VTT Technical Research Centre of Finland

Exploiting and Evaluating Live 360° Low Latency Video Streaming Using CMAF

Uitto, Mikko; Heikkinen, Antti

Published in:
2020 European Conference on Networks and Communications, EuCNC 2020

DOI:
10.1109/EuCNC48522.2020.9200954

Published: 01/09/2020

Document Version
Peer reviewed version

Exploiting and Evaluating Live 360° Low Latency Video Streaming Using CMAF

Mikko Uitto
VTT Technical Research Centre of Finland Ltd
Kaitoväylä 1, 90570 Oulu, Finland
Email: firstname.lastname@vtt.fi

Antti Heikkinen
VTT Technical Research Centre of Finland Ltd
Kaitoväylä 1, 90570 Oulu, Finland
Email: firstname.lastname@vtt.fi

***Abstract*—In this paper, we describe the implemented 360° live video streaming architecture and evaluation setup running in the 5G test network (5GTN), which can be considered beneficial for live educational services and providing a fast situation awareness in medical sector. The concentration is on providing more global HTTP adaptive streaming (HAS) architecture using also CDN components rather than having only a point-to-point solution. With this setup, we evaluate the common media application format (CMAF) usage in HAS context in order to have reliable video transmission method for high bitrate 360° video as well as setting the end-to-end (e2e) latency as low as possible. By this, both uplink and downlink direction are investigated by using wired, wireless, or mobile connection. In addition, the experienced e2e latency is measured. The extensive placement of synchronized network measurement indicators from distinct network nodes help to identify the bottlenecks in mobile video delivery path and verify the CMAF-based usefulness in streaming optimization. 5G key performance indicators delay and throughput are the key parameters in 360° live streaming, which can be useful in forming the overall, live view of an educational surgery or even from an accident scene. In such scenarios, mobile networking can be the easiest and often the only option for outputting the important information to relevant users, such as students or doctors.**

I. Introduction

The recent advances in mobile network technology have opened new target of applications and usage in various industry sections, which are often relying on traditional wired networking methods. Traditional fields such as medical healthcare, heavy industry in harbours, and authorities can benefit from high-speed wireless connection especially for live video streaming scenarios. For example, receiving hospital can benefit from live video when monitoring the patient real-time in ambulance in order to provide better first-sight vision. Also, real-time educational surgery from a treatment room is possible if real-time feedback for students is a requirement. Furthermore, remote operations in conjunction to such scenarios are also dependent of the live video stream. Although the evolution of current wireless local area networks (WLANs), such as in 802.11ad, enable significant throughputs up to several gigabits per second their use in mobile live streaming scenarios is often limited due to range of signal coverage. The improved network mobility with current LTE/5G technology enables not only place-independent client access to streaming services, but also mobile video production scenarios in the field.

Live video streaming techniques today varies depending on the target applications, use case and slightly of the used access network technology. Earlier more traditional real-time based protocols, such as real-time streaming (RTSP) or real-time messaging protocol (RTMP) in conjunction with real-time transport (RTP) and user datagram (UDP) protocols have been used in such streaming scenarios, but they often need dedicated data transmission path where access against firewalls and network address translation (NAT) is allowed. In addition, streaming to unknown number of audience for these protocols without using more efficient multicast/broadcast based protocols is considered inefficient when thinking network data usage. Thus, transmission errors for vital medical content should be avoided when considering information sharing from the ambulance towards target hospital.

In order to overcome firewall issues alongside with erroneous data possibility, hypertext transfer protocol (HTTP) on top of transmission control (TCP) has become more generalized in live video transmission. Pull-type transmission initialized by the client is more efficient for the service providers. Many of the popular service providers, such as YouTube, are relying on HTTP streaming as it favours for guaranteed transmission procedure and certain flexibility in terms of the transmission setup although with cost of higher delay. This is one of the key questions in our video streaming work; how to minimize the latency by using the available open-source components and optimal setup parameters. In our work, we define the network delay as one-way and stream latency as the e2e delay between camera capture and display in client.

HTTP Adaptive streaming (HAS) dominates over-the-top (OTT) based consumer video distribution today both in live as well as video on demand (VoD) streaming and is often combined with content delivery networks (CDN)s. In HAS the video stream is divided into a sequence of short media segments and delivered to the client using common HTTP servers. With traditional HAS, typical latency is over 6 seconds and it mainly depends on the media segments length, which are typically 1 to 10 seconds. The encoder needs to create a full segment before sending it to the CDN before client can download it. Usually client's player downloads two or more segments to the playback buffer to guarantee uninterrupted playback. Hence, the encoding and playback are generating most of the e2e latency.

One option to reduce the e2e latency without shortening segment duration is to use chunked transfer encoding, where the data in the segments is composed of a series of chunks and the encoder can send the chunks to the player immediately after their encoding [1]. For example if encoder produces 5-second segments at 25 frames per second (fps) and puts one frame

per chunk it can be forwarded after 40ms without the need for waiting 5s until the whole segment is ready. Commen media application format (CMAF) [2] is a MPEG standard to simplify delivery of HAS so that Apple's HTTP Live Streaming (HLS) [3] and MPEG's Dynamic Adaptive Streaming over HTTP (DASH) [4] can use the same media segments and chunked encoding feature.

Swaminathan et al. [5] presented first studies how HTTP chunked encoding approach can reduce the latency in live video streaming. They used 1s chunks and achieved 2.5s latency. Bouzakaria et al. [6] validated the very low latency streaming in their system where encoder and DASH-aware web server are in the same computer. When the web server receives a request from a client it parses the segment to detect thee fragments and sends these as a chunk to the client. They can achieve only 240ms latency in this system. Essaili et al. [7] presented a prototype implementation of chunked transfer encoding in low latency DASH delivery system. They evaluated the system distribution latency in WiFi connection with different fragment duration.

One of the good 5G enhanced mobile broadband (eMBB) use cases contains high throughput video streams, such as 360° streams that can be beneficial of forming situation awareness of a scene. 4K and 8K resolution require huge amount of bits as well as bandwidth, but are still manageable with the current LTE and 5G new technologies. 360° live streaming is usually one option for realising virtual reality (VR) streaming, which has been also conducted using DASH [8] [9].

In this paper we present our system architecture for low latency 360° CMAF streaming using DASH to be used for example medical and healthcare related scenarios. In section II, the proposed system architecture will be presented that is functional in 5G test network (5GTN). Section III presents the detailed evaluation setup with the applied measurement tools and results. Finally, a conclusion with future work aspects will be done.

II. System architecture

The chunked CMAF low latency live video streaming system architecture contains a live capturing camera, video encoder, live packetizer, origin, edge, and HTML5 player. The overview of the system architecture is presented in Fig. 1 and it is operational in 5GTN. 5GTN [10] is one of the hybrid 4G-5G test network infrastructures coordinated by 5G Test Network Finland (5GTNF), located at VTT's 5G laboratory and University campus areas in Oulu, Finland. It is currently connected to multiple test sites and introduces several 5G components, such as multi-access edge computing (MEC), and evolved multimedia broadcast multicast service (eMBMS) with CDN. Radio access technology (RAT) is LTE Advanced (LTE-A) using the carrier aggregation (CA) technique operating in frequency division duplex (FDD) mode on around 2.6 GHz and time division duplex (TDD) aroung 2.3 GHz. Currently the core network functionalities are handled mainly by evolved packet core (EPC) simulator. 5G new radio (NR) is expected to be integrated in the 5GTN during spring 2020.

A live video camera is connected to the encoding computer through HDMI to USB capture card. The modern cameras are also capable of producing and transmitting directly the encoded video stream e.g. via RTMP, but we want to include also small, powerful, separate encoding PC in the architecture in order to have possibility e.g. to form audio feedback channel in the future. In addition, network delay measurement realization is easier with such machine and not directly from application-restricted camera. The *Live Encoder* compresses the live video feed with low latency settings and streams the video to the RTMP server located in the *Live Packetizer*. The encoding PC is connected to the 5GTN and is referred as the uplink (UL) server.

The *Live Packetizer* contains RTMP server and CMAF packetizer. The RTMP server receives the live video feed and serves the stream to the CMAF packetizer. It packs one frame to a fragment (chunk) and pushes the fragment using HTTP PUT chunked transfer encoding to the *Origin* server. The *Origin* is a part of the open-source CDN together with edge servers and enables effective HTTP video delivery when multiple users need to access the same content. The *Origin* receives the HTTP PUT request from the CMAF packetizer. The *Edge* server will shield the *Origin* server from load that simultaneous requests from distinct players aka clients can cause. When a client requests a video segment using HTTP PULL from the server, the *Edge* server distributes video segments in chunks to the player. If the requested data is not available in the *Edge* server, it requests the data using HTTP PULL from the *Origin* and forwards it to the *Video Player*. The *Edge* stores the requested data for a certain time according to its caching rules. The *Origin* and the *Edge* both use the chunked transfer encoding.

Open-source HTML5 player Dash.js [11] supports CMAF low latency streaming and is utilized in our architecture as the *Video Player*. It is a reference DASH client implementation for playing videos via Javascript libraries in a browser using Media Source Extensions API. The low latency implementation in Dash.js uses W3C Fetch API and chunked transfer encoding. HTML5 player is also connected to the 5GTN and is referred as the downlink (DL) application.

III. Evaluation and results

A. Evaluation setup

This section presents our evaluation setup that is a detailed version of the system architecture in Fig. 1. The video streaming path introduced next was evaluated against delay in different networks as test cases: wired (Ethernet), WLAN (802.11ad), and mobile (LTE-A), which are available in our test network (5GTN). Live video content was used as the input, but drastic movement in the feed was avoided for keeping the bitrate relatively constant and to have comparable set of runs.

The evaluation setup for chunked CMAF low latency live 360° video streaming system has been built using open source software and the overview is presented in Fig. 2. All the servers run Ubuntu 18.04 operating system. 360° streamer depicted in the figure consists of Insta Pro 2 360° camera and Intel NUC *Live Encoder* with Intel Core i7-8809G CPU connected via HDMI to USB 4K capture card. The camera functions with real-time stitching forming the video with 4K resolution at 30 fps in equirectangular format. The *Live Encoder* captures the video from USB 3.0 port in 4:2:0 YUV format nearly 3 Gbit/s. The video is encoded using x264 software encoder with 13

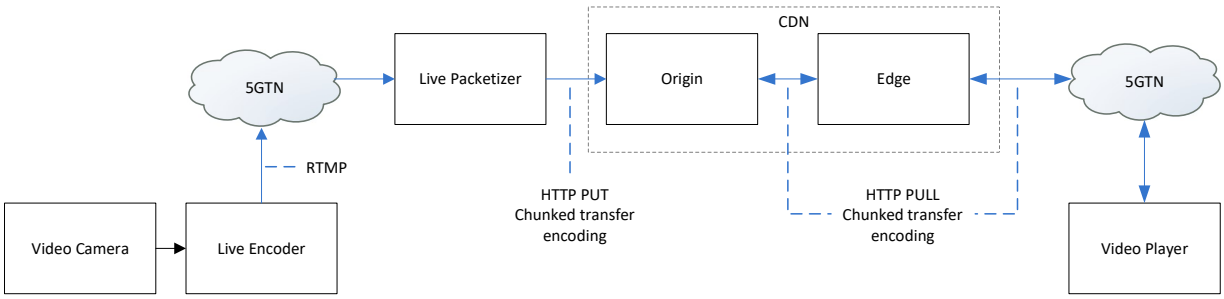

Fig. 1: System architecture for low latency CMAF video streaming.

Mb/s target bitrate and low latency settings (preset=superfast, tune=zerolatency, profile=high, bframes=0, ref=3, scenecut=0). The *Live Encoder* is connected to the network using either wired, WLAN or mobile connection depending on the test case. The encoder streams the video to the *Live Packetizer* (see Fig. 2) via RTMP.

The *Live Packetizer* contains RTMP server (Nginx with rtmp-module) [12] and FFmpeg for chunked CMAF packaging. The RTMP server receives the live video feed and serves this stream to FFmpeg. FFmpeg generates 4-second long segments and packs one frame to a chunked fragment and pushes the fragment using HTTP PUT request method to the *Origin* server. The *Origin* server contains two components: an ingest and a shield (edge). The ingest is based on the open source Streamline server written in Go [13] and it receives the HTTP PUT request from CMAF packetizer. The shield is based Nginx and it will protect the ingest server from load that players' simultaneous requests can cause by caching the requested data for two seconds. The shield server distributes video to the player using chunked transfer encoding. As a *Video Player* we used laptop with Ubuntu 18.04 and Intel Core i7-4800MQ CPU that runs Dash.js player version 3.0.1 in Chrome browser version 80.0 and FFplay media player. These players does not include support for 360° videos and they show the video in 4K resolution.

The delay measurements were divided into two methods: measurements with network performance indicator Qosium and measurements using screen clock capturing. We wanted to identify how e2e latency is divided in the live video streaming system. In the first method, the streaming video camera captures the current time in milliseconds from screen 1. Player plays the video in screen 2 that is alongside screen 1. We take a picture of the screens and calculate the time difference, which is e2e latency. We used FFplay when we measured the latency in the middle of the system. The FFplay was used with format flag *nobuffer* that reduces the latency introduced by buffering and flag *low_delay* that forces low latency playback. We used Dash.js when we measured the e2e latency with CMAF. We set the *liveDelay* to 1.0s (target delay), *liveCatchUpMinDrift* to 0.05s (Minimum delay deviation allowed before catch-up activation), and *liveCatchUpPlaybackRate* to 0.5% (Max. catch-up rate).

Secondly, Qosium [14] was used for measuring the uplink and downlink network performance as it can provide a real-time, passive, and light-weight measurement method for reliable assessment without generating additional congestion traffic into the network. Qosium consists of two components, namely as Probe and Scope. The Scope is responsible for visualizing and/or gathering the results, but it is not part of the measurement path and requires only network connection to one of the probes. The Probes are installed and run in the nodes that serve as endpoints of the measured network path and pass barely the measured traffic through them. Fig. 2 shows the Probe and Scope nodes in our evaluation case. The uplink measurement path holds Probes in *Live Encoder* and *Live Packetizer*, and downlink measurement Probes in *Origin* and *Video Player*, respectively. Scope was placed in external laptop for gathering the results, connected to Probe located in *Origin*. In our measurements, delay, and throughput are the main indicators visualized in the results. Thus, indicators such as packet loss, delay, and jitter were also monitored. The video bitrate was fitted to the uplink capacity in order to avoid packet losses.

TABLE I: Used video and network parameters.

Video		**Network**	
Resolution	3840x2160 (4K)	Wired network	1 Gbps Ethernet
Bitrate	13 Mbps (CBR)	WLAN AP	802.11ad @ 2.4 GHz
Frame rate	30 fps	LTE BS	eNB pico cell @ 2.6 GHz
Coding	H.264	LTE Frequency	Band 7 @ 10 MHz
Packetization	CMAF (DASH)	LTE Modulation	2x2 MIMO
DASH segment length	4s	LTE EPC version	EPC simu LTE WTS

One of the important issues when using Qosium (or any other delay-to-point measurement tool) for measuring the network delay is setting the time synchronization as accurate as possible for distinct machines. Basically three alternatives exist: network time protocol (NTP), precision time protocol (PTP), and global positioning system (GPS) time. Our several pre-measurement tests indicated that NTP was not accurate enought for synchronization and decided to use PTP. Open-source PTP daemon [15] was used in our measurements. First we set a common PTP master machine in our 5GTN from which the PTP slaves (Probe machines) get the clock synchronization via wired Ethernet network. Naturally, we configured the routing for each component in the evaluation setup so that video was routed via desired network interface in each test case.

B. Results

As depicted in the previous section, the rough division for delay measurements was divided between Qosium network

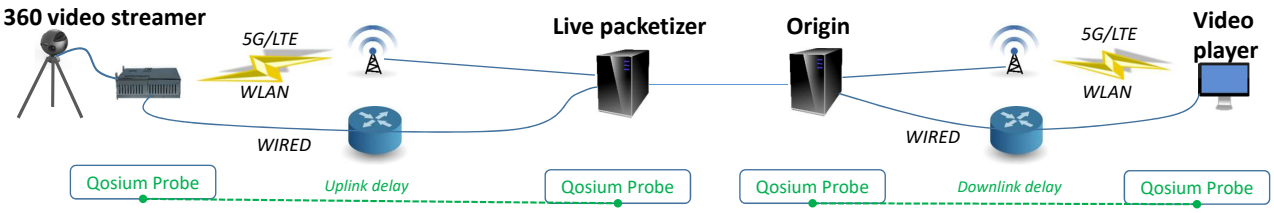

Fig. 2: Evaluation setup.

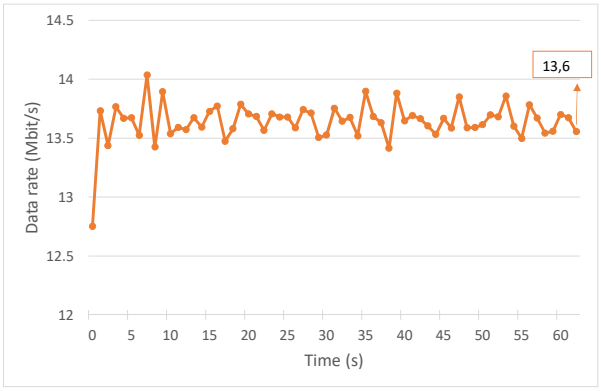

Fig. 3: Measured average bitrate for the live video stream.

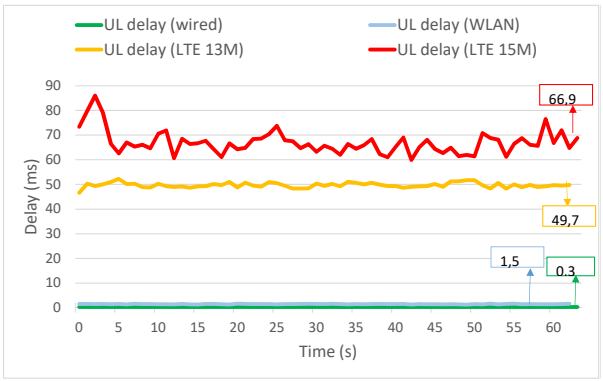

Fig. 4: Measured average video delays in the UL.

measurements in the application layer and true latency measurements by taking screen captures from live digital clock both between sender and client. The evaluation was performed using wired, WLAN and mobile network inside 5GTN. For mobile, we also wanted to see how increased bitrate influences to delays in uplink and downlink. Some of the essential video and network parameters are illustrated in Table I. In addition to this, the mobile network capacity was approximately 20 and 60 Mbit/s for uplink and downlink, respectively.

1) Network delay: The network measurements were accomplished by using Qosium that gathers and measures the through-traffic with application layer packets. The paths of interest were uplink transmission between video encoder and live packetizer as well as downlink transmission between *Origin* and *Video Player*. The delay between packetizer and origin was not measured since the connection in our architecture is always wired via Ethernet/fiber and therefore this delay is considered extremely small. We measured 10 distinct 60s live streaming runs for each of the network scenarios; wired, WLAN, and mobile. Finally, an averaged results were conducted. The averaged data (bit) rate for our live measurements is presented in Fig. 3 and measured with Qosium as the output rate from video encoder before the uplink transmission. The box in the graph shows the total average bitrate, which is 13.6 Mbit/s. Constant bitrate (CBR) factor enabled in the encoder works relatively well and the bitrate stays within 200-300 Kbit/s median. Thus, the earlier FFmpeg encoding experiments done with lower resolution seem to maintain the bitrate even better towards the target rate.

Fig. 4 presents the uplink delays with average 60s values against each of the network types. In the uplink scenario RTMP stream is transmitted on top of TCP. As can be seen, the uplink delay in wired and WLAN scenarios is extremely fast compared to mobile measurements that is approximately 50ms for the 13 Mbit/s stream. In addition, 15 Mbit/s stream leads even increased delay. We derived that this arises from close up eNB congestion by the video traffic leading to TCP retransmissions in IP level and increased delays in eNB. As the average delay for our WLAN 802.11ad wireless router is close to 1ms, it can be considered close to upcoming, real 5G values launched in the 5GTN.

Fig. 5 shows similarly the downlink delays against different network types for CMAF packetized RTMP stream. The downlink delay is relatively the same for 13 Mbit/s and 15 Mbit/s and much lower than in the uplink side is caused mainly by eNB packet scheduling and high data rate video.

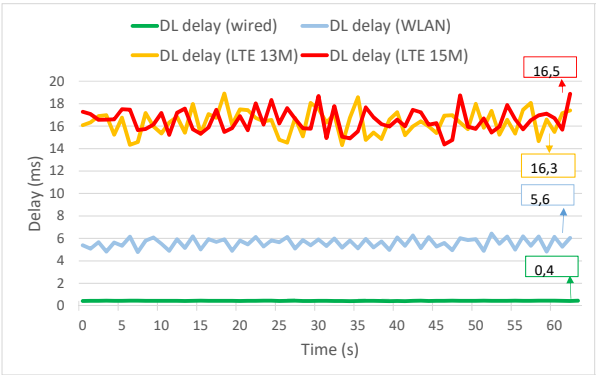

Fig. 5: Measured average video delays in the DL.

TABLE II: Measured capturing and e2e latencies.

Test case	Measured e2e latency	buffer size	actual delay
capturing latency	0.43s	-	-
e2e RTMP latency	0.58s	-	-
e2e CMAF latency (wired)	1.29s	0.70s	0.59s
e2e CMAF latency (mobile)	2.01s	1.29s	0.72s

2) End-to-end latency: In the latter case we first measured capturing latency between 360° camera and *Live Encoder*, illustrated in Table II. FFplay was used to playback the video from video capture interface in the *Live Encoder*. In this case the latency includes only capturing and playback. Then we measured e2e RTMP latency when FFplay in laptop plays video from RTMP server. Now the latency includes capturing, encoding, delivery to RTMP server, delivery to player and playback. Finally, we measured the total e2e CMAF latency from camera to the Dash.js running in the laptop with Chrome browser. In this case the latency consists of capturing, encoding, delivery to RTMP server, live packetization, delivery to origin, delivery to player, and playback.

Each measurement took 60s to calculate the average size of the player buffer and we took a picture of the screens every three seconds. Based on the pictures we calculated the e2e latency. We repeated the measurement 10 times to ensure the correct averaged results for wired and mobile tests. Table II presents the average values for the latencies. In each test the player's playback rate was 1 and there were no stalls during the playback. In the early e2e measurements with the browser, we noticed that the realized latency actually varies depending on the moment browser is started, which also influenced to player's buffer size. In the wired network the measured e2e latency varied from 0.95s to 1.82s and the average was 1.29s. The player buffer size varied form 0.36s to 1.23s and the average was 0.70s. In the mobile network the measured e2e latency varied from 1.74s to 2.28s and the average was 2.01s. The buffer size varied from 1.1s to 1.5s and the average was 1.29s. The player's buffer size had a significant impact on e2e latency when the actual e2e latency was relatively constant. Therefore, we conducted that the actual e2e latency (camera capture + encoding + packetizing + network + playout) can be calculated by subtracting buffer size from the measured e2e latency. The actual e2e latency in the wired network was 0.59s and in the mobile network 0.72s. However based our measurement the player buffers 0.6s more on mobile than on wired network when the actual e2e latency is 0.12s higher.

IV. Conclusion

This paper illustrated our live system architecture and evaluation for realizing 360° video streaming setup using fast video capture with real-time encoding, live packetizing with CMAF connected to CDN, and low latency player. This setup was tested against wired, wireless as well as mobile LTE-A connectivity using the 5GTN in Finland.

The extensive results gathered both by using network performance monitoring as well as camera capturing indicators show that the current setup is capable for outputting as low as one second end-to-end latency by using chunked CMAF packetization, which enables faster, fragment-level media transmission also in HTTP streaming. This transmission procedure combined with minimized player buffering allows good performance with CMAF, which usually leads to several seconds of experienced delay. One of the use cases within our streaming setup is considered in medical/health care scenarios, where live, near real time 360° streaming can be beneficial remote consultation or remote learning scenarios.

The authors plan to continue the development of the existing testbed and refine each of the components even better with 5G new radio (NR). 360° low latency video player implementation and integration into VR glasses will be one of the improvements. Mobility tests especially in the uplink transmission will be conducted and novel configuration or even novel multiconnectivity algorithms for guaranteeing the uplink capacity will be investigated.

Acknowledgment

This work was supported by the European Commission in the framework of the H2020-ICT-19-2019 project 5G-HEART (Grant agreement no. 857034) and national PRIORITY and 5G-FORCE projects funded partially by Business Finland. The authors would like to thank for the support.

References

[1] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," Internet Requests for Comments, RFC Editor, RFC 7230, June 2014. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7230.txt

[2] MPEG, "Information technology — multimedia application format (mpeg-a) — part 19: Common media application format (cmaf) for segmented media," ISO/IEC 23000-19:2018, 2018.

[3] R. Pantos and W. May, "HTTP Live Streaming," Internet Requests for Comments, RFC Editor, RFC 8216, June 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8216.txt

[4] T. Stockhammer, "Dynamic adaptive streaming over HTTP standards and design principles," in *ACM MMSys'11*, San Jose, California, USA, feb 2011, pp. 133–144.

[5] V. Swaminathan and S. Wei, "Low latency live video streaming using http chunked encoding," in *2011 IEEE 13th International Workshop on Multimedia Signal Processing*, December 2011, pp. 1–6.

[6] N. Bouzakaria, V. Concolato, and J. Le Feuvre, "Overhead and performance of low latency live streaming using MPEG-DASH," in *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, July 2014, pp. 92–97.

[7] A. E. Essaili, T. Lohmar, and M. Ibrahim, "Realization and evaluation of an end-to-end low latency live dash system," in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, June 2018, pp. 1–5.

[8] M. Hosseini and V. Swaminathan, "Adaptive 360 vr video streaming: Divide and conquer," in *2016 IEEE International Symposium on Multimedia (ISM)*, Dec 2016, pp. 107–110.

[9] D. Podborski, E. Thomas, M. Hannuksela, S. Oh., T. Stockhammer, and S. Pohm, "360-degree video streaming with mpeg-dash," *SMTPE Motion imaging Journal*, vol. 127, pp. 20–27, Aug. 2018.

[10] 5GTN. (2019) 5GTN - 5G test network. [Online]. Available: https://5gtn.fi/

[11] DASH-IF. (2020) dash.js javascript reference client. [Online]. Available: https://reference.dashif.org/dash.js/

[12] R. Arutyunyan. (2017) Nginx rtmp-module. [Online]. Available: https://github.com/arut/nginx-rtmp-module

[13] C. K. Henry. (2019) Streamline low latency dash preview. [Online]. Available: https://github.com/streamlinevideo/low-latency-preview

[14] Kaitotek. (2019) Qosium. [Online]. Available: https://www.kaitotek.com/qosium

[15] K. Correll. (2020) Ptpd, precision time protocol daemon. [Online]. Available: https://ptpd.sourceforge.net