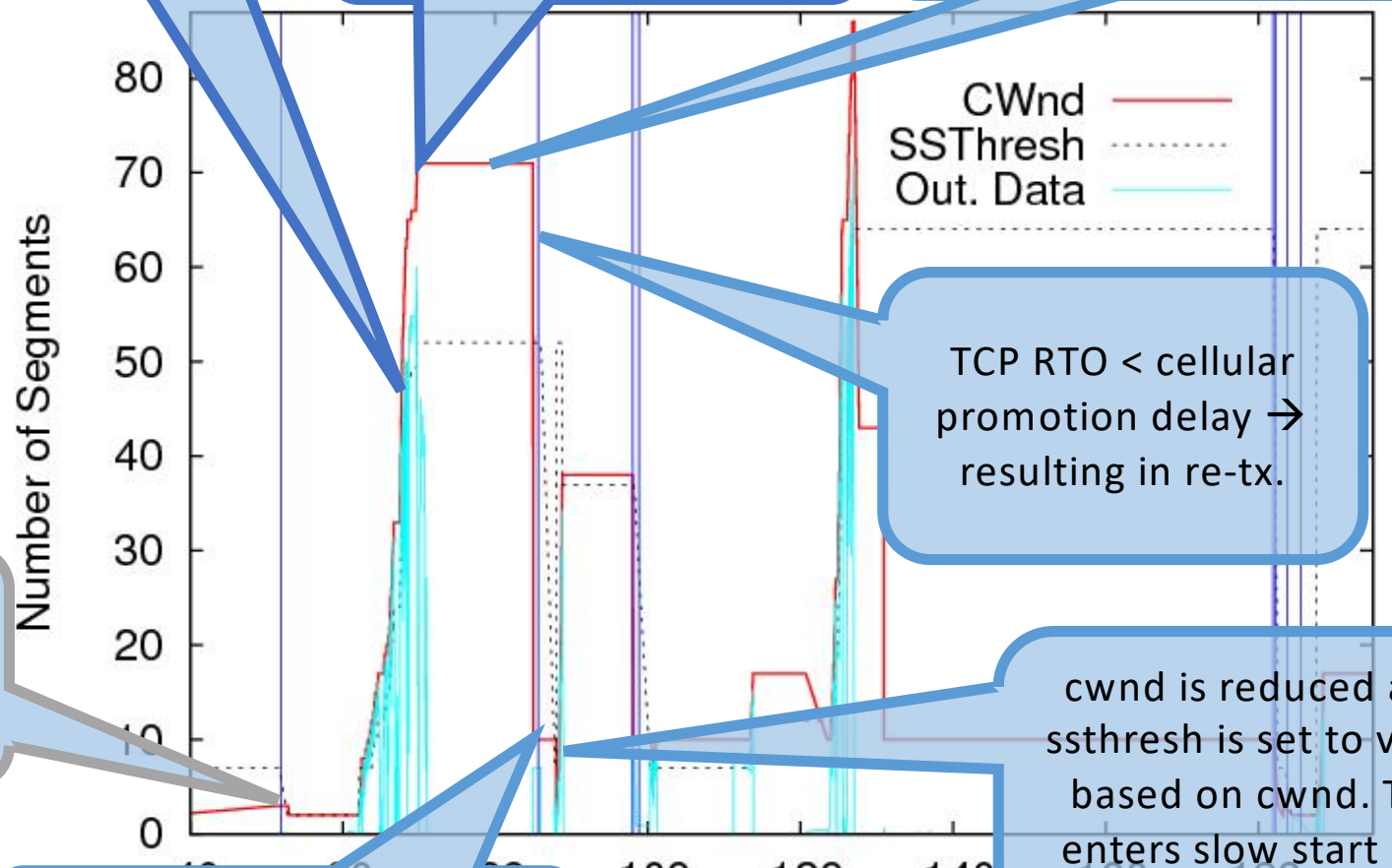
SPDY performs better than HTTP consistently with page load time improvements ranging from 4% to 56% with WiFi

TCP Congestion Window fluctuates with SPDY

Expectation would be for cwnd and ssthresh to grow and stabilize.



Congestion window with SPDY tick tock…



Grow as data is transferred

cwnd>ssthresh. TCP stays in congestion avoidance.

No data transferred and cellular connection goes to idle.

TCP RTO < cellular promotion delay → resulting in re-tx.

Cwnd and ssthresh are initially small due to multiple re-tx between 0-60 sec.

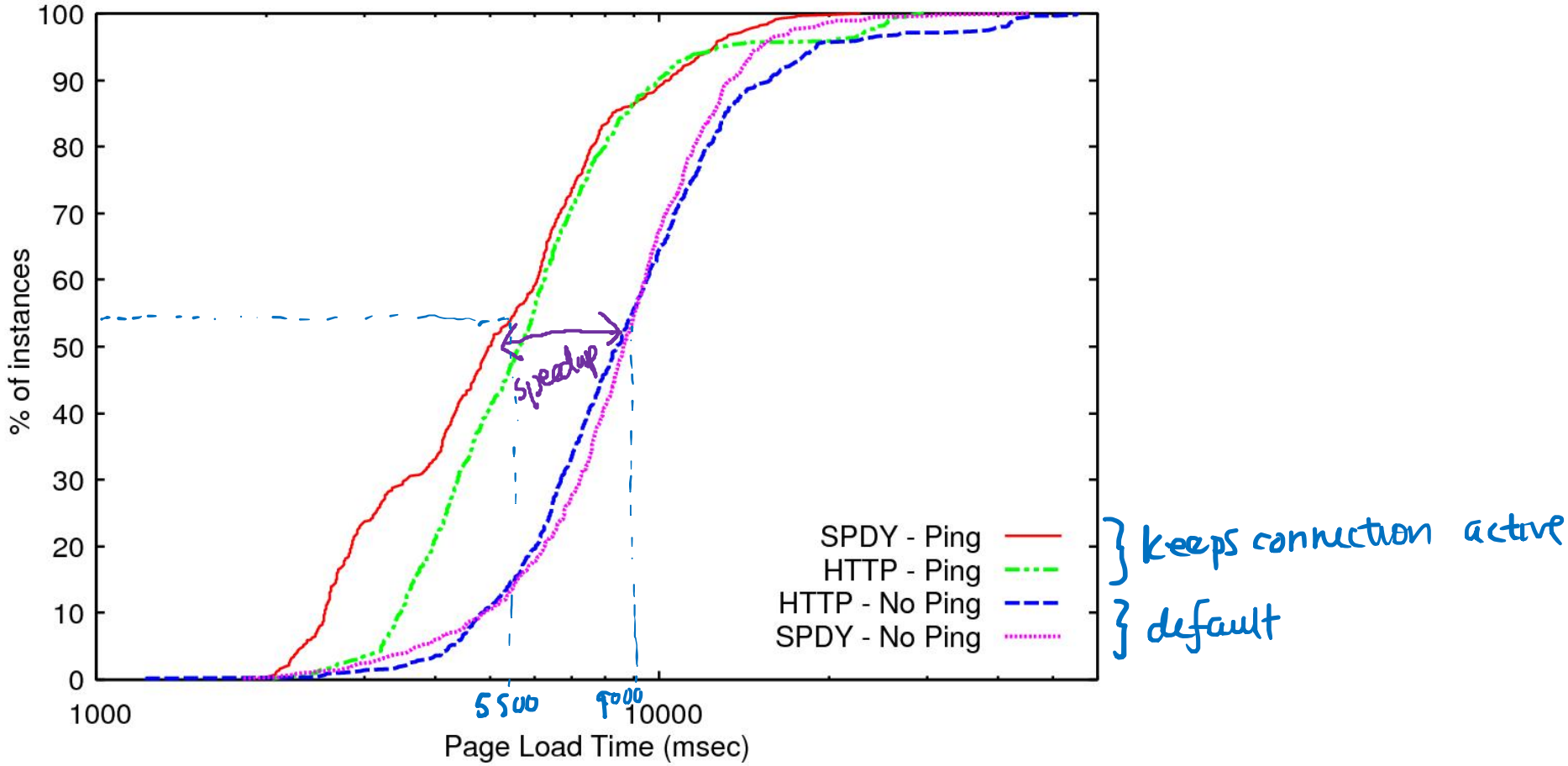
Idle triggers tcp_slow_start_after_idle. cwnd = 10

cwnd is reduced and ssthresh is set to value based on cwnd. TCP enters slow start and quickly rebounds.

- RTO = TCP re-transmission timeout
- Cellular promotion delay = delay when cellular connection moves from idle to active (~2 s)
- Re-tx = Packet retransmission

Impact of Cellular State Machine

Idea: send pings to make the connection continuously active.



Having device always in "active" mode results in lower page load times.

Sources

1. “SPDY: An experimental protocol for a faster web,” Google, http://www.chromium.org/spdy/spdy-whitepaper.
2. “How Speedy is SPDY?,” Wang et al., *NSDI* 2014.
3. “Towards a SPDY’ier Mobile Web?,” Erman et al., *IEEE Trans. Networking*, 2015.
4. *High Performance Browser Networking*, Ilya Grigorik, O’Reilly, 2013.
5. *Computer Networking: A Top-Down Approach* (6th ed), James Kurose and Keith Ross, 2011.