Adaptive Rate Control for Live streaming using SRT protocol

Roberto Viola*, Angel Martin, Juan Felipe Mogollón,
Alvaro Gabilondo*, Javier Morgade and Mikel Zorrilla
Vicomtech Foundation
Basque Research and Technology Alliance (BRTA)
San Sebastián, 20009 Spain
Email: rviola@vicomtech.org
*PhD Candidate at UPV/EHU

Jon Montalbán
Dpt. Electronic Technology
University of the Basque Country (UPV/EHU)
San Sebastián, 20018 Spain
Email: jon.montalban@ehu.eus

Pablo Angueira
Dpt. of Communications Engineering
University of the Basque Country (UPV/EHU)
Bilbao, 48013 Spain
Email: pablo.angueira@ehu.eus

***Abstract*—Media delivery represents one of the main challenges for future networks which aim to converge Broadcast and Broadband video traffic into a common telecommunication network architecture. Nowadays, contents streamed over Internet are delivered in two different manners depending on the application: Video on Demand and Live Streaming. For the former, HTTP-based streaming technologies, such as Dynamic Adaptive Streaming over HTTP (MPEG-DASH), are widely employed for unicast and broadcast communications. It also enables Adaptive Rate Control on the client device allowing players to select a representation and bitrate matching the capabilities of the network at any moment. For the latter, MPEG-DASH does not provide low latency for Live streaming when compared to a Broadcast service. Secure Reliable Transport (SRT) is proposed by SRT Alliance to overcome such limitations of unicast and broadcast communications. Nevertheless, it misses the adaptation of the content to the available network resources. In this paper, we show an implementation of Adaptive Rate Control for SRT protocol which exploits periodical network reports in order to adapt the content encoding process. The evaluation includes a real deployment of the solution and a comparison with a legacy SRT stream.**

***Index Terms*—Rate Control, Traffic and performance monitoring, Secure Reliable Transport, Video coding and processing.**

I. Introduction

MPEG-DASH [1] and other HTTP-based alternatives are widely employed solutions for media services. It is compatible with existing HTTP-based Internet infrastructure and allow resolution and encoding bitrate selection to mitigate network performance fluctuations in unmanned networks. These solutions perfectly fit for Video on Demand (VOD) scenario where the latency between content packaging and playback is not an issue. On the contrary, they are not suitable when latency constraints come into play. Live streaming applications, such as video surveillance and video conference, cannot work with tens of seconds of delay of HTTP-based solutions.

Real Time Streaming Protocol (RTSP) and Real Time Messaging Protocol (RTMP) are legacy protocols for Real-time streaming which enable lower latency than HTTP-based solutions. Nevertheless, they are designed to work in unicast mode, meaning that the communication is based on a server-client delivery where the server sends the content in push mode. This communication model fails when players scale up to broadcast concurrency rates, since the server should push as many unicast steams as the number of connected players. Moreover, these solutions suffer of network restrictions applied by network functions, such as firewall and NAT, blocking the delivery of those streams.

Secure Reliable Transport (SRT) protocol [2] is the proposal of SRT Alliance to fill this gap. It lets to gain scalability of Broadcast delivery while guaranteeing low latency required for Live streaming. SRT has been designed to work in both push and pull mode, then allows to stream content even when firewalls and NATs network functions are present. The protocol also includes forward error correction (FEC) [3] which enforces resilience from transmission errors. SRT server also employs network reports from the client to adapt packet overhead. Thus, the server upload speed depends on network throughput and packets are not lost when the available throughput is enough to send the content to the client. Lost packets are re-transmitted only if the network throughput can absorb such overhead.

SRT does not interfere with content encoding, then network reports are never exploited to adapt resolution and encoding bitrate of the content as it happens in HTTP-based solutions. Enabling content adaptation on top of SRT allows two main advantages. First, in case of network degradation, it shields

978-1-7281-2150-5/19/$31.00

from playback stalls by reducing the bitrate of the content to be send, as MPEG-DASH does. Second, it allows to send a representation matching the client display features. This work proposes a real implementation of an Adaptive Rate Control for SRT streams. This solution includes two relevant contributions:

- A server-side Adaptive Rate Control implementation on top of Open Source framework for SRT streaming applications. This Adaptive Rate Control exploits the network reports employed by SRT protocol to enable the adaptation of the resolution and encoding bitrate of the content.
- Evaluation of the effects on user's Quality of Experience (QoE) when compared the proposed solution to a legacy one.

The rest of this paper is organized as follows. First, section II presents the background of Video Streaming solutions and performance metrics. Then, section III shows the implementation of our Adaptive Rate Control on top of SRT streams. In section IV we describe the experiments and present the results. Finally, in section V we expose the conclusions and future work.

II. Related Work

A. Overview of Video Streaming

MPEG-DASH [1] was developed by MPEG and standardized by ISO/IEC. MPEG-DASH is a pull-based streaming technology over HTTP, where the client requests the content from a conventional HTTP server which stores it split into segments and encoded at many representation levels. A segment consists in a unique ISO Base Media File Format fragment, usually called MP4 fragment, which is the minimum playable data. First, the client fetches a manifest file, referred as Media Presentation Description (MPD), and parses it to be aware of the different representations of the content. Then, the client downloads the segments corresponding to the representation that matches the device capabilities and user preferences in terms of resolution, language, codec and bitrate. Each time the client requests the next segment, it can switch to a different representation depending on network performance to avoid playback degradation and maximize user's QoE. Thus, the Adaptive Rate Control is fully managed by the client during the streaming session. However, the delay of MPEG-DASH streams is high since, by design, it is not possible to go behind the segment duration, which usually ranges from 2 to 30 seconds. Thus, MPEG-DASH is not suitable for real-time applications. Apple HTTP Live Streaming (HLS) and Microsoft Smooth Streaming are other solutions working with a similar workflow to MPEG-DASH, where the format of the manifest file differs, and experience delays with the same order of magnitude.

The Common Media Application Format (CMAF) [4], proposed by ISO/IEC, tries to overcome such latency limitations of MPEG-DASH by introducing a Low Latency mode, namely Low Latency or Chunked CMAF. Chunked CMAF allows the presence of several MP4 fragments inside one segment. Consequently, the buffering done by the client is shorter as it can start to play the content even if the segment is not completely downloaded, it just needs to have a MP4 fragment. In [5] and [6], two different implementations of Chunked CMAF are presented. The former, [5] still evidences latency in the order of 1 second, the latter, [6] reduces latency behind one second by generating fragments containing just one frame. The use of just one frame introduces a heavy overhead inside the communication since MP4 header must be replicated each time a fragment is sent. Moreover, it comes at cost of reduced QoE since smaller fragments causes a smaller playout buffer which more easily can go empty [7]. Nevertheless, Chunked CMAF is not currently used by the media industry, but it is a promising solution for the future.

In any case, HTTP-based solutions were not designed for real time applications. Thus, achieving similar latency performance as real-time designed protocols, based on Real-time Transport Protocol (RTP) and User Datagram Protocol (UDP) [8], is complicated. RTP Control Protocol (RTCP) [9] was designed to work jointly with RTP. Hence, RTCP does not transfer streaming data, but it provides RTP with an out of band channel to get feedback on the network statistics, enabling RTP to control the transferring rate. However, such adaptation is made at the network interface level. Then, it does not imply changes on the bitrate of the encoder that could make the difference to adapt the throughput to the available network bandwidth, preventing packet losses. RTP-based solution has also some drawbacks. On the one hand, RTCP is designed to work with unicast streams and, on the other, RTCP is only compatible to push communications mode. Thus, it is difficult to scale as the number of clients increases and when firewalls and NATs are present.

Periscope, one of the most common live streaming services, overcomes these issues by using a hybrid RTMP and HLS solution [10]. Here, the streaming protocol is chosen depending on the volume of clients and latency trade-off. For streaming sessions involving few clients, RTMP is preferred to reduce latency. When the number of clients increases, HLS is exploited to reduce overheads at the server.

SRT [2] protocol, proposed by SRT Alliance, is the media industry solution to transfer live broadcast streaming under the constraint of low latency. The protocol also includes a mandatory encryption to enforce the security. SRT allows both push and pull modes which means that, in case of network traversal barriers, pull mode could bridge them. Moreover, the use of a FEC mechanism [3] enforces resilient communication. Network feedback reports are also exploited to tune the packet overhead and provide protection against transmission errors. In case of packet losses, they are re-transmitted or discarded depending on the configured maximum latency and on the network possibilities to support such overhead.

Finally, SRT includes many advantages compared to conventional real time protocols, but it still lacks the capability to adapt the bitrate throughput of the content when the network bandwidth changes. Network reports are exploited to tune the

transferring rate, but they are not accessible by the encoding process. Consequently, resolution and encoding bitrate of the content are neither adapted at the server nor at the client as it happens in HTTP-based solutions.

B. Performance metrics

All the proposed streaming technologies have a common aspect, they need to focus not only on reducing the latency to allow live streaming, but also on maximising the QoE to retain user when satisfying expectations. The QoE is a key aspect for user satisfaction and retention when rating streaming services. An exhaustive QoE evaluation requires a demographic perception study to get a Mean Opinion Score (MOS) [11].

Nevertheless, there are many studies in literature which demonstrate that the use of objective performance metrics is helpful to provide an estimation of user's QoE [12].

In [13] the authors consider stalling time, number of representation switches and inter-switching time as objective metrics to estimate user's QoE. Recently, the work [14] also includes initial buffering. In both cases, the proposed performance metrics are applicable only for HTTP-based streaming applications involving content adaptation. Since our solution aims to include the same feature on top of SRT protocol, the same performance metrics can be assessed.

III. Adaptive Rate Control Implementation

The system architecture for delivering SRT streams is depicted in Figure 1. The system is composed by a Live Source, a SRT Media Server and a SRT Player.

The Live Source is the node which provides the content to the processing and delivery pipeline. Here, many different entities can act as a Live source, e.g. a camera or a video software editor.

The SRT Media Server processes the content ingested by the Live Source and delivers it to the SRT Player after encoding and packetizing it into an SRT-compliant stream. Thus, it accomplishes the following tasks:

- Encoding: it encodes the content into a live H.264 bitstream [15]. In a legacy SRT solution, the video frame resolution and encoding bitrate is chosen when launching the encoding process and kept unaltered during all the process. In our approach, both resolution and bitrate can be dynamically changed during the streaming session to provide different representation levels of the same content.
- Muxing: H.264 bitstream is packetized into a MPEG-2 Transport Stream (MPEG-TS) container [16].
- Encryption: MPEG-TS is encrypted though 128/256 bit Advanced Encryption Standard (AES) [17]. This is a mandatory feature included in SRT to enforce end-to-end security.
- Delivery: SRT employs User Datagram Protocol (UDP) to transmit data over the network to the client since it guarantees lower latency than Transmission Control Protocol (TCP), which is commonly used by HTTP-based streaming solutions. However, UDP is not reliable since it does not provide mechanisms to compensate for transmission errors. Then, SRT includes Forward Error Correction (FEC) and re-transmission mechanisms on top of UDP delivery to allow the SRT Player to recover from lost or corrupted packets.
- Monitoring: SRT server receives network reports from the SRT Player which contain information related to network status (bandwidth and delay) and packet transmission (sent, lost or re-transmitted packets number). In a legacy SRT solution, reports are only employed to tune the sending transmission rate and schedule the transmission of new and/or lost packets. In our approach, network reports are also captured and employed to select the appropriate representation level (resolution and bitrate) to be used by the encoding process.

We employ GStreamer [18] in its v1.14 stable release to develop our Adaptive Rate Control-enabled SRT Media server. We select and setup the following plugins to accomplish the above tasks:

- *H.264 encoder*: the setup of the encoder is key to allow the adaptation of the representation (resolution and bitrate) of the content according to the measured network statistics. *Keyframes* (I-frames) do not require any other frames to be decoded, so the player can always start decoding a stream from a *keyframe*. Thus, *keyframes* are essential for live streaming to start playing the content as soon as possible when the player starts receiving the content. Moreover, in our Adaptive Rate Control-enabled stream, when it switches the representation level, it introduces a discontinuity. Thus, it makes new frames, with a different resolution and encoding bitrate, not possible to be decoded based on the previous frames. The player needs a new *keyframe* to decode the stream every time the representation level changes. Then, the Adaptive Rate Control forces the encoder to introduce a *keyframe* every time a representation switch is performed.
- *MPEG-TS muxer*: it packetizes H.264 encoded frames into MPEG-TS chunks. Each chunk cannot contain data at different representation levels. Then, it is mandatory that each MPEG-TS chunk starts with a *keyframe*.
- *SRT server sink*: it receives MPEG-TS chunks from the muxer, it encrypts and encapsulates them into UDP packets before sending them to the player. This plugin gets active, sending packets, only when a client is connected. It also monitors the network by getting network statistics measured during the transmission of the video stream to the client. A legacy SRT server sink uses statistics only to adapt the network overhead of the transmission, to reduce packets lost and to avoid re-transmissions. Additionally, the proposed Adaptive Rate Control-enabled solution exploits this information to dynamically change the setup of the encoder plugin to switch to a suitable representation level.

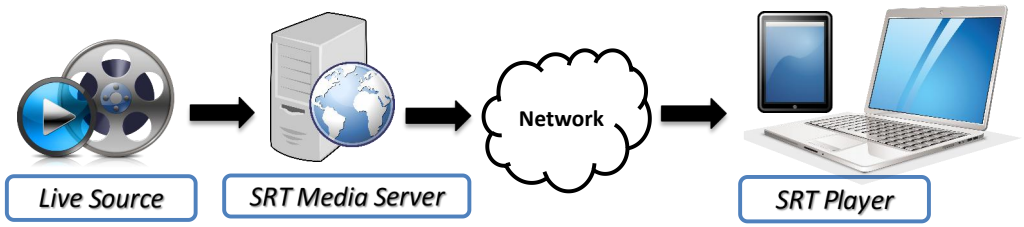

Fig. 1: System architecture for SRT streaming.

Finally, SRT player is implemented with GStreamer release (v1.14) [18]. This version provides SRT client and decoding capabilities. Thus, it can play SRT legacy streams. Moreover, the aggregation of the Adaptive Rate Control to the SRT Media server does not cause any relevant change in the protocol. This means that developing a custom client application is not required and the generated SRT stream can be played by any SRT-compliant player.

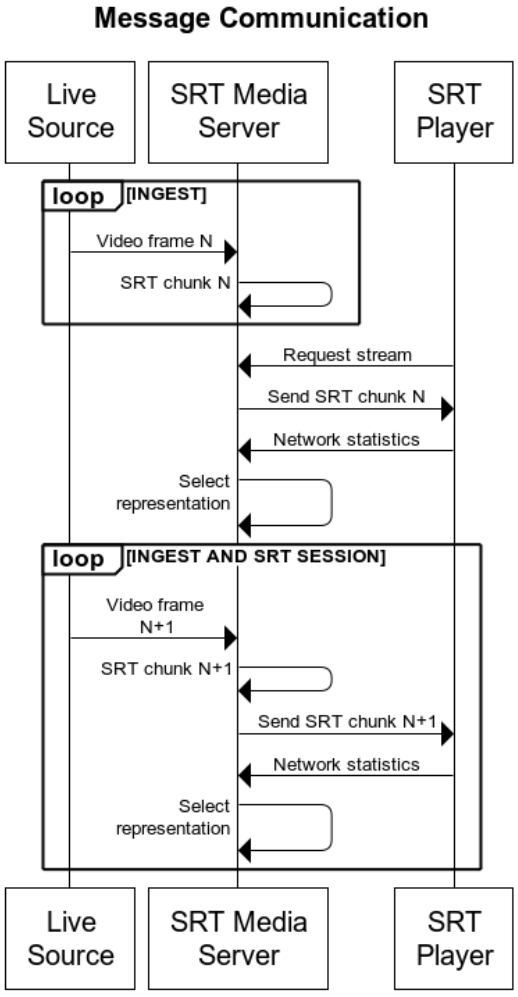

Fig. 2: Sequence diagram.

The communication between the systems in the setup is shown in Figure 2. SRT Media Server starts to encode and packetize video frames when the Live Source is connected. Video frames are H.264 encoded and MPEG-TS muxed, then packetized into SRT chunks. Nevertheless, data are not transmitted until an SRT player connects to the SRT Media Server. It means that chunks can be discarded by the server if there are no players. The SRT Media Server only stores the most recent chunks with a buffer size of "maximum allowed latency". This operation is necessary in order to guarantee that only the most recent chunks are sent, then achieve a low latency live streaming. In GStreamer, maximum allowed latency is configurable, but we decide to keep it to its default value which is 125 ms. Once the SRT player is connected, SRT Media Server starts delivering the content to the player. While receiving the content, the SRT player stores it in the playback buffer before decoding and displaying it. The playback buffer is set to 1 seconds to balance low latency, packet losses reliability and changeable network conditions.

To adapt the representation, the implemented Adaptive Rate Control accesses network statistics from the SRT server sink plugin and exploits the information to tune the configuration of the H.264 encoder. This evaluation is performed by the Adaptive Rate Control once per second. The decision algorithm of the implemented Adaptive Rate Control is shown in Algorithm 1. The algorithm takes the last measured network bandwidth (bw_t), round-trip delay time (rtt_t) and send rate ($rate_t$) from SRT network reports, the current employed representation level (rep_t) and the list of all the available ones ($\{rep_{list}\}$). Bandwidth and round-trip delay time are employed to evaluate the maximum allowed network throughput ($throughput_{max_t}$) through the Equation 1.

$$throughput_{max_t} = bw_t * \frac{1}{1 + \frac{rtt_t}{2}} \quad (1)$$

Then, for each available representation, the required network throughput to allow its transmission is calculated. It is important to note that each representation means a different throughput configured by the encoding bitrate. Furthermore, the encoding bitrate needs to accommodate a gap to allow that protocols messages and extra information from other levels of the ISO/OSI model, added to the encoded video payloads, still meet network bandwidth thresholds.

Consequently, to establish if it is possible to stream a specific representation to the client, we should compare the maximum allowed network throughput ($throughput_{max_t}$)

Algorithm 1 Adaptive Rate Control

function ADAPTIVERATE(bw$_t$, rtt$_t$, rate$_t$, rep$_t$, $\{rep_{list}\}$)
Input: bw$_t$ ▷ measured bandwidth
Input: rtt$_t$ ▷ measured delay
Input: rate$_t$ ▷ measured send rate
Input: rep$_t$ ▷ current representation
Input: $\{rep_{list}\}$ ▷ available representations
Output: rep$_{t+1}$ ▷ next representation
throughput$_{max_t}$ ← bw$_t$, rtt$_t$ ▷ network throughput
for all rep^i ∈ $\{rep_{list}\}$ **do** ▷ for each representation
bitrate^i_t ← rep^i, rate$_t$, rep$_t$ ▷ minimum network bitrate
if (throughput$_{max_t}$ > throughput^i_t) **then** ▷ network admits the representation
rep$_{t+1}$ ← rep^i ▷ next representation

with the throughput that the representation would generate ($throughput^i_t$). Since we cannot access the employed throughput before streaming the content, we estimate it from the current send rate ($rate_t$) provided by the network reports. Equation 2 estimates the necessary network throughput ($throughput^i_t$) to allow the representation (rep^i) bitrate to be streamed. The ratio between current send rate ($rate_t$) and current representation encoding bitrate (rep_t) is the current overhead, then we multiply it per the representation encoding bitrate (rep^i).

$$throughput^i_t = rep^i * \frac{rate_t}{rep_t} \quad (2)$$

If the estimated throughput ($throughput^i_t$) is lower than the maximum allowed throughput ($throughput_{max_t}$), it means that the representation can be sent. The output of the algorithm is the selected representation to be employed at the H.264 encoder. The encoder is configured to immediately generate a *keyframe* and switch at the new selected representation resolution and encoding bitrate.

IV. RESULTS

The experimental setup employed for testing the implemented Adaptive Rate Control is presented in Figure 3. The overall setup comprises the following nodes:

- STR Media Server and Traffic Control: this is a unique physical node which embeds two logical systems. We employ a Docker containerization [19] to run different functions in separated environments. A Docker container running Ubuntu 19.04 OS includes the Adaptive Rate Control-enabled SRT Media Server developed through GStreamer framework [18]. The container communicates with the host machine running Ubuntu 16.04 OS which forwards data to the physical network interface. On the host machine, we periodically modify bandwidth and latency of the network interface. Traffic Control [20] is the utility to change the uplink capacity. To emulate an LTE network, we use the *European Broadband user experience* dataset collected and publicly provided by the Joint Research Centre (JCR) of the European Commission [21]. The dataset provides both bandwidth and latency that we apply through Traffic Control utility. The interval between two consecutive network changes is set to 100 ms.
- Network switch: this node provides wired network access to both server and player nodes to communicate each other. It forwards all the incoming traffic on both sides.
- SRT Player: this node run a GStreamer application to receive the SRT stream and play it.

TABLE I: Set of representations employed in the experiments.

Index	bitrate (kbps)	resolution	framerate (FPS)
1	1200	640x360	24
2	2250	1280x720	24
3	4500	1920x1080	24

TABLE II: Number of switches (S_{Nb}), number of freezes (F_{Nb}), average freeze duration (F_{avg}), initial delay (D) and average representation bitrate (R_{avg}) for both legacy and Adaptive Rate Control-enabled SRT streams.

SRT server	S_{Nb}	F_{Nb}	F_{avg}(ms)	D(ms)	R_{avg}(kbps)
Legacy	0	122	820	972	4500
Adaptive Rate Control	197	87	727	986	4028

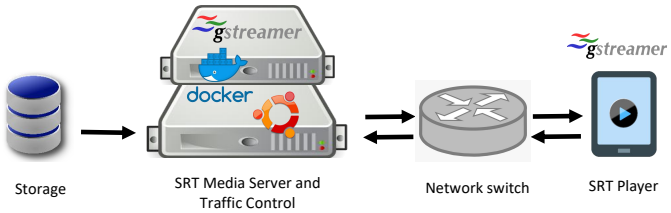

Fig. 3: Experimental setup.

We perform different experiments to compare the Adaptive Rate Control-enable SRT stream with a legacy SRT stream. We use a locally stored Big Buck Bunny test sequence to feed the SRT Media Server. Its raw version is provided by Xiph.Org Foundation [22]. For the Adaptive Rate Control-enabled stream, we established three different representation levels to be employed by the H.264 encoder, while legacy one employs only the higher one. The representations are shown in Table I.

The duration of each SRT streaming session lasts 594 seconds which is the duration of the employed test sequence. The results of the two strategies in terms of representation switches, freezes, initial delay and average representation bitrate are shown in Table II.

In terms of switches, Adaptive Rate Control solution performs 197 representation switches, while it is not applicable to the legacy SRT stream. Figure 4 shows the distribution of the switches across the streaming session. Legacy stream is stable to 4500 kbps encoding bitrate, while seems that Adaptive Rate Control one never uses the lowest encoding bitrate (1200 kbps) but moves between the other two (4500

and 2250 kbps). Thus, it causes that the average representation bitrate is 10% lower when using the Adaptive Rate Control (4028 kbps against 4500 kbps). In terms of initial delay, there is not a noticeable difference since the Adaptive Rate Control does not introduce any delay while starting the streaming session. On the contrary, our Adaptive Rate Control solution outperforms legacy one while considering number of freezes and their duration. Here, the proposed solution scores 29% less freezes events, and their average duration is 11% shorter.

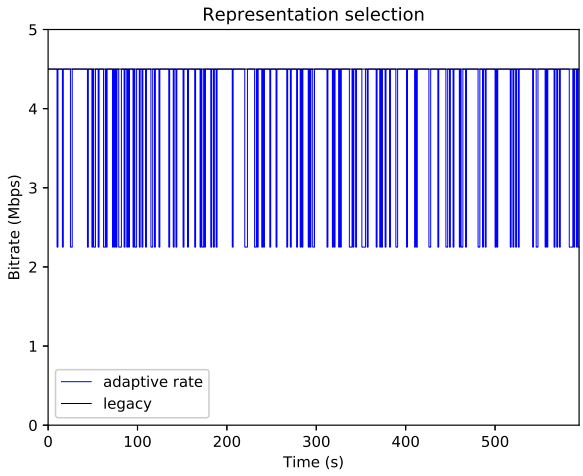

Fig. 4: Representation bitrate selection.

These results show that our solution sacrifices average representation bitrate in order to reduce freezes events. Fewer freezes events lead to a smoother video playback for the end user.

V. Conclusions and Future Work

This paper proposes an Adaptive Rate Control for SRT protocol to deliver live streaming content, while coping with transmission variability due to network degradation or issues.

The proposed solution is integrated with Open Source GStreamer multimedia framework and tested by altering the network capabilities according to real LTE measurements provided by a publicly available dataset.

Compared to a legacy SRT solution, the results show that Adaptive Rate Control-enabled SRT delivery experiences fewer freeze events by enabling switching operations to lower representation bitrates. Thus, it reduces the average representation bitrate to prioritize playback smoothness.

References

[1] Sodagar, I. (2011). The mpeg-dash standard for multimedia streaming over the internet. IEEE multimedia, 18(4), 62-67.

[2] SRT Allicance, Secure Reliable Transport Protocol, 2018. [Online]. Available: https://github.com/Haivision/srt/files/2489142/SRT_Protocol_TechnicalOverview_DRAFT_2018-10-17.pdf

[3] Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M., & Crowcroft, J. (2002). The use of forward error correction (FEC) in reliable multicast. RFC 3453, December.

[4] ISO/IEC 23000-19:2018 Information technology – Multimedia application format (MPEG-A) – Part 19: Common media application format (CMAF) for segmented media. 2018. [Online]. Available: https://www.iso.org/standard/71975.html

[5] Wei, S., & Swaminathan, V. (2014, March). Low latency live video streaming over HTTP 2.0. In Proceedings of Network and Operating System Support on Digital Audio and Video Workshop (pp. 37-42).

[6] El Essaili, A., Lohmar, T., & Ibrahim, M. (2018, June). 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) (pp. 1-5). IEEE.

[7] Viola, R., Gabilondo, A., Martin, A., Mogollón, J. F., Zorrilla, M., & Montalbán, J. (2019, June). QoE-based enhancements of Chunked CMAF over low latency video streams. In 2019 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB) (pp. 1-6). IEEE.

[8] Jacobson, V., Frederick, R., Casner, S., & Schulzrinne, H. (2003). RTP: A transport protocol for real-time applications. [Online]. Availabble: https://tools.ietf.org/html/draft-ietf-avt-rtp-03

[9] Ott, J., & Perkins, C. (2010). Guidelines for extending the RTP control protocol (RTCP). [Online]. Available: https://tools.ietf.org/html/draft-ott-avt-rtcp-guidelines-01

[10] Wang, B., Zhang, X., Wang, G., Zheng, H., & Zhao, B. Y. (2016, November). Anatomy of a personalized livestreaming system. In Proceedings of the 2016 Internet Measurement Conference (pp. 485-498).

[11] ITU-T Recommendation P.800: Mean opinion score (MOS) terminology. (2016).

[12] Alreshoodi, M., & Woods, J. (2013). Survey on QoE\QoS correlation models for multimedia services. arXiv preprint arXiv:1306.0221.

[13] Claeys, M., Latre, S., Famaey, J., & De Turck, F. (2014). Design and evaluation of a self-learning HTTP adaptive video streaming client. IEEE communications letters, 18(4), 716-719.

[14] Lentisco, C. M., Bellido, L., & Pastor, E. (2017). QoE-Based Analysis of DASH Streaming Parameters Over Mobile Broadcast Networks. IEEE Access, 5, 20684-20694.

[15] Wiegand, T., Sullivan, G. J., Bjontegaard, G., & Luthra, A. (2003). Overview of the H. 264/AVC video coding standard. IEEE Transactions on circuits and systems for video technology, 13(7), 560-576.

[16] Sarginson, P. A. (1996). MPEG-2: Overview of the systems layer.

[17] Chown, P. (2002). Advanced encryption standard (AES) ciphersuites for transport layer security (TLS). RFC 3268, June.

[18] GStreamer: open source multimedia framework. https://gstreamer.freedesktop.org/

[19] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux journal, 2014(239), 2.

[20] Traffic Control. http://manpages.ubuntu.com/manpages/xenial/man8/tc.8.html

[21] Chawdhry, P., Folloni, G., Luzardi, S., & Lumachi, S. (2016) European Broadband user experience. European Commission, Joint Research Centre (JRC). [Dataset]. Available: http://data.europa.eu/89h/jrc-netbravo-netbravo-od-eu-broadband

[22] Xiph.Org Foundation. Xiph.org Video Test Media. https://media.xiph.org/video/derf/