Comparative study of Video Streaming in HTTP/1.1, HTTP/2.0, HTTP/2 Adaptive Push and HTTP/3

Gautham Gunapati: 113274189, Anuroop Gubbala: 113260238, Yashwanth Madaka: 114353641

The code for this project is located here : Github , Presentation Link : Google Slides

1 Abstract

The use of OTT platforms has widely increased in the past few years. With the increase in popularity of OTT platforms like Netflix, Hulu, Prime, etc, video streaming has become increasingly popular. Video streaming contributes the highest bandwidth usage around the world. And in the past decade, HTTP protocols developed a lot. In this project, we made a decent attempt to setup, learn and compare video streaming under different HTTP protocols.

2 Introduction

In this project we have built 3 different servers which utilizes HTTP/1.1, HTTP/2.0 and HTTP/3 protocols. Additionally, we also implemented adaptive server-Push in HTTP/2 [HWZ+21]. We conducted multiple experiments under different protocols under varying latency and download speed in order to mimic the real world network conditions. At the end, we made a comparative study and analyzed the performance of video streaming under different protocols. We also tried to investigate whether or not using server push is advantageous for video streaming and analyze if there are any performance gains.

3 Experimental Setup

Initially, we started with video segmentation and HTTP/1 server setup. We have run few experiments to verify the setup. Later on we implemented HTTP/2 and HTTP/2 Adaptive server push. At the same time, we observed that the use of HTTP/3 youtube and added this protocol to the comparative study.

3.1 Video Segmentation

1. We used multimedia packager available in GPAC[GPA] called MP4Box. To prepare the existing videos for CMAF MPEG-DASH format. We fragmented video based on time for each segment using the option of DASH-FRAG ,thereby generating a MPD file.
2. This MPD file is used by the client for requesting next sequence numbers of the segments.
3. In our setup, we took a video of length 3 min 55 seconds. We divided the video into segments of 5 seconds interval. Each segments of the video is stored at the server with 4 qualities - 144p, 360p, 720p, 1080p. which are the usual available qualities for adaptive video streaming.

3.2 Client Setup

1. We built a basic HTML page consisting of a DASH video player(open source)[DAS].
2. In order to mimic the real world network conditions, we setup custom profiles in the chrome browser with varied download rates - 500 kbps, 1000 kbps, 2000 kbps and 4000 kbps.

3.3 Server Setup

1. We have written HTTP/1.1 and HTTP/2.0 in Node.js. For experimenting in HTTP/3, we used opensource python library called AIOQUIC[QUI].
2. For setting up latency between client and server, we used linux traffic control. We set the latency to 200 ms for all the experiments.
3. For capturing the data segments requested by each client and its metadata (time, size), we are storing them in a CSV format near server side to later use it for data analysis.

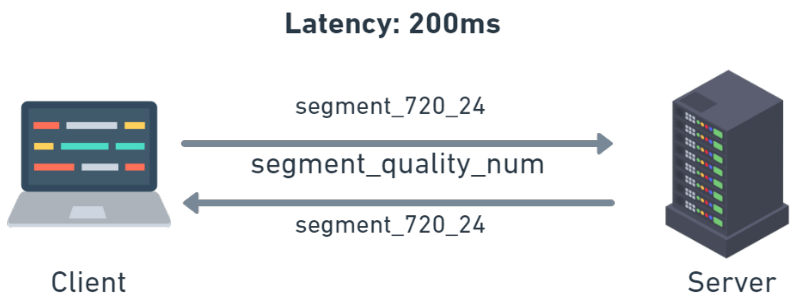


Fig 3.3.1. Client requests a segment that is delivered by server.

4 Approach

Before running our experimental setup, we made sure that both the client and server are connected to the same WiFi. To keep it simple, we haven't made any changes at the client side. We are using open-source player DASH for video playing. To test with adaptive server-push, we implemented server push logic in Node.js on top of HTTP/2 server.

4.1 Video

Quality index and corresponding bit-rates.

Quality	Bit-rate
144	400.0kbps
360	800.0kbps
720	1.6Mbps
1080	2.4Mbps

4.2 HTTP/1.1

we developed a simple HTTP/1.1 server developed in Node.js. For conducting experiments Chromium Browser is used as a client and open source DASH.js player is used to stream video.

4.3 HTTP/2.0

The same client was used again, but a HTTP/2 server written in Node.js was used to process the requests. In these tests, we didn't implement server push.

4.4 HTTP/2 Server Push

We have designed an adaptive server push algorithm[WS14] on top of HTTP/2 server which pushes the additional segments for a request from client. The implementation was totally done at the client level based on the state of the client.

4.4.1 Adaptive Server Push Algorithm

1. We implemented algorithm in such a way that there is a cap on the maximum number of segments that can be pushed for a request based on it's quality.

2. We initially started using adaptive push when there is the quality is greater than 360p, but later realised that HTTP/2 and HTTP/2 adaptive push behaved the same at lower download speeds. To show the variance between them, we are even doing adaptive server push for lower quality segments requested also. Below table shows the maximum cap set on the value k (number of segments pushed) for each quality.

Quality	Max K-Value
144	1
360	2
720	3
1080	4

3. Number of segments to be pushed was decided based on the buffer occupied (checked based on difference between the time of current stream requested and actual time of video being played) at the client side and when there is no change in segments quality continuously 3 times, we increment the number of segments pushed for a request. We repeat the same until we reach the max cap value of segments pushed for a particular quality. When a segment quality reduces at any instance of Adaptive push, the push index is set back to zero.[FSL21]

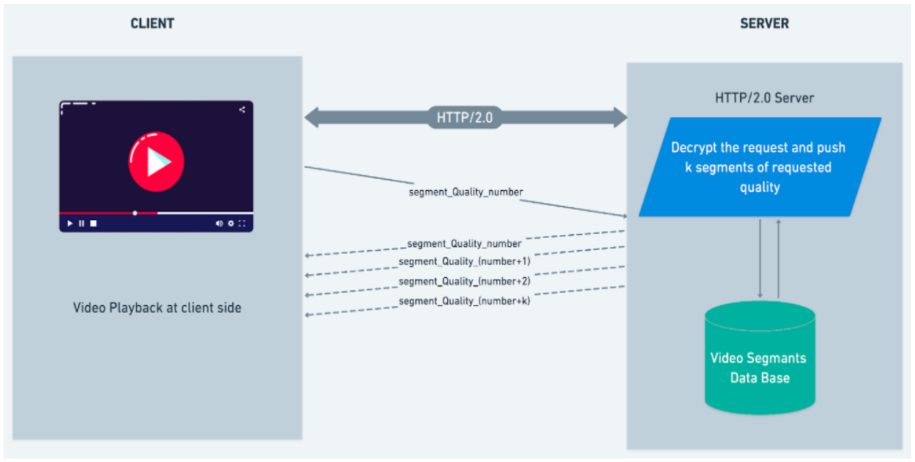

CLIENT
SERVER
HTTP/2.0
HTTP/2.0 Server
Decrypt the request and push k segments of requested quality
segment_Quality_number
segment_Quality_number
segment_Quality_(number+1)
segment_Quality_(number+2)
segment_Quality_(number+k)
Video Playback at client side
Video Segmants Data Base

Fig 4.4.1. Adaptive HTTP/2 server-push.

4.5 HTTP/3

We first tried to implement HTTP/3 in Node.js, but were unable to succeed because of the minimal documentation and we didn't find any proper open source implementation of the same. Hence, we used aioquic[QUI], which is a library in python which supports HTTP/3. So, we used it as a black-box and modified the source code to support video segment requests. For running experiments experiments we launched Chromium browser with extra flags enabled to support HTTP/3.

4.6 Experiments

For each Protocol(4 protocols) * For each download rate (4 speeds) * 10 experiments =**160 experiments**

HTTP/1.1 , HTTP/2.0, HTTP/2 push,HTTP/3.0 $\rightarrow$ 4 Protocols

500, 1000, 2000, 4000 kbps $\rightarrow$ 4 Downlaod rates

144p, 360p, 720p, 1080p $\rightarrow$ 4 different segment quality rates.

The experiment commands are return in README.MD of our github source code.

4.7 Extracting Data

For each and every experiments for all the above protocols, we extracted the following information at server based on the client requests.

1. Segment quality of each request.
2. Timestamp of the request.
3. Size of the segment requested.

A Node.js was written in the server to capture the above data for HTTP/1.1 and HTTP/2.0 and HTTP/2 server push protocols. For HTTP/3, a python script was written to do the same. The data was stored in a CSV format and was later used to perform data analysis using pandas and seaborn.

5 Results

5.1 Graphs

5.1.1 HTTP/1.1

The below figure shows the results of the tests ran under HTTP/1.1 server. The plot shows the change in quality of the video over time. We can see that the fluctuation in the quality of the video is very minimal.

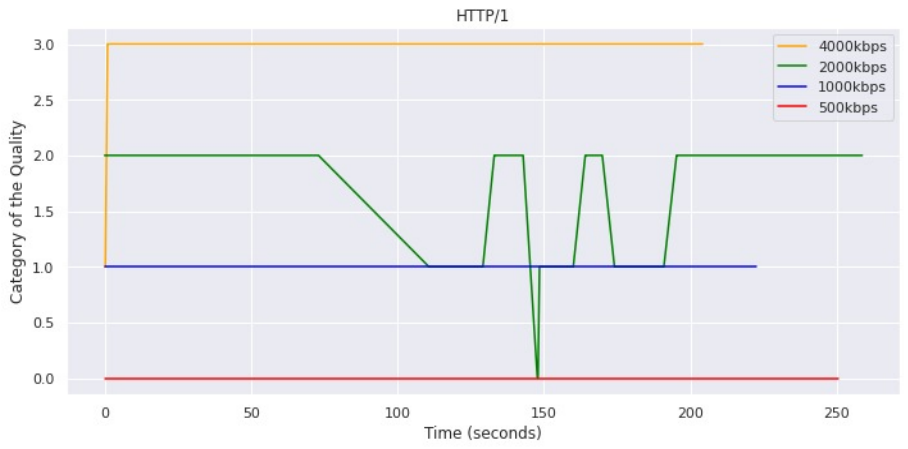


Fig 5.2.1. Quality of video over time for all the download speeds for HTTP/1.1

5.1.2 HTTP/2.0

The below figure shows the results of the tests ran under HTTP/2.0 server. The plot shows the change in quality of the video over time as the video progresses. We can see that there is a lot of fluctuation and the quality level is not as stable as HTTP/1.1.

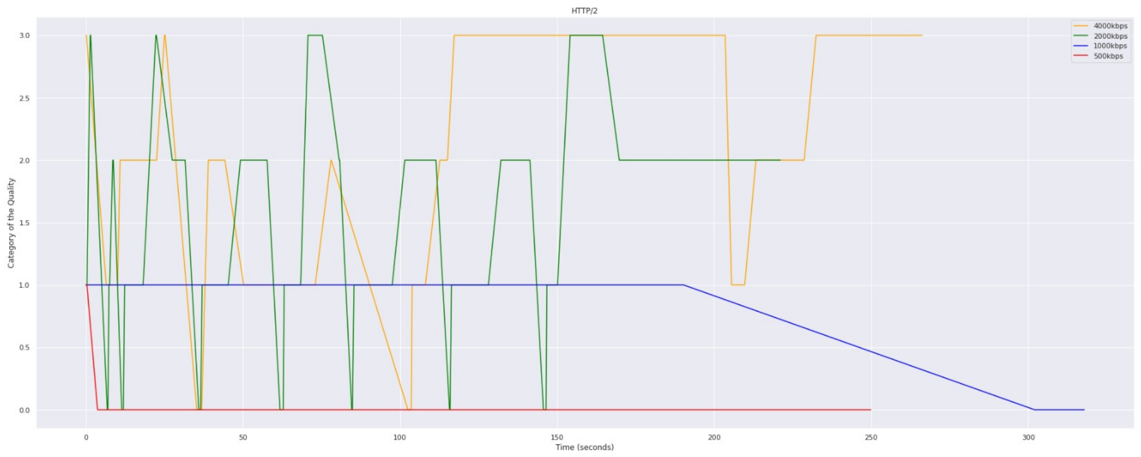


Fig 5.2.2. Quality of video over time for all the download speeds for HTTP/2.0

5.1.3 HTTP/2 Server Push

The below figure shows the results of the tests ran under HTTP/2 server with adaptive server push implemented in it. The plot shows the change in quality of the video over time. We can see that there is a lot of fluctuation for all the download speeds. An important observation to note is that even when the download speed is high i.e.4Mbps, the segment quality dips to lowest quality 144p. This is due to the fact that when additional segments are pushed, the ABR algorithm misinterprets that the available bandwidth has reduced and the client starts requesting for video segments of lower quality.

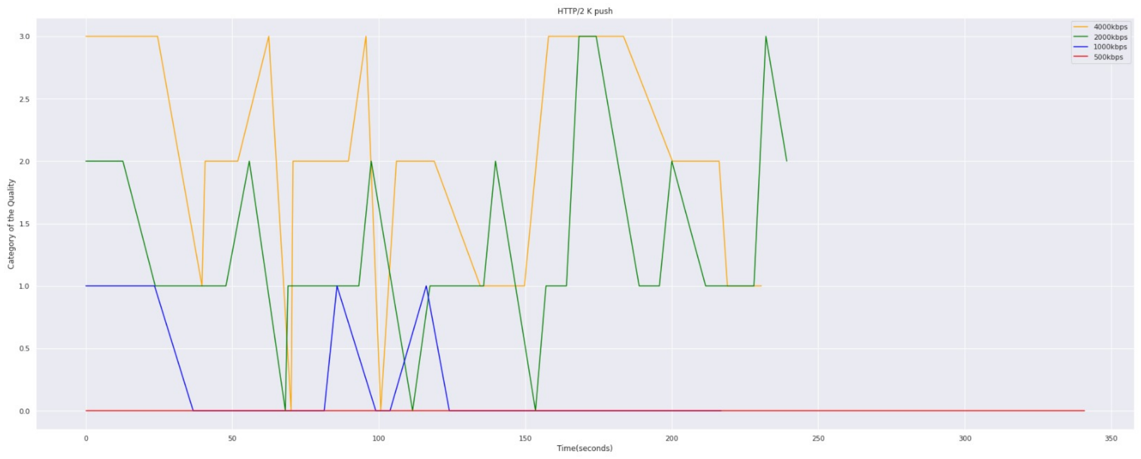


Fig 5.2.3. Quality of video over time for all the download speeds for HTTP/2 Server Push

5.1.4 HTTP/3

The below figure shows the results of the tests ran under HTTP/3 server. The plot shows the change in quality of the video over time. We can see that for lower download speeds and higher download speeds, there is very less or no fluctuation in quality of video segments.

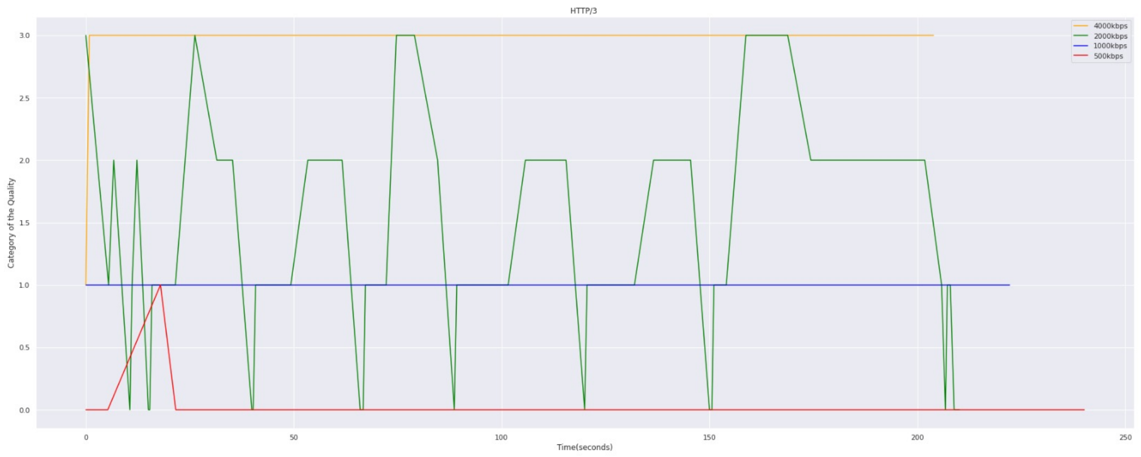


Fig 5.2.4. Quality of video over time for all the download speeds for HTTP/3

5.2 Comparisons

In this section, we compare the performance of each protocol under different download speeds and try to infer which one performs better and if there is any advantage of using server push over HTTP/2.0.

5.2.1 Quality vs Download Speed

We have calculate the average quality of the segments requested for all the protocols under different download speeds. The figure shows that the average quality of the segments requested in HTTP/1.1 and HTTP/3 is comparatively higher than HTTP/2.0.

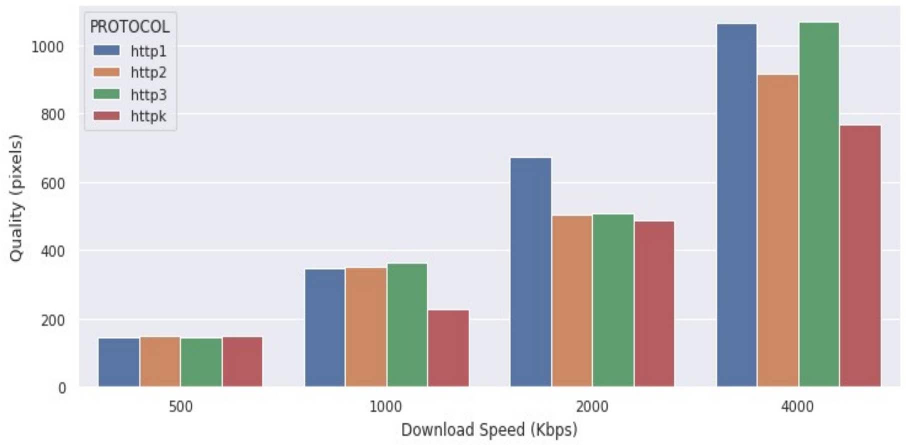


Fig5.3.1. Comparison of Quality under different download speeds.

From the above information, it is evident that HTTP/2.0 has comparatively lower quality than other protocols. To dig deeper into why this is happening, we compared the variance in video quality of segments to get a better picture.

5.2.2 Variance vs Download Speed

We have calculated the variance which tells how many times the quality has changed on an average for experiments at given download speed and latency. The below figure shows that HTTP/1.1 and HTTP/3 have the lower variance when compared to HTTP/2.0.

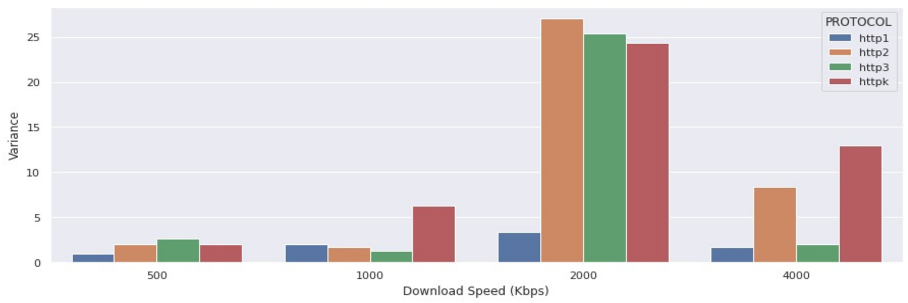
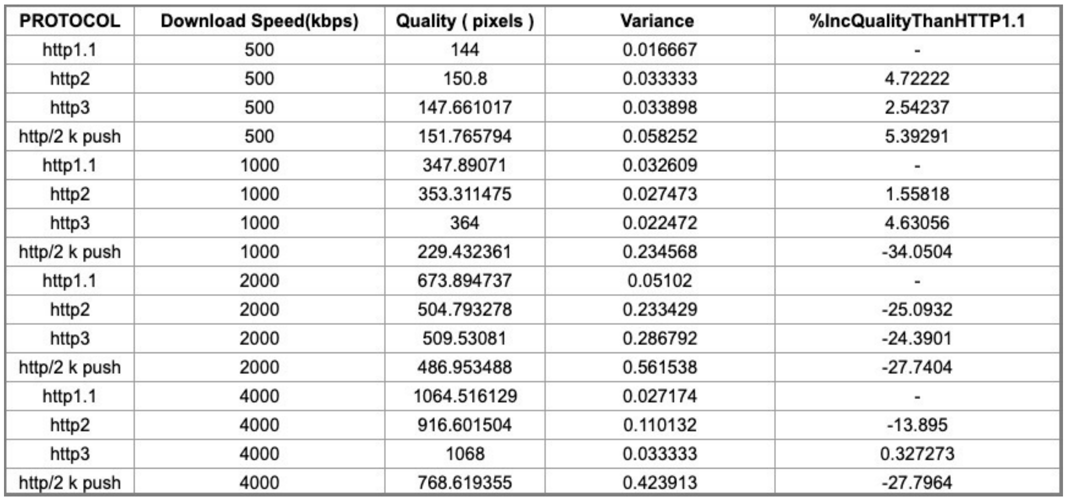


Fig5.3.2. Comparison of Variance in Quality under different download speeds.



PROTOCOL	Download Speed(kbps)	Quality (pixels)	Variance	%IncQualityThanHTTP1.1
http1.1	500	144	0.016667	-
http2	500	150.8	0.033333	4.72222
http3	500	147.661017	0.033898	2.54237
http/2 k push	500	151.765794	0.058252	5.39291
http1.1	1000	347.89071	0.032609	-
http2	1000	353.311475	0.027473	1.55818
http3	1000	364	0.022472	4.63056
http/2 k push	1000	229.432361	0.234568	-34.0504
http1.1	2000	673.894737	0.05102	-
http2	2000	504.793278	0.233429	-25.0932
http3	2000	509.53081	0.286792	-24.3901
http/2 k push	2000	486.953488	0.561538	-27.7404
http1.1	4000	1064.516129	0.027174	-
http2	4000	916.601504	0.110132	-13.895
http3	4000	1068	0.033333	0.327273
http/2 k push	4000	768.619355	0.423913	-27.7964

Fig5.3.2. Comparison of Variance in Quality under different download speeds.

Since the variance is more in HTTP/2.0, we compared the number of requests to check whether there are more video segments requested or not.

5.2.3 Total No. of Requests vs Download Speed

From the below figure, we observe that the number of requests in the case of HTTP/2 adaptive push (HTTP-k) is the lowest. This is due to the fact that

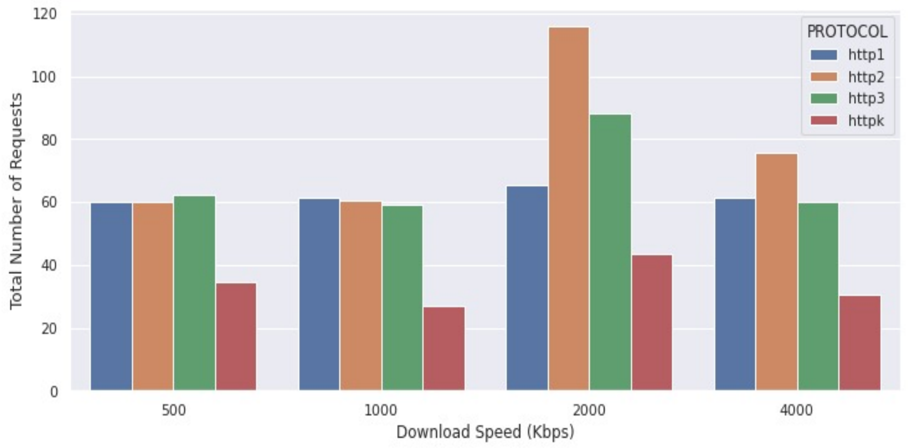


Fig5.3.3. Comparison of Total number of segment requests under different download speeds.

5.2.4 Total Time vs Download Speed

We extracted the total time taken to complete all the video segments to be requested. This clearly shows that adaptive server push increases the **stalling rate** at lower download rates. As the downlaod rates increase, the difference between the times taken among all the protocols is reduced.

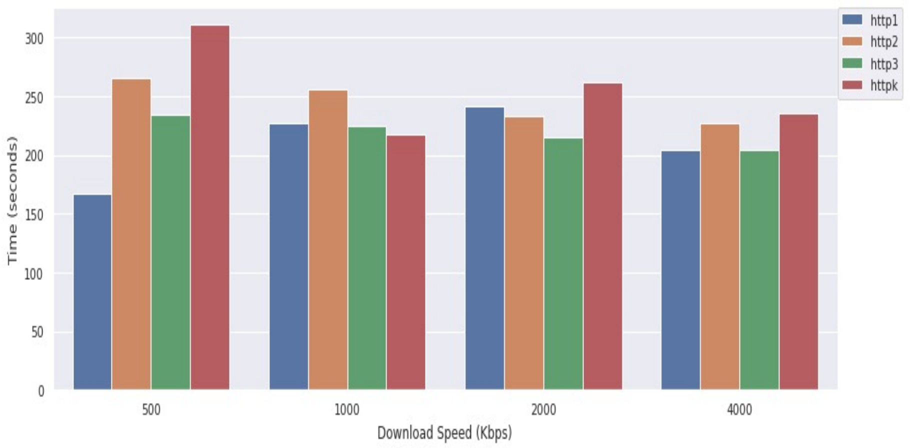


Fig5.3.4. Comparison of Total time taken under different download speeds.

6 Observations

1. The HTTP/1.1 has the most stable results of all the different download rates.
2. HTTP/2 has the highest variance and number of segments requested when there is no adaptive push.
3. Though HTTP/2 adaptive server push reduces the number of segments of segments requested when compared to HTTP/2, but the variance is higher and average segment quality is lower when compared to other protocols.
4. HTTP/3 results are comparable to HTTP/1 (except for download speed of 2000 kbps). but the variance of segments requested int HTTP/3 is slightly on a higher side.

7 Extra 30

1. We have made an attempt to conduct a comparative study of all the 3 protocols (HTTP/1.1, HTTP/2, HTTP/3) for video streaming.
2. We haven't found any comparative study for the same before and assuming it is a **first attempt**.
3. We have extended our initial problem statement and included **HTTP/3** protocol in our study.

8 Conclusion

To conclude, it is observed that there is a decrease average video quality when using HTTP/2. We are not conclusive on what is the major reason which contributes for this. Adaptive server-push also doesn't show any significant improvement in the performance except for **reducing the number of video segments requested**. We see that HTTP/2 is not a better alternative for video streaming when compared to HTTP1.1 or HTTP/3 based on only our experiment data. This might also be due the softwares we used (Dash.js) are more compatible with HTTP/1.1.

Also HTTP/3 performs slightly little better than HTTP/1.1 as the download speeds are increasing. As the world is moving towards 5G and the increase in bandwidths might be the reason for why some companies like Facebook and Youtube have already moved to QUIC and HTTP/3 for their video streaming service.[FB]

References

[DAS] DASH. https://github.com/dash-industry-forum/dash.js?

[FB] FB. https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/.

[FSL21] Huei-Wen Ferng, Shan-Hsiang Shen, and Chih-Wei Lai. Video streaming over http/2: Design and evaluation of adaptive server-paced push. *Journal of Communications and Networks*, 23(2):106–116, 2021.

[GPA] GPAC. https://github.com/gpac/gpac/wiki/mp4box.

[HWZ$^+$21] Shouqin Huang, Zhiwen Wang, Weizhan Zhang, Haipeng Du, and Qinghua Zheng. Adaptive video streaming using dynamic server push over http/2. In *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 673–678, 2021.

[QUI] QUIC. https://github.com/aiortc/aioquic.

[WS14] Sheng Wei and Viswanathan Swaminathan. Cost effective video streaming using server push over http 2.0. In *2014 IEEE 16th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–5, 2014.