Media QoE Enhancement With QUIC

Géza Szabó, Sándor Rácz
TrafficLab, Ericsson Research, Budapest, Hungary
Email: {geza.szabo,sandor.racz}@ericsson.com

Daniel Bezzera, Igor Nogueira, Djamel Sadok
Federal University of Pernambuco Recife, Brazil
Email: {daniel.bezerra,igor.nogueira,djamel}@gprt.ufpe.com

***Abstract*—The demonstrated system is an online video streaming service using QUIC protocol as transport. Video buffer status information is gathered at the client side and it is transferred to the video content server via the object priority assignment method of QUIC. Our proposal is to utilize the priority information for congestion control (CC) influence as well. In our realization, the server extracts the priority information and utilizes it to adjust the aggressiveness level of the CC of TCP emulation in QUIC. The measured gain in initial buffering time (IBT) in the demonstrated system is in the range of 6-49% depending on the video properties and network environment. [1] shows the demonstration.**

I. Introduction

A popular usage pattern in mobile environment is consuming the content of online video sharing sites. As the resolution of smart phones screens constantly grows, the high resolution screens require high resolution videos to provide satisfying Quality of Experience (QoE) for the users. There are various technologies to stream the video content to the user. In this demonstration, we will focus on the Dynamic Adaptive Streaming over HTTP (DASH) [2]. There are various Key Performance Indicators (KPIs) for video QoE. These KPIs include e.g., screen resolution, initial buffering time (IBT), stall time, etc. The connection of the KPIs and the QoE is manifold, and there are various papers, metrics defined on them. As a simplified view on this issue, one can say that the higher the bandwidth the user is provided with, the better the QoE gets in most cases.

One approach to improve video QoE is to utilize the available bandwidth more efficiently, another approach is to seize more bandwidth for the video. [3] proposed a custom LTE scheduler for the YouTube service that allocates further resources to users with low buffer levels. In this demonstration, we extend the above idea further by focusing on a best-effort network use case in which cooperative middle nodes are not prerequisites. We demonstrate that the direct communication of the video app and the QUIC network stack can also result in QoE enhancement.

QUIC is an experimental protocol [4] aimed at reducing web latency over that of TCP. To introduce ways of QoS signaling both in HTTP/2 and QUIC a client can assign a priority for a new stream by including prioritization information in the headers frame that opens the stream. At any other time, the priority frame can be used to change the priority of a stream. However, utilizing the strength of user defined priorities is not straightforward for web developers. In a state-of-the-art web browser, e.g., Chromium the priority information cannot be set directly by the web page. Instead, the object priorities are assigned by the web browser to URL request based on the content-type of the object [5]. Later, the priorities are mapped

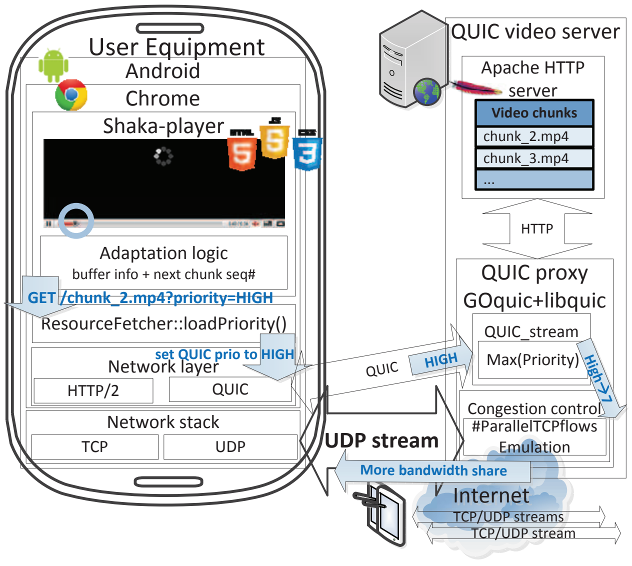

Fig. 1: The demonstrated system

by the network layer from the URL requests to the QUIC protocol. Finally, the assigned priorities are optionally handled by the QUIC server.

Stream priority schedules the service sequence of streams within the QUIC session, but cannot take or give resources from concurrent flows via this method. In our demonstrated system we utilize the existing stream priority mechanism and apply it to alter the CC of the server to make the video segments to be 'pushed down' more aggressively.

The demonstrated system works in an environment with shared bottleneck and serving buffer, that is like the majority of the Internet traffic (except e.g., the radio scheduling in LTE). In our demonstration, we show how video playout buffer level information extracted at the terminal and translated into priority information that is fed back to the video server that can reduce the time needed to start video playout.

II. Demonstrated Features

The working mechanism of the demonstrated system is shown in Figure 1. The *client* is a vanilla Nexus 7 Android 6.0 tablet. The HTML5 based Shaka-player [6] extended with our custom adaptation logic fetches chunks with the priority level indicated in the GET parameters of the URL. Each chunk priority is calculated on-the-fly based on the playout buffer level of the Shaka-player. In case of the buffer level is below 20 sec, then high priority is requested; between 20 and 80 sec

978-1-4673-9955-5/16/$31.00 ©2016 IEEE

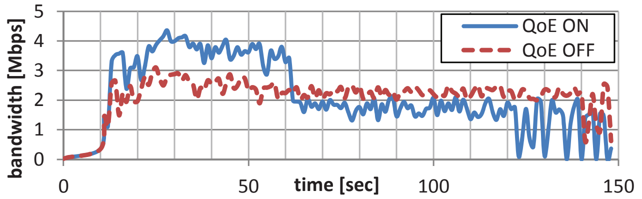

Fig. 2: The effect of altering the aggressiveness of the QUIC stream in case of our QoE enhancement is used (blue) compared to the baseline (red)

bandwidth limit [Mbps]	IBT [sec]	IBT' [sec]	Resolution [p]	Gain [%]
0.75	4.80	4. 52	238	6.15
1.5	3.32	3.05	360	8.74
3	3.35	2.25	480	49.23
5	2.04	1.91	720	6.75
8	2.43	1.95	1080	24.86

TABLE I: Gain ((IBT-IBT1')/IBT1') in IBT with the demonstrated QoE enhancement feature for various bandwidth limited cases (IBT: default, IBT': with QoE enhancement)

medium, and above 80 sec buffer level, the low priority is requested. The requests from the adaptation logic are passed towards the browser. The current mechanism for managing priorities in Chromium is spread between several different subsystems. `ResourceFetcher` [5] of Chromium implements an arbitrary priority assignment i.e., the best practice of object handling sequence according to Blink developers. Our custom `ResourceFetcher` grabs the priority information by parsing the URL and assigns the QUIC stream priority received in the URL request to the network layer. The network layer sends the priority information to the server in the request via the standard QUIC way.

The QUIC *server* acts as reverse proxy that handles the QUIC connections and communicates with the local Apache server via HTTP. The QUIC server uses the `libquic` [7] and `goquic` [8] libraries to implement the necessary protocol features the same way that Chromium does. The QUIC stream priority is parsed from the request. So far we are completely inline with the QUIC draft. In the next phase, we take one step further from the draft and extend the name space of the priority information. We customized the CC of the QUIC stream (`quic_spdy_stream.cpp`) and the aggressiveness of the QUIC session level CC is updated based on the received stream priorities. Practically, we setup the `paralellTCPflowsemulation` parameter in the CC in the function of the stream priority. The browser maps the object priorities to QUIC priorities on a 6 level scale (0 is the highest, 6 is the lowest). We keep the semantic of the object priority, thus high priority is used for important chunks by setting `paralellTCPflowsemulation` to 5. For medium and low priority we setup 2, 1 as CC values respectively. Note, that this is not a within-flow scheduling parameter as in the current draft, but among-flow scheduling usage. As the parameter alters the aggressiveness of the UDP stream, increasing it results in a less fair behavior, seizing more resources from other flows in a network with shared bottleneck, while decreasing the aggressiveness parameter results in even more altruistic behavior.

III. Demonstrated gains on initial buffering time

In the demonstrated scenario a user starts to watch an online video in a loaded network on an Android device and the video streaming experience increases compared to the state-of-the-art case. The network with a shared bottleneck is wifi access point. The wifi access point throughput is shaped for 5 Mbps and an FTP download is present all the time in the background by a separate device to emulate the fully loaded network status.

Figure 2 shows the effects of altering the aggressiveness of the QUIC stream in case of our QoE enhancement is used (blue) compared to the baseline (red). The beginning of the QUIC stream contains downloads of the necessary HTML and JS files. Later, the video chunks are downloaded with high aggressiveness. As the buffer fills in, the client requests chunks with lower priority resulting in decreased server aggressiveness. It is possible to go below the default TCP aggressiveness, thus we can share back resources in case of large buffers.

The quantified effects of the demonstrated feature are summarized in Table I. The results are for demonstration purposes, not intended to stand for as fully evaluated scenarios. The measured initial buffering times were the average of the measured IBT after 50 seek events for each bandwidth limitation scenario (without caching of the chunks). It can be seen that the possible gains can be in the 7-60% range depending on the video chunk size distribution. The final resolution to which the adaptation logic converged during the video differs in the various bandwidth limited cases.

IV. Benefits of the demonstrated system

Our proposal utilizes the existing QUIC protocol capabilities thus there is no information leakage about the user to any middle node. The QUIC session bandwidth share changes in the function of the QUIC aggressiveness parameter of the CC. If there is user consent, this limited information is only sent to the trusted content provider that is in the encrypted part of the protocol. The content provider can optionally take into account the received information.

References

[1] "Our demo video," retrieved: Jan, 2016. [Online]. Available: https://youtu.be/82boc9q92JA

[2] "ISO/IEC 23009, MPEG-DASH," retrieved: Jan, 2016. [Online]. Available: http://mpeg.chiariglione.org/standards/mpeg-dash

[3] J. Navarro-Ortiz, P. Ameigeiras, J. Lopez-Soler, J. Lorca-Hernando, Q. Perez-Tarrero, and R. Garcia-Perez, "A QoE-Aware Scheduler for HTTP Progressive Video in OFDMA Systems," *Communications Letters, IEEE*, vol. 17, no. 4, pp. 677–680, April 2013.

[4] "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2, draft-tsvwg-quic-protocol-00," retrieved: Jan, 2016. [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00

[5] "Assigning object priority," http://tinyurl.com/znv3gbf.

[6] "Shaka-player," retrieved: Jan, 2016. [Online]. Available: https://github.com/google/shaka-player

[7] "libquic," retrieved: Jan, 2016. [Online]. Available: https://github.com/devsisters/libquic

[8] "goquic," retrieved: Jan, 2016. [Online]. Available: https://github.com/devsisters/goquic