HTTP priority design team

HTTP WG, IETF 106 Singapore
Design Team Goals

**Requirement:** Determine a solution for HTTP/3 to have some client-to-server priority hinting mechanism that it can ship with. This can be a minimal approach.

The following are potentially in scope (though not all are required):

- Mechanism to indicate that H2 priorities are not being used
- Mechanism to indicate what kind of priority hinting is being used
- Priority hinting mechanism that is non-minimal
- A plan to backport the new priority hinting to H2

**Out of scope:** Changes that would add complexity that we’re not confident in that would risk shipping HTTP/3.
Motivation: HTTP/2, The Wild West

https://speeder.edm.uhasselt.be/www18/
Simulation Results
H2 on large pages (>1MB)

Higher line is faster

Chrome’s use of H2 is best of browsers, old Edge’s is worst (fair Round Robin)

(btw: fair RR is also H2’s default behaviour…)

https://speeder.edm.uhasselt.be/www18/
Can we do better?

https://h3.edm.uhasselt.be/
Can we do better?

Table 2: Mean speedup ratios compared to \( rr \) per other prioritization scheme from Figure 12. Higher mean values are better. #PH = number of placeholders used in this scheme.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>#PH</th>
<th>Mean All</th>
<th>Mean ATF</th>
<th>Mean 1000K</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrr</td>
<td>0</td>
<td>1.05</td>
<td>1.49</td>
<td>1.28</td>
</tr>
<tr>
<td>fifo</td>
<td>0</td>
<td>1.27</td>
<td>1.93</td>
<td>1.57</td>
</tr>
<tr>
<td>dfifo</td>
<td>5</td>
<td>1.27</td>
<td>2.30</td>
<td>1.72</td>
</tr>
<tr>
<td>firefox</td>
<td>6</td>
<td>1.07</td>
<td>1.22</td>
<td>1.25</td>
</tr>
<tr>
<td>p+</td>
<td>3</td>
<td>1.17</td>
<td>2.20</td>
<td>1.64</td>
</tr>
<tr>
<td>s+</td>
<td>8</td>
<td>1.14</td>
<td>1.45</td>
<td>1.56</td>
</tr>
<tr>
<td>spdyrr</td>
<td>5</td>
<td>1.14</td>
<td>1.96</td>
<td>1.57</td>
</tr>
<tr>
<td>bucket</td>
<td>0</td>
<td>1.20</td>
<td>2.13</td>
<td>1.82</td>
</tr>
<tr>
<td>bucket</td>
<td>0</td>
<td>1.20</td>
<td>2.49</td>
<td>1.83</td>
</tr>
<tr>
<td>zeroweight</td>
<td>0</td>
<td>1.15</td>
<td>2.8</td>
<td>1.9</td>
</tr>
</tbody>
</table>

ATF = “above the fold”, critical resources

- Can do better than Chrome, with **simpler** schemes
- Server-side (re-)prioritization is **very** powerful, but difficult across browsers in H2
- Flexibility still needed (heterogeneous sites, HOL blocking)

https://h3.edm.uhasselt.be/
https://blog.cloudflare.com/better-http-2-prioritization-for-a-faster-web/
Experimental measurements
Background

All experiments done with gQUIC between Chrome and Google servers

Control group is SPDY priorities: 8 buckets, round robin within a bucket (spdyrr)
gQUIC default

Experiment groups

- Chrome H2 : Linked list (dfifo) - buckets, sequential within a bucket
- FIFO - lowest stream ID first
- LIFO - highest stream ID first
- Round Robin (rr)
YouTube QoE

LIFO

Android 3.34% higher rebuffer rate than SPDY, reduction in video resolution

Desktop 2.6% higher rebuffer rate than SPDY, reduction in video resolution

All other schemes were statistically insignificant

For reference, QUIC had 15.3% fewer rebuffers on Android, 18% on Desktop
Flywheel Data Compression Proxy

- All HTTP requests are proxied to Google servers over a QUIC connection
  - in a sense, the "best case" for prioritization
  - HTTP only; HTTPS requests are not proxied
  - Android Chrome users only

- Summary:
  - Chrome H2 > SPDY > {FIFO, LIFO, RoundRobin}
  - Improvements range from 0.4% faster to 1.7% faster
## Flywheel Data Compression Proxy

**FirstContentfulPaint relative to H2**  
(statistically-significant changes only, with 95% CIs, **green = H2 is faster**)

<table>
<thead>
<tr>
<th>grp</th>
<th>25th percentile Percentage Difference</th>
<th>50th percentile Percentage Difference</th>
<th>75th percentile Percentage Difference</th>
<th>90th percentile Percentage Difference</th>
<th>95th percentile Percentage Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td>-0.40% [-1.77, 0.96] %</td>
<td>0.64% [0.15, 1.12] %</td>
<td>0.29% [-0.18, 0.76] %</td>
<td>0.75% [0.17, 1.34] %</td>
<td>1.77% [0.70, 2.85] %</td>
</tr>
<tr>
<td>LIFO</td>
<td>-0.81% [-2.50, 0.88] %</td>
<td>0.64% [-0.12, 1.40] %</td>
<td>0.87% [0.43, 1.32] %</td>
<td>1.22% [0.47, 1.97] %</td>
<td>1.77% [0.44, 3.10] %</td>
</tr>
<tr>
<td>RoundRobin</td>
<td>0.61% [-1.03, 2.25] %</td>
<td>0.88% [0.22, 1.53] %</td>
<td>0.79% [0.17, 1.41] %</td>
<td>0.96% [0.19, 1.74] %</td>
<td>1.40% [0.10, 2.69] %</td>
</tr>
<tr>
<td>SPDY</td>
<td>0.40% [-0.70, 1.51] %</td>
<td>0.48% [-0.13, 1.08] %</td>
<td>0.62% [0.25, 1.00] %</td>
<td>0.64% [-0.14, 1.41] %</td>
<td>0.30% [-0.70, 1.29] %</td>
</tr>
</tbody>
</table>
Flywheel Data Compression Proxy

FirstContentfulPaint relative to SPDY
(statistically-significant changes only, with 95% CIs, green = SPDY is faster, red = SPDY is slower)

<table>
<thead>
<tr>
<th>grp</th>
<th>25th percentile Percentage Difference</th>
<th>50th percentile Percentage Difference</th>
<th>75th percentile Percentage Difference</th>
<th>90th percentile Percentage Difference</th>
<th>95th percentile Percentage Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td>-0.81% [2.14, 0.52] %</td>
<td>0.16% [-0.55, 0.87] %</td>
<td>-0.33% [-0.81, 0.15] %</td>
<td>0.12% [-0.72, 0.96] %</td>
<td>1.47% [0.48, 2.47] %</td>
</tr>
<tr>
<td>H2</td>
<td>-0.40% [-1.50, 0.70] %</td>
<td>-0.48% [-1.07, 0.12] %</td>
<td>-0.62% [-0.99, -0.25] %</td>
<td>-0.63% [-1.40, 0.14] %</td>
<td>-0.30% [-1.29, 0.69] %</td>
</tr>
<tr>
<td>LIFO</td>
<td>-1.21% [-2.73, 0.31] %</td>
<td>0.16% [-0.56, 0.88] %</td>
<td>0.25% [-0.23, 0.72] %</td>
<td>0.58% [-0.01, 1.17] %</td>
<td>1.47% [0.59, 2.36] %</td>
</tr>
<tr>
<td>RoundRobin</td>
<td>0.20% [-1.39, 1.79] %</td>
<td>0.40% [-0.19, 0.98] %</td>
<td>0.17% [-0.30, 0.63] %</td>
<td>0.33% [-0.31, 0.96] %</td>
<td>1.10% [-0.01, 2.20] %</td>
</tr>
</tbody>
</table>
AMP

- AMP clicks from the Google Search results page
  - Android Chrome users only
  - Only AMP clicks that were not prerendered

- Summary:
  - SPDY > {Chrome H2, FIFO, LIFO, RoundRobin}
  - Improvements range from 0.5% faster to 1.4% faster
FirstContentfulPaint relative to SPDY
(statistically-significant changes only, with 95% CIs, green = SPDY is faster)

<table>
<thead>
<tr>
<th>grp</th>
<th>25th percentile</th>
<th>50th percentile</th>
<th>75th percentile</th>
<th>90th percentile</th>
<th>95th percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Percentage Diff</td>
<td>Percentage Diff</td>
<td>Percentage Diff</td>
<td>Percentage Diff</td>
<td>Percentage Diff</td>
</tr>
<tr>
<td>FIFO</td>
<td>-</td>
<td>-</td>
<td>0.32% [0.01, 0.64]%</td>
<td>0.32% [-0.35, 0.99]%</td>
<td>0.69% [-0.43, 1.82]%</td>
</tr>
<tr>
<td>H2</td>
<td>0.35% [-0.41, 1.10]%</td>
<td>0.41% [-0.40, 1.22]%</td>
<td>0.32% [-0.15, 0.80]%</td>
<td>1.08% [0.37, 1.78]%</td>
<td>1.20% [-0.19, 2.58]%</td>
</tr>
<tr>
<td>LIFO</td>
<td>0.35% [-0.44, 1.13]%</td>
<td>0.41% [-0.11, 0.93]%</td>
<td>0.51% [0.19, 0.83]%</td>
<td>0.88% [0.21, 1.54]%</td>
<td>1.45% [0.37, 2.52]%</td>
</tr>
<tr>
<td>RoundRobin</td>
<td>0.17% [-0.62, 0.96]%</td>
<td>0.00% [-0.69, 0.69]%</td>
<td>0.19% [-0.20, 0.58]%</td>
<td>0.76% [0.06, 1.46]%</td>
<td>0.94% [-0.32, 2.21]%</td>
</tr>
</tbody>
</table>
Summary

New design should therefore:

- Be simpler than HTTP/2 tree
- Work for both H2 and H3
- Allow for expressing both Chrome H2 and SPDY schemes
- Allow easy server-side (re-)prioritization
- Not use Round Robin as the default

The priority draft ([draft-kazuho-httpbis-priority](draft-kazuho-httpbis-priority)) includes all of these.
Proposed Design
an update to draft-kazuho-httpbis-priority
Extensible Priorities

**Goal:** Extensible without changing every client every time

=> Unique Key-value pairs, encoded using Structured Headers

Initially specifies 2 fields, ‘urgency’ and ‘progressive’

‘urgency’ parameter is an integer between -1 and 6

‘progressive’ is 0 or 1

  If 0, fifo within an urgency, 1 indicates round-robin
Urgency semantics

The draft details how these are intended to be used in Section 4.1

-1 prerequisite
0 default
1 to 5 supplementary
6 background

Semantics enable an origin server to effectively re-prioritize without knowing the priority of every other request.

Semantics “hopefully” create more consistency across browsers
Two Key Use Cases

Client to Server over multiplexed HTTP

One Common Goal: Provide scheduling hints to the sender
Headers as an API

Headers are the standard API for an application using HTTP

Applications could also have a specific API, that’s out of scope for the DT

However, Headers are End-To-End on the wire

Introduces complexities, still need a frame for re-prioritization

Solution: Senders locally consume application headers

Only frames are used to prioritize hop-by-hop

If a server receives this header from a client, it can ignore it

Open Questions: Could/should this be a pseudo-header?

Can/should this be exposed in the web API? (Whose decision?)
Wire Encoding Goals

The initial priority frame needs to be delivered prior to the HEADERS frame.

Client should send first requests with initial priorities even before it receives the server's SETTINGS.

Allow Reprioritization even after a request has been sent.
New Frame: HTTP/2

**R**: Reserved 1-bit field

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------------------------------------------------------+
| R |                        Stream ID (31)                       |
+---------------------------------------------------------------+
|                                           Priority Field Value (*) ... |
```

*Only* sent on the control stream, because of HTTP/2 extension constraints
MUST be sent immediately preceding corresponding headers

A server only has to remember the most recent
Reprioritization also on the control stream
New Frame: H3

T: Bit to indicate request Stream ID or Push ID

Initially sent on the request stream before headers
Reprioritization on the control stream
Proxy to Origin

‘priority’ header can be sent by proxy
  Indicates current priority on the *previous* hop

‘priority’ response header sent by origin
  Indicates to override the client’s priority
  Allows specifying a priority if the client specifies nothing

Example Deployment described in [better-http-2-prioritization-for-a-faster-web](http://example.com)
Issue [#57](http://example.com)
Negotiation with SETTINGS

Key use cases:
1) The client can indicate it does not use H2 priorities
2) The server expresses what information from the client it wants

Challenge: Either side may send first, neither can wait for the others

Ordered sequence of 8-bit identifiers, with the server’s preference dominating
  Up to 4 values in H2 (32 bits), 7 values in H3 (62 bits)

Draft defines:
  H2-TREE
  URGENCY - May be renamed EXTENSIBLE
Some smaller issues still TBD

Should ‘urgency’ start at -1? A higher value corresponds to higher priority?

<table>
<thead>
<tr>
<th></th>
<th>-1 [0] 1 2 3 4 5 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>option a</td>
<td>0 [1] 2 3 4 5 6 7</td>
</tr>
<tr>
<td>option b</td>
<td>7 [6] 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

What is the best encoding of the key-value pairs?

Design Team Goals: Review

**Requirement:** Determine a solution for HTTP/3 to have some client-to-server priority hinting mechanism that it can ship with. This can be a minimal approach.

The following are potentially in scope (though not all are required):

- Mechanism to indicate that H2 priorities are not being used
- Mechanism to indicate what kind of priority hinting is being used
- Priority hinting mechanism that is non-minimal
- A plan to backport the new priority hinting to H2

**Out of scope:** Changes that would add complexity that we’re not confident in that would risk shipping HTTP/3.
What’s next?
Update the draft to reflect this proposal
Determine if/how to break it up into multiple docs
Close smaller issues

Comments, Questions, Suggestions?
Add to H3, keep as extension?

Thanks to all design team members!  Group Notes