INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11 MPEG2015/M37255
October 2015, Geneva, Switzerland

Source	**Telecom ParisTech**
Status	**For consideration at the 113th MPEG Meeting**
Title	**Use of HTTP/2 Push for DASH bootstrap**
Author	Cyril Concolato, Jean Le Feuvre, Nassima Bouzakaria

1 Introduction

During its 112th MPEG meeting, MPEG issued the 2nd Working Draft of MPEG-DASH FDH. This contribution focuses on the technical comments regarding the use of the HTTP/2 Push feature in that working draft and proposes to update it. Other editorial comments on the working draft can be found in m37248.

2 Review of the use of HTTP/2 Push in MPEG-DASH

HTTP/2 Push is a feature which allows servers to start delivering resources to clients without the client requesting them. It is based on PUSH_PROMISE frames and on the SETTINGS_ENABLE_PUSH settings.

The settings means that clients and servers actually declare their ability to process PUSH_PROMISE frames. If one of them cannot process them, they will not be used. However, settings can be updated during the lifetime of an HTTP/2 session. Finally, settings apply to a connection not to a stream, at a given time, either push is available for all resources or it is for none. A SETTINGS_MAX_CONCURRENT_STREAMS value of zero also disables server push, but does not disable the sending of PUSH_PROMISE frames. In all cases, a PUSH_PROMISE can be refused individually by the client by issuing a RST_STREAM frame.

Current announcements of HTTP/2 deployments do not support HTTP/2 PUSH. It may therefore be important to design a DASH solution which works with HTTP/2 without the push feature and which improves as the push feature is deployed.

The HTTP/2 standard notes that: "Server push is semantically equivalent to a server responding to a request". Given that HTTP/2 is a Full Duplex Protocol, the server issuing a PUSH_PROMISE for a resource can be replaced by the client making the request for it. We therefore recommend mentioning that approach in the WD. However, there are differences between using client requests and using server push, as follows:

- In one case the client needs to know the URL (and possibly byte range) to request the resource. In the other case, the server also needs to know the URL (and byte range) to push the resource.
  In DASH, URLs are known because they are documented in the MPD either explicitly (using SegmentList, SegmentBase with index ranges …), or implicitly using

SegmentTemplate. This means that for using server push, a server has to parse the MPD, or that a specific communication with the DASH generator is made (sockets, files …). Such specific communication may be made at the origin server, but will be difficult to do at edge servers. At the edge server, policies could be defined so that push promises coming from upper servers are relayed, but this would not be acceptable in all cases.

- The client has to determine when to issue the request to avoid 404, while the server (possibly using a communication channel with the DASH generator already mentioned) may know earlier the availability of a segment. In such case, the push feature could indeed help reduce the latency for receiving a segment. Note that the HTTP long polling approach as described in m36180 may avoid this latency issue for a peer-initiated request on a live edge segment.
- The use of server push features also probably reduces the upstream traffic, because fewer requests are made. The cost of the upstream traffic in HTTP/2 may be lower than for HTTP/1 due to the header compression mechanism and the benefit of using push is not so clear.
- Finally, one noticeable restriction of PUSH is that: "PUSH_PROMISE frames MUST only be sent on a peer-initiated stream that is in either the "open" or "half-closed (remote)" state."
  For DASH, this means that the PUSH PROMISE for a subsequent segment has to be sent before the closing of the response to the current request. This limits its usage. The URL (or names) of all segments which are meant to be pushed, for instance in a K-push, have to be known when sending the PUSH_PROMISE frames. This is especially problematic for live cases using segment timeline, since knowing the URL/byte-range in advance for several segments might not be possible…

In summary, the server push feature used for general segment delivery does not seem to be so beneficial (except for latency and overhead reasons to be evaluated) compared to issuing get requests for those segments. However, the server push feature could be more useful to push segments which are known to be available in advance, for instance at the same time as the MPD, like initialization segments. This is exactly what it was meant for: delivering static JavaScript or CSS resources available when the HTML page is.

3 Evaluation of the use of HTTP/2 Server Push for pushing initialization segments

This section evaluates the gain obtained by using server push to push initialization segments. These results are extracts from a research paper presented at MMSP 2015 at the same time as the 113th MPEG meeting [1].

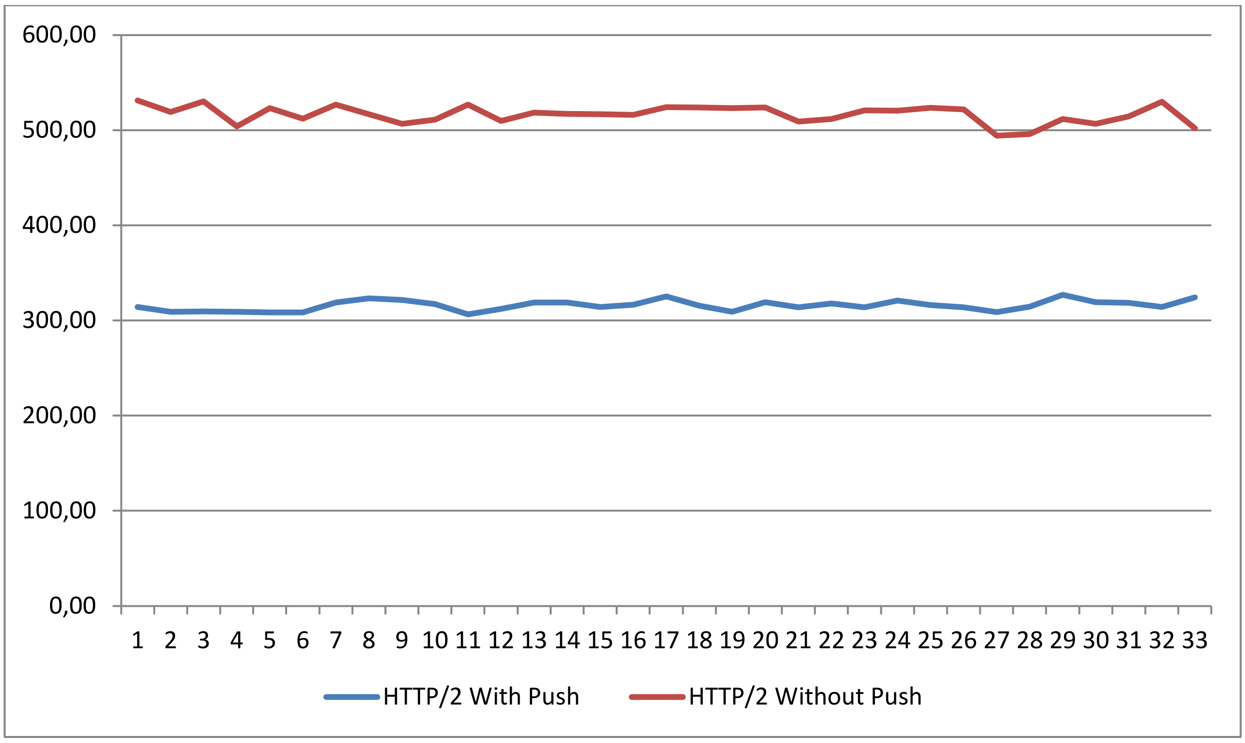
We deployed a Web Server supporting standard HTTP/2 using NodeJS and its http2 module. We instructed the server to start pushing all the initialization segments listed in the MPD when a request was made for this MPD.

We use the DASH.js client running in Chrome.

We deployed client and server in a network with an RTT of 50ms and a bandwidth of 2 Mbps. The server was using a TCP init congestion window of 10 TCP segments because this is realistic of what HTTP/2 server would use.

We used the DASH-IF Test vectors, only the non-encrypted ISOBMFF Live profile ones, and only the first period was used and then measured the delay between the initial MPD request and the end of reception of the last IS.

The results are shown below as the time difference (in milliseconds) between when the client makes the request for the MPD and when it receives the last IS. The x-axis shows the sequence number where sequences have been sorted by the total size (MPD+IS+HTTP headers)/



We see that whatever the sequence, the download delay is almost constant. This is due to the fact the MPD and IS are small (<10kB). This delay is high (>300ms) because of the use of TLS. However, we see that the HTTP/2 Push approach allows for a reduction of the delay by ~200ms. This is due to the fact that no additional round trip is needed for getting the initialization segments. However, we should make the following remarks:

- The DASH.js client makes the requests for all its needed IS at the same time, but Chrome does not yet seem optimized to benefit from the full duplex feature of HTTP/2 and still waits for IS Video to be received to actually requests the audio IS, as it would do in HTTP/1.1 to avoid the head of line blocking problem. The delay if this behavior would be optimized could probably be divided by 2, reaching 100ms.
- The NodeJS implementation of the HTTP/2 suffers from issues. An analysis of the HTTP/2 logs seems to indicate that the server is blocked for 20-30ms upon sending of each individual resource. This means that a better HTTP/2 server stack could lower the delay difference to 40-50ms.

4 Conclusion

We provided an analysis of the use of the server push feature in DASH. This analysis indicates that server push might be useful in scenarios when the URL is not known in advance, in particular during the start-up phase of a DASH session when MPD and IS are downloaded.

The experimental results reported in this contribution indicate that delivering some (even all) IS on HTTP/2 at the same time as the MPD can be beneficial to reduce the start-up delay in DASH sessions, even if the exact gain depends on the client/server implementations of HTTP/2.

We therefore recommend MPEG to include in FDH the ability for a client to indicate that it is willing to be pushed some or all IS upon requesting the MPD. The exact IS pushed could also be trimmed by the server using the existing HTTP "Accept" headers (such as codecs, language) or the HTTP Client-hints [2].

[1] "*Fast DASH Bootstrap"*, N. Bouzakaria, C. Concolato and J. Le Feuvre, MMSP, Xiamen, China, October 2015.
[2] "*HTTP Client Hints*", I. Grigorik, http://igrigorik.github.io/http-client-hints/