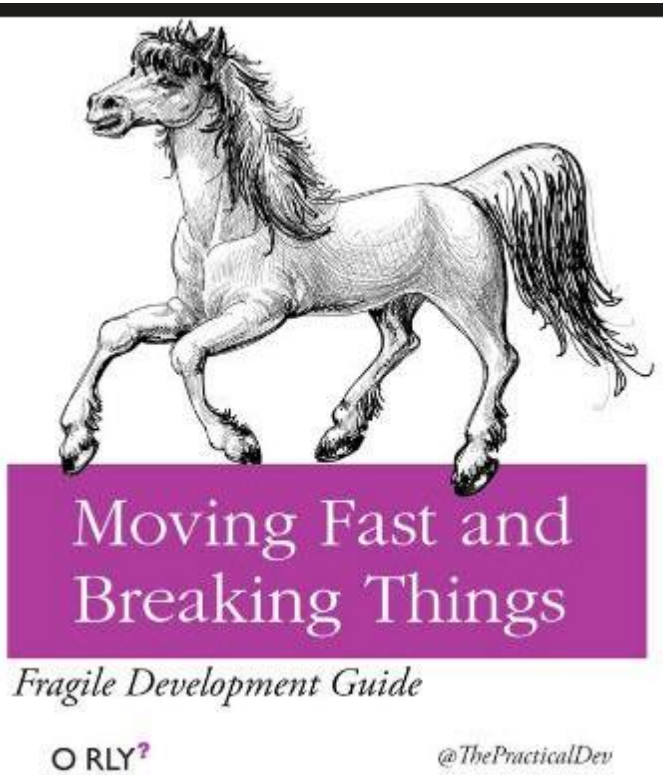


Concatenation, Embedding and Sharding:

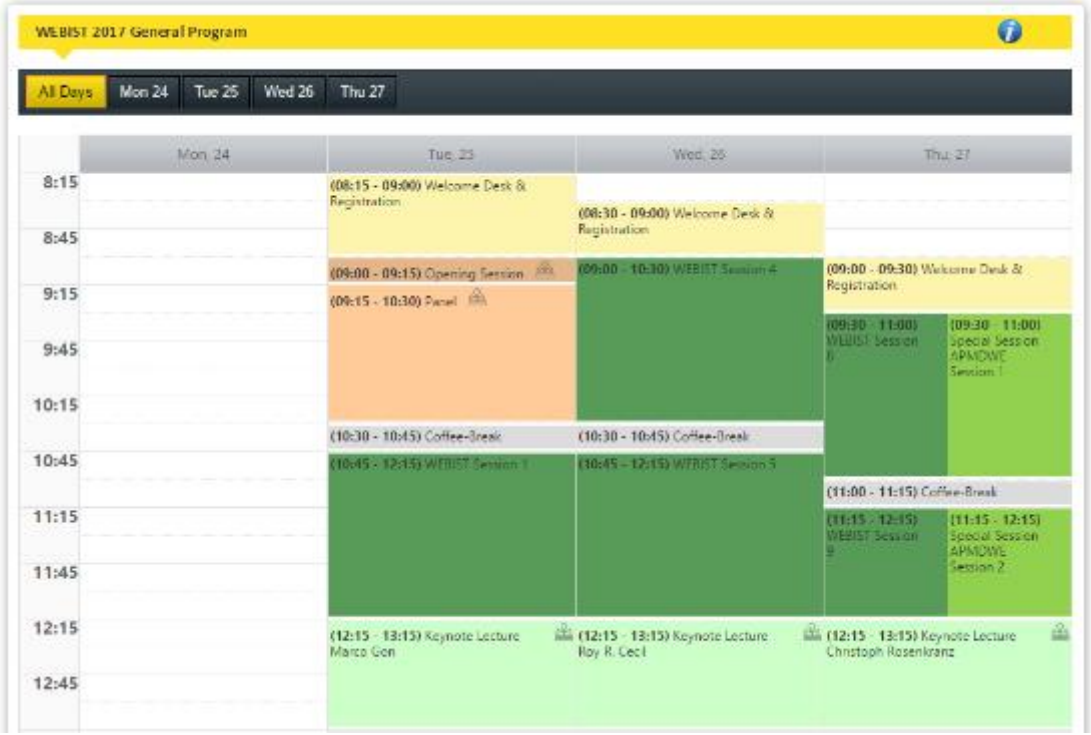
Do HTTP/1 Performance Best Practices Make Sense in HTTP/2?

UHASSELT
KNOWLEDGE IN ACTION

EDM | imec

Robin Marx
PhD Researcher
robin.marx@uhasselt.be

PRIMORIS
EVENT MANAGEMENT SYSTEM



EDM imec UHASSELT KNOWLEDGE IN ACTION

Web Performance

- HTTP/2 is the new kid in town!
- Replaces good old trusted HTTP/1.1
  - Currently 13.3% of sites is on H2*

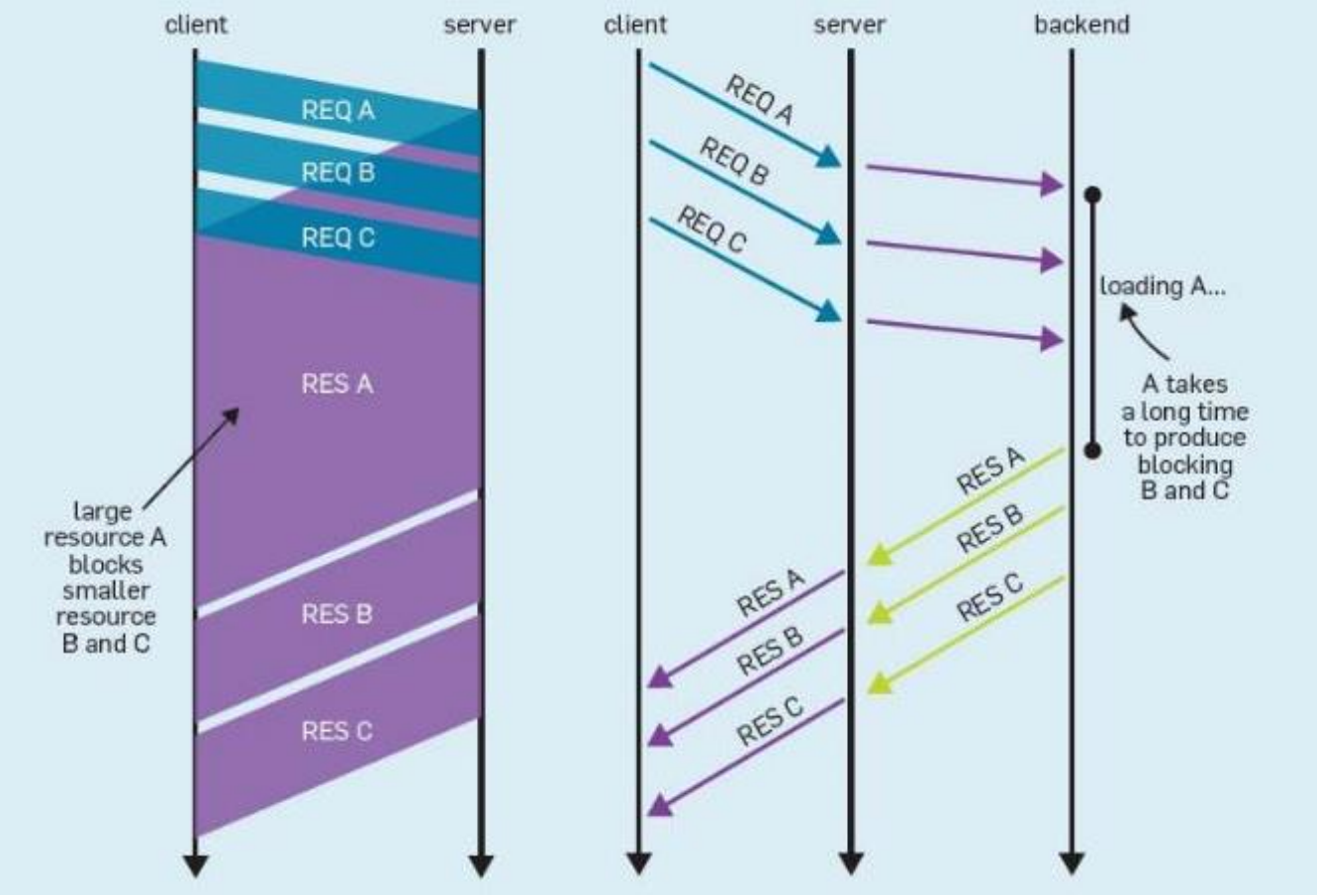
- How is HTTP/2 different from HTTP/1.1?
- Is h2 always faster than h1? And why?

- Do we need to change our sites to get the most out of h2?

*https://w3techs.com/technologies/details/ce-http2/all/all

HTTP/1.1 : the bad and the ugly 1

- Single resource at a time per TCP connection
- "Head of Line" (HOL) blocking



http://deliveryimages.acm.org/10.1145/2390000/2380673/figs/f5.jpg

HTTP/1.1 : The solution!

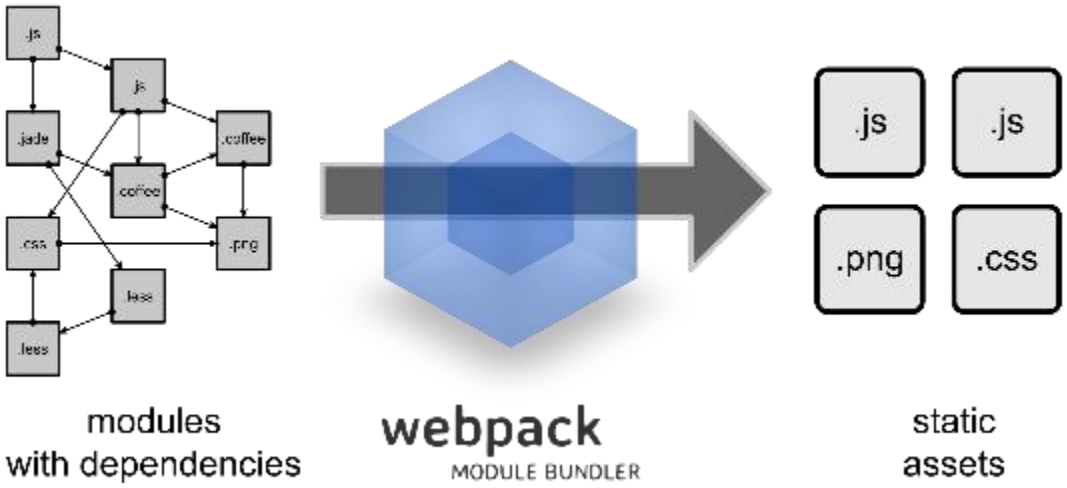
- 6 parallel TCP/HTTP connections per domain

http://www.splicemarketing.co.uk/_blog/Splice_Blog/post/the-arrival-of-http2/

HTTP/1.1 : 6 is not enough!

- Best Practice 1 : Concatenation
- = merge files together to reduce # resources

- 1.js + 2.js + 3.js = scripts.js
- Downside: reduced cacheability

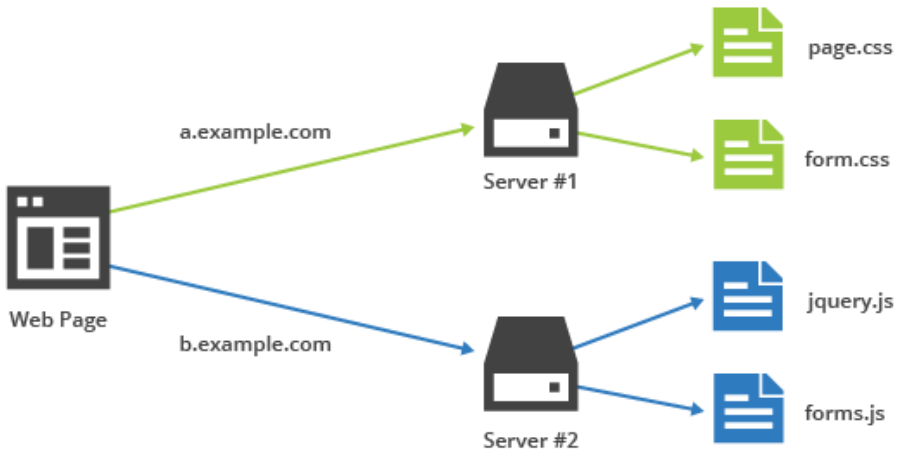


https://www.phase2technology.com/blog/bundle-your-front-end-with-webpack/

HTTP/1.1 : 6 is not enough!

- Best Practice 2 : Domain Sharding
- Content Delivery Networks (CDNs)
- img1.site.com, img2.site.com, static.site.com

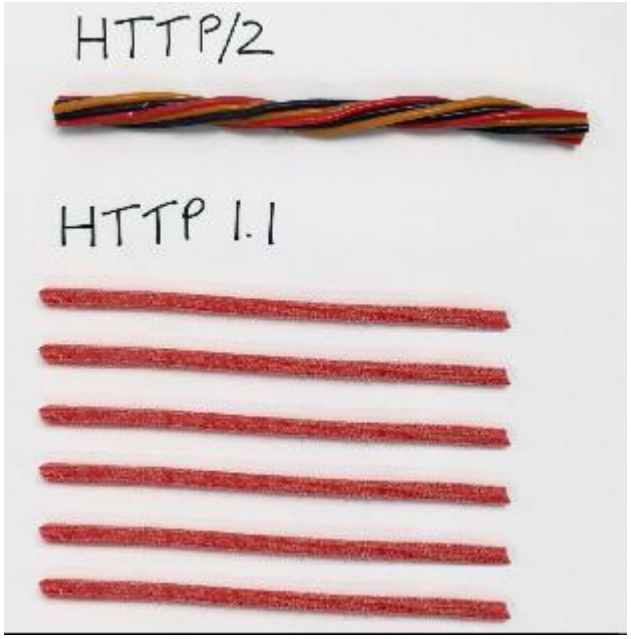
- Downside: serverside overhead

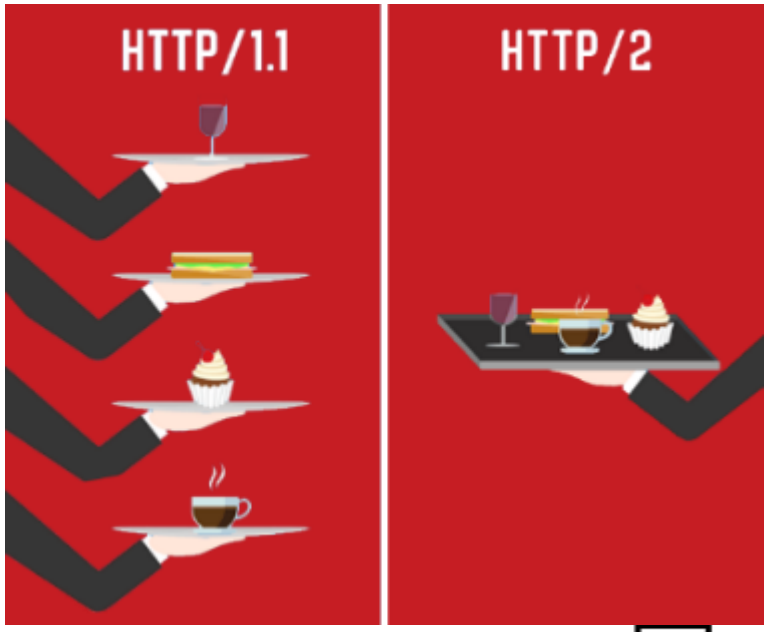


https://blog.cloudflare.com/content/images/2015/12/domain-sharding-1.png

HTTP/2 : The Challenger

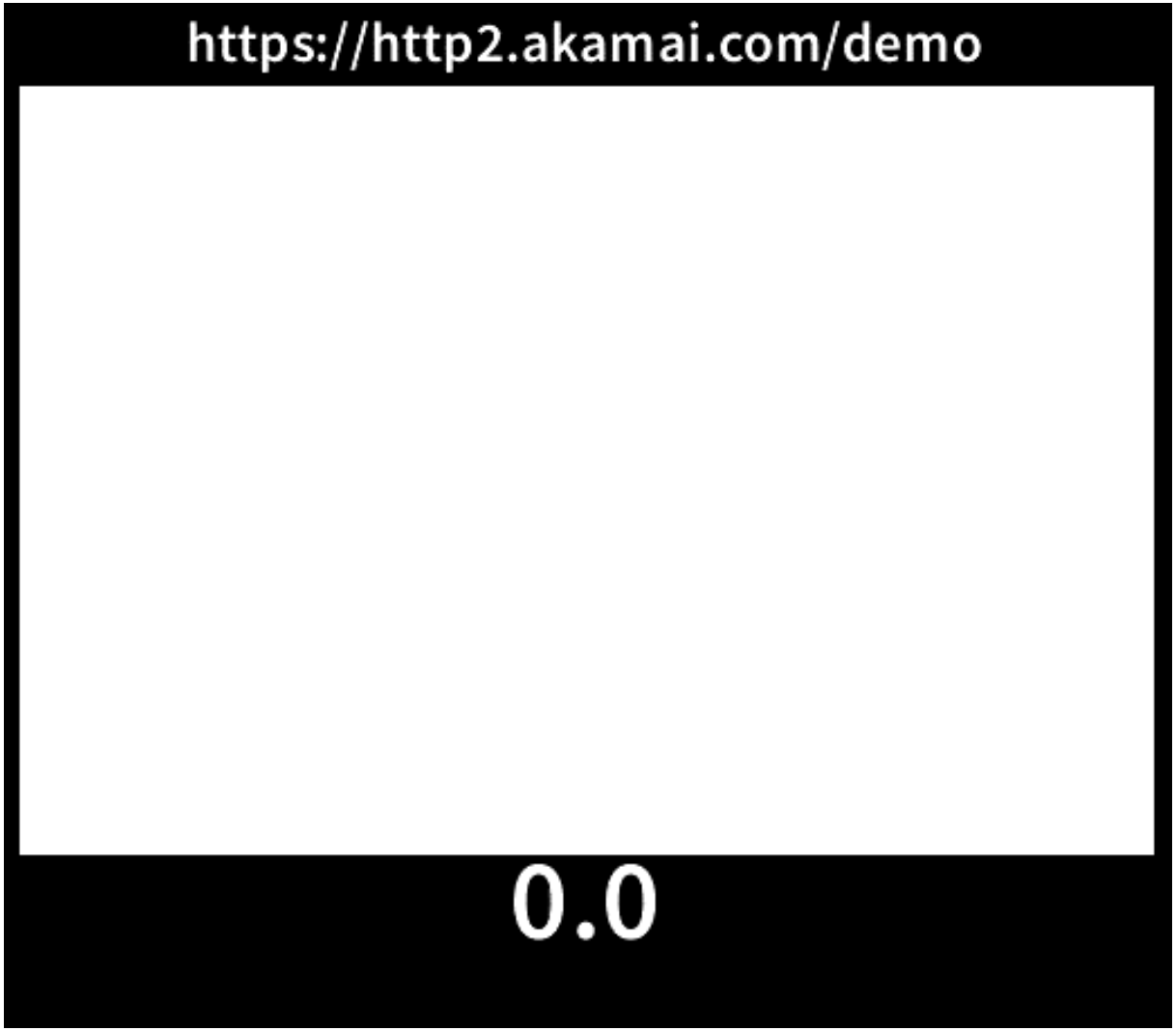
- Started as SPDY, standardized in 2016
- Back to 1 TCP connection
  - Priority-based multiplexing of resource streams





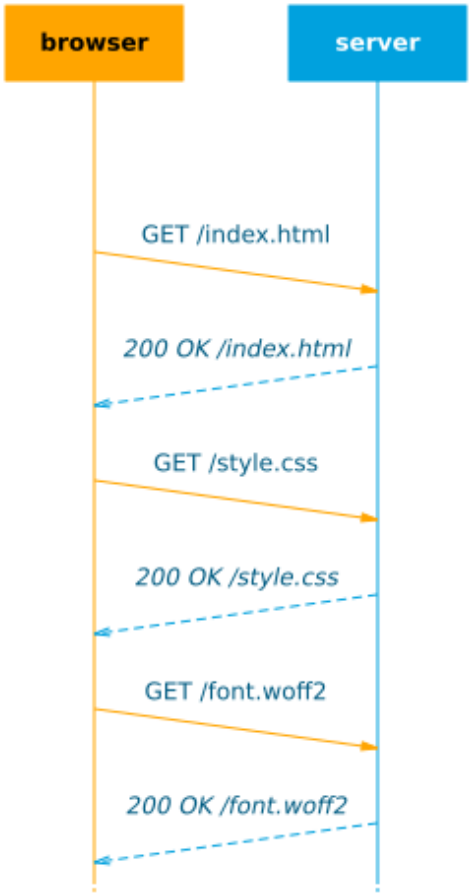
http://www.splicemarketing.co.uk/_blog/Splice_Blog/post/the-arrival-of-http2/

HTTP/2 : Solves HOL blocking!



HTTP/1.1 : the bad and the ugly 2

- Round trip times cause most of the delay
- “dependency graph” / resource discovery



Index.html
references
Style.css
references
Font.woff2

-> min. 3 round-trips needed before rendering can start!

https://calendar.perfplanet.com/2016/http2-push-the-details/

HTTP/1.1 : Round trip times

- Best Practice 3 : Resource embedding
- = Inline resource inside HTML file

- <link rel="stylesheet" href="style.css" />
  VS
- <style> #myDiv{ color: #FFBEEF; } </style>

- Downside: reduced cacheability

HTTP/2 : The Challenger

- Ability to Push resources along

https://calendar.perfplanet.com/2016/http2-push-the-details/

HTTP/2 : The bad and the ugly

- In theory, H1 best practices are now obsolete
  - High-performance browser networking (Grigorik, 2013)
- But: Single TCP connection
  - More sensitive to loss than # parallel connections
- But: TCP in-order guarantee (kernel-level)
  - ~Head-of-line blocking when retransmit
- Problems expected on bad connections
- Difficult to estimate actual impact…

Epic battles in contradictory previous work

Epic battles in contradictory previous work

1. Implementations change
   - SPDY vs H2
   - Server and Browser support
2. Test setups are limited
   - Single browser, server, metric, network, tool, platform, …

? Protocol-inherent or implementation-based ?

The Speeder Framework

TEST ALL THE THINGS!

The Speeder Framework

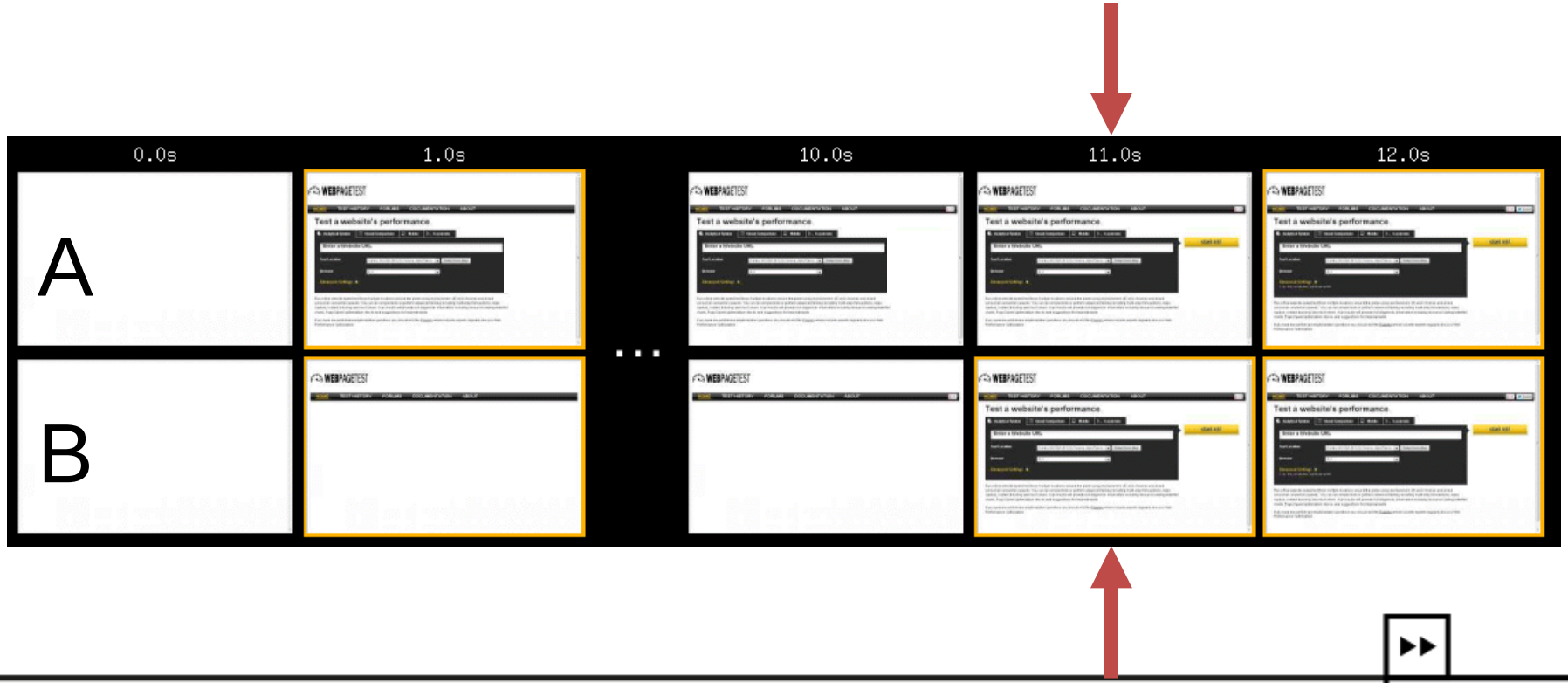
Table 1: Speeder software and metrics.

Protocols	HTTP/1.1 (cleartext), HTTPS/1.1, HTTPS/2
Browsers	Chrome (51 - 54), Firefox (45 - 49)
Test drivers	Sitespeed.io (3), Webpagetest (2.19)
Servers	Apache (2.4.20), NGINX (1.10), NodeJS (6.2.1)
Network	- `DUMMYNET` (cable and cellular) (Webpagetest) - fixed `TC NETEM` (cable and cellular) - dynamic `TC NETEM` (cellular) (Goel et al., 2016)
Metrics	All Navigation Timing values (Wang, 2012), SpeedIndex, firstPaint, visualComplete, other Webpagetest metrics (Meenan, 2016)

TEST MOST OF THE THINGS!

loadEventEnd vs SpeedIndex

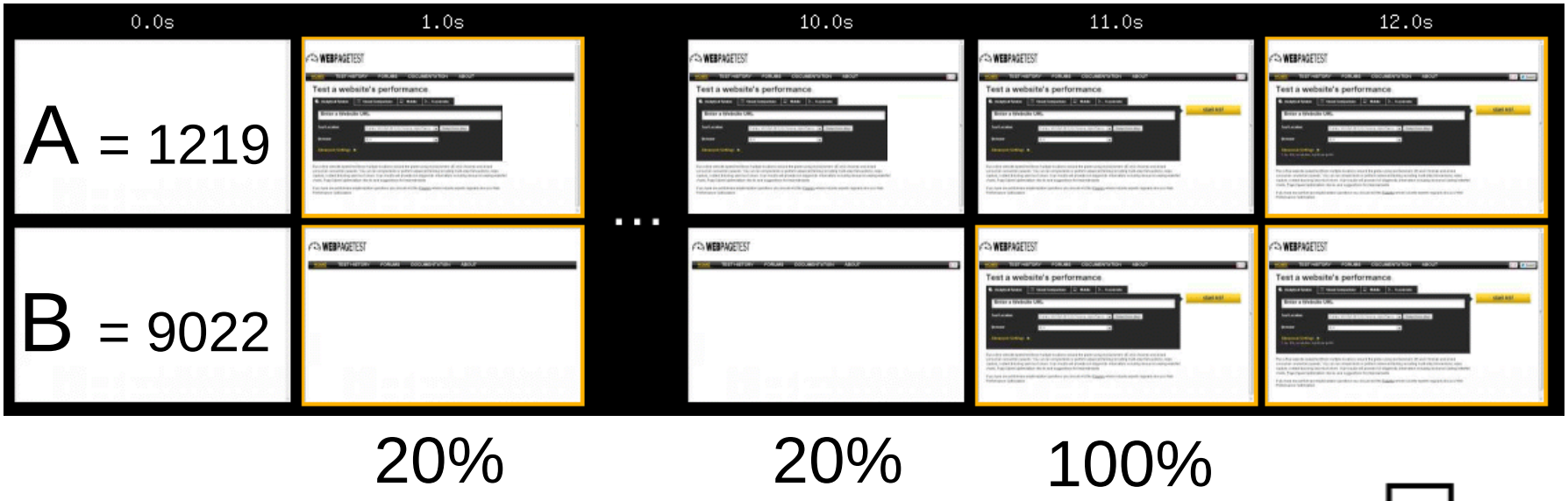
- loadEventEnd
  - document.onload
  - = Everything is downloaded and rendered



https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index

loadEventEnd vs SpeedIndex

- SpeedIndex
  - % of visual complete over time (video frames!)
  - Quick to render = better for user experience
  - Measured in milliseconds = LOWER IS BETTER



https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index

The Speeder Framework

Table 1: Speeder software and metrics.

Protocols	HTTP/1.1 (cleartext), HTTPS/1.1, HTTPS/2
Browsers	Chrome (51 - 54), Firefox (45 - 49)
Test drivers	Sitespeed.io (3), Webpagetest (2.19)
Servers	Apache (2.4.20), NGINX (1.10), NodeJS (6.2.1)
Network	- `DUMMYNET` (cable and cellular) (Webpagetest) - fixed `TC NETEM` (cable and cellular) - dynamic `TC NETEM` (cellular) (Goel et al., 2016)
Metrics	All Navigation Timing values (Wang, 2012), SpeedIndex, firstPaint, visualComplete, other Webpagetest metrics (Meenan, 2016)

TEST MOST OF THE THINGS!

The Speeder Framework

- Automate this setup using Docker
  - Light-weight stateless VMs based on install scripts
  - Easy to Update!
  - Easy to Share!

- Available on speeder.edm.uhasselt.be
- Go forth and reproduce!

Recap

1. 3 Best Practices to combat H1 downsides
2. HTTP/2 performance is unclear
3. Test many permutations with Speeder

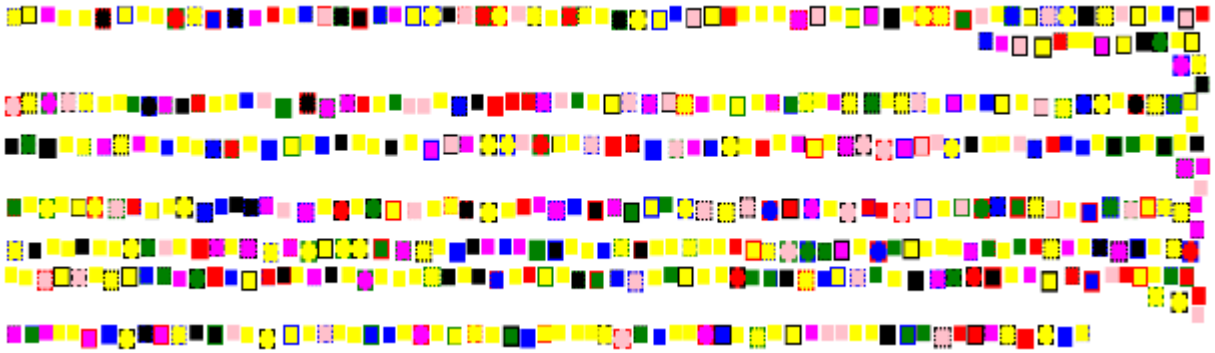
? Impact on developer workflow ?

- Synthetic test cases for best practices
- Realistic test cases

1. Concatenation

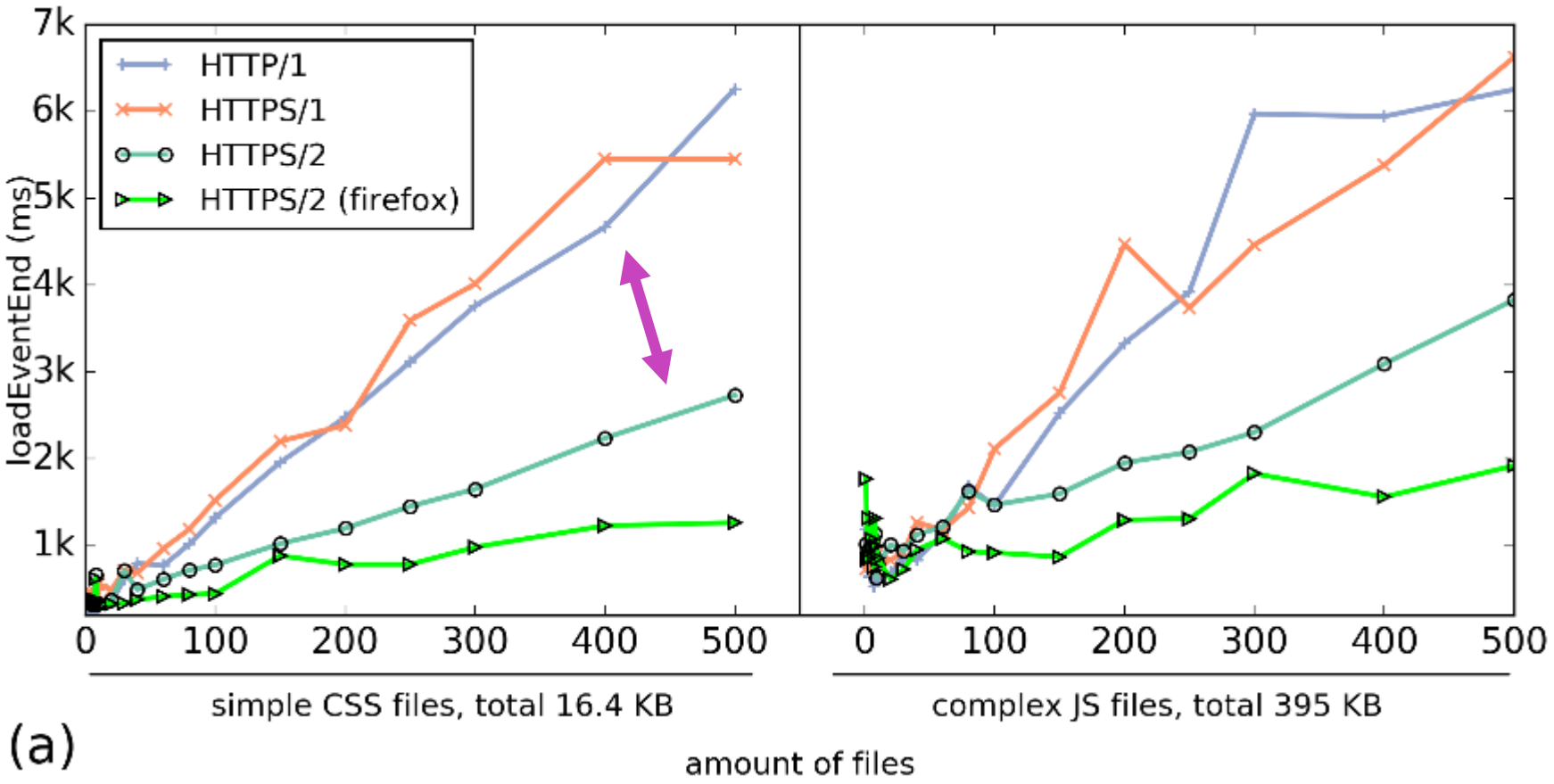
- CSS/JavaScript : 1 – 500 files,
  simple and complex

1 line per



multiple lines
per

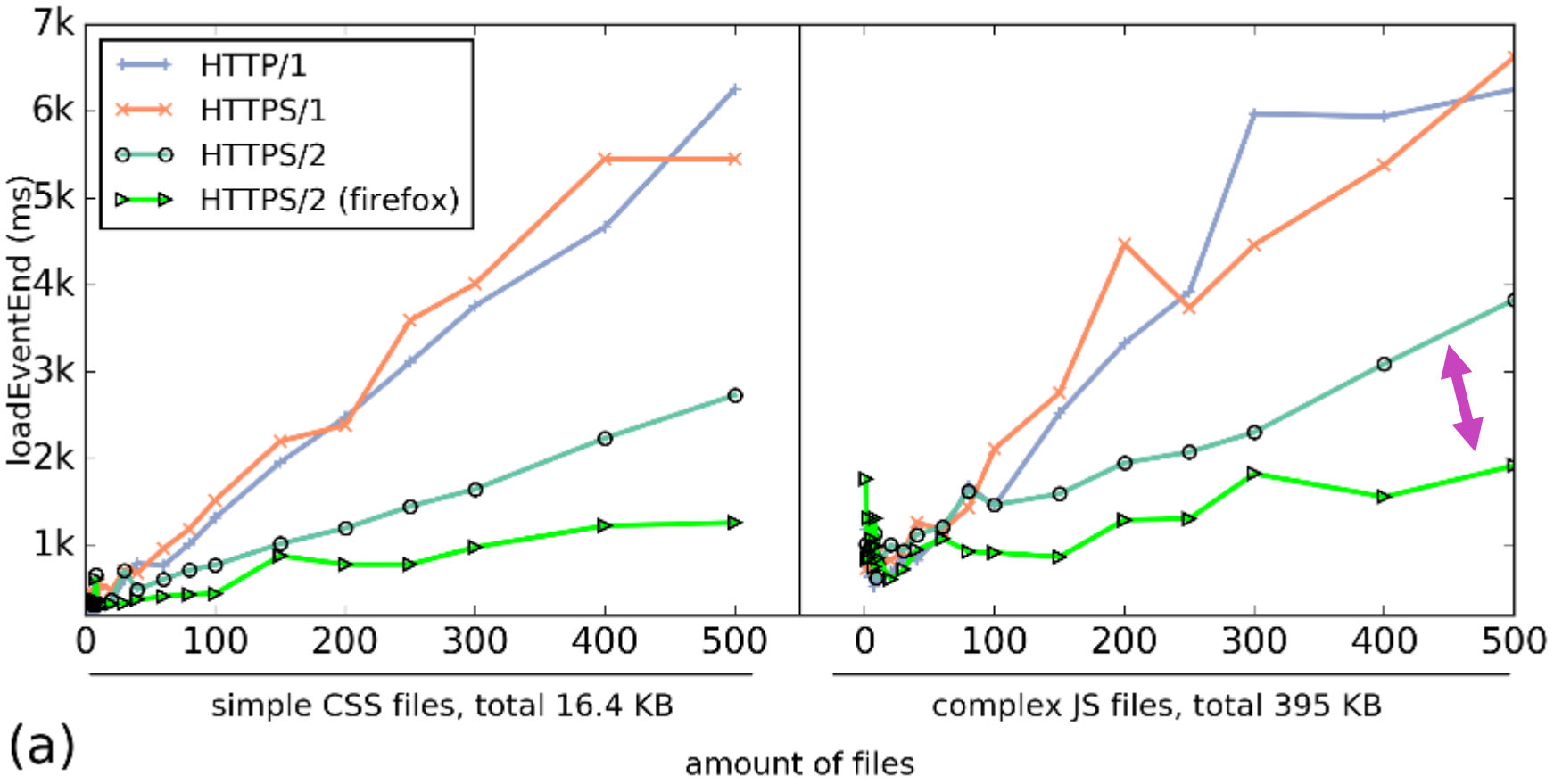
1. Concatenation : 500 CSS/JS



CSS Simple

JS Complex

1. Concatenation : 500 CSS/JS

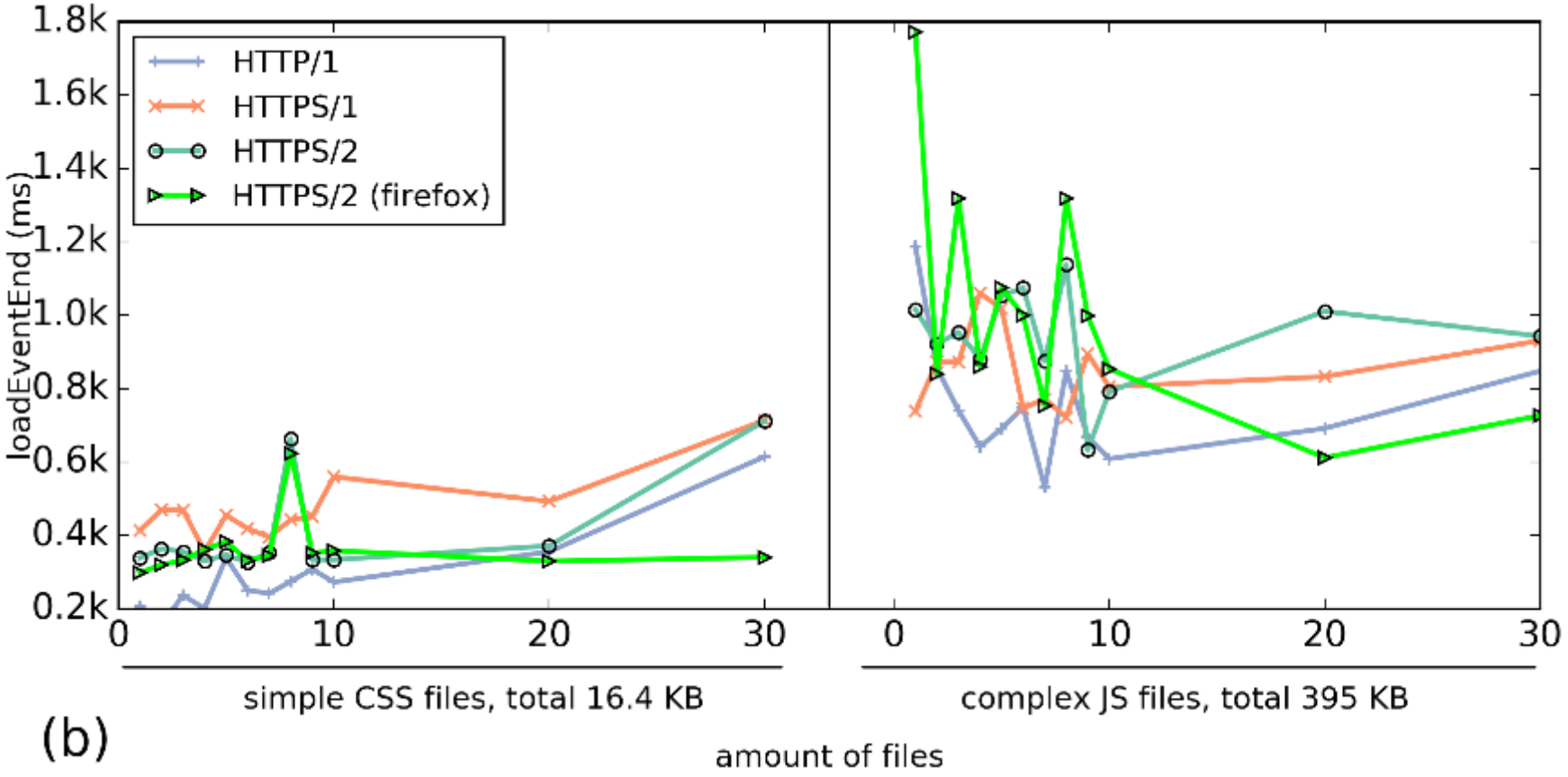


CSS Simple

JS Complex

EDM imec UHASSELT KNOWLEDGE IN ACTION

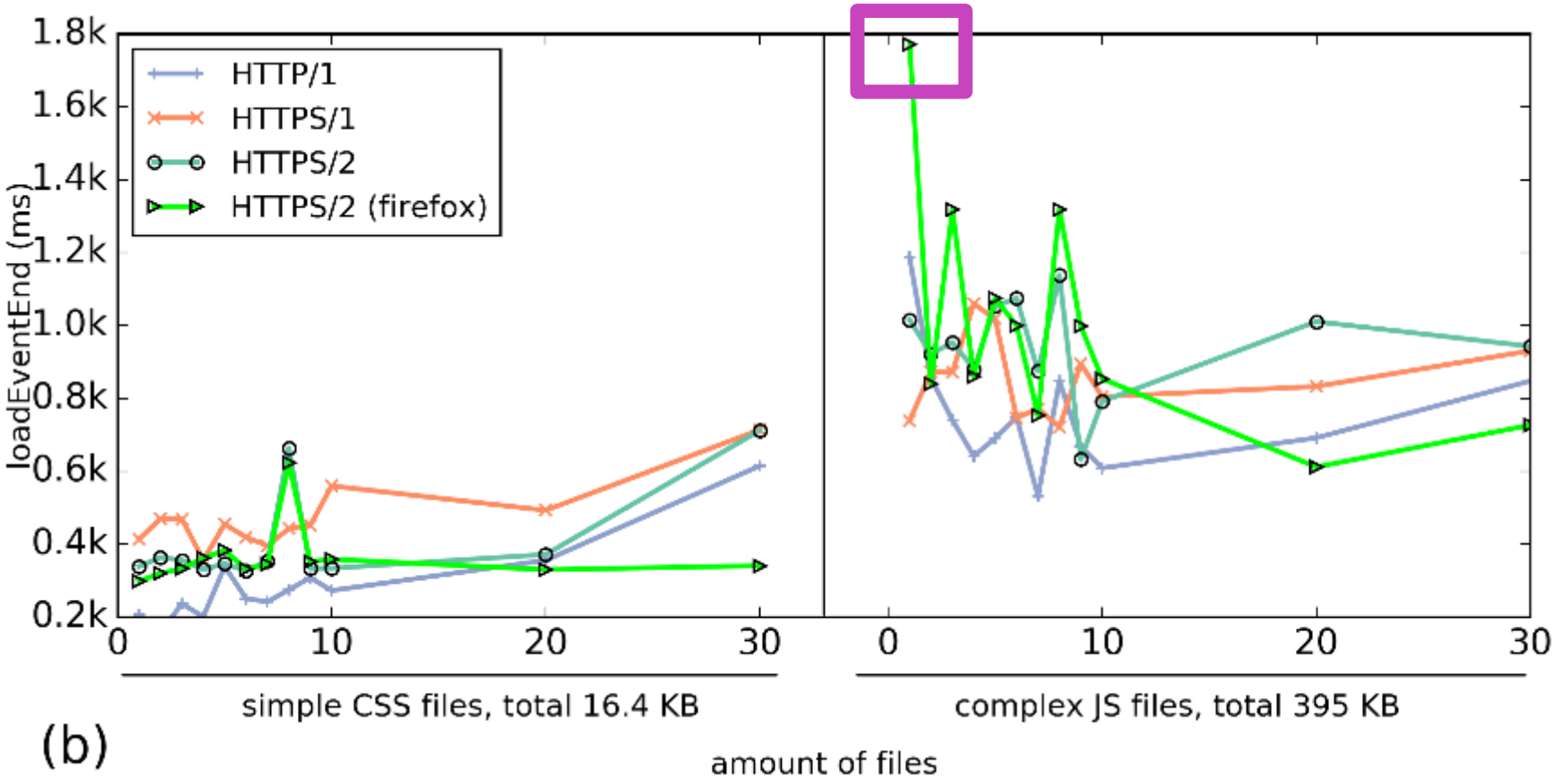
1. Concatenation : 30 CSS/JS



CSS Simple

JS Complex

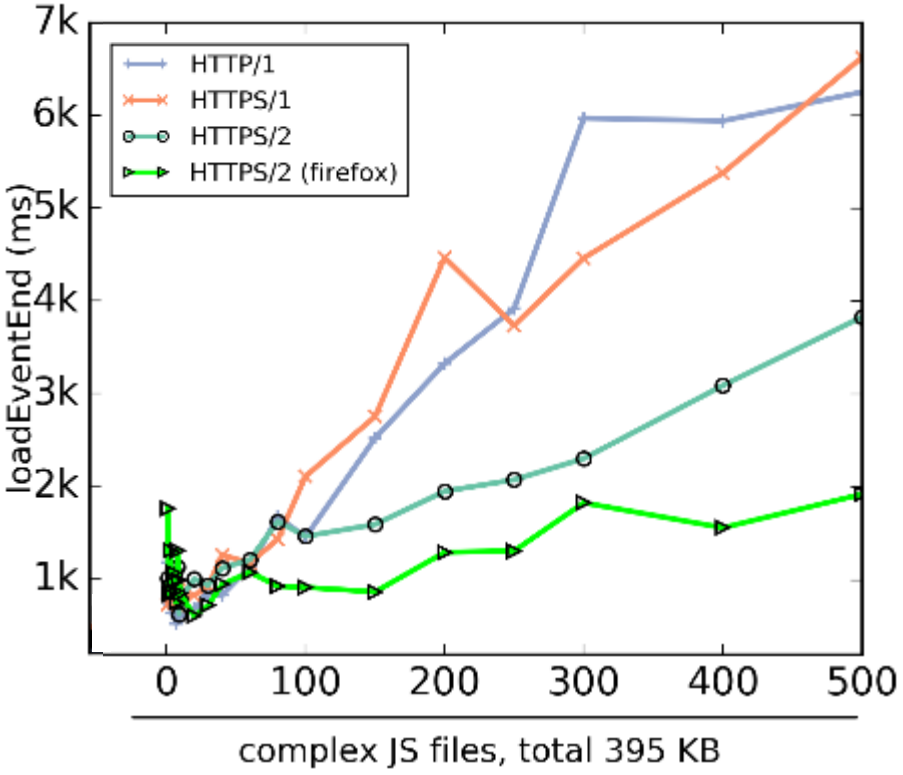
1. Concatenation : 30 CSS/JS

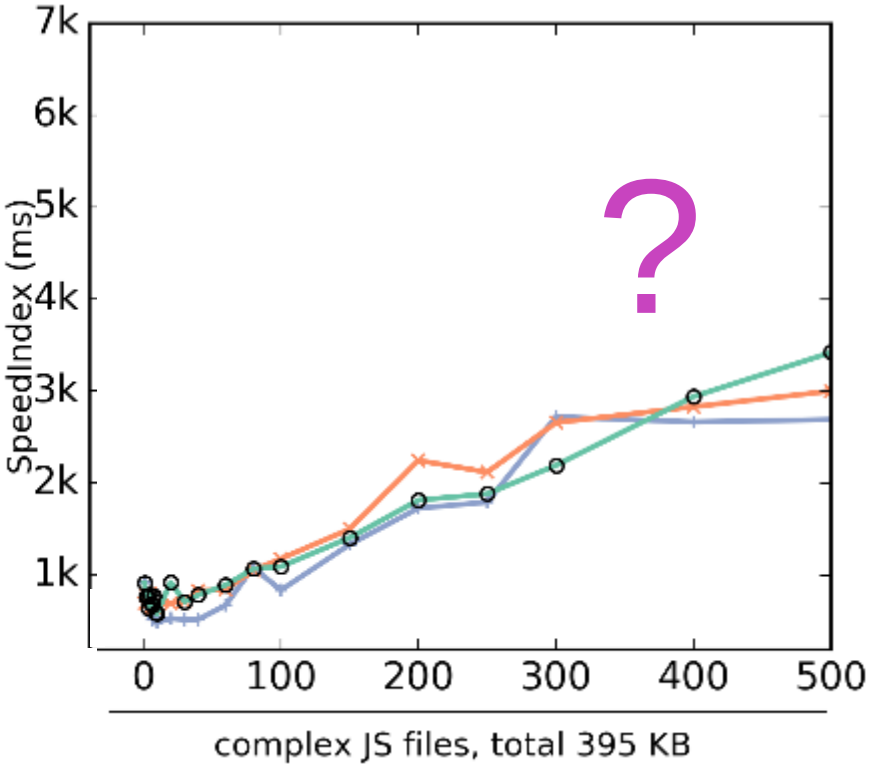


CSS Simple

JS Complex

1. Concatenation : JS SpeedIndex

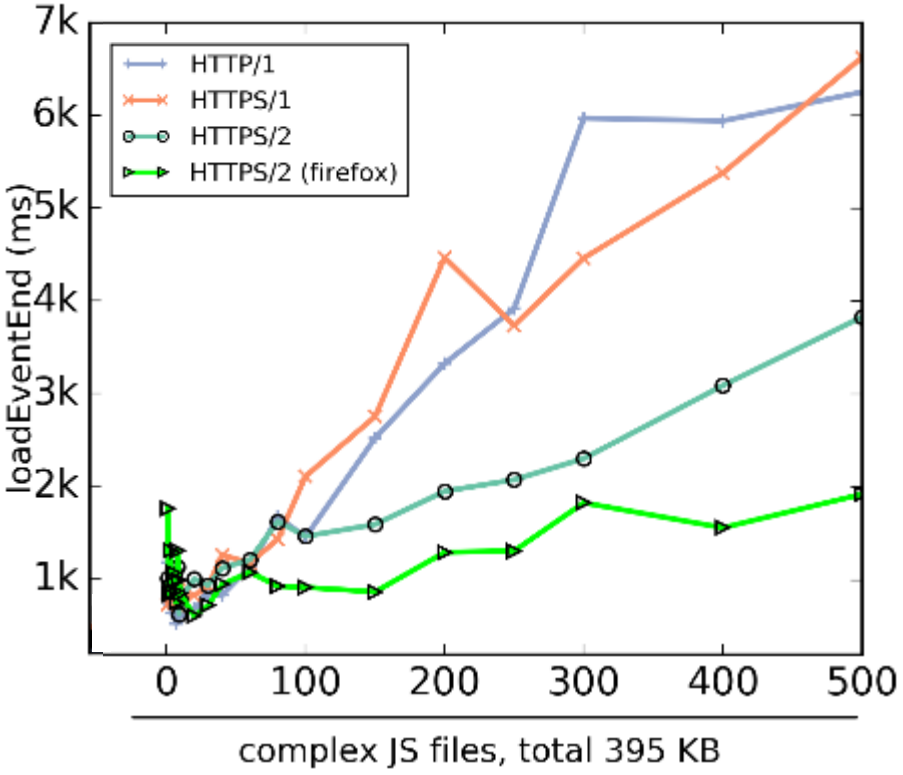


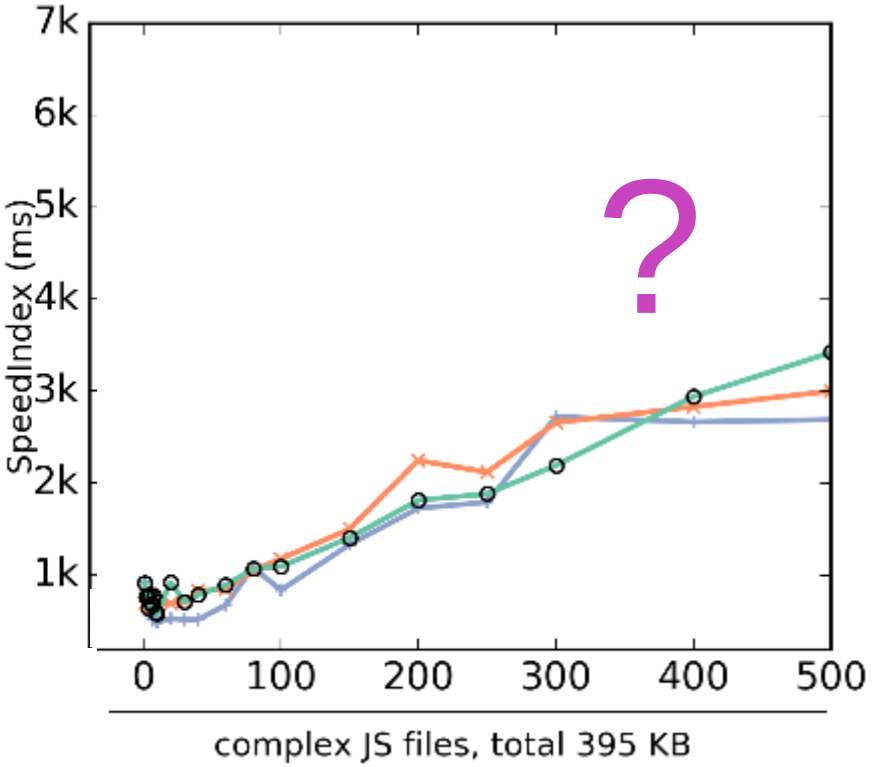


loadEventEnd

SpeedIndex

1. Concatenation : JS SpeedIndex



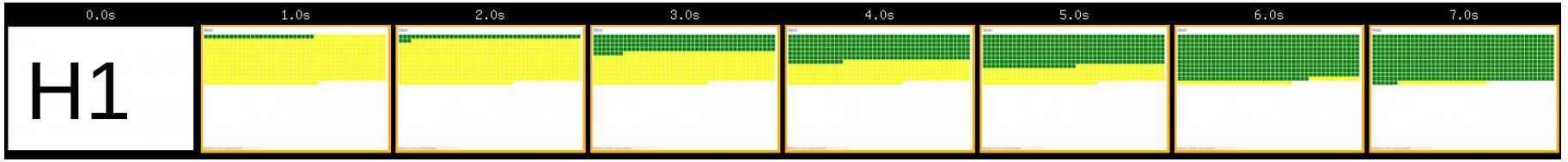


H1 is much lower = rendering starts ASAP

H2 is ~ the same = rendering is delayed

1. Concatenation : JS SpeedIndex

- 500 individual files



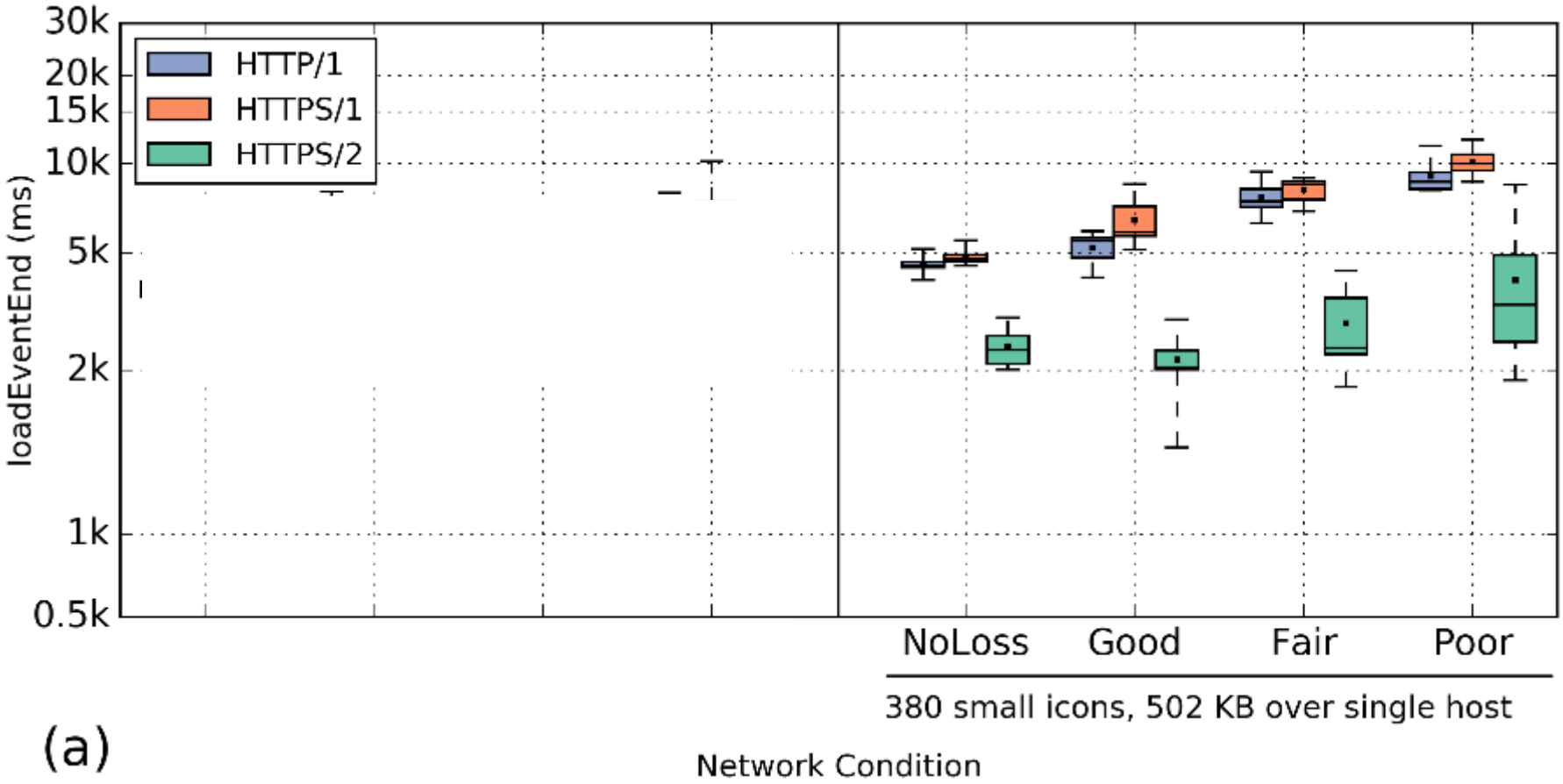
SpeedIndex H1 = 2729
SpeedIndex H2 = 3217

2. Sharding

- Images : 380 small, 42 medium, 30 large
- Sharded over 4 hosts (24 H1 conns, 4 H2)

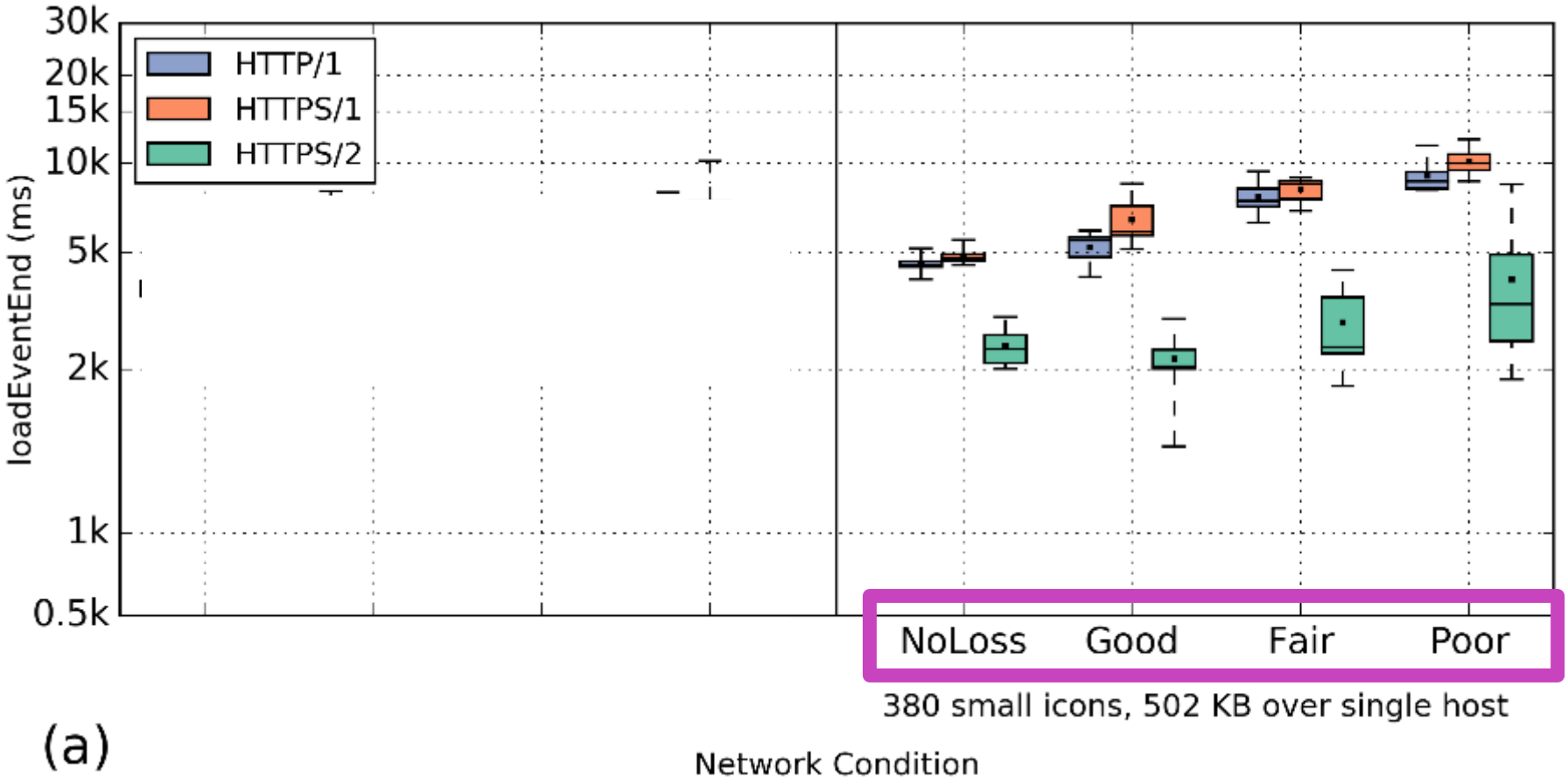
EDM | imec | UHASSELT KNOWLEDGE IN ACTION

2. Sharding : 380 small images



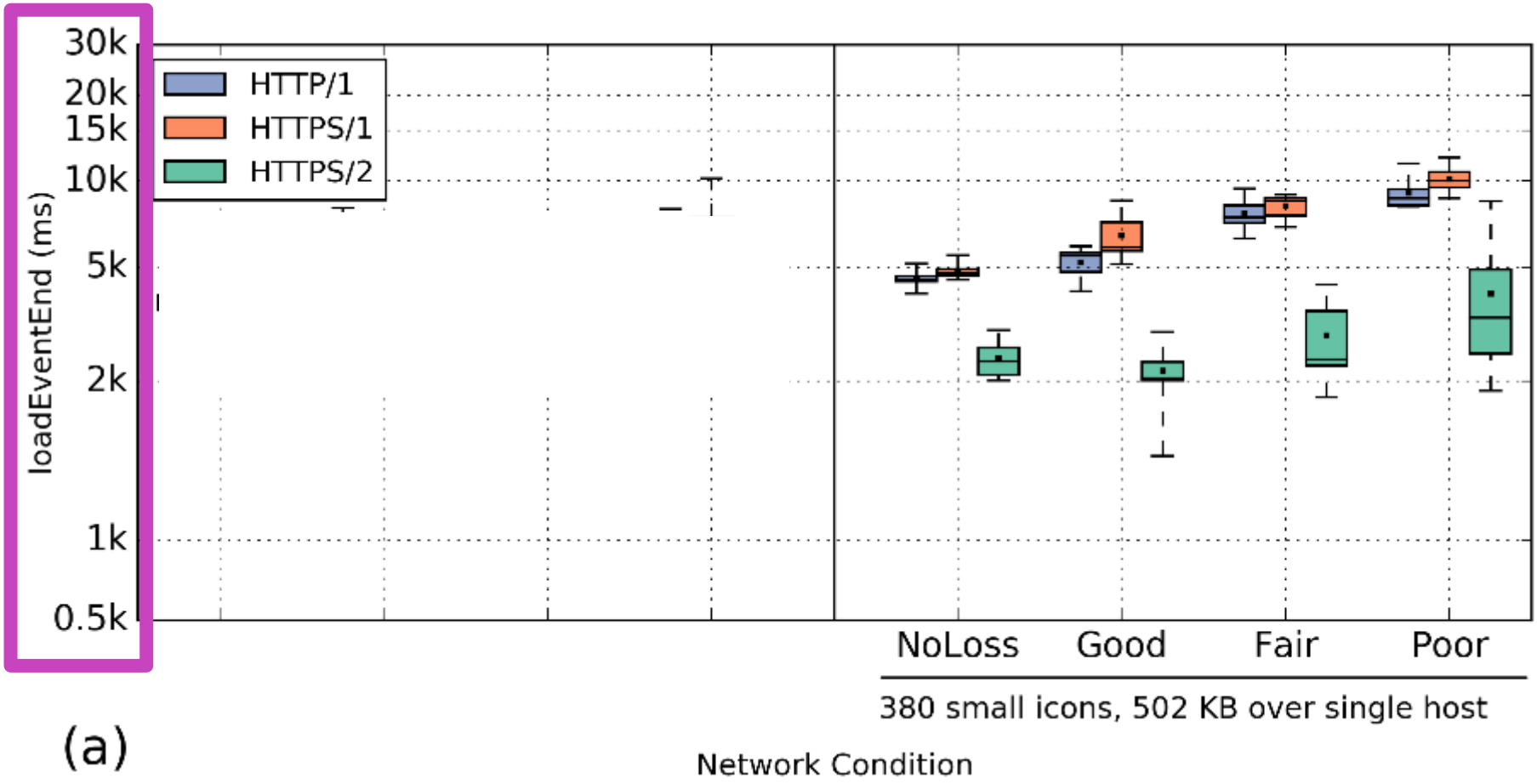
NOT OPTIMIZED

2. Sharding : 380 small images



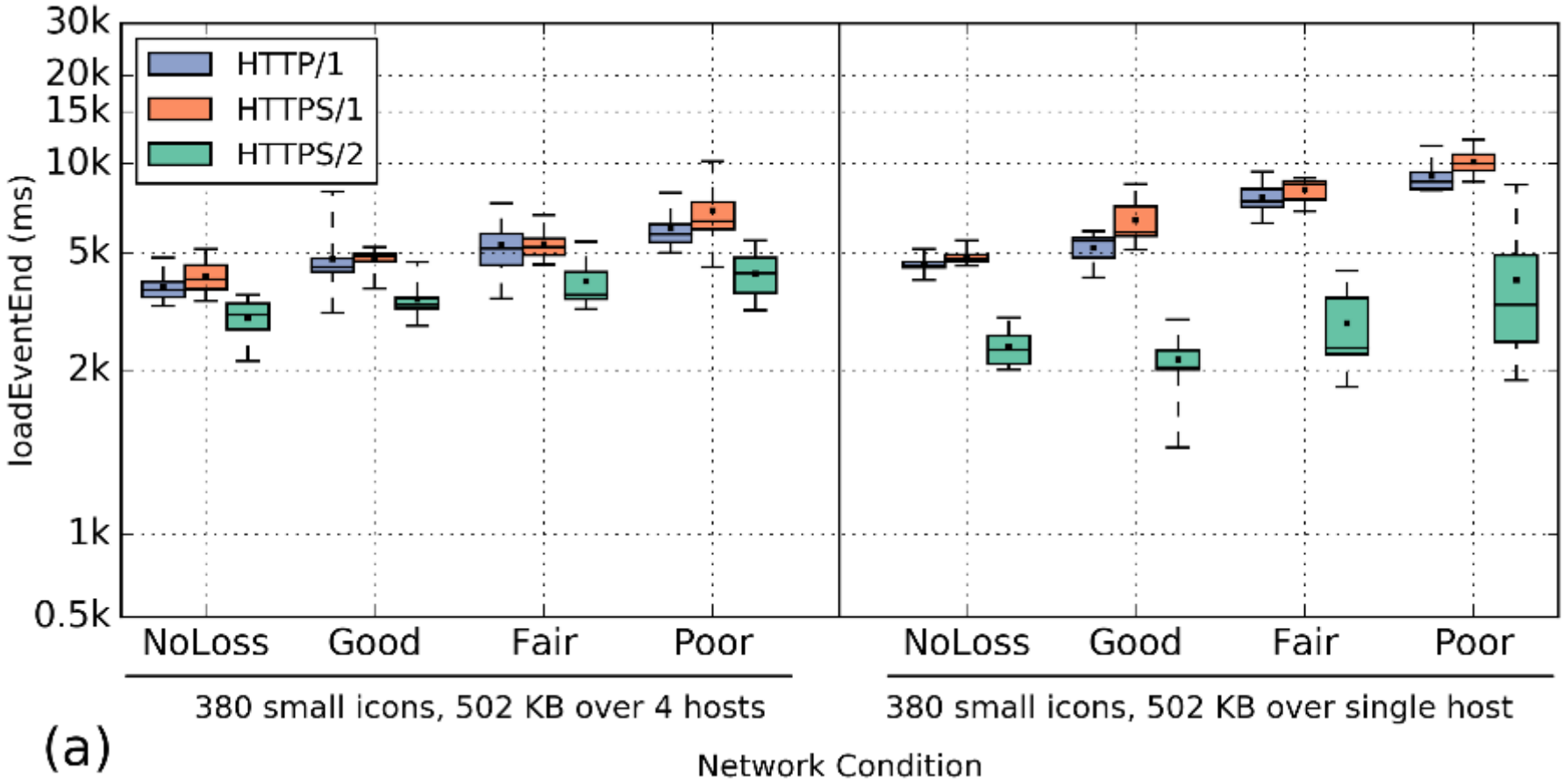
NOT OPTIMIZED

2. Sharding : 380 small images



NOT OPTIMIZED

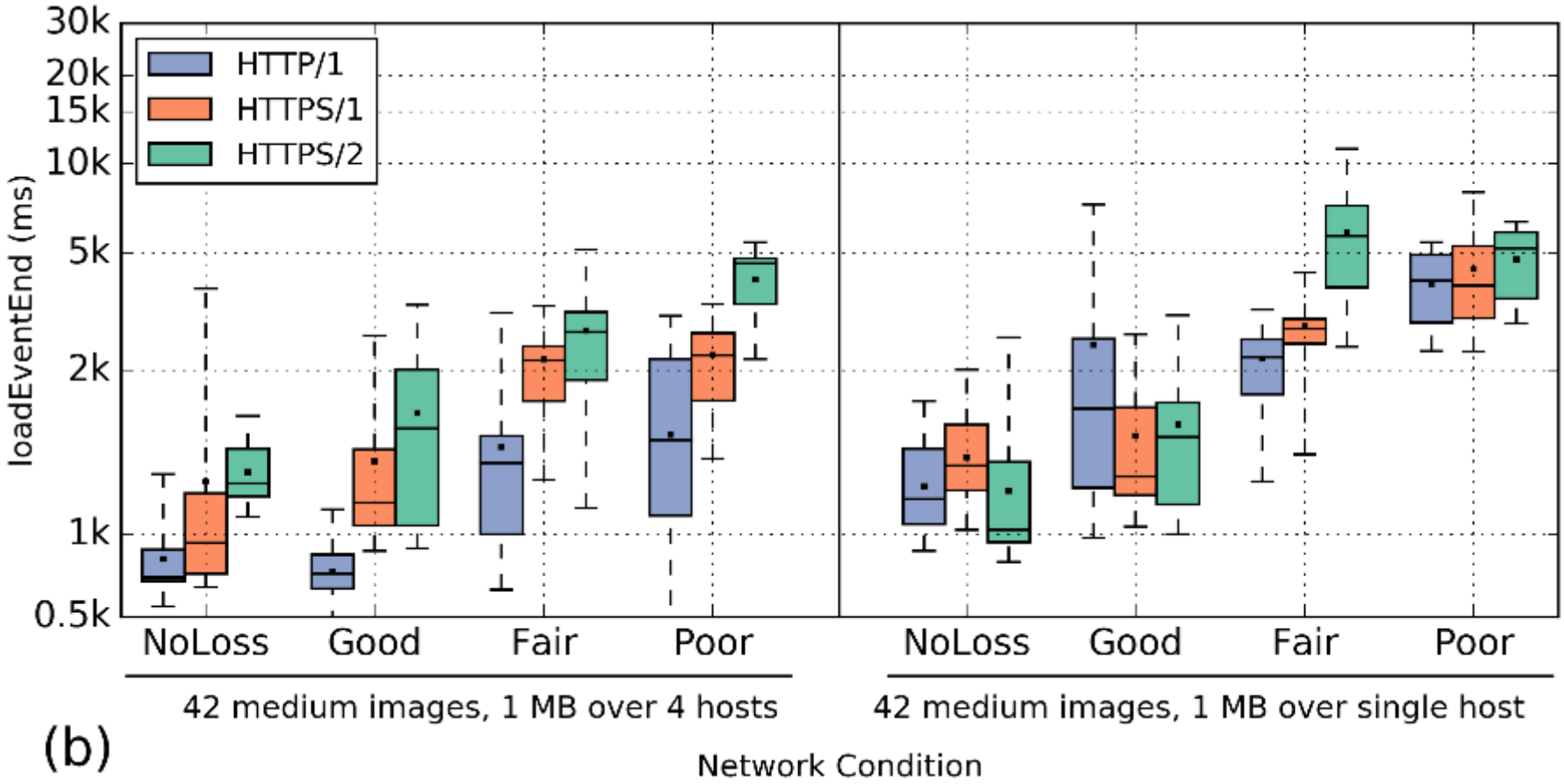
2. Sharding : 380 small images



OPTIMIZED

NOT OPTIMIZED

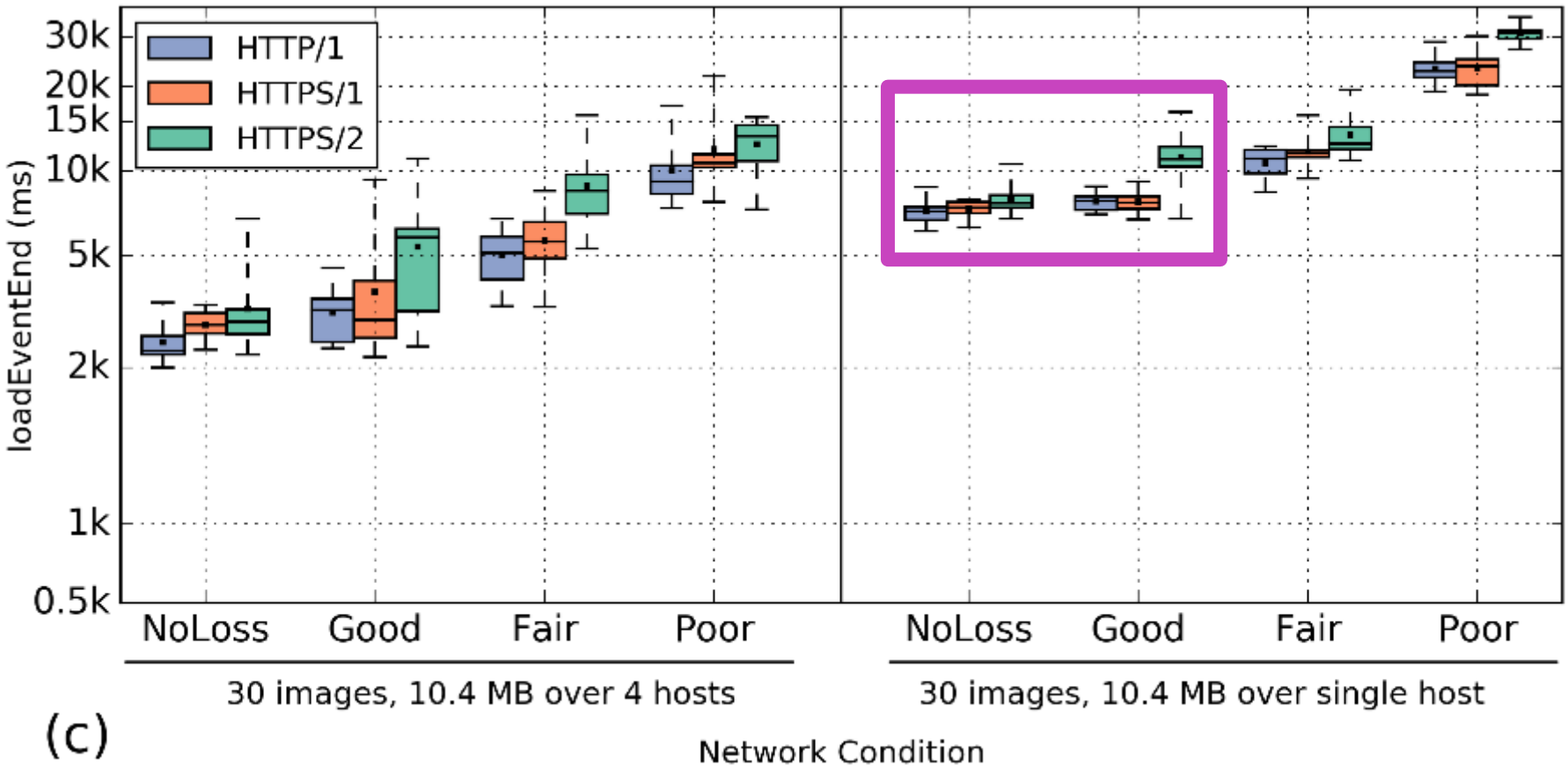
2. Sharding : 42 medium images



OPTIMIZED

NOT OPTIMIZED

2. Sharding : 30 large images

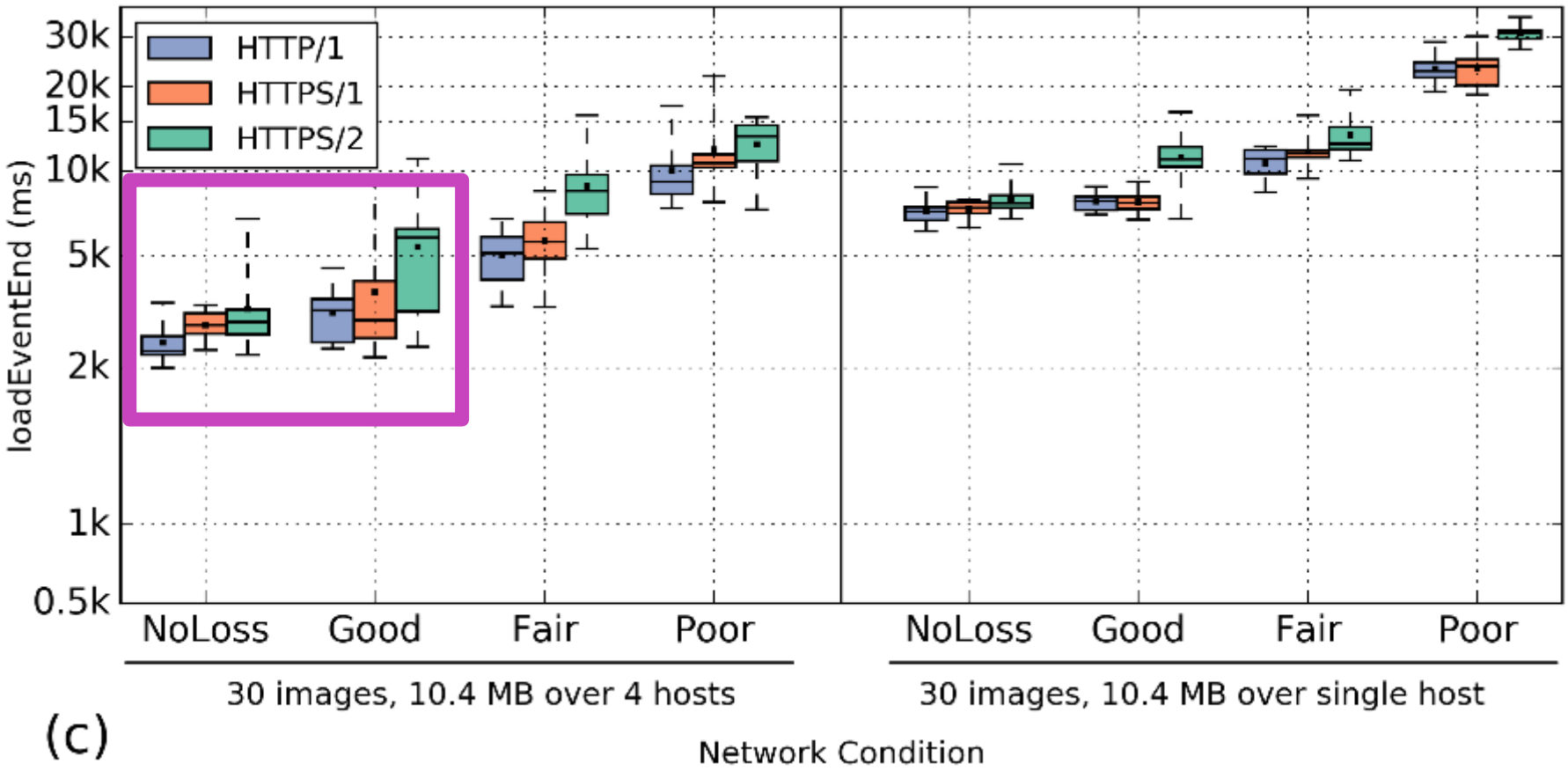


OPTIMIZED

NOT OPTIMIZED

EDM imec UHASSELT KNOWLEDGE IN ACTION

2. Sharding : 30 large images



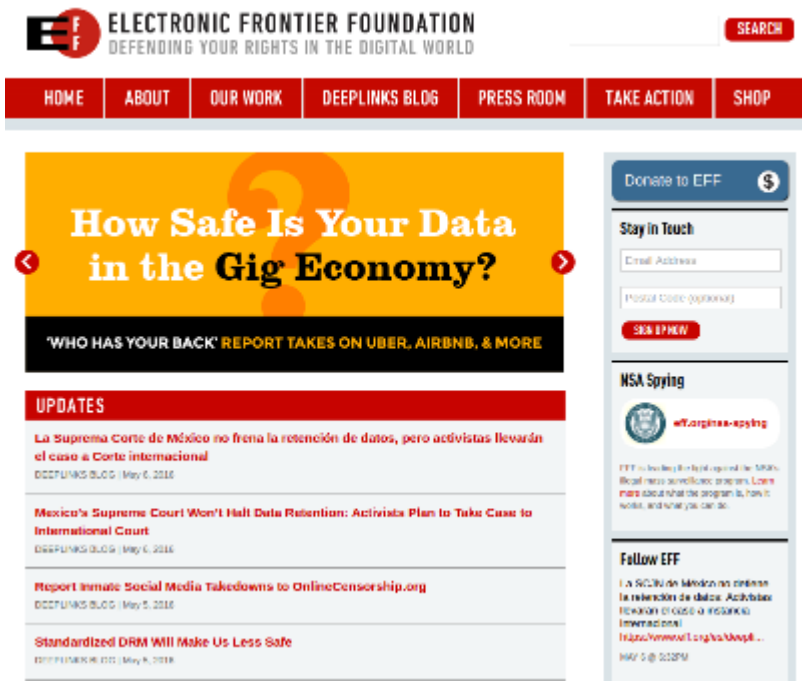
OPTIMIZED

NOT OPTIMIZED

EDM imec UHASSELT KNOWLEDGE IN ACTION

3. Embedding and Push

- Embedding : critical CSS
- Push : various test cases

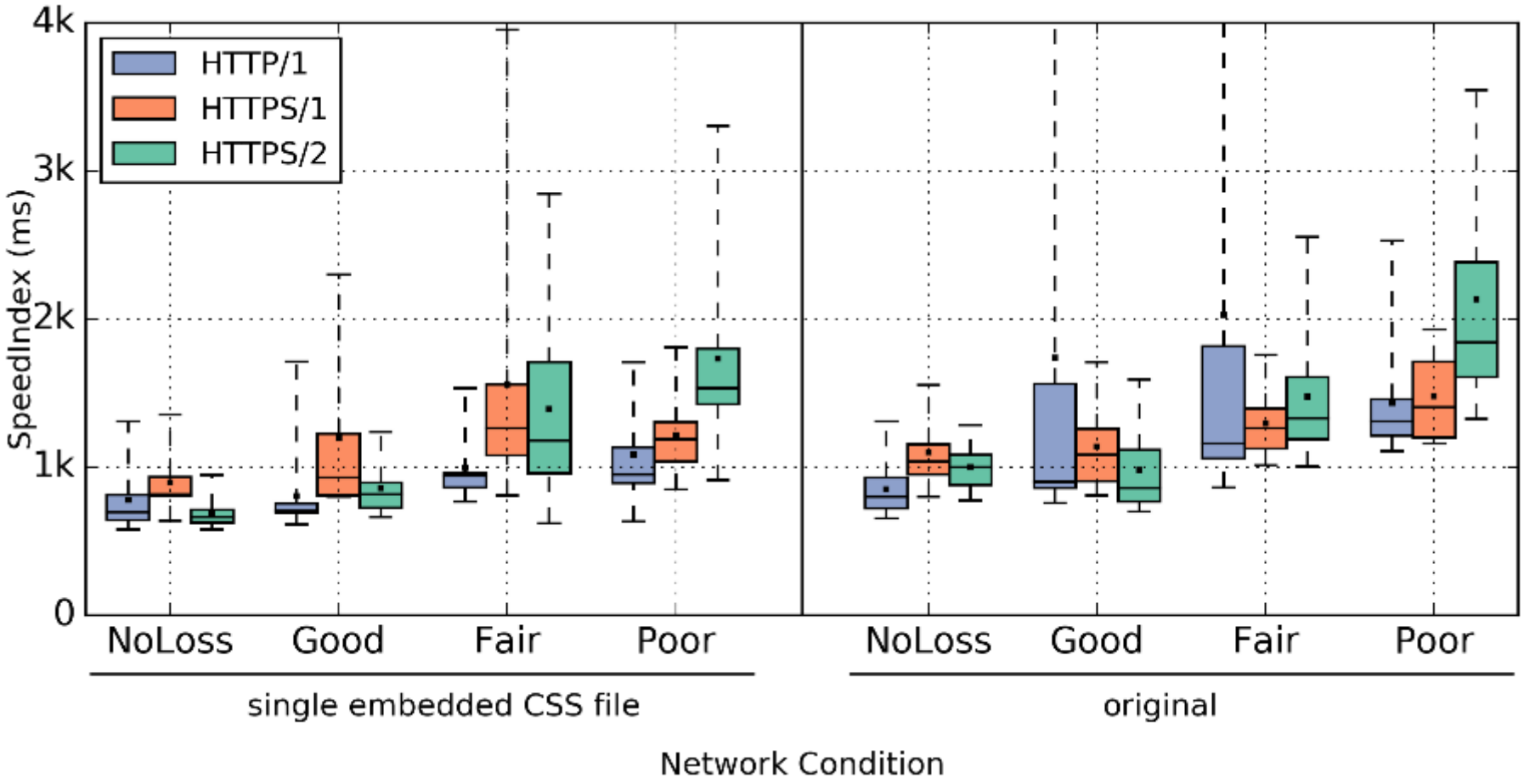


FULL SITE

CRITICAL CSS

https://github.com/bradleyfalzon/h2push-demo

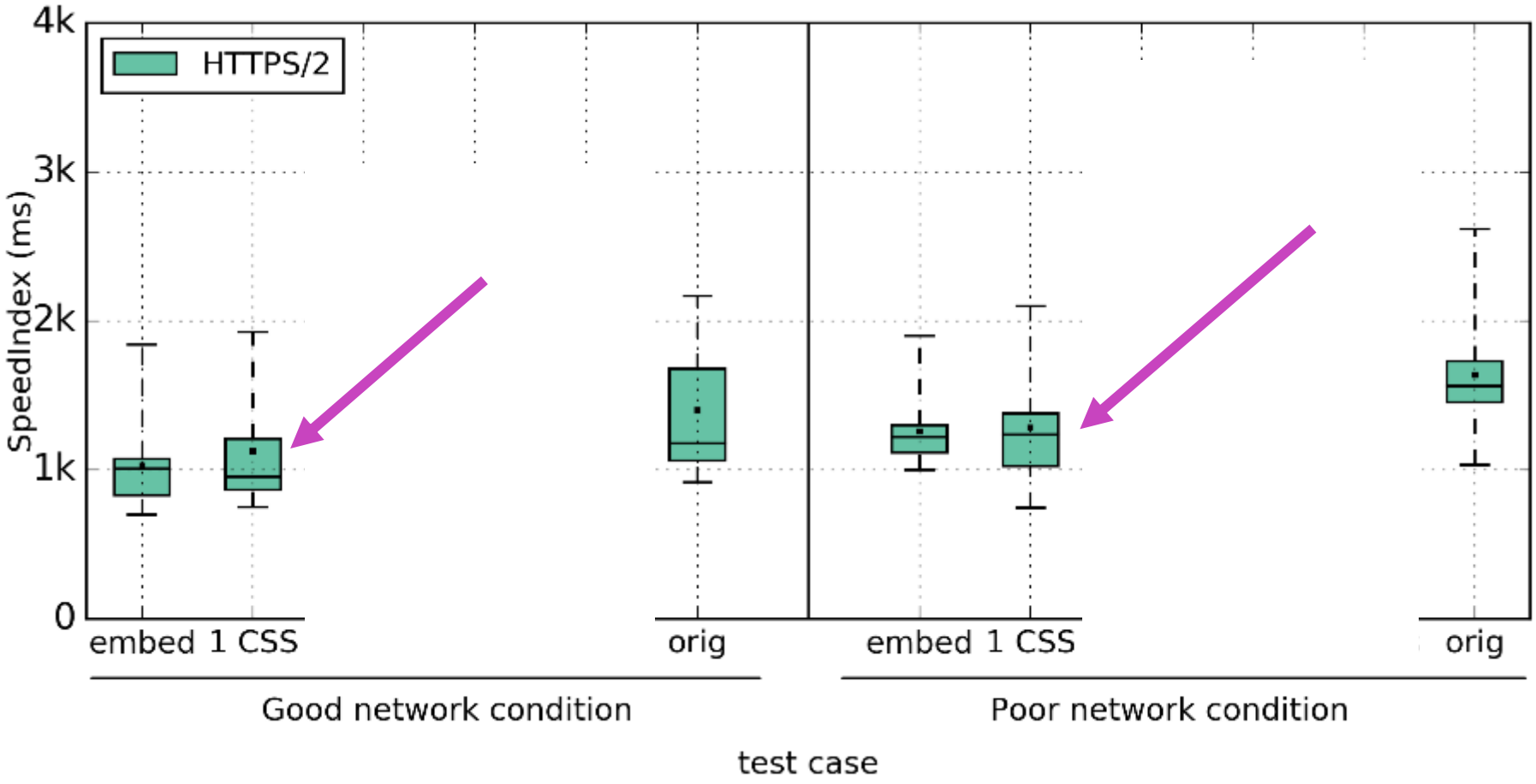
3. Embedding



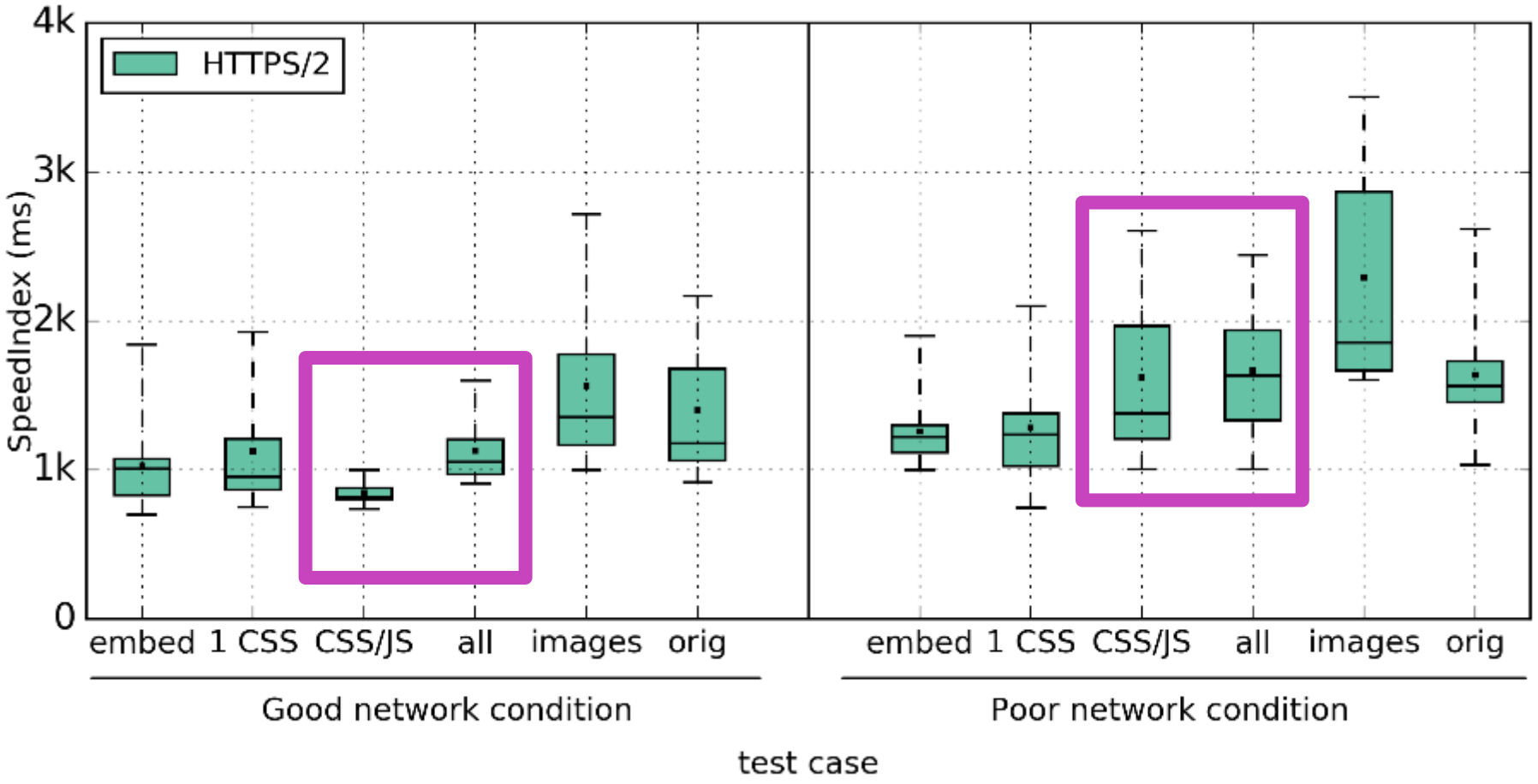
OPTIMIZED

NOT OPTIMIZED

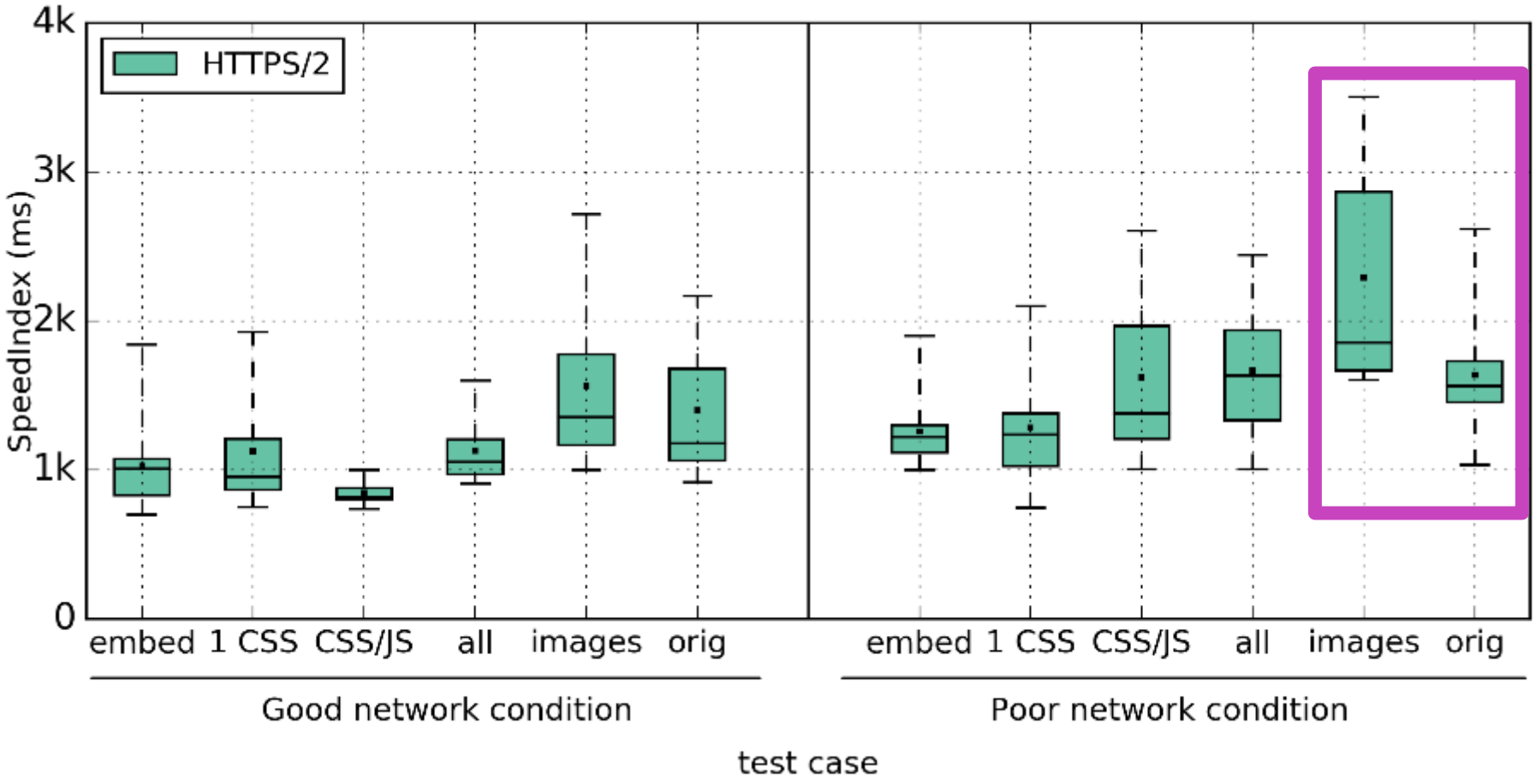
3. Push


HTTPS/2
SpeedIndex (ms)
4k
3k
2k
1k
0
embed 1 CSS
orig
embed 1 CSS
orig
Good network condition
Poor network condition
test case

3. Push

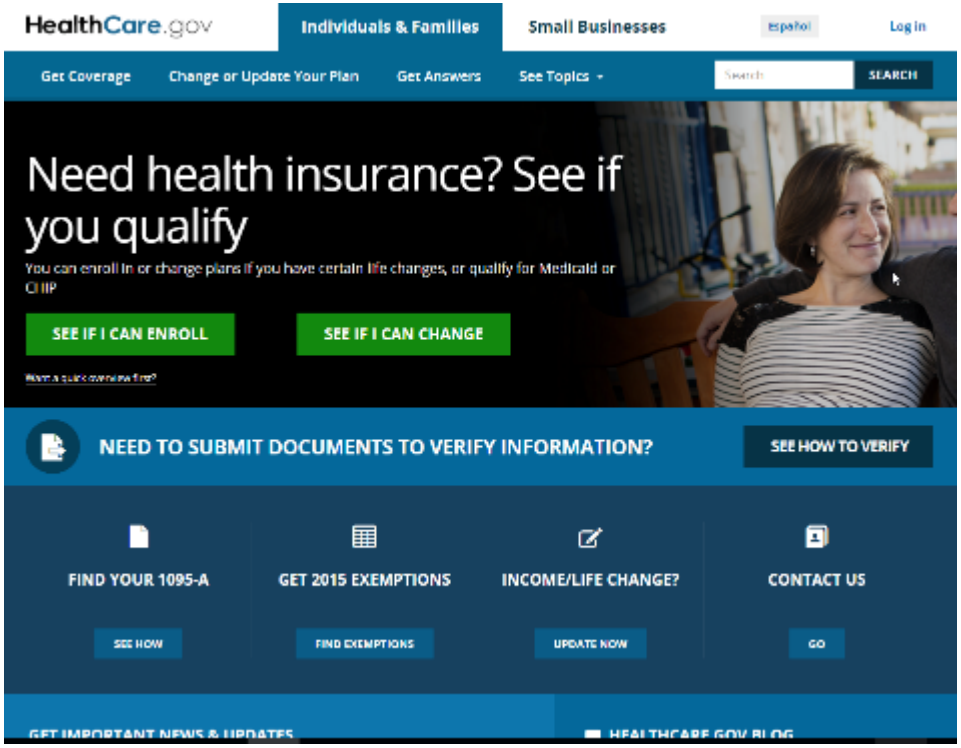

HTTPS/2
SpeedIndex (ms)
4k
3k
2k
1k
0
embed
1 CSS
CSS/JS
all
images
orig
embed
1 CSS
CSS/JS
all
images
orig
Good network condition
Poor network condition
test case

3. Push


HTTPS/2
SpeedIndex (ms)
4k
3k
2k
1k
0
embed
1 CSS
CSS/JS
all
images
orig
embed
1 CSS
CSS/JS
all
images
orig
Good network condition
Poor network condition
test case

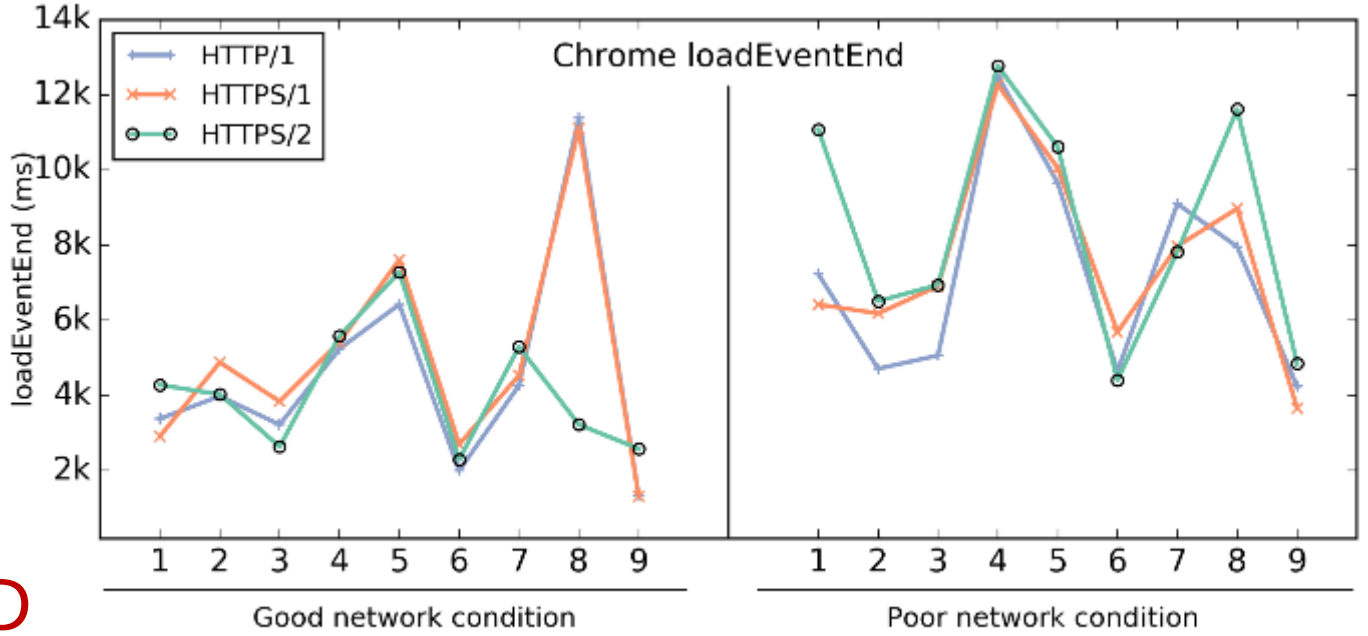
4. Realistic websites

- How would a developer test H2 on his site?
- No sharding for main assets (~H2 should win)



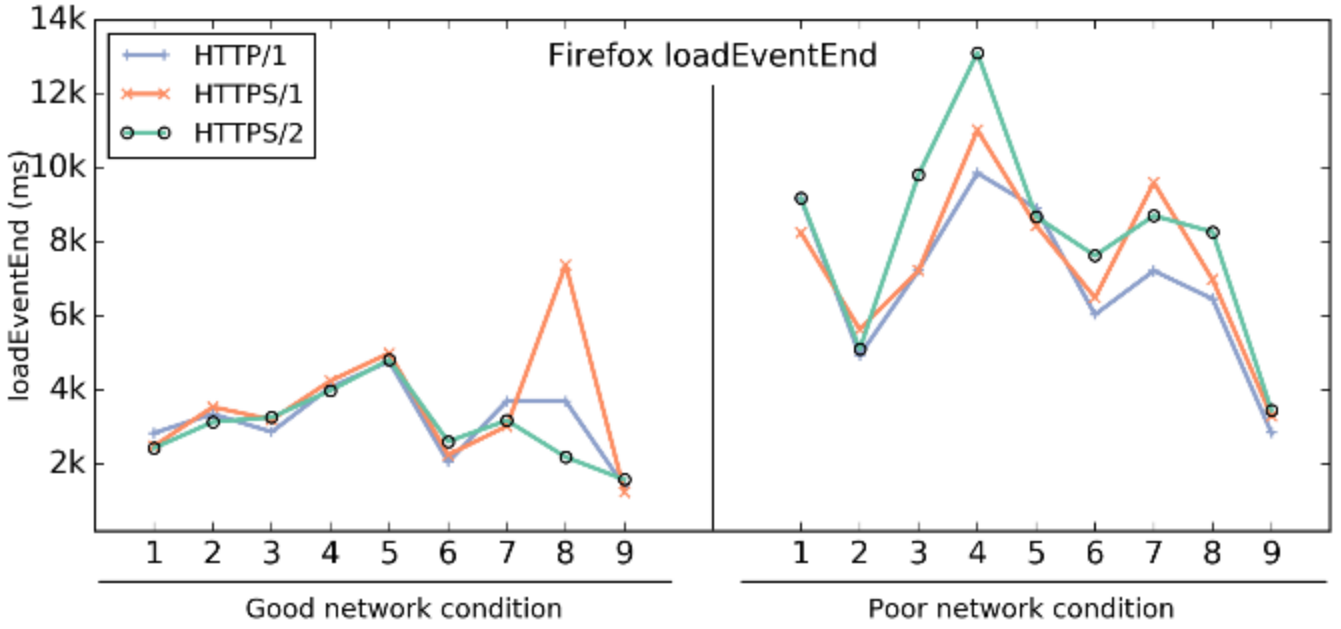
EDM imec UHASSELT KNOWLEDGE IN ACTION

4.



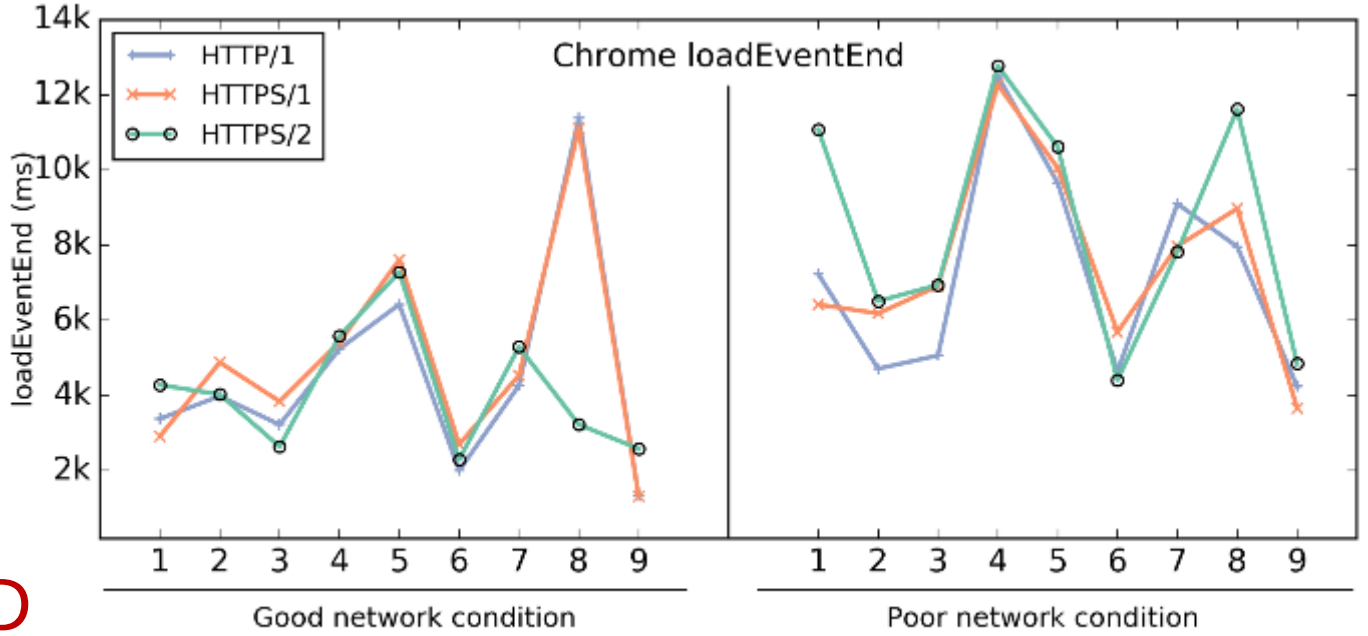
GOOD

POOR



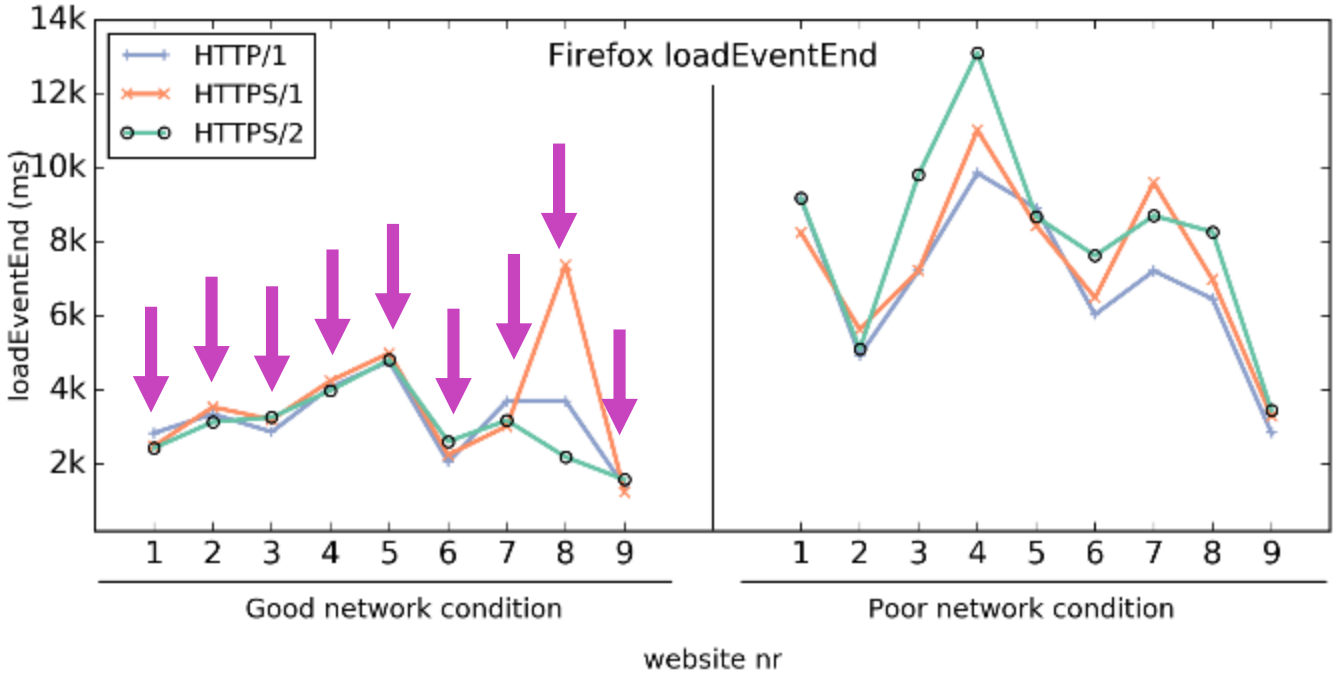
UHASSELT
KNOWLEDGE IN ACTION

4.



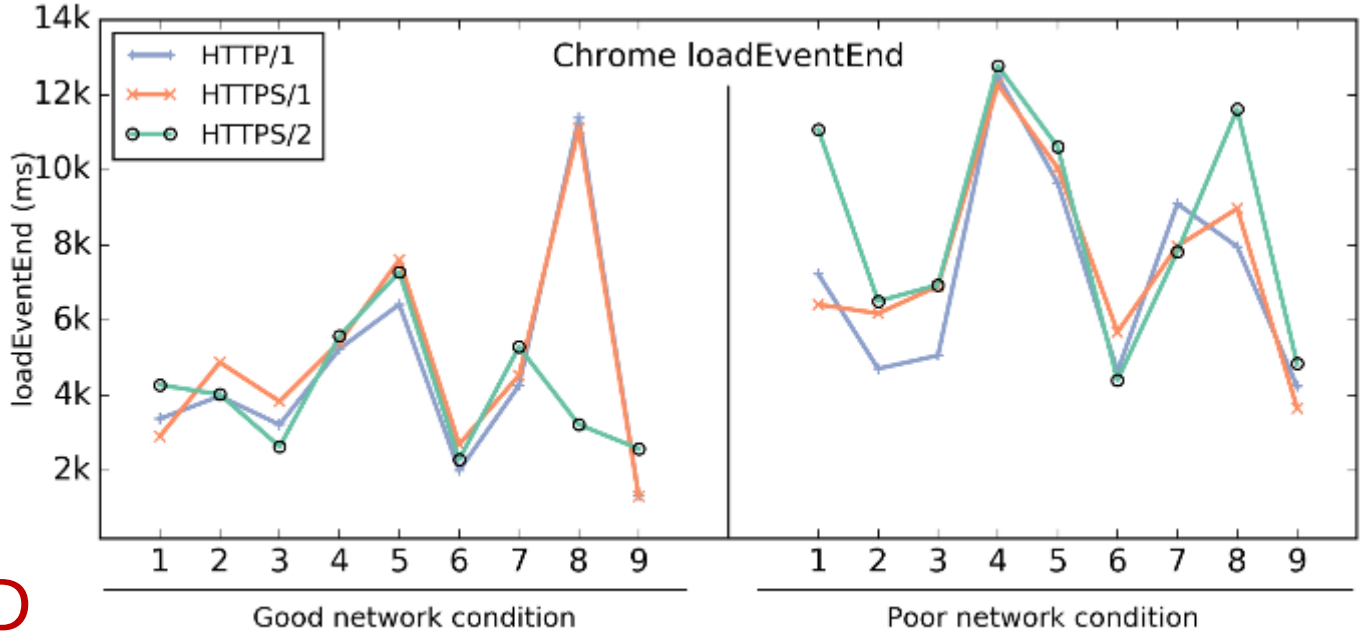
GOOD

POOR



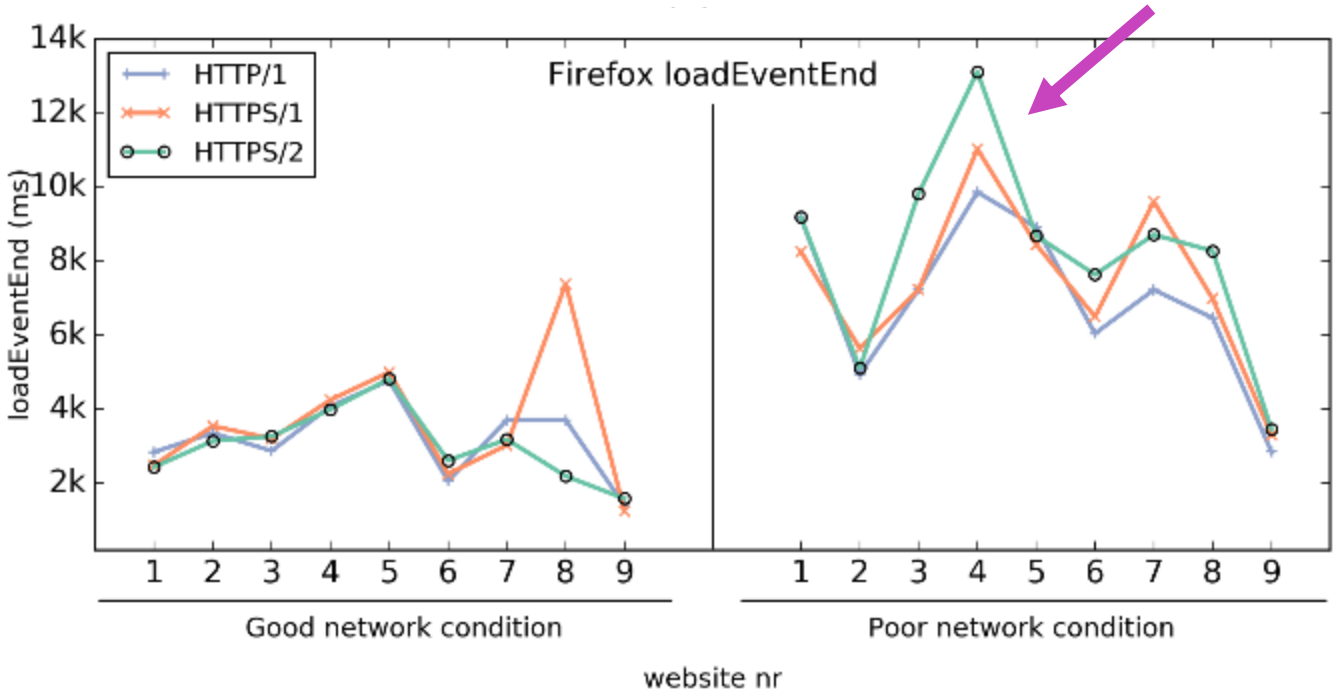
UHASSELT
KNOWLEDGE IN ACTION

4.



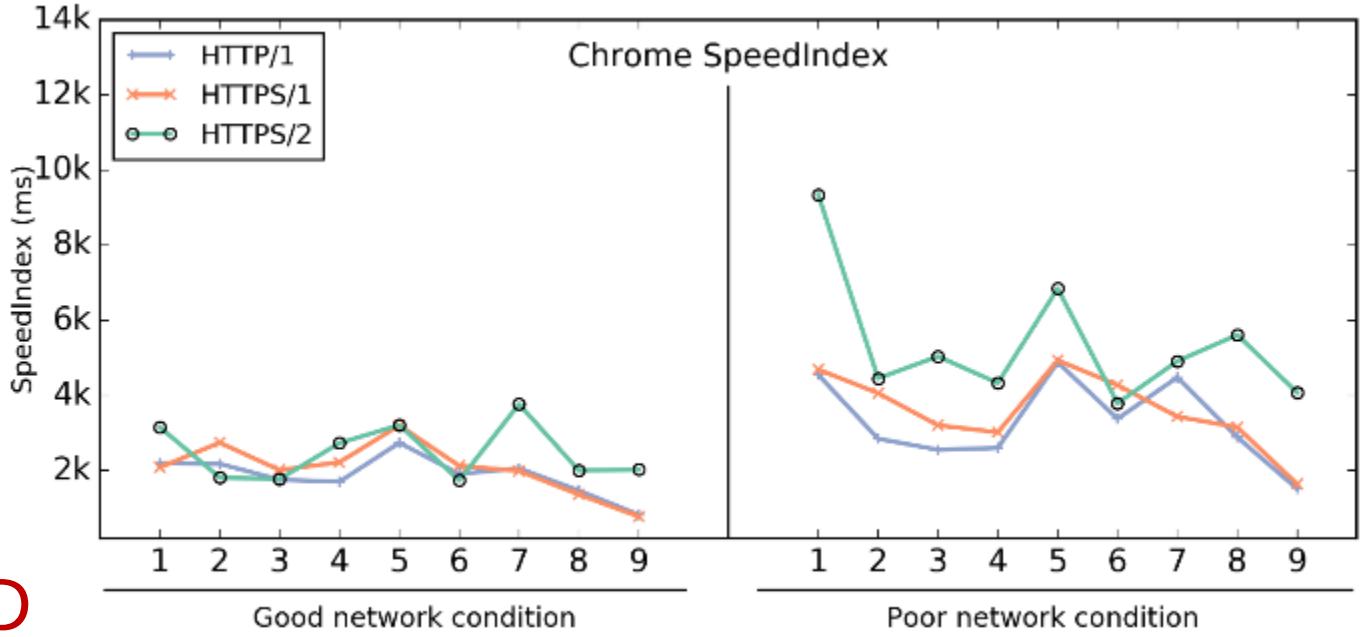
GOOD

POOR



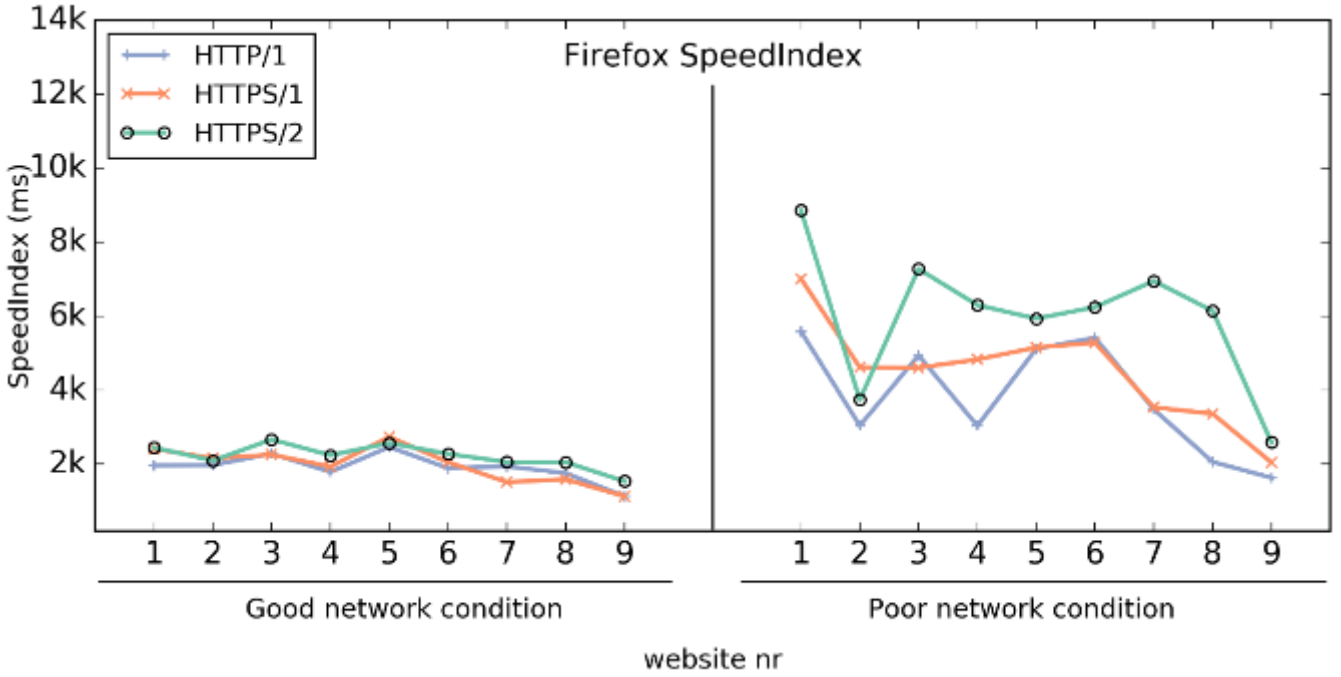
UHASSELT
KNOWLEDGE IN ACTION

4.



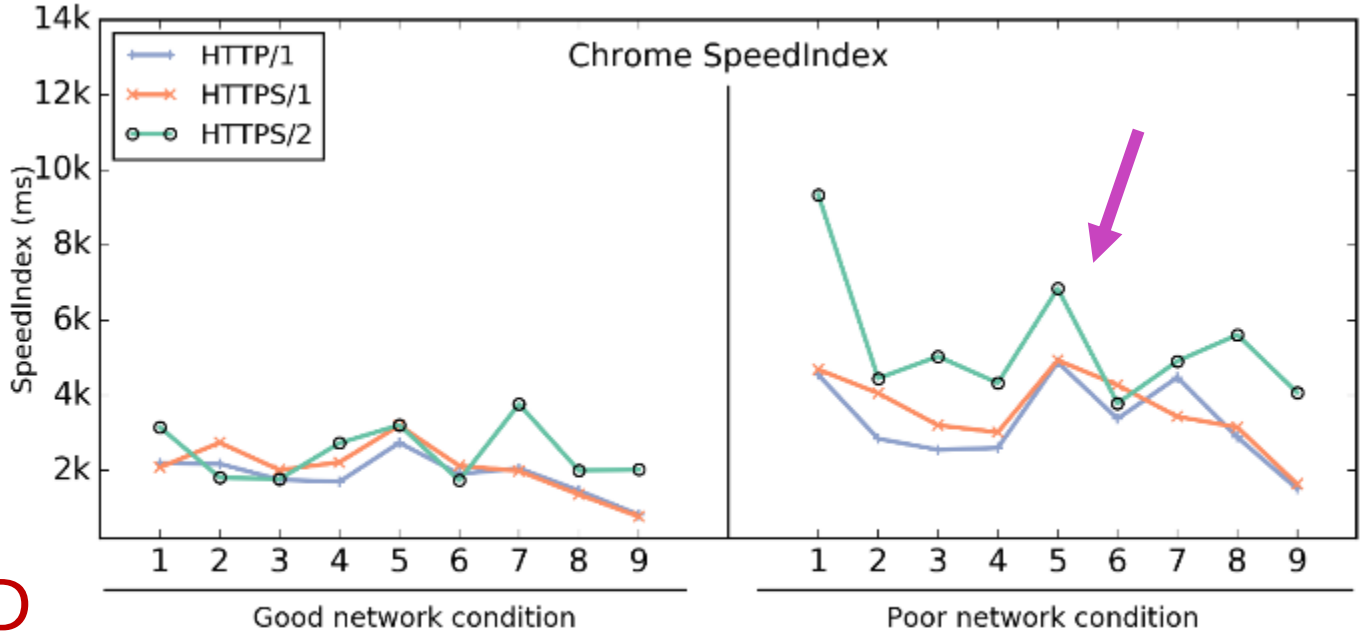
GOOD

POOR



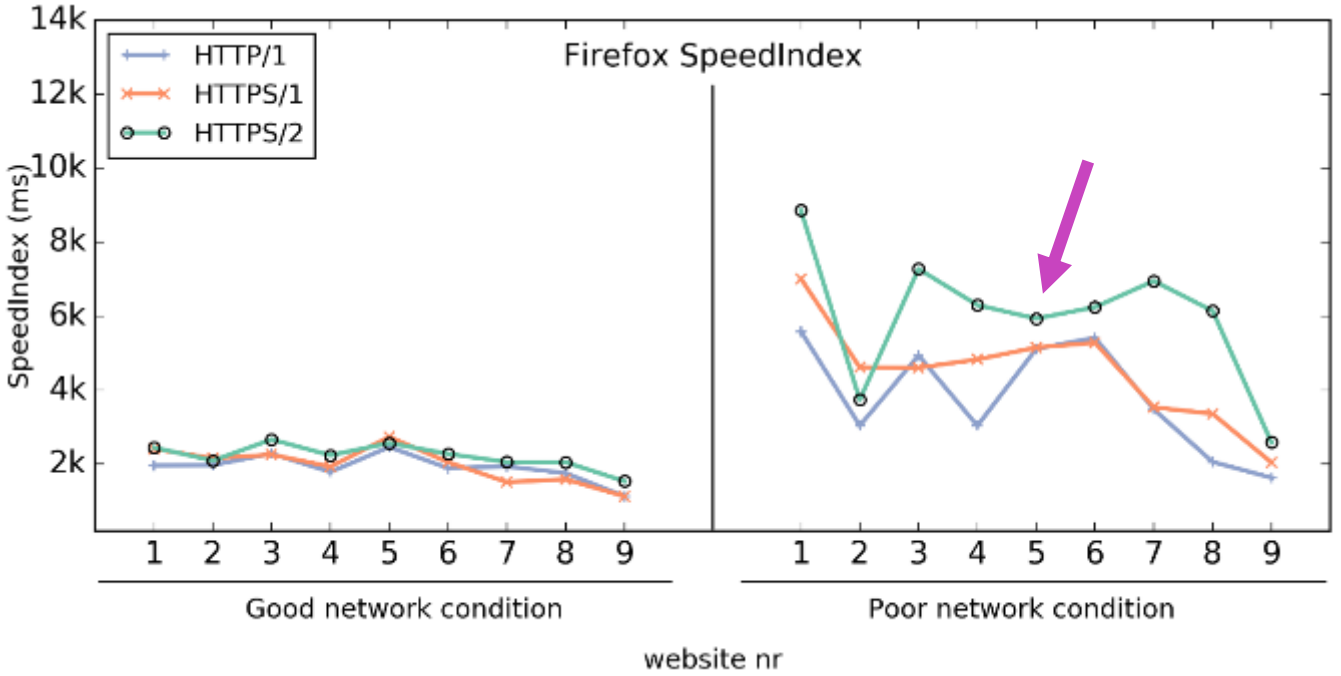
UHASSELT
KNOWLEDGE IN ACTION

4.



GOOD

POOR



UHASSELT
KNOWLEDGE IN ACTION

Conclusions : HTTP/2

- Better for large amounts of small files
- Worse for larger files

- Suffers from bad networks but so does HTTP/1

- For most "average" testcases, performance is similar between H1 and H2, with (much) less overhead (# connections) from H2

Conclusions : HTTP/1 Best Practices

- Concatenation: implementation-limited
- Sharding: protocol-inherent
- Embedding: can be replaced by Push

- Most "average" H1 websites can be transferred to H2 without much work ☺

Conclusions : Outlook

- More fine-grained caching (web components, Single Page Apps, …)
- Bigger websites remain difficult (4k images, VR/gaming, …)

- QUIC is already being standardized!

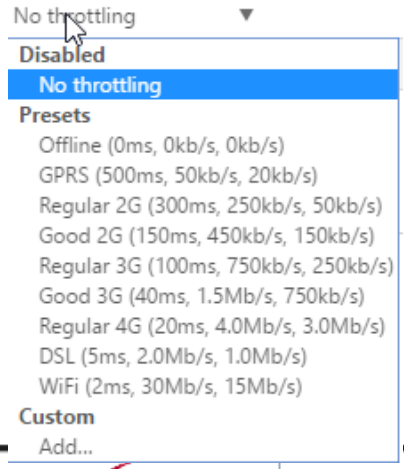
Ask me about:

- Cellular network emulation
- QUIC protocol
- TCP slow start and embedding/push
- HPACK header compression
- Browser rendering behavior
- Website optimization anecdotes
- Docker / The Speeder Framework
- HTTPS vs HTTP cleartext
- Other uses of HTTP/2 Push

Cellular network emulation

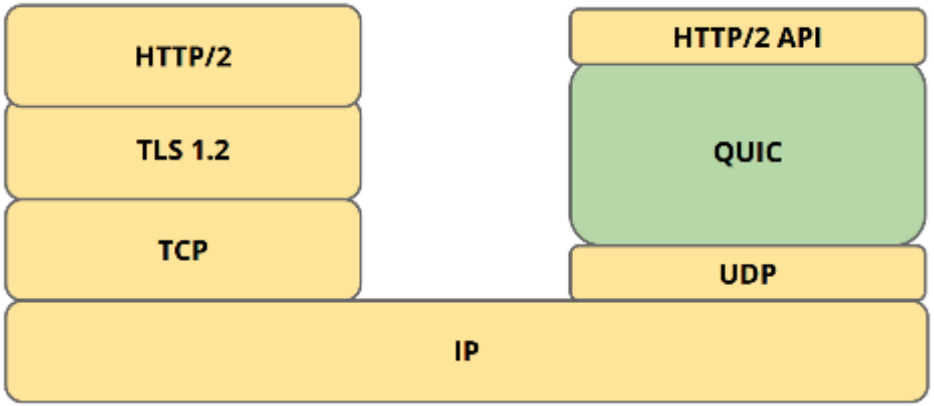
- "Http/2 performance in cellular networks" (Goel et al., 2016) – MobiCOM
- Traces of real networks @ CDN edge
- Modulate `TC NETEM` parameters every 70 ms
- https://github.com/akamai/cell-emulation-util

- Fixed model based on Chrome Devtools
  - Add loss
  - Typically clearer but similar trends



QUIC protocol

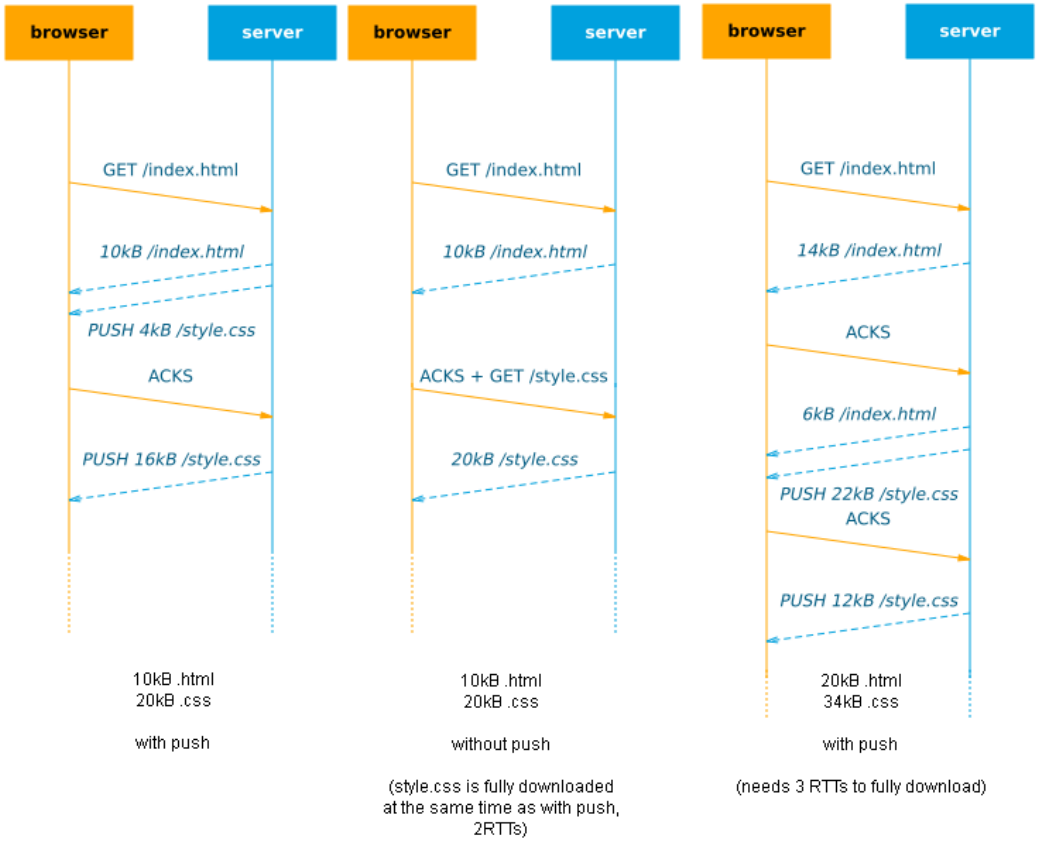
- QUIC = Quick UDP Internet Connections
  - "Re-implement TCP in application layer" ~TCP 2.0
  - Removes TCP HOL-blocking and adds smarter flow-control / less sensitive to loss
  - FEC, Connection UUID/handoffs, multipath, …



- https://ma.ttias.be/googles-quic-protocol-moving-web-tcp-udp/

TCP slow start and embedding/push

- TCP slow start uses congestion window
  - Initial value is ~14KB in modern linux
  - Grows when data is ACK'ed (takes roundtrip)
- If data > 14KB limit, additional RTTs needed!



HPACK header compression

- HTTP headers are often the same
  - Especially important for large cookies
- HPACK uses dynamic dictionary encoding
  - Learns during connection, builds dict at both sides

Table 4: Total bytes sent by Google Chrome (~ HTTP headers) and ratio to total page size. For many small files, the HTTP header overhead is significant.

File count	Protocol	1 host BytesOut	4 hosts BytesOut	Total page size	4 hosts % of total page size
42 files	h2s	649	1362	1075000	0.1%
	h1s	2786	3346	1075000	0.3%
400 files	h2s	29580	38680	610000	6%
	h1s	165300	177600	610000	19%

Browser rendering behavior

- "Render blocking"
  - CSS and sync JS : could change DOM/CSSOM!
- Mitigation: Place on bottom of <body>

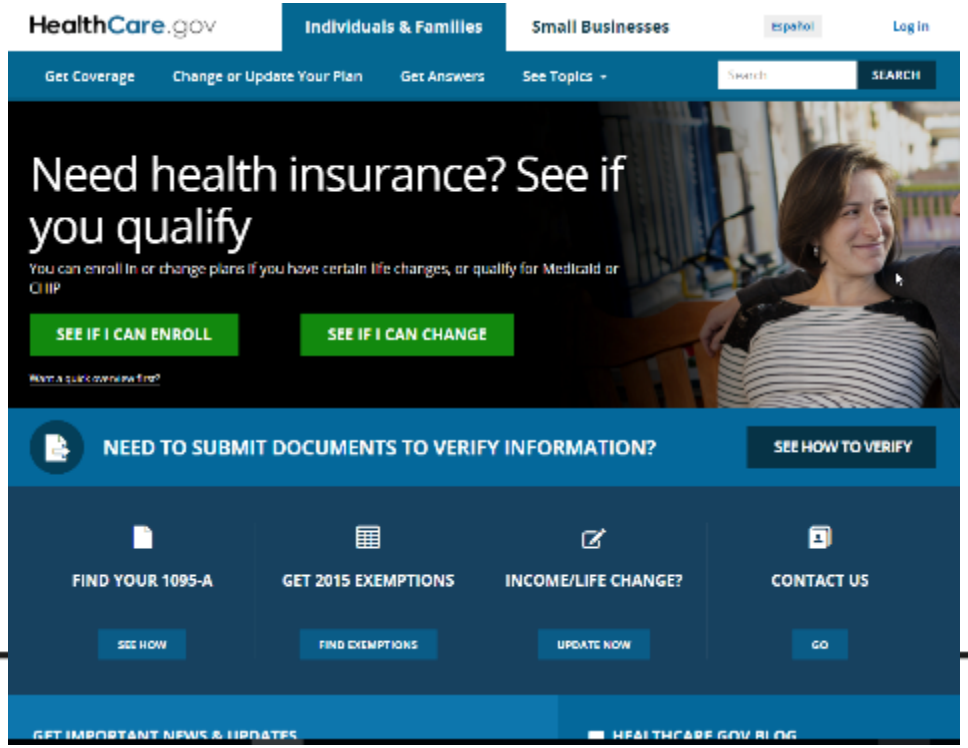
Table 3: Observed browser render-blocking behavior. Chrome blocks on too much, while Firefox doesn't seem to block on anything.

	Expected <head>	**Chrome** <head>	**Firefox** <head>
CSS	blocking	blocking	*progressive*
JavaScript	blocking	*progressive*	*progressive*

	Expected bottom	**Chrome** bottom	**Firefox** bottom
CSS	progressive	*blocking*	progressive
JavaScript	progressive	progressive	progressive

Website Optimization anecdotes

- Hero image halfway on the page
  - 6 HTTP/1 connections had been taken up, HTTP/2 gives all images same priority
  - Concatenation : only benefits HTTP/1
  - Moving image up in HTML : benefits both



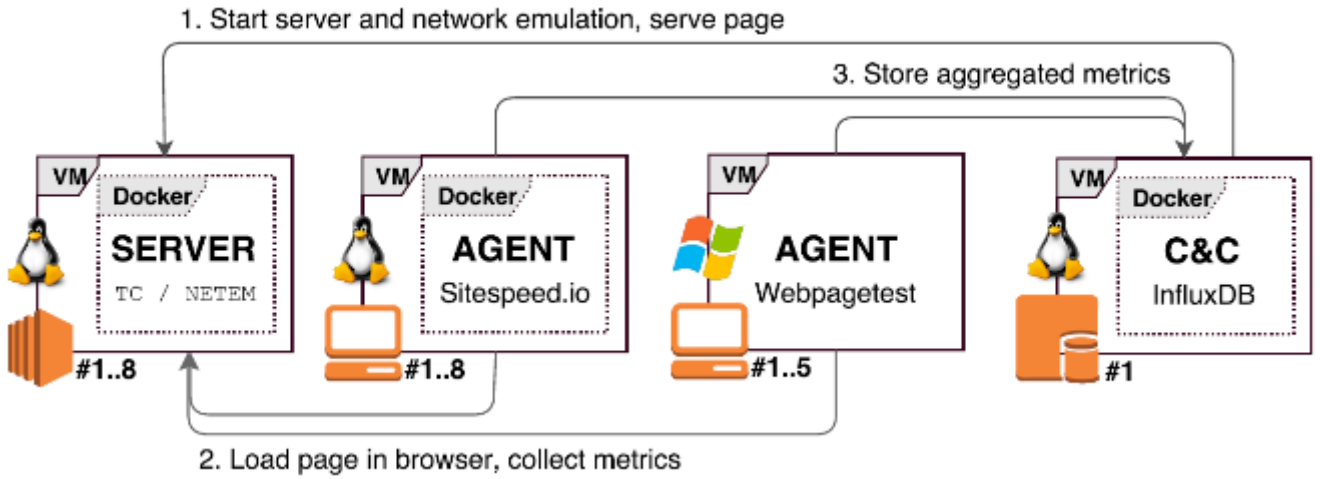
Website Optimization anecdotes

- Embedding is Best Practice!
  - Let's embed our fonts as well! Base64-style!
  - Embeds > 100KB in fonts that are not cached…
  - Every new page load, flash-of-unstyled-content despite embedding

imec UHASSELT KNOWLEDGE IN ACTION

Docker / The Speeder Framework

- 3 Docker containers
  - Base (C&C), Backend and Agents
  - Dockerfile is easy to update, rebuild, deploy
- 5 private WebPageTest instances
- C&C starts instances and restarts all running software before each test
- Spread over 16 Linux VMs and 5 Windows VMs



Docker / The Speeder Framework

- Simple GUI for starting tests

0 tests sheduled

Testrun:
debug_test

Label:

Run count:
2

Option 1 : choose a locally served testcase (use Speeder backend setup)

Testcases: 0 / 286

- 2v1_aggregation 0 / 57
- 2v1_aggregationComplex 0 / 57
- 2v1_aggregationLibrary 0 / 13
- 2v1_domainSharding 0 / 10
- 2v1_gzlib 0 / 16
- 2v1_http2Priority 0 / 4
- 2v1_images 0 / 11
- 2v1_inlining 0 / 33
- 2v1_inliningComplex 0 / 33
- 2v1_loadingJS 0 / 8
- 2v1_redirect 0 / 9
- 2v1_sanitychecks 0 / 1
- 2v1_serverPush 0 / 24
- 2v1_template 0 / 2
- H2priorities_H2oWithVsWithoutHttp2Prios 0 / 1
- H2priorities_test1debug 0 / 1
- VoDLib_NginxVsNode 0 / 4
- drupal_main 0 / 0
- examples_customconfigs 0 / 2

Backends:

- node
- nginx
- apache
- custom

Frontends:

- sitespeed
  - chrome
  - firefox
  - slimer
- browsertime_chrome
- SRUMdriver_chrome
- SRUMextension_chrome
- wpt
  - chrome
  - firefox
  - ie

Protocols:

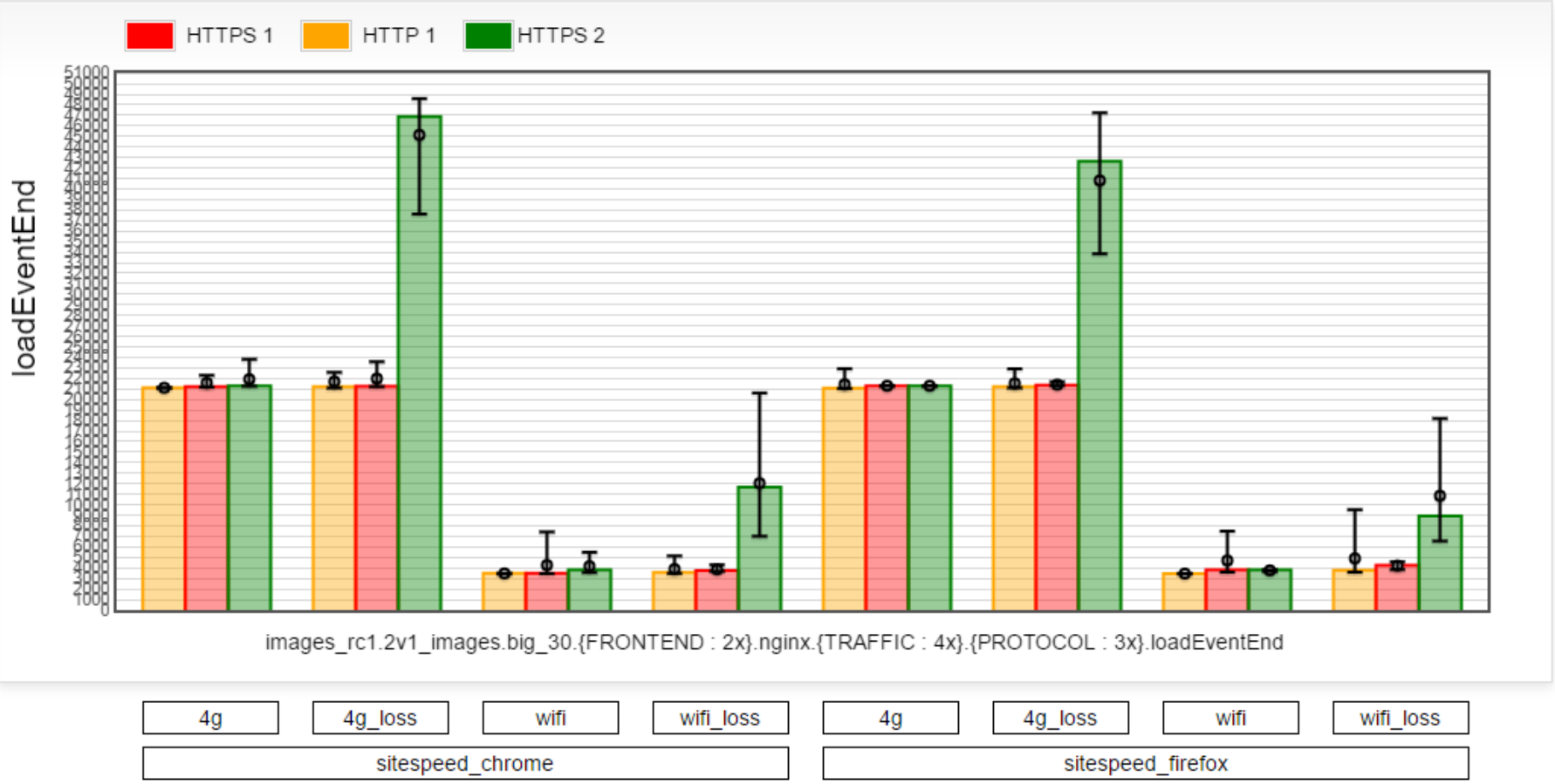
- HTTPS 2
- HTTPS 1.1
- HTTP 1.1

Traffic:

- none
- wifi
- wifi_transatlantic
- wifi_transatlantic_loss
- 4g
- 4g_transatlantic
- 4g_transatlantic_loss
- 3g_loss
- 2g_loss
- cellular_noloss
- cellular_good
- cellular_fair
- cellular_passable
- cellular_poor
- cellular_verypoor

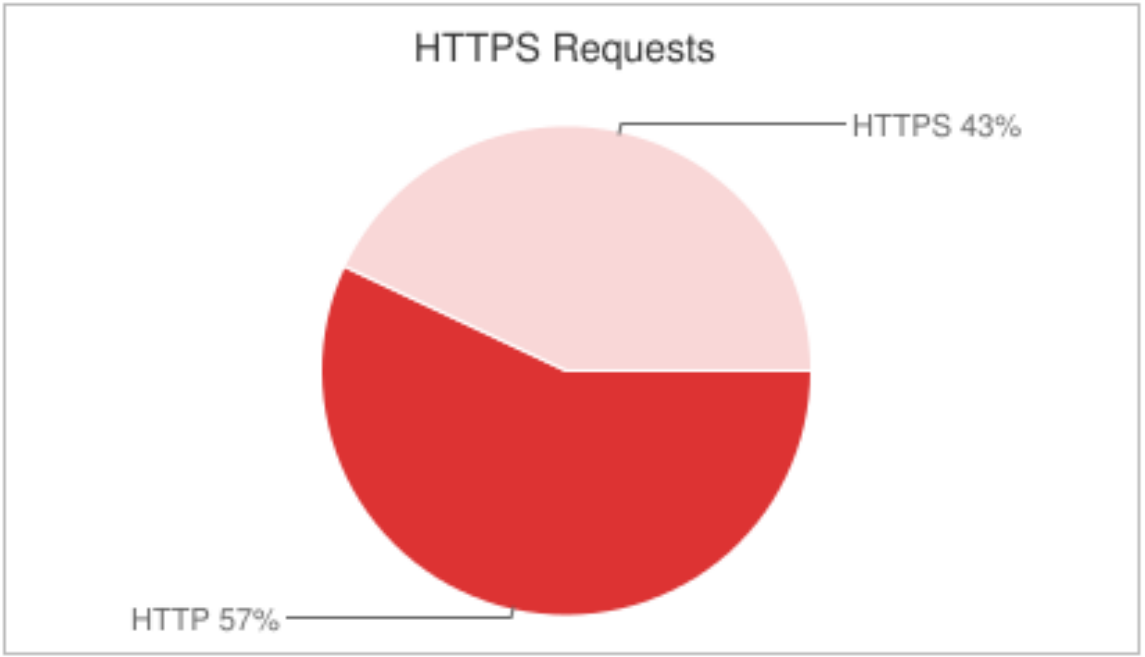
Docker / The Speeder Framework

- Quick graphing of results



HTTPS vs HTTP cleartext

- Push towards a safer web
- H2 cleartext isn’t supported by any browser
- Many of the newer (performance) features only work over HTTPS



httparchive.org

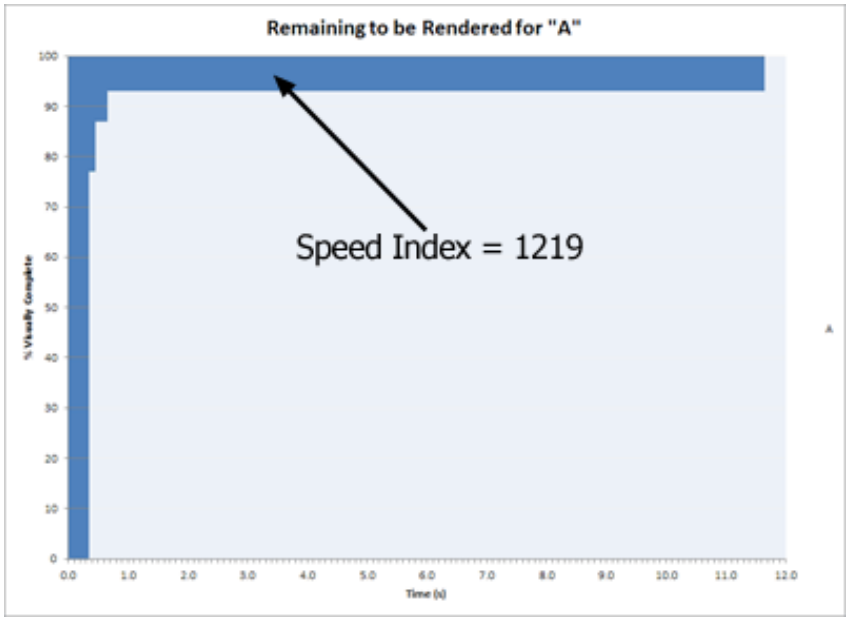
Other uses of HTTP/2 Push

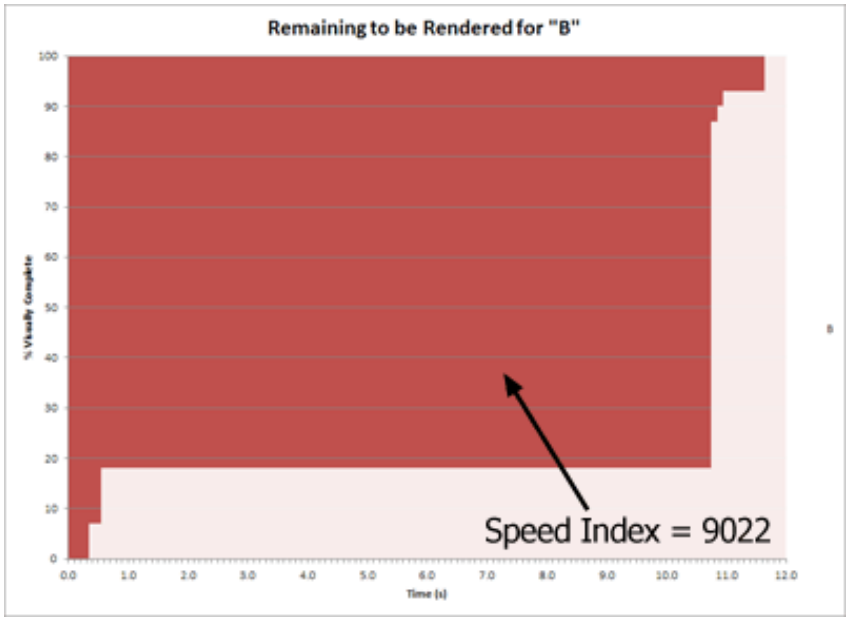
- Warm vs cold connections
- Single Page Apps / REST APIs
- Service Workers
- Video streaming
- CDN-side

- But: Caching problems!
- But: Server support!
- But: Re-prioritization!

loadEventEnd vs SpeedIndex

- SpeedIndex
  - % of visual complete over time (video frames!)
  - Quick to render = better for user experience
  - Measured in milliseconds = LOWER IS BETTER





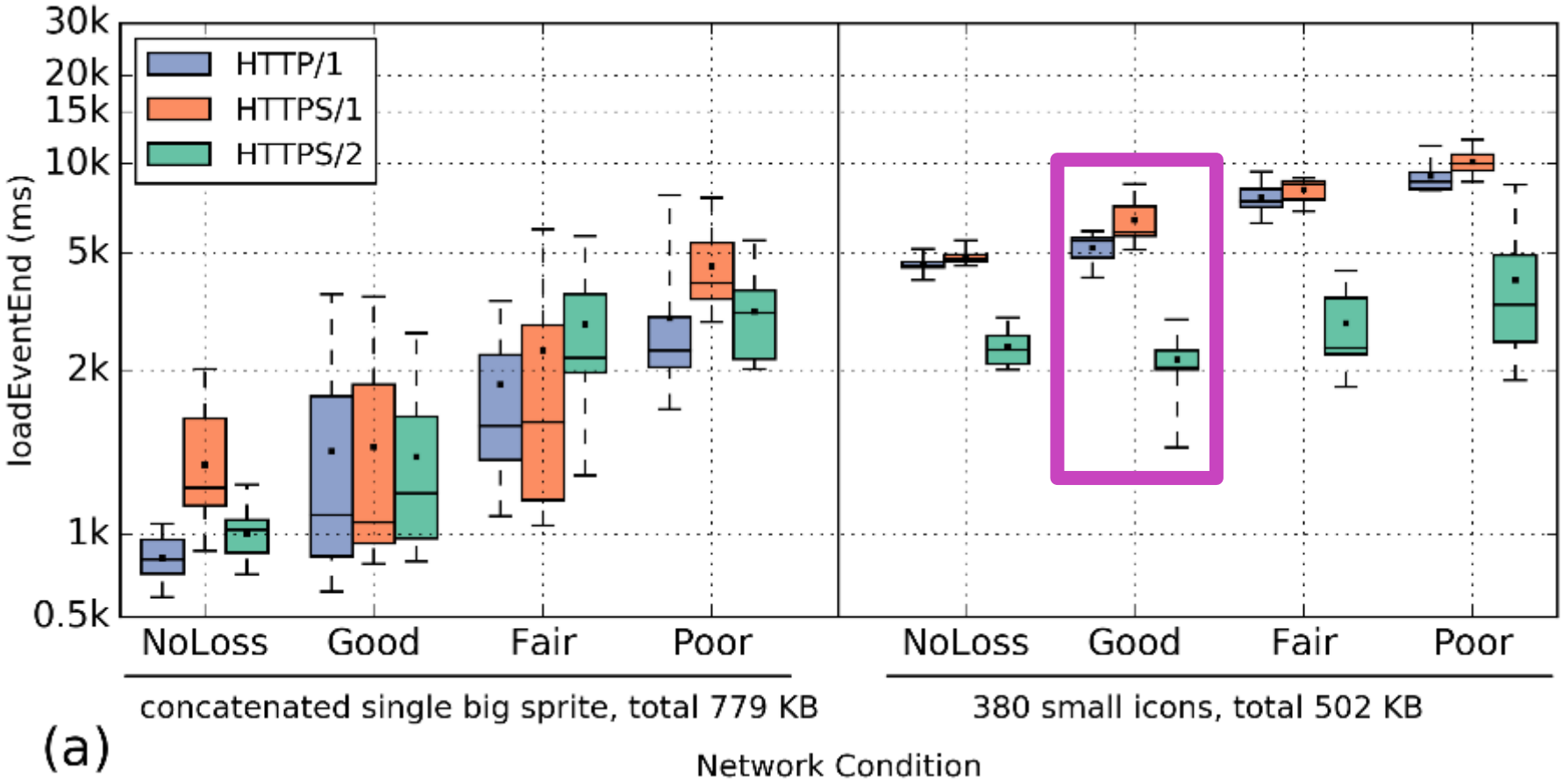
https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index

1. Concatenation

- Images : 380 small, 42 medium, 30 large
- CSS/JavaScript : 1 – 500 files, simple and complex

http://www.netflix.com

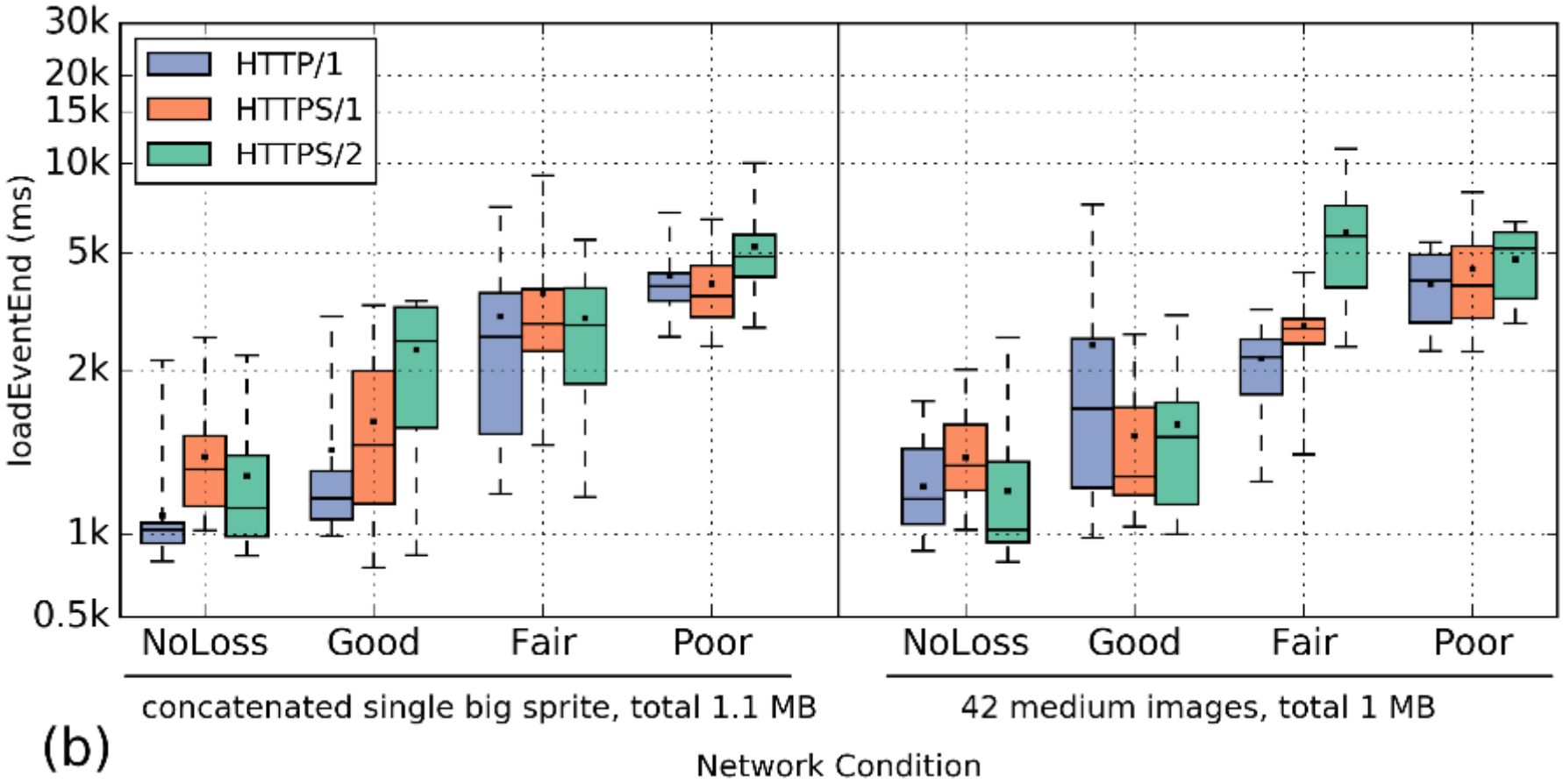
1. Concatenation : 380 small images



OPTIMIZED

NOT OPTIMIZED

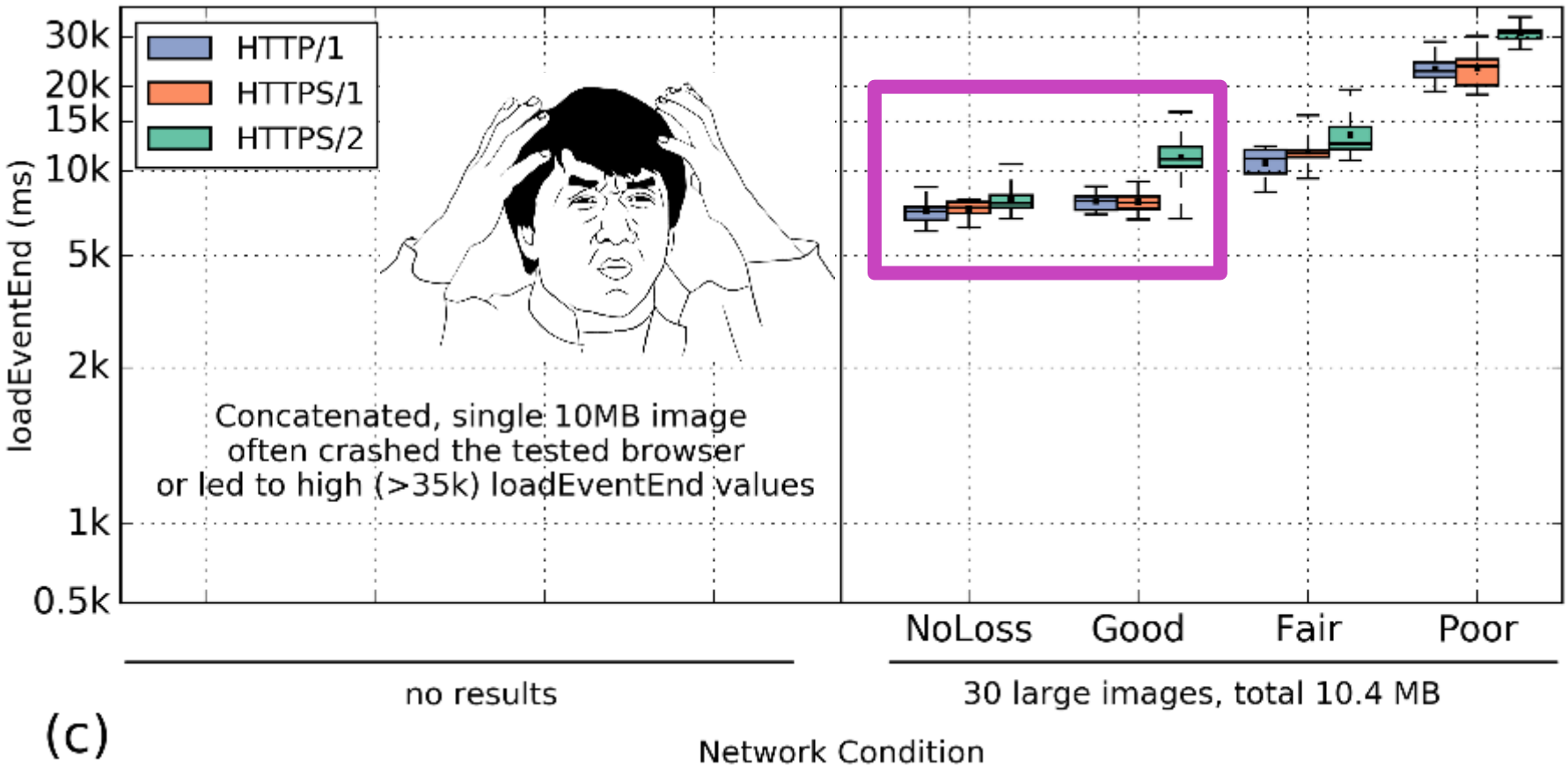
1. Concatenation : 42 medium images



OPTIMIZED

NOT OPTIMIZED

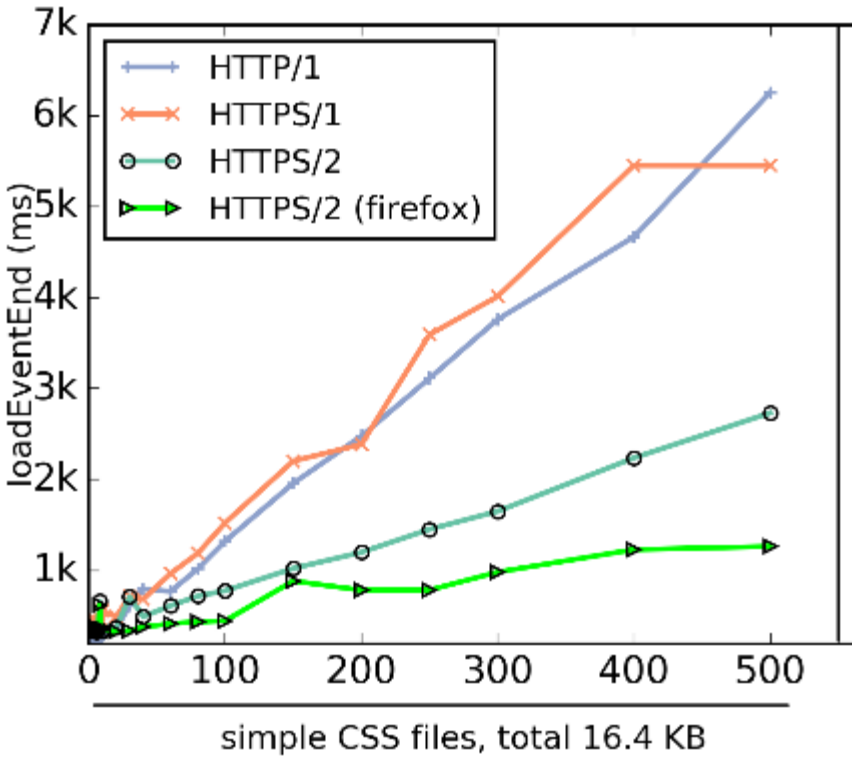
1. Concatenation : 30 large images

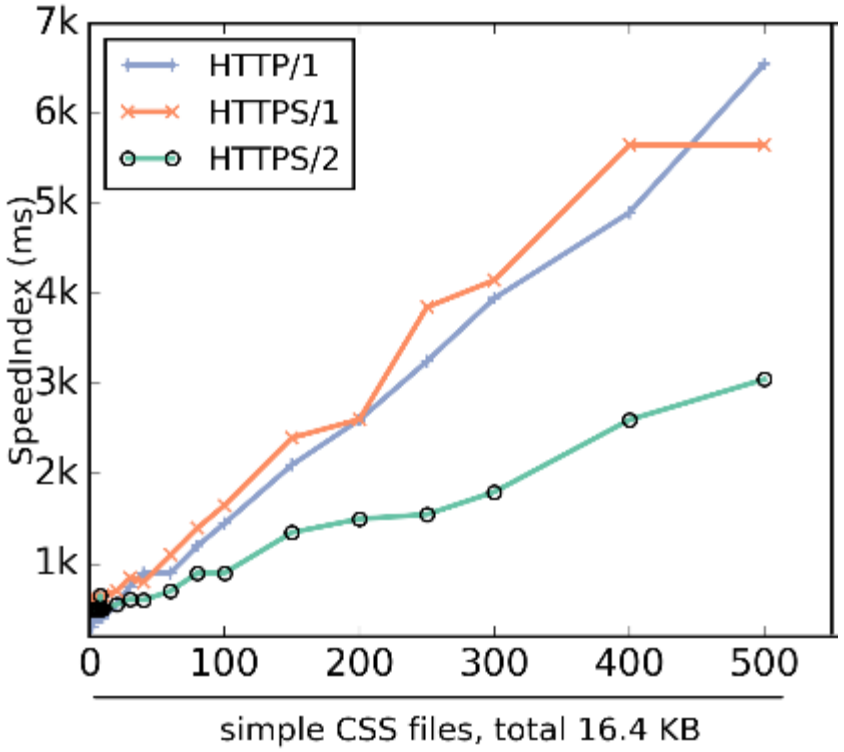


OPTIMIZED

NOT OPTIMIZED

1. Concatenation : CSS SpeedIndex

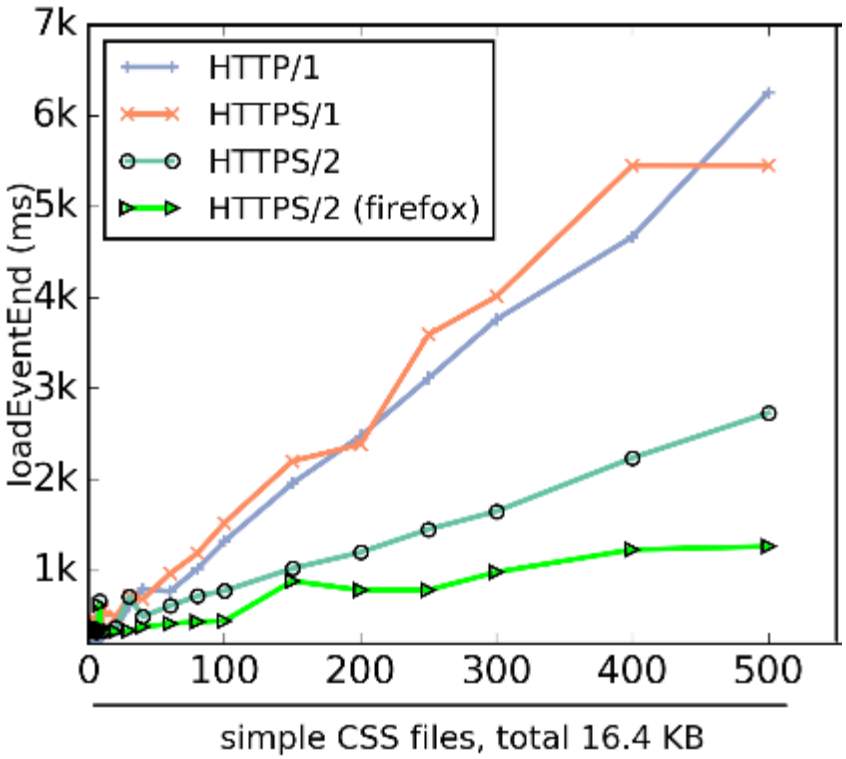


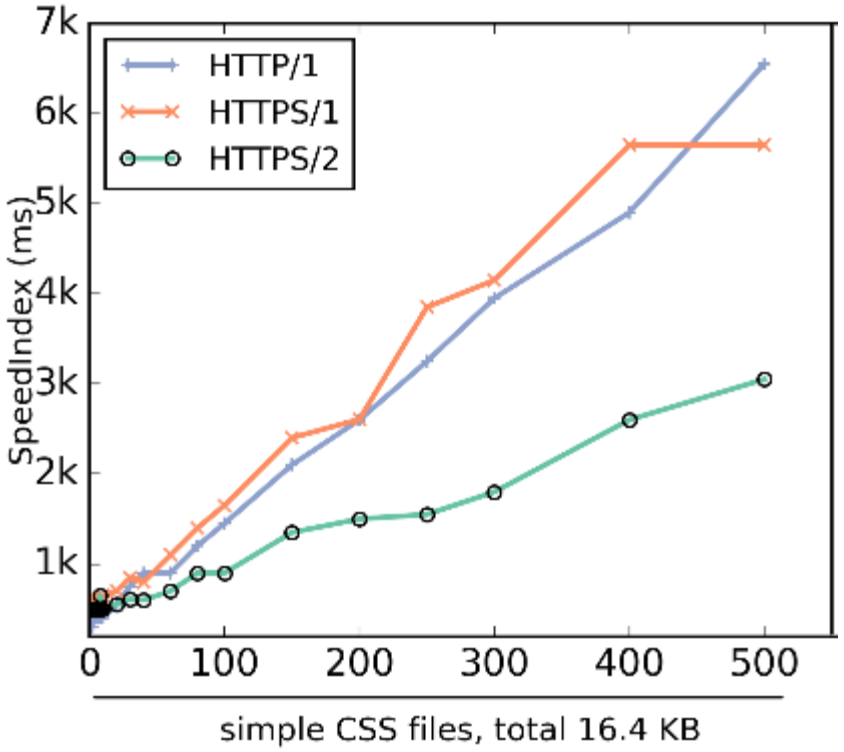


loadEventEnd

SpeedIndex

1. Concatenation : CSS SpeedIndex





Both are the same = rendering started at onLoad
=> Page was white until all CSS was loaded

4. Realistic websites

- Many factors contribute to load times
  - Polaris (Netravali et al.), Shandian (Wang et al.)

- Other related work exists
  - http://isthewebhttp2yet.com (Varvello et al.)

Can we simply transfer HTTP/1 sites to HTTP/2 ?

Image Sources

- http://deliveryimages.acm.org/10.1145/2390000/2380673/figs/f5.jpg
- https://dab1nmslvvntp.cloudfront.net/wp-content/uploads/2017/01/1484692838webpack-dependency-tree.png
- https://blog.cloudflare.com/content/images/2015/12/domain-sharding-1.png
- https://calendar.perfplanet.com/2016/http2-push-the-details/
- http://www.splicemarketing.co.uk/_blog/Splice_Blog/post/the-arrival-of-http2/
- http://s1377.photobucket.com/user/barteks11/media/Marvel%20gifs/h20wkj2-iron-man-vs-captain-america-who-sides-with-who-in-marvel-s-civil-war-jpeg-151871_zpsarkko07f.jpg.html
- https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index
- https://twitter.com/officialrtsf
- https://twitter.com/ThePracticalDev
- http://www.netflix.com