Cache-digests, etc.

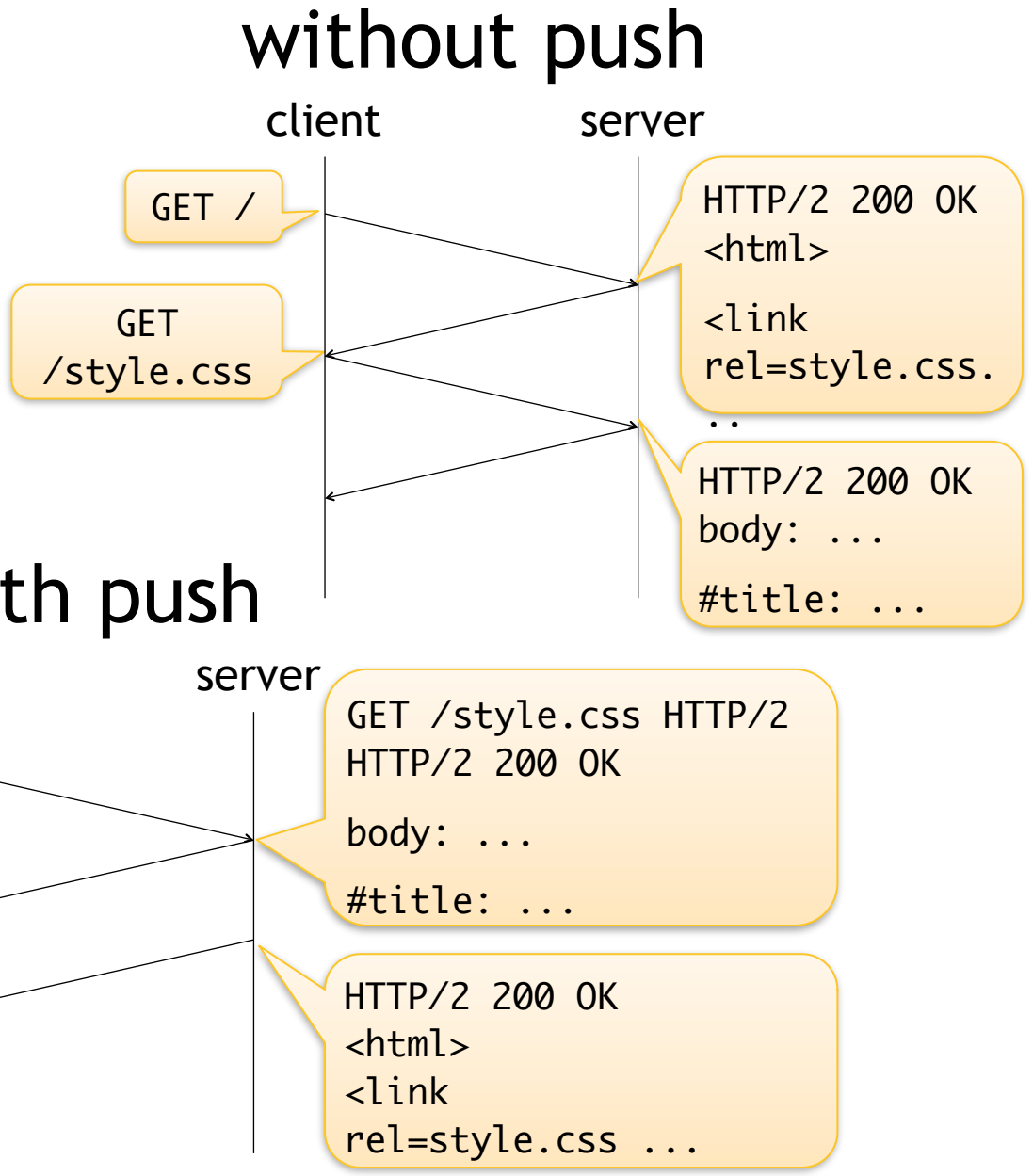
DeNA Co., Ltd.

Kazuho Oku

Three use-cases of push

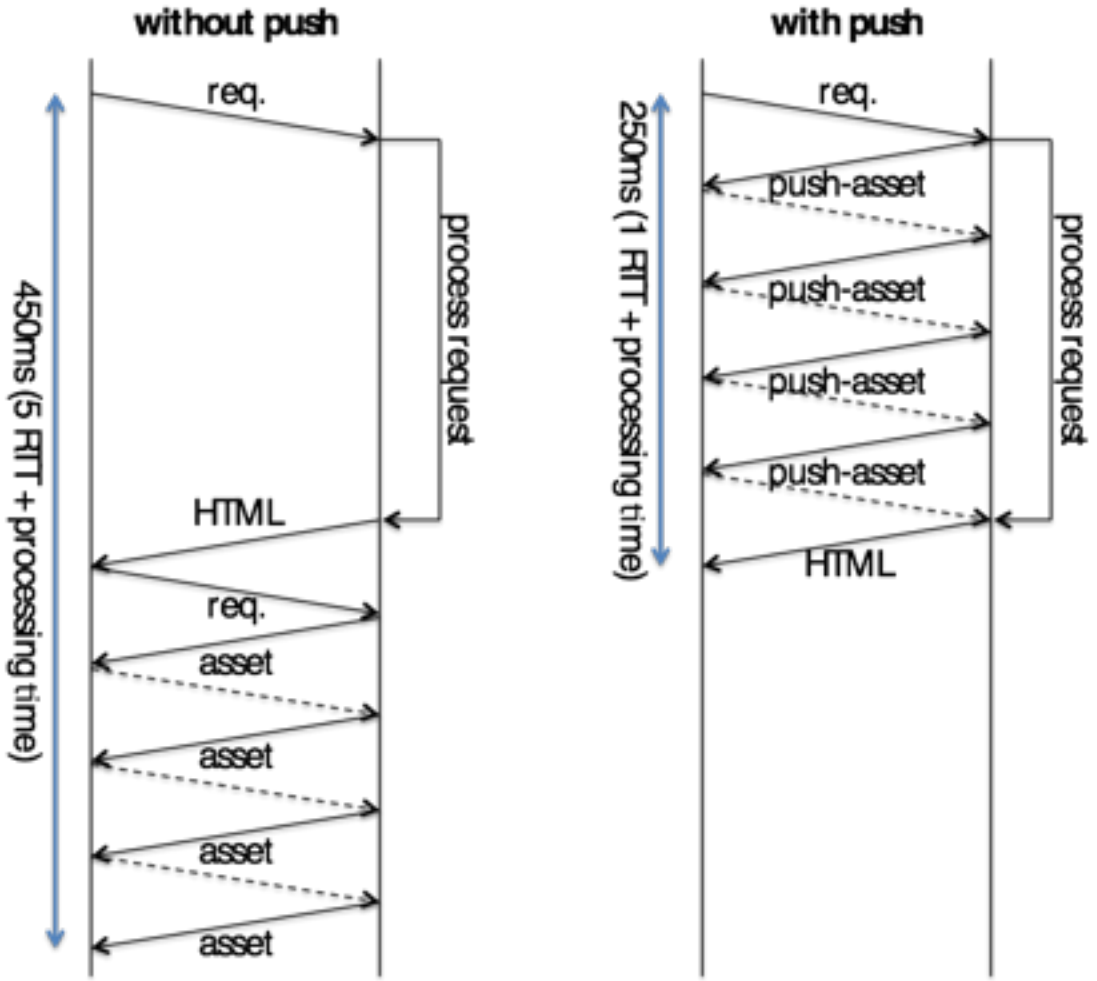
Send critical responses first

1. send CSS, JS
2. send HTML (can be rendered progressively)



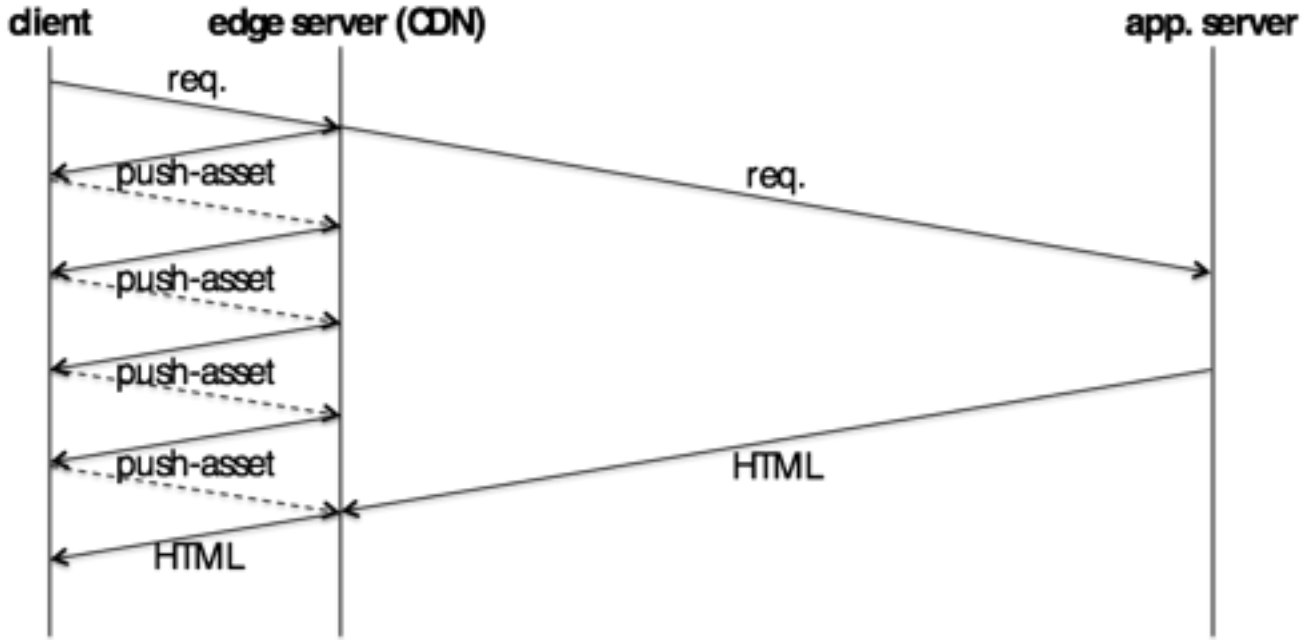
Conceal request processing time

- web applications involving DB access, etc.



Conceal network distance

- CDNs' use-case
  - utilize the conn. while waiting for app. response



Cache Digests at a glance

- a compressed list of URLs cached by the client
  - sent by client
  - used by server to determine what *not* to push
- uses Golomb-coded sets (compressed bloom filter)
  - encodes delta of pre-sorted hash values (H_n)
    - range of the hash values is: $0..2^{N+P}-1$
    - $floor((H_n - H_{n-1} - 1) / P)$ encoded using unary coding
    - remainder encoded as binary of log2(P) bits
  - size: below (2+P)*N bits
- sent as a HTTP/2 frame following the first request

Example

- create digest of URLs at false positive rate: $1/2^7$
- calculate SHA256
  - https://example.com/style.css (SHA256: baf9…)
  - https://example.com/script.js (SHA256: 1174…)
- truncate to 2 (N) * log2(2^7) bits and sort
  - 0x11, 0xba
- emit log2(2), log2(2^7) in 5 bits, then the deltas of sorted list (subtracted by 1)
  - 00001 00111 1 0010001 01 010 1000

    N P 0x11 0xa8 (0xba-0x11-1)

Implementations

- Server
  - H2O: cookie (pre-00), header (ietf-00)
  - mod_h2: header (kazuho-00)
  - nghttp2: frame (ietf-00)
- Client
  - cache-digest.js: header (ietf-00, ServiceWorker)
  - nghttp2: frame (ietf-00)

from: Report on Cache digests for HTTP/2 Push

- by Zunzun AB.
  - Almost 70% goes to domains that have between 0 and 5 assets in cache.
  - The next 15% contains domains that have between 5 and 10 assets in cache.
  - And the final, thinnest slice goes to sites that have between 120 and 180 assets in cache.
  - With P equal 1/32, … the largest digest wouldn't be longer than 167 bytes.

Command-line tool

- https://github.com/h2o/cache-digest.js
  - SW-based impl. comes with a command-line tool

```
% node cli.js -p 7 https://example.com/style.css \
        https://example.com/script.js | od -t x1
0000000    09  e4  55  00
```

Changes since the initial draft

- GCS encoding
- four flags
  - STALE, VALIDATORS, COMPLETE, RESET

Changes to GCS

- max range has been restricted to 0 .. 2^{62}-1
  - by restricting N and P to 5 bits (was 8)
    - note: max value is 2^{N+P}-1
  - servers can use 64bit int ops
- truncate sha256(URL) instead of taking modulo
  - follows the pattern in HMAC, SHA224, etc.
- encode unarys as 000…1 (was 111…0)
- ToDo: recommend better way to decide P

STALE and VALIDATORS

- STALE flag:
  - if set: payload contains digest of stale responses
  - if not set: contains digest of fresh responses
- VALIDATORS flag:
  - if set: hash = sha256(URL + Etag)
  - if not set: hash = sha256(URL)

The usage matrix

		STALE	
		0	1
VALIDATORS	0	push responses cached as fresh	???
	1	push / update responses cached as fresh	update / revalidate responses cached as stale

(STALE, VALIDATORS) = (0, 0)

- i.e. list of H(URL) of fresh responses
- on GCS hit: do nothing
- on GCS miss: push 200 with big expiration date
    - so that it would be fresh until eviction
        - under the assumption that some clients might only send a digest of fresh responses

(STALE, VALIDATORS) = (0, 1)

- i.e. list of H(URL + etag) of fresh responses
- on GCS hit: do nothing
- on GCS miss: push 200
  - to force-update the client cache
    - if the client is capable of accepting push to replace an existing fresh response

(STALE, VALIDATORS) = (1, 0)

- i.e. list of H(URL) of stale responses
- on GCS hit: push not-modified?
  - but the client might have an outdated response
- on GCS miss: push 200

no good, should always use (1,1) for stale responses

(STALE, VALIDATORS) = (1, 1)

- i.e. list of H(URL + etag) of stale responses
- on GCS hit: push not-modified on GCS hit
  - but how?
- on GCS miss: push 200 on GCS miss

Two things to consider

- how to push not-modified
  - essential if we want to push responses that might become stale-cached
- how to force-update a fresh response
  - or we should merge the two flags into one, if we agree to ignore such possibility

Pushing not-modified

Timing requirements for push

- server SHOULD send PUSH_PROMISE prior to sending Link: rel=preload that refers to the pushed response
- server SHOULD NOT reset a pushed stream, even if cache-digest indicates it has a response in cache
  - cache-digest has false positives
  - response may get evicted after generating digest
- 2nd requirement is hard to meet, esp. when server needs to lookup etag of a potentially pushed resp.
  - client needs to issue GET after receiving pushed not-modified, if no matching response was found

At least 3 ways to push not-modified

- unconditional GET + 304 w. etag
- HEAD + 200 w. etag
- Link: etag=xxx

Unconditional GET + 304 w. etag

- pros:
  - PUSH_PROMISE is indifferent w. 200
    - server can first send PUSH_PROMISE, then decide whether to send 200 or 304
- cons:
  - client needs to issue GET if matching response was not found in cache
  - compatibility issue: existing clients recognize 304 as 200 with 0-byte content
    - could limit use of 304 to when stale digests are received

HEAD + 200 w. etag

- pros:
  - no compatibility issues
- cons:
  - server needs to determine etag before sending PUSH_PROMISE
    - since in case of GCS miss it needs to push a GET
  - client needs to issue GET if matching response was not found in cache

Link: etag=xxx

- i.e. server specifies the etag of a preloaded link
  - client should use a cached response with a matching etag, or if not found in cache, should fetch the link (that might be push by the server)
- pros:
  - no need to consider the case of reissuing GET
  - can also be used for updating a fresh response
  - works without push / cache-digests
- cons:
  - server needs to determine etag before sending the header

COMPLETE flag

- cache-digest transfers tri-state for every URL:
  - CACHED: GCS hit (may have false positives)
  - UKDETERMINED: GCS miss wo. the flag set
  - NOT_CACHED: GCS miss with the flag set
- a server might push assets only known to be NOT_CACHED, or might push those UNIDENTIFIED as well
- the flag covers either the digests identified by the STALE flag
  - i.e. both (STALE, VALIDATORS)=(x,0) *and* (x,1)

RESET flag

- digests accumulate as CACHE_DIGEST frames arrive
  - use-case: caching proxy receives cache objects from a peer cache
- RESET flag used to reset *all* data
  - use-case: user clears the browser cache

Initiating push

Problem

- webapps know what to push
  - they build concatenated JS / CSS using “asset-pipeline”
- HTTP/2 server recognizes link: rel=preload response header and pushes the assets
- webapps cannot send the link header, until it finishes processing the request (by accessing database, etc.)

Solution in H2O

- recognize link: rel=preload headers in 100 response
- webapps run in following steps:
  1. send 100 + link: rel=preload
  2. process the request (e.g. query the database)
  3. send 200 + HTML

```
HTTP/1.1 100 Continue
Link: </style.css>; rel=preload

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
<HTML>
<title>Hello from my webapp!</title>
...
```

Do the same for browsers?

- nice complement to H2 push
  - can be used to instruct 3p preload links ASAP
  - browsers not supporting H2 push could leverage from this