Annals of Telecommunications (2024) 79:197–210
https://doi.org/10.1007/s12243-023-00981-8

Using packet trimming at the edge for in-network video quality adaption

Mustafa Tüker[1] · Emre Karakış[1] · Müge Sayıt[1] · Stuart Clayman[2]

Received: 30 May 2023 / Accepted: 26 July 2023 / Published online: 11 September 2023

Abstract

This paper describes the effects of running in-network quality adaption by trimming the packets of layered video streams at the edge. The video stream is transmitted using the BPP transport protocol, which is like UDP, but has been designed to be both amenable to trimming and to provide low-latency and high reliability. The traffic adaption uses the Packet Wash process of Big Packet Protocol (BPP) on the transmitted Scalable Video Coding (SVC) video streams as they pass through a network function which is BPP-aware and embedded at the edge. Our previous work has either demonstrated the use of Software Defined Networking (SDN) controllers to implement Packet Wash directly, or the use of a network function in the core of the network to do the same task. This paper presents our effort to deploy and evaluate such a process at the edge, highlighting the packet trimming algorithm and showing the packet trimming effects on the streams. We compare the performance of transmitting video using BPP and the Packet Wash trimming, against alternative transmission schemes, namely UDP and HTTP adaptive streaming (HAS), presenting a number of quality parameters. The results demonstrate that providing traffic engineering using in-network quality adaption using packet trimming, provides high quality at the receiver.

Keywords Edge computing · High-speed packet processors · Packet trimming · Traffic engineering · SVC Video

1 Introduction

Recent emerging technologies in networking such as the use of softwarized and virtualized network functions and edge computing, together with new protocols, provides a flexible infrastructure which can be tailored to the various requirements specific to Internet applications. Multimedia applications, being one of the most popular applications on the Internet, have become demanding of resources. This is amplified with the growth in user expectations and the characteristics of challenging applications such as remote surgery, AR/VR, and mobile broadcasting. Therefore, although current video streaming applications and standards achieve a good performance today, those future applications will continue to force network and service operators to evolve towards applications providing lower latency, higher reliability, and more adaptivity.

The high traffic volume created by video streaming applications can cause an increase in packet losses, which results in increased latency due to the retransmissions. Although there are a number of techniques for addressing this in the server and the client, *Packet Trimming* is a promising technique that can be used to reduce the number of packets lost [1]. Packet Trimming is a relatively new idea which relies on very fast hardware to adjust the size of a packet during its transmission over the network [2] [3]. When the network becomes limited, the packets are trimmed according to the bottleneck link capacity rather then dropping the whole packet, and the packets are transferred without the cost of any retransmission.

Currently, none of the standard protocols support packet trimming directly. Big Packet Protocol (BPP) is a protocol that can utilize trimming in its design [4]. BPP has been

✉ Stuart Clayman
s.clayman@ucl.ac.uk

Mustafa Tüker
tukermustafa@gmail.com

Emre Karakış
emrekarakis@gmail.com

Müge Sayıt
muge.sayit@ege.edu.tr

[1] International Computer Institute, Ege University, Izmir, Turkey

[2] Department of Electronic and Electrical Engineering, University College London, London, UK

designed as one of the protocols used for the low latency and high reliability applications in future networks. From its introduction in 2018 [4], there are a number of studies related to BPP, which show its usage. The use of the Packet Wash process, where chunks in packets are trimmed, has been shown to reduce latency in [2]. However, such a processes of trimming packets does have an impact on the receiving application, as not all the data arrives, therefore the application has to adapt to such behaviour.

In this paper, we propose an architecture, which utilizes Packet Trimming and the BPP protocol to provide low latency with high Quality of Experience (QoE) for video streaming applications. We consider a scenario, where there are clients having different resolutions and rendering capabilities, hence, the quality of the video should be adapted. Although, video streaming, using HTTP Adaptive Video Streaming (HAS), is the most prevalent and successful application in which the clients adapt the video quality on the basis of observed network parameters, in this study, we implement in-network video quality adaption with the packet trimming approach. An H264 SVC encoder [5] has been chosen as it allows us to create layered videos necessary for use with BPP. We have demonstrated that the combination of SVC and BPP is an effective way to enhance the performance of video streaming applications [6]. Our approach provides continuous delivery, with low latency, while still maintaining guaranteed quality at the receiver [7]. Although there are more modern codecs than H264 SVC, such as H266 [8], it is currently not possible to find any publically available H266 SVC implementations.

In general, Wide Area Networks (WAN) have the capacity to transfer video with high quality; however, the bottleneck links are at the edge where the clients connect to the network. As each client is different, their network attributes are different, and the rendering capabilities of each device can be different. We need to be able to handle this, so a streaming system has many aspects to factor in. By considering these, in our architecture the server always transfers the video at the highest quality, and we rely on the edge to address these client aspects. We utilize the Packet Wash facilities in BPP to implement packet trimming, and we virtualize the BPP-aware network functions at the edge. When the packets arrive to the edge, the BPP-aware function handles them by implementing packet trimming based on the characteristics of each client, and then sends the potentially trimmed packets to the client. The video transmitted is encoded by using a layered codec, namely H.264 Scalable Video Coding (SVC) which makes video packet trimming suitable because of its characteristics that allows quality adaptation by extracting layers.

No one had previously tested sending media streams using packet trimming, but we designed and built an implementation of such a system, and provided some preliminary evaluations in our previous work [9] and [10]. Those initial results showed that in-network video quality adaption can be a promising technique that can meet the requirements of emerging and future video streaming applications. In those studies, we used an SDN controller that implements the BPP functions. However, those studies also showed that the implementation architecture has a remarkable impact on QoE. Processing in the core network creates more load at the center when handling multiple streams, it is also far from the client, and it is harder to deal with the various client differences. Hence, we propose a new architecture that provides higher QoE, is able spread the processing load, and is closer to the clients. The contributions of this paper are threefold. First, we show the potential advantages of in-network quality adaptation by comparing it to HAS. The next contribution is an architecture that uses the edge and virtualized BPP functions for delivery, showing the packet trimming algorithm and the effects of packet trimming on the streams. We compare various protocols, and provide experimental results showing the QoE obtained, and we also discuss the issues caused by implementing the packet trimming process directly in an ONOS SDN controller, showing that deep packet inspection and packet updates in ONOS is highly CPU intensive and is not scalable. This new architecture is compatible with 5 G/6 G and NFV and provides scalability and with optimized results.

This paper is an extension of our paper presented at the 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN 2023) [11], and is structured as follows: Background is next, with a description of the BPP Process in Sect. 3, a comparison of in-network quality adaptation versus HAS in Sect. 4, a description of doing BPP processing at the edge in Sect. 5, and an evaluation of this in Sect. 6. The conclusions and further work are in Sect. 7.

2 Background

Layered video codecs such as Scalable Video Coding (SVC) enables video sequences with various qualities from one encoded video file[5]. After the video is captured, the video frames are encoded with various parameter settings, which produces a number of quality alternatives within the video, by taking advantage of the similarities between the different versions of the same frame. Using SVC video can have a beneficial impact on video transmission [12].

Currently, the majority of video transmission is usually done in one of 2 ways: (i) as discrete packets using RTP over UDP, which is unreliable and can have loss at the receiver, or (ii) as data streams using HTTP over TCP, which is reliable, but can have delay / latency at the receiver. There are advantages and disadvantages to each approach. With UDP, there is a view of the network that presents loss at the receiver. The receiving application has to deal with this packet loss in the network, but the application does have direct control over requests for resends, if they are needed. Using UDP works well for interactive video, when *low latency* is important. With TCP, there is a view of the network where there is no loss; however, there is delay/latency. The receiver application is responsible for dealing with this delay, and is commonly done by using buffering techniques. However, when using TCP the application is presented with a stream of bytes, it has no control over requesting resends which is directly implemented in the TCP stack of the kernel. Using TCP works well for video playback, where there is no interactivity.

To support the transmission of media data, a number of protocols have been defined for carrying audio and video. The RTP/RTCP protocols provide a mechanism that allows interacting media streaming entities to share information [13] such as payload type, lost packet information, timestamps and sequence numbers. RTP [14] was devised to carry media streams with a UDP transport, and is a protocol whose approach is based on Application Layer Framing (ALF) [15]. RTP packets have relative timestamps, and it supports both Sender reports that have a mapping relative to NTP timestamp, which can be used this for syncing, and Receiver reports which present packet loss/jitter. RTCP was devised to provide out-of-band statistics and control information for RTP sessions. RTSP is a presentation-layer protocol that allows end-users to interact with media servers via pause and play capabilities. RTSP is a stateful protocol used for media delivery[16], using RTP. WebRTC is a combination of standards, protocols, and JavaScript APIs that enables Real-Time Communications via a web browser [17], with a latency of sub-500 milliseconds. WebRTC uses SRTP (Secure Real-time Transport Protocol) to ensure that the exchanged data is secure and authenticated.

The HTTP protocol sits over TCP [18], and is commonly for transmitting video, such as those used by CDNs, include HTTP adaptive video streaming (HAS) which is today's de-facto streaming technology. Dynamic adaptive streaming over HTTP (DASH) [19] is a standard developed by MPEG, for providing interoperability between the participants of HTTP adaptive video streaming (HAS) deployments. As DASH runs on top of HTTP, it is inherently a TCP-based protocol. MPEG-DASH was devised in order to deal with various network and TCP behaviours, due to packet loss over time. The DASH server is designed to send segments of video, of a few seconds length, at the requested quality. Each client starts by requesting the lowest quality, and then progressively goes to a higher quality if the client calculates that there is low congestion and that network bandwidth is available. The client can dynamically adapt the quality of the video being sent by requesting segments of a different quality, and placing those video segments into the decoder for viewing. To accommodate various bandwidths, the source video is encoded multiple times with a range of qualities. The higher the quality, the higher the data rate, the bigger the file, and the more bandwidth used. A side-effect of using numerous files of video together with TCP, is that the low latency real-time behaviour of video delivery can be compromised by both the size of the video files and the built-in retransmission mechanisms of TCP. A number of efforts are looking at addressing this low latency issue [20], and some results are already appearing [21] and [22].

One of the main benefits of DASH is that it can utilize the caching capabilities of HTTP, allowing the use of large distributed cache infrastructures spread across the network. Specialized CDN companies provide such distribution and caching facilities, often with embedded nodes close to the users.

Another approach that has been evaluated for optimizing the video delivery when there is congestion in the network is video transcoding close to the user. In [23], the authors present a system which reduces video streaming costs in HTTP adaptive streaming by enabling lightweight transcoding at the edge. Overall, there are a number of bitrate adaptation schemes for streaming media over HTTP which have been surveyed in [24].

The QUIC (Quick UDP Internet Connections) transport protocol [25] was introduced to improve the QoE of web services, and provides the HTTP protocol running over UDP rather than over TCP. Although QUIC has been suggested as a good candidate for video transmission, in [26], the authors evaluated YouTube streaming and conclude that there is no evidence of a QoE improvement when using QUIC, compared to HTTP over TCP. In [27], the authors considered real-time video traffic with QUIC, and found that QUIC was still too reliable, sometimes performed worse than TCP for video streaming, and that the ABR schemes utilized with QUIC operated poorly compared to TCP. They concluded that reliable transports are ill-suited for video streaming, and suggest an unreliable version of QUIC.

The combined use of both SVC and QUIC has been evaluated in [28]. The authors reiterate that HTTP suffers from Head-of-Line blocking, and three-way handshake delay due to TCP, and that QUIC, which run over UDP can tackle these issues. They have determined that with the fast recovery and with the elimination of HOL blocking abilities, QUIC can support multiple streams to provide better performance where there is packet loss.

One of the design goals of BPP is the definition and implementation of application specific networking behaviour, manifested at the level of an individual packet or a flow. This is facilitated by new functions in enhanced network devices

[4]. In BPP, each packet is grouped into a header and a set of chunks rather than just a sequence of bytes. An important feature of a BPP-aware network node is that given specific commands, the node can drop particular chunks from the payload, with the BPP header having particular fields containing meta-data for indicating the commands. Depending on the commands and the load on the network, some chunks can be dropped during transmission. With BPP, the strategy is to reduce the load on the network by reducing the consumed bandwidth, while keeping the flow of packets continuously arriving at the receiver. BPP was devised as a new type of transport layer protocol, which is more like UDP than TCP, but provides *partial reliability*.

BPP was designed to be used for low latency and high reliability applications [4], where having retransmission makes resending times too slow. It has been evaluated in a number of studies to determine its effectiveness. In [29], combining BPP with TSN (Time Sensitive Networking) was used to overcome the limitations of TSN's scalability and complexity issues. In [30], BPP was used for computation offloading in Mobile Edge Networks. Neither of those papers addressed multimedia over BPP. Although BPP was designed for media transmission, our previous work [9] and [10] was the first actual implementation of video streaming using BPP, and for determining the effects of sending video over BPP. We demonstrated that BPP is better than TCP and UDP for latency and outages, and that BPP is a reasonable alternative for streaming video, especially when taking into account some of the high precision requirements. In [6] we demonstrated that in-network adaption can be provided when using SVC combined with BPP. A detailed description of the techniques and mechanisms used for carrying the streams of SVC video from servers to clients, using the BPP Packet Wash mechanism is presented in our paper [7]. As there are no BPP-aware devices, we send BPP payloads over UDP.

Work on Packet Trimming has become of interest as hardware has become fast enough to do in-transit packet updates. Handley et al. considered trimming for the Data Center [1]. They define NDP, a novel data-center transport that achieves near-optimal completion times for short transfers and high flow throughput in a wide range of scenarios. With NDP, the switches trim the packets to just headers and then priority forward the headers. The authors observe that "Due to packet trimming, it is very rare for a packet to be actually lost". To improve round trip times across the network, in [3] the authors suggest that trimming the whole payload and only keeping the header, can only work well in Data Centers where the dropped payload may be retransmitted fast enough to the node that dropped the payload. Such a data center approach does not generalize to full WANs. Selectively trimming the packet, rather than the whole payload, offers a less drastic mechanism that would work in the wide area network. For enhancing end-to-end transport, a new transport protocol QUCO is defined [31], which reacts to congestion by selectively dropping parts of a payload packet, combined with mitigation mechanisms to handle the loss of part of the payload. Their packet trimming scheme reduces the variation in the number of packets going through the network, and allows for tighter targets to be set on the number of packets in transmission, and also on the size of the switch buffers. QUCO has less delay and less delay variations than TCP, with a resulting reduction in jitter. Very recent work for implementing a hardware mechanism for packet trimming has investigated how trimming can be implemented in programmable switches which were not specifically designed for trimming [32]. That implementation uses P4 on the Tofino switch ASIC, and shows that it is possible to undertake trimming and demonstrates that trimming can be integrated into a production-grade datacenter switch.

Edge Computing is considered an extension of cloud computing, whereby additional computational, data handling, and networking resources are placed closer to the end systems. As a consequence, the processes for data processing, networking, data management, and storage can occur between the end systems and the cloud servers, not just at the centralized cloud servers. Edge Computing can be extremely useful for low-latency applications, as well as applications that generate an enormous amount of data that cannot be practically transferred to cloud servers in real-time, due to proximity to the clients, or bandwidth or time limitations [33]. In recent times, centralized applications have evolved towards service-oriented architectures and microservices, with small independent and loosely coupled systems that deal with a very specific tasks being virtualized and executed autonomously, especially at the edge.

3 BPP processing

To execute the required BPP processing of packets, BPP enabled network nodes are necessary. A server is responsible for constructing the BPP packets, by creating the header, the chunks, and setting the *significance value* of each chunk. Our paper [7] explains this packet filling process in detail. The significance values are set for each chunk of each BPP packet, and they show the importance of each chunk within the packet. The significance value provides the network node with the information needed when it is deciding to keep the chunk, or trim it from the packet. When we use the Packet Wash process of BPP to trim packets during their transmission over the network, the chunk removal is undertaken by considering both the importance of the chunk to the video content, indicated by the significance value, and the available bandwidth at the time of the calculation.

Table 1 Significance values used for the transmitted video

Video layer	Frame type	Temporal layer	Significance value *SIG*
L0	I	T0	1
L1	I	T0	2
L2	I	T0	3
L0	P	T1	2
L1	P	T1	7
L2	P	T1	12
L0	P	T2	3
L1	P	T2	8
L2	P	T2	13
L0	P	T3	4
L1	P	T3	9
L2	P	T3	14
L0	P	T4	5
L1	P	T4	10
L2	P	T4	15

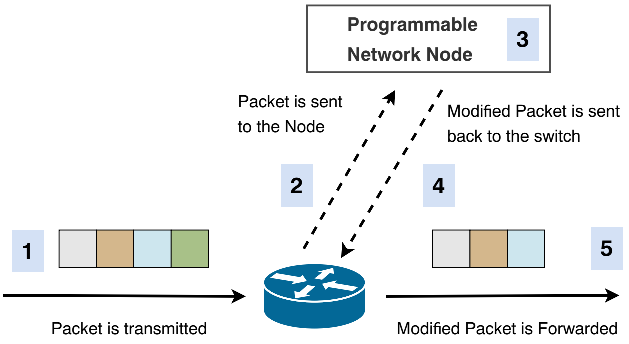


Fig. 1 BPP processing

For SVC video, we set the significance value embedded into the chunks by taking into account both the impact of each chunk on the decoding process and the resulting QoE. The video chunks which have the highest importance are: (i) all the chunks that consist the base layer, (ii) the chunks for all layers of an I frame, and (iii) all meta-data chunks. These are set with the lowest significance value, which indicates such importance to the network node. This means that those chunks should not removed during transmission.

The information about the video layers, the frame types, and the temporal layers of the video, and the associated significance value (SIG) is given in Table 1. By considering the importance of each video layer, each frame type, and each temporal layer, we allocate a significance value for each of them. The significance value (SIG), shown in column 4 of the table, is placed by the server into the SIGi field of each chunk's meta data. As all the I frames (which have a SIG of 1,2, or 3) and the base layer of all frames (with a SIG of 1-5) always need to be received by the clients, in order to be able to play the video with continuity, the server sets a trimming threshold value of 5. This allows the network function to trim the chunks where the significance value is above 5, in the situation where there is not enough available bandwidth.

Figure 1 shows the steps of the Packet Wash process in the network node, whereby chunks can be trimmed if the total amount of data transferred in a specific time period exceeds the available bandwidth. $\boxed{1}$ Packets are constructed with a BPP header and a number of chunks, and transmitted across the network. $\boxed{2}$ When the packet arrives at a network node, it is sent for processing. Each programmable network node enumerates the number of bits transferred within each time period, using the size of the packets. If the value exceeds the available bandwidth, then it determines that trimming should occur. $\boxed{3}$ The node checks each packet to evaluate which chunks should be trimming from the packet, according to the available bandwidth measurements. Chunks are trimmed one at a time, by considering the *significance value* field of each chunk, such that the size of a packet should be reduced to below the specified limit. Chunks whose *significance value* is lower than a threshold are not trimmed in any circumstances, and so the packet size may not be reduced as much as desired. This means later packets will need more trimming. $\boxed{4}$ The packet, modified or not, is sent back to the switch, and $\boxed{5}$ forwarded onwards. The client is responsible for collecting the chunks from the packets, and reconstructing a valid data stream, as shown in step 3 of Fig. 2.

The effects of Packet Wash trimming ensures that the client receives a continuous stream of packets, and so always has some meaningful data to process. This approach of using the bandwidth is clearly different from UDP, where packets are dropped when there is congestion. Although there is no more usable bandwidth when sending with BPP, our previous work has shown that it can be utilized and managed more effectively.

We presented work on in-network quality adaptation in our paper [6]. That showed that it was a promising approach when compared to both TCP and UDP. In [34], we showed the full effects of Packet Wash on SVC video in limited bandwidth environments.

HTTP adaptive video streaming (HAS), being the one of the most popular video streaming applications, provides an efficient client driven adaptation when network conditions change. HAS clients adapt the quality on the basis of both the observed and the internal parameters, hence minimizing the negative impacts of network condition changes on QoE.

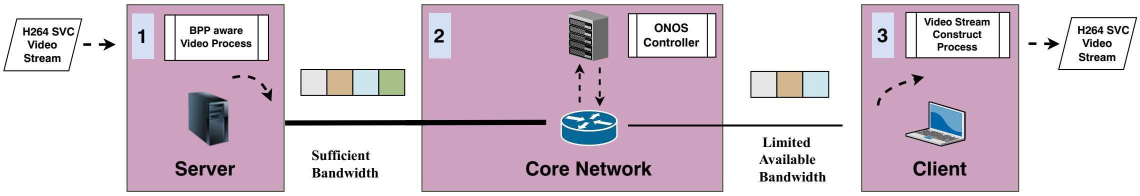


Fig. 2 ONOS controller for BPP processing

In this paper, we focus on the potential advantages of in-network quality adaptation, at the edge, over quality adaptation at the client.

4 In-network quality adaption versus HAS

To show the potential advantages of in-network quality adaption, we compare it to HTTP adaptive streaming (HAS) in order to determine its effectiveness. Only by doing this, we can determine if such an approach has the benefits we are looking for.

The first approach, we considered, in [9] and [10], utilized an ONOS SDN controller [35] to implement the BPP packet wash process, as a function embedded in the ONOS controller.

The video used in all the experiments is *Big Buck Bunny*. The bitrate distribution for video qualities are slightly different for HAS and other protocols, due to encoding aspects. For the HAS experiments, we used the packetized video for HAS systems given in [36]. The bitrates of the qualities of the packetized encoded video used with HAS are 1 Mbps, 2 Mbps, and 4.2 Mbps. In the other experiments, using BPP, UDP, and TCP, the bitrates of the video layers are 1.1 Mbps, 1.9 Mbps, and 5.6 Mbps. However, these bitrates are increased slightly after the packetization process. We also decreased the bandwidth values in accordance with the bitrate of the packetized video, in order to provide a fair comparison in HAS experiments.

4.1 On the performance of in-network quality adaption

We conducted a set of experiments with a number of network conditions, based on the architecture in Fig. 2, using the ONOS SDN controller and Mininet. Mininet [37] is an environment which creates virtual networks, running a real kernel, switch, and application code. It allows for the evaluation of SDN systems using OpenFlow and P4.

We compared the quality adaption using BPP against quality adaption with HAS, as well as having regular transmission with UDP and TCP, where there is no quality adaption. In these experiments, the bandwidth of the path between the core network and the client is *limited* and set to different values, namely 1.5 Mbps, 2 Mbps, 2.5 Mbps, and 3 Mbps. There is also one set of experiments with dynamic network

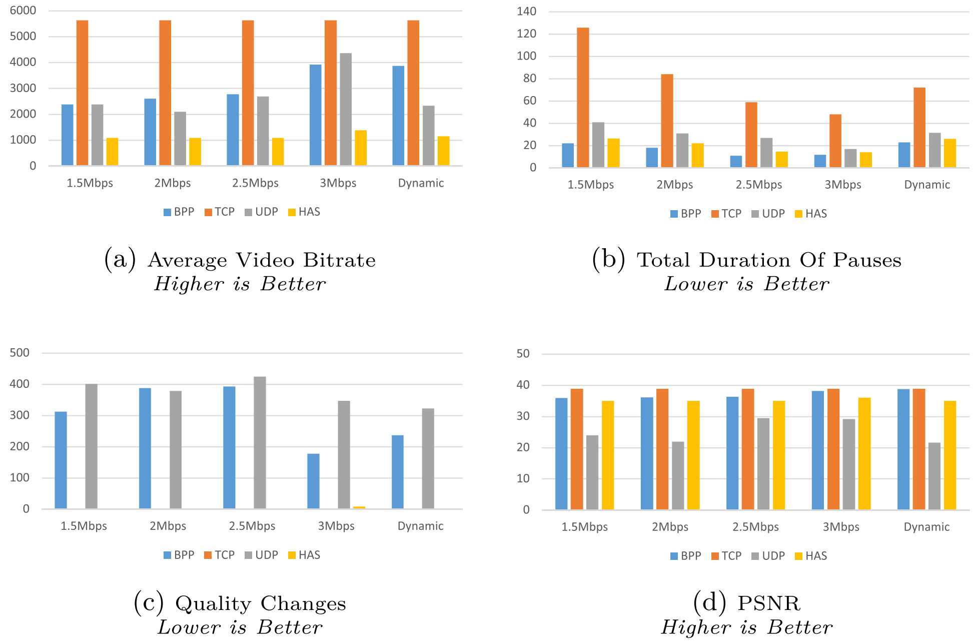


Fig. 3 Quality parameters using ONOS controller for BPP processing

(a) Average Video Bitrate
Higher is Better

(b) Total Duration Of Pauses
Lower is Better

(c) Quality Changes
Lower is Better

(d) PSNR
Higher is Better

conditions, where the bandwidth changes between 1 Mbps and 3 Mbps over time.

The current implementations of the H264 SVC decoder are rather primitive, limited, and fragile, compared to the H264 AVC version. Even the available H264 SVC encoder/decoders are still mostly experimental, so it is hard to do full video decoding and full evaluation. For these experiments the PSNR data comes as text debugging output from the encoder and decoder, and therefore the collected PSNR data is a best effort evaluation.

In Fig. 3, the different QoE parameters observed by different protocols are presented. In Fig. 3a, the average bitrate of the played video on the client's side are given. In terms of this QoE parameter, TCP outperforms other approach since TCP client always get the video at the highest quality. However, this behaviour has significant costs, with a high level of duration of pauses, which is the one of the attributes that has a huge negative impact on QoE, as seen in Fig. 3b. Due to its design, the HAS client applies a conservative approach to keep the duration of pauses at minimum, where the clients always request the video at the lowest quality. As the TCP and HAS implementations do not change the quality over time, there is almost no observable quality changes with these experiments, in Fig. 3c, whereas BPP and UDP have many changes. In Fig. 3d, the PSNR values observed for each protocol are given. We see that TCP has the highest PSNR value since the server sends video at the highest quality during the whole session. BPP and HAS values for PSNR are similar, where the PSNR values obtained with BPP is slightly higher than HAS.

Although the different QoE parameters observed in the experiments provide some basic information about the nominal perceived quality on the client side, an *overall QoE value* gives a better idea about the general perceived quality. To calculate this overall QoE value, a linear function is defined based on one devised in [38]. The importance of different QoE parameters is specified by setting different weights to the different terms, based on their effects on perceived quality.

The *QoE value* formula is given in Eq. 1 and it calculates the overall QoE for the segments numbered between k and K. In the formula, α, β, and γ are the weights used for the QoE parameters. Q_k represents the video quality in terms of PSNR or video bitrate for the k^{th} segment. l_k is the layer of the k^{th} segment, therefore the second term in the formula given the quality change level between two consecutive segments. dp_k represents the duration of pauses experienced during the playout of the k^{th} segment, and it is calculated by the function given in Eq. 2. In that function, ts_k and p_k are the timestamp value of the received time and the playout time of the k^{th} segment, respectively.

Table 2 Overall QoE values for different transmission schemes and varying bandwidths

	Bandwidth 1.5 Mbps	2 Mbps	2.5 Mbps	3 Mbps	Dynamic
BPP	57.7	59.5	63.4	68.8	63.7
TCP	14.8	35.8	48.3	53.8	41.8
UDP	23.5	24.7	41.2	46.4	24.1
HAS	56.9	59	62.7	64.9	56.9

The overall QoE values calculated for this set of experiments are given in Table 2. In the QoE calculation, the PSNR value is used as the Q_k values in Eq. 1. The weights in the equation, for α, β, and γ, are 2, 0.01, and 0.5, respectively. These numbers are selected based on the positive/negative effects of the QoE parameters. The research done on perceived quality [39] showed that the bitrate is the most important parameter that affects QoE, and that users prefer quality changes over a duration of pauses. The table shows that the highest overall QoE values are observed with BPP in fixed bandwidth experiments; however, HAS values that are very close to the BPP values. With dynamic bandwidth conditions, we observe that BPP outperforms other approaches, giving an insight into how in-network quality adaption can be beneficial compared to client-based adaptation.

$$QoE_1^K = \alpha.\sum_{k=1}^{K} Q_k - \beta.\sum_{k=1}^{K-1} |l_{k+1} - l_k| - \gamma.\sum_{k=1}^{K} dp_k \quad (1)$$

$$dp_k = \begin{cases} ts_k - p_k & ts_k \geq p_k \\ 0 & otherwise \end{cases} \quad (2)$$

4.2 The effects of implementing bpp in the controller

The performance results show that BPP adapts the quality in an efficient way, as there is a lower total duration of pauses and a higher received average bitrate, when compared to HAS. However, HAS managed to keep the number of quality switches to a minimum, when compared to BPP.

The experiments given in the previous section were conducted with one server, one client, and one OpenFlow enabled switch. The controller sets flow rules in the switches, between the server and client, at the beginning of the streaming session, to forward the packets. We observe that if there is just one client on the network, the SDN controller can manage the BPP processing for each packet of the stream. However, if the number of clients increases, problems arise and the controller starts consuming a high level of CPU resource. Additionally, sending a packet to the controller from a switch causes an increase in the end-to-end delay due to an extended transmission delay between the controller and the switch.

We assume that the network operator and the video streaming company are in co-operation, so the controller gets knowledge about the highest video bitrate from the server. This information helps the controller to determine if the bandwidth is enough to transfer the video with the highest quality. If the available bandwidth of a link is too low to send the video with the highest quality, then the packets should be trimmed at that point. During the session, it periodically measures available bandwidth.

In order to limit the computational complexity on ONOS and reduce the end-to-end delay, we developed an approach which would only send packets to the controller when a BPP operation was needed, rather than sending every packet.

To support this, the controller *removes* the flow rule related to the video stream from the switch, if the bandwidth becomes limited. When a packet is received by the switch *and* there are no flow rules, it sends a message to the controller to ask for the flow information related to the video stream. As the response, the controller sends the trimmed packet, coupled with the output port information.

We then implemented this enhanced approach and performed several experiments using Mininet. However, the experiments show that the scalability of this approach is quite limited since it requires Deep Packet Inspection (DPI) operations that trims the payload and headers.

Given the number of issues with processing BPP streams in the controller, including having too much load with so few streams, we investigated another potential solution. We proceeded to a new approach and evaluation, which is to provide in-network quality adaption by implementing BPP functions as a virtual network function. We replaced the ONOS controller with a single virtual router implementing a BPP-aware function, located between the server and the client. In our paper [34] we show how well that approach works.

5 BPP processing at the edge

In this paper, we generalized the idea of the virtualized BPP function implementation and propose a system for larger networks with more clients connected, where a virtualized BPP function is used for the BPP operations.

5.1 Virtualized BPP function

In general, the bottleneck links are those links that connect clients to the network. Therefore, in the proposed system, the virtualized BPP-aware network functions are installed at the edge routers. In our system, the edge network is an SDN domain where the SDN controller is responsible for managing the network. It periodically measures the available bandwidth of the links that clients use to connect to the network. This available bandwidth information is fed into the BPP function.

In this system, shown in Fig. 4, the server sends the video at the highest quality, considering there are many clients having different characteristics, such as different bandwidth conditions, device resolutions, and rendering capabilities. The packets, which carry all the layers of the video, are transferred through the network until they arrive to the edge. At the edge, the packets are sent to the virtualized BPP function, where the packets are trimmed according to the characteristics of each client, such as the current link bandwidth between the edge and the client, and then the function forwards the potentially trimmed packets to the client.

When the virtualized BPP function receives a packet, it determines whether the packet should be trimmed and if it is to be trimmed, how many chunks should be removed from the packet. The virtualized BPP function uses the available bandwidth information of each client based on the information received from the controller. Packet trimming is done by using a simple volume checking mechanism, given in Algorithm 1.

5.1.1 Packet trimming algorithm

For each packet that the function receives, it checks if the number of bytes sent in the current second is greater or less than the available bandwidth. If the number of bytes sent is below the available bandwidth, then the packet can be forwarded as-is; however, if it would be greater than the available bandwidth, then it determines that bytes should be trimmed from the packet. The trim operation is done by removing some chunks from the packet, based on their *signif-*

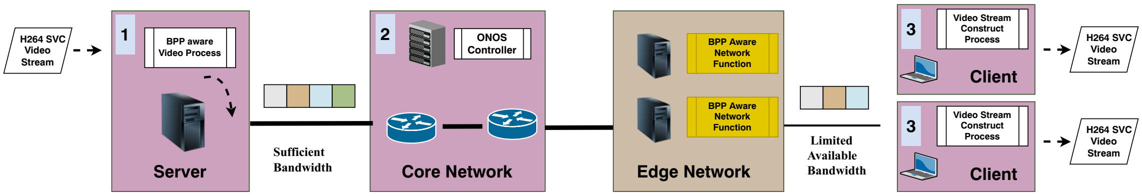


Fig. 4 Edge network deployment of VNFs

Algorithm 1 Packet Trimming Algorithm

```
// Set timeStart as current time in milliseconds
$timeStart \leftarrow Clock()$
$bytesRecvThisSecond \leftarrow 0$; $bytesSentThisSecond \leftarrow 0$
for each packet received do
    ▷ Timing
    // Get current time in milliseconds
    $now \leftarrow Clock()$
    // Millisecond offset between now and timeStart
    $timeOffset \leftarrow now - timeStart$
    // What is the offset in this second (as floating point)
    $secondOffset \leftarrow timeOffset\ /\ 1000$
    ▷ Pre processing of packet
    $packetLength = length(packet)$
    $bytesRecvThisSecond \mathrel{+}= packetLength$
    // The ideal no of bytes to send at this offset into a second
    $idealSendThisSec \leftarrow availableBandwidth * secondOffset$
    // How far below the ideal are we
    $below \leftarrow idealSendThisSec - bytesSentThisSecond$
    ▷ Check current second
    if $(timeOffset >= 1000)$ then
        // Crossed a second boundary. Reset variables for new second
        $timeStart \leftarrow now$
        $secondOffset \leftarrow 0$
        $bytesRecvThisSecond \leftarrow 0$; $bytesSentThisSecond \leftarrow 0$
    end
    ▷ Decision making and forwarding
    if $(below > 0)$ then
        // Below ideal so there is capacity so forward without trimming
        $bytesSentThisSecond \mathrel{+}= packetLength$
        $forward(packet)$
    else
        // We need to drop something, so trim chunks from the content
        $(newPacket, droppedAmount) = trim(packet, idealSendThisSec, below)$
        // droppedAmount is how much content was actually trimmed
        $bytesSentThisSecond \mathrel{+}= packetLength - droppedAmount$
        $forward(newPacket)$
    end
end
```

icance value, as 3 step $\boxed{3}$. As the size of the chunks that can be trimmed may be different to the number of bytes that need to be trimmed, the granularity of the process is not perfect. Either more bytes maybe trimmed, if the chunks are large, or it can take a number of packets to be trimmed in order to match the correct bandwidth level.

The BPP function runs the algorithm for each second during video transmission. There are 4 main phases: (i) getting *Timing* information when a packet arrives; (ii) doing *Pre processing of packet* to get the packet size and the determining the ideal number of bytes to send at the particular offset into a second; (iii) to *Check the current second* to see if a second boundary has been crossed; and (iv) the *Decision making and forwarding* which processes each packet.

If the volume of the packets that are transmitted in the current second is less than or equal to the available bandwidth, i.e. the *below* value > 0, then the received packet is transmitted to the client without implementing packet trimming. Otherwise, some chunks need to be dropped from the packet, so trimming occurs. The $trim()$ function considers the current ideal send volume, the chunks, and their significance values, and returns a packet and a dropped amount. These packets, modified or not, are then forwarded to the clients.

5.1.2 Packet trimming effects

The effects of using the Packet Trimming Algorithm are shown here. In Fig. 5 there are graphs showing the data flow rates for different available bandwidths. In Fig. 5a we see how the per second processing described in Algorithm 1 can visualized. The data is presented for the first 5 s of a flow to highlight the behaviour. The magenta line shows the *ideal* amount which could be sent, at a particular offset into a second, which is correlated to the available bandwidth. The green line shows how much is actually *sent* onwards. On each second boundary, the values are reset for the new second.

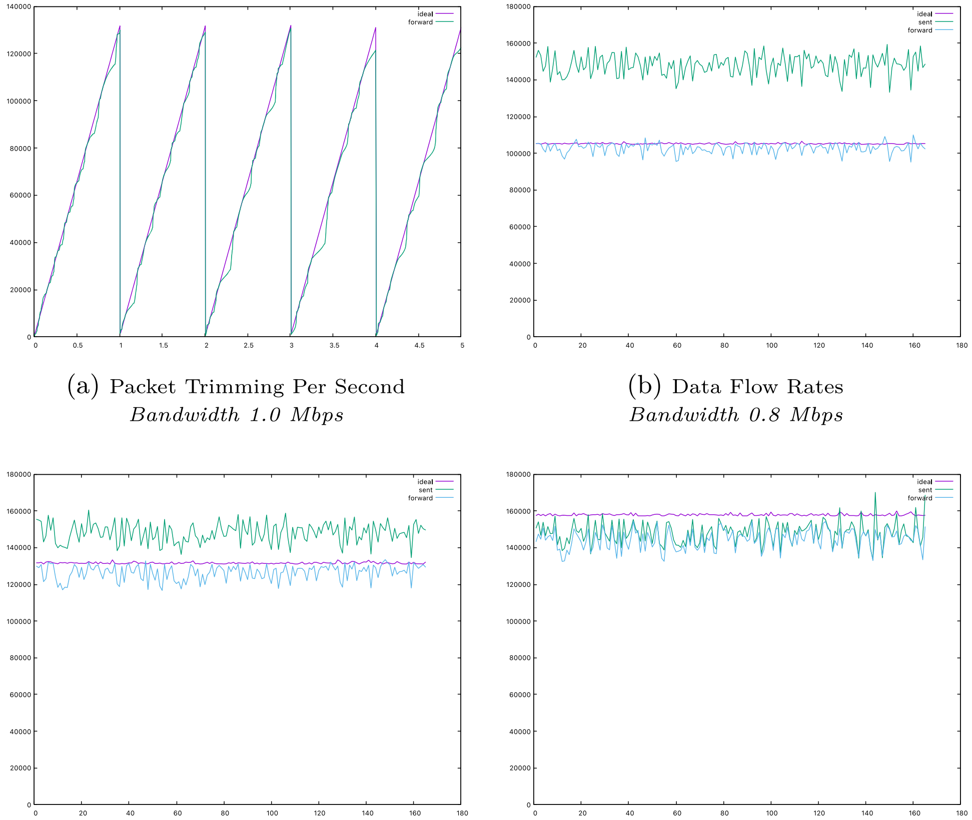
The other 3 figures show the data flow rates, when the bandwidth is set to 0.8 Mbps, 1.0 Mbps, and 1.2 Mbps. In each figure, the y-axis shows Bytes/sec and the x-axis shows Time in seconds, and the amount of data *sent* by the server is green, the *ideal* amount which could be sent is magenta, and the amount actually *forwarded* is in cyan.

The video used in these experiments is encoded at 1.1 Mbps, which equals 144000 bytes / second. In all three of these figures, we can see that the the amount of data *sent* by the server in green, averages that rate. The amount actually *forwarded* in cyan, always averages bewlow the available bandwidth, which is the *ideal* amount, in magenta. We can see how the algorithm trims chunks as described, and that as the bandwidth reduces, the more data is trimmed. For 1.2 Mbps, very little is trimmed, but for 0.8 Mbps there is a considerable difference between the sent and the forwarded.

5.2 The performance of implementing BPP at the edge

In Fig. 6, we present a comparison between the two different approaches to process packet trimming. For this purpose, the video is streamed between the server and 2 clients, with the

Fig. 5 Data flow rates showing trimmed chunks Bytes/sec (y-axis) time in secs (x-axis) *Sent (green) - Ideal (magenta) - Forward (cyan)*



(a) Packet Trimming Per Second
Bandwidth 1.0 Mbps

(b) Data Flow Rates
Bandwidth 0.8 Mbps

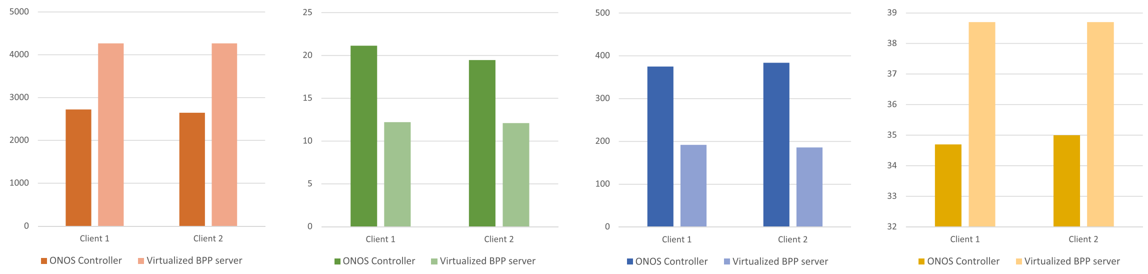
(c) Data Flow Rates
Bandwidth 1.0 Mbps

(d) Data Flow Rates
Bandwidth 1.2 Mbps

results for Client 1 and Client 2. The QoE parameters which are measured on the client side are collected.

The darker bars labeled “ONOS Controller” represent the results that are observed when using an ONOS controller which has a module running the BPP process. On the lighter bars labeled as “Virtualized BPP server”, the BPP processing is done by a virtualized function at the edge.

In these experiments, the 2 clients are connected to the network over different edge links, and the available bandwidth on those links is 2.5 Mbps. When the packet trimming operation is done by the virtualized BPP function, all of the QoE parameters have better values, compared to the experiments with the ONOS controller, as seen in Fig. 6. In addition to that, although the clients received the video at a higher quality, the duration of pauses observed with the virtualized BPP server is still less than ONOS controller due to the latency added by the BPP process on the controller as seen in Fig. 6b.



(a) Average Video Bitrate
Higher is Better

(b) Total Duration Of Pauses
Lower is Better

(c) Quality Changes
Lower is Better

(d) PSNR
Higher is Better

Fig. 6 Comparing BPP process performance – *ONOS controller versus virtualized BPP server at the edge*

Overall, we see that the QoE obtained by trimming packets at the edge is better than using the ONOS controller, as utilizing the virtualized BPP functions at the edge provides higher QoE parameters than the those with the ONOS controller.

6 Performance evaluations

In this section, we give the performance evaluation of the virtualized BPP function which is compared with the other protocols, UDP and HAS. In the BPP and UDP experiments, the server sends the video at the highest bitrate.

6.1 Experimental setup

The video used here is encoded with slightly different parameters than those used previously. The bitrates of the base layer, first enhancement layer, and second enhancement layers are 0.9 Mbps, 1.9 Mbps, and 4.4 Mbps, respectively. However, once the overhead of packetization using BPP is added, we obtain similar bitrates to the encoded video used in HAS experiments.

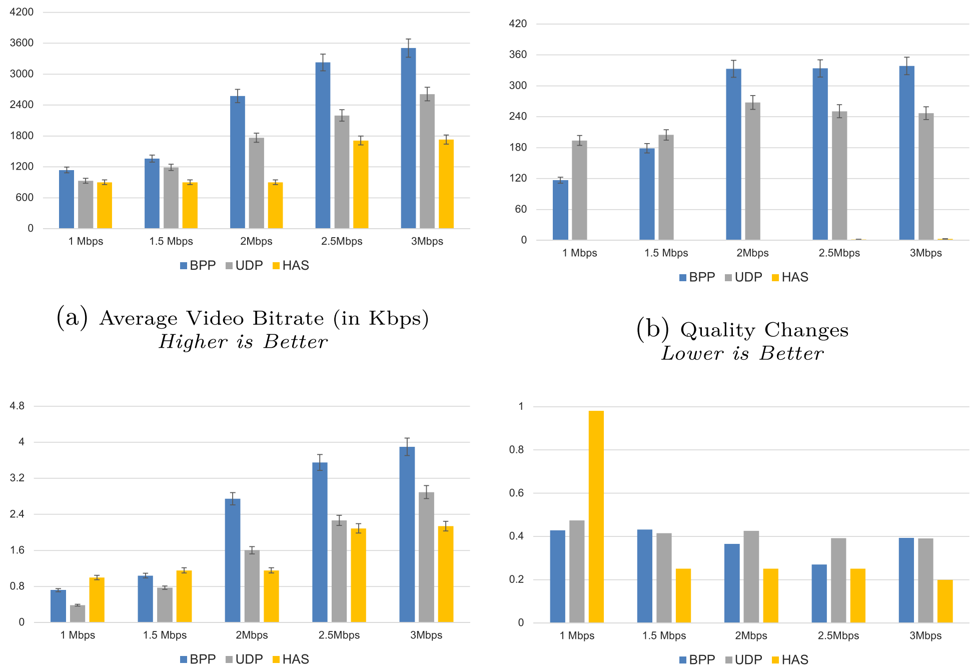
As previously stated, the available H264 SVC encoder/decoders are still experimental, so they cannot always do full video decoding, and thus we cannot always recreate the video stream. The tools we have do give some information; however, it is limited. A consequence of this, newer QoE video metrics such as VMAF, SSIM, or ITU-T rec. P. 1203 cannot be evaluated.

In the experiments, there are 6 clients connected to the server, where runs are conducted using a fixed bandwidth value with a range between 1 Mbps and 3 Mbps in fixed bandwidth tests. We also run experiments with dynamic bandwidth. In Dynamic Scenario 1, the bandwidth value starts with 3 Mbps, then drops to 2 Mbps, and then drops to 1 Mbps. In Dynamic Scenario 2, the bandwidth value starts again with 3 Mbps but then drops to 1 Mbps and increases to 2 Mbps. Each bandwidth value change occurs with a 14 s interval. The total video duration is 42 s. Each experiment is reconfigured for HAS and UDP using the same networks conditions. HAS is inherently adaptive, but there is no quality adaptation in the experiments with UDP. The initial buffering time is set to 600 ms for all approaches, to provide a transmission with low latency. The performance results observed in these experiments follows, where the obtained parameters are averaged and presented in the graphs with a 0.95 confidence interval.

6.2 Comparative performance evaluation

The QoE parameters are collected and averaged in this set of experiments for each protocol. In Fig. 7a, the average received video bitrate values are presented for each protocol. The clients with UDP do not adapt quality, so the bitrate values of the UDP clients are higher than HAS but this causes UDP clients to experience high level of duration of pauses. The clients using BPP play the video with a higher quality compared to UDP and HAS clients. Although HAS clients

Fig. 7 Quality parameters in fixed bandwidth conditions



(a) Average Video Bitrate (in Kbps)
Higher is Better

(b) Quality Changes
Lower is Better

(c) Overall QoE
Higher is Better

(d) QoE Fairness
Higher is Better

have the lowest bitrate, other QoE parameters observed with HAS clients show that the ABR algorithm aims to keep duration of pauses and the number of quality changes at minimum. The number of quality changes, in Fig. 7b, is high with BPP since the quality can change on a frame-by-frame basis, rather than with HAS where the quality change is limited to a 2 s segment. Clearly, there can be a very large number of packets transmitted every 2 s.

In addition, the *Overall QoE value* is calculated by using a modified version of Eq. 1. We added a new QoE parameter to the calculation, based on the number of interruptions which are a direct cause of the duration of pauses. Because it is a parameter that negatively affects the QoE, we subtract the number of interruptions value in the formula. In the enumeration of the *QoE value*, the average received video bitrate is used as the Q_k parameter. We normalized all QoE parameters because the bitrate unit is in Kbps, which is higher than the other parameters, namely the number of the quality switches and the duration of pauses. In these QoE calculations, the α parameter, which shows the importance of the video bitrate is set to 5. We also change the penalty for the number of quality switches, the number of interruptions, and the duration of pauses, and use the weights 0.01, 0.5, and 0.5, respectively.

When we examine the overall QoE values, Fig. 7c, we see that BPP outperforms other approaches with the fixed bandwidth experiments, expect in the tests with very limited bandwidth, where it equals to 1 Mbps. In this highly limited bandwidth test, HAS clients always keep the quality at the lowest and this conservative approach helps to get higher QoE value.

We also measure the QoE fairness value and give the related graph in Fig. 7d. We see that the best QoE fairness value is obtained with HAS, where the bandwidth equals to 1 Mpbs. In this test group, all clients requested the lowest quality for all segments and get very similar QoE values. For other bandwidth settings, we see that BPP and UDP provide slightly higher QoE fairness than HAS, where the QoE values are higher with BPP than UDP as seen in Fig. 7c.

In Figs. 8a and Fig. 8b, we provide QoE related results for the tests conducted with dynamic environment. The observed QoE values are higher with BPP than the other approaches. We observe QoE fairness behaviour similar to those given in fixed bandwidth tests. These results show that BPP performance behaves similarly in different networks conditions by providing higher QoE values than the other approach. But there is still room to enhance the performance by taking into account some QoE parameters, namely the number of quality switches and the number of interruptions.

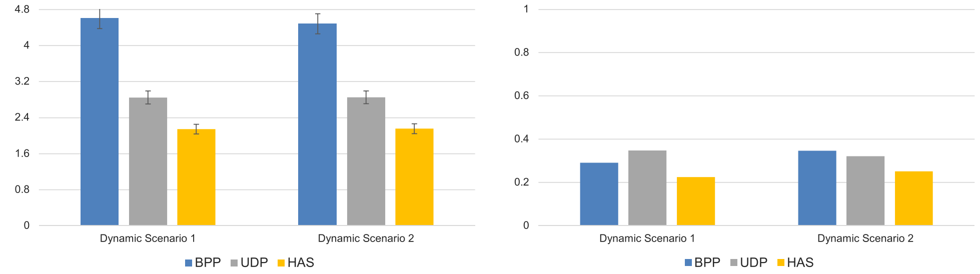
7 Conclusions

In this paper, we have shown the advantages of having in-network quality adaptation by presenting a number of performance results and comparing it to HAS. The comparative experiments show that in-network video quality adaptation is a promising approach that can meet the requirements of future video streaming applications. The results show that BPP adapts the quality in an efficient way, such that higher average received bitrate can be obtained when compared to HAS. However, HAS managed to keep the number of quality switches minimized, compared to BPP.

An architecture that utilizes virtualized BPP functions at the edge, for video delivery, has been presented. We showed the use of an ONOS controller as a solution to implement in-network quality adaptation, but on the other hand, it does add a huge burden to the controller since it also has the responsibilities to manage the network. The experimental results have shown that implementing in-network quality adaptation at the edge, and by using a virtualized BPP function, provides scalability and an improvement in QoE.

We compared a number of protocols in this paper, and demonstrated good performance via the experimental results, which has shown that the QoE obtained by trimming packets at the edge is better than using an ONOS controller. In the experiments utilizing the virtualized BPP functions, the average video bitrate is higher than those experiments utilizing the ONOS controller. The insights observed from this study show that in-network video quality adaption might provide enhanced QoE. Overall, the QoE value could be even higher

Fig. 8 QoE related results in dynamic network conditions



(a) Overall QoE (Dynamic) *Higher is Better*

(b) QoE Fairness (Dynamic) *Higher is Better* if more refined approaches, which consider the number of quality changes, are developed. Doing quality adaptation at the client will always have the advantage of receiving information about internal parameters, where one of the most important among them being the buffer level.

For future work, we will consider a number of improvements. We will adapt the algorithm that trims the chunks from the packets, and make it more refined. Currently, it only pays attention to the available bandwidth and the bandwidth used, and does not consider the number of quality changes. This number is quite high in our experiments, but for better QoE values, and for perceptual reasons, it is ideal to reduce the number of quality changes for smoother delivery. We will also investigate the use of packet trimming using hardware systems, if these become available, in order to provide higher throughput. Furthermore, to overcome the limitation of available working H264 SVC decoders, we will investigate the availability of H.266 SVC or AV1 SVC codecs which can be connected to a BPP transport, in order that we may evaluate a standardized QoE model, such as ITU-T rec. P. 1203, VMAF, or SSIM. With a working SVC decoder we can also gather Mean Opinion Scores (MOS) data experimentally from human subjects.

Acknowledgements Dr S. Clayman is funded by the TUDOR project (UK DCMS). This work is funded by TUBITAK Electric, Electronic and Informatics Research Group (EEEAG) under grant 121E373

Availability of data and materials Not applicable.

Declarations

Conflicts of interest The authors declare no competing interests.

References

1. Handley M, et al. (2017) Re-architecting datacenter networks and stacks for low latency and high performance. ACM SIGCOMM, 29–42 (Los Angeles, CA, USA)
2. Dong L, Li R (2019) In-packet network coding for effective packet wash and packet enrichment. IEEE Globecom Workshops (GC Wkshps) 2019:1–6
3. Westphal C, Makhijani K, Li R (2019) Packet trimming to reduce buffer sizes and improve round-trip times - extended abstract. Buffer Sizing Workshop (BS'19) (ACM)
4. Li R, Clemm A, Chunduri U, Dong L, Makhijani K (2018) A new framework and protocol for future networking applications. NEAT 2018 - Proc. of ACM Workshop on Networking for Emerging Applications and Technologies
5. Schwarz H, Marpe D, Wiegand T (2007) Overview of the scalable video coding extension of the H.264/AVC standard. IEEE Trans Circuits Syst Video Technol 17(9):1103–1120
6. Clayman S, Sayit M (2021) In-network scalable video adaption using big packet protocol. Proceedings of the 12th ACM Multimedia Systems Conference, MMSys '21, 363–368 (ACM)
7. Clayman S, Sayıt M, (2023) Low latency low loss media delivery utilizing in-network packet wash. J Netw Syst Manage 31(1):29. https://doi.org/10.1007/s10922-022-09712-1
8. Zhu B et al. (2021) A real-time H.266/VVC software decoder. Proceedings of IEEE International Conference on Multimedia and Expo (ICME)
9. Clayman S, Tuker M, Arasan H, Sayıt M (2021) The future of media streaming systems: transferring video over new IP. IEEE 22nd Intl. Conf. on High Performance Switching and Routing (HPSR) (Paris)
10. Clayman S, Tuker M, Arasan H, Sayıt M (2021) Managing video processing and delivery using big packet protocol with SDN controllers. IEEE Conf. on Network Softwarization – Netsoft (Tokyo)
11. Clayman S, Tuker M, Karakş E, Sayıt M (2023) In-network video quality adaption using packet trimming at the edge. 2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 161–168 (Paris)
12. Yang S, Wei Z, Ling L (2011) Multimedia communication and scalable video coding. Fourth Intl Conf on Intell Comput Technol Automation 616–619 (Shenzhen, China)
13. Crowcroft J, Handley M, Wakeman I (1999) Internetworking multimedia (CRC Press)
14. RTP: a transport protocol for real-time applications. https://tools.ietf.org/html/rfc3550
15. Clark DD, Tennenhouse DL (1990) Architectural considerations for a new generation of protocols. SIGCOMM, 200–208 (ACM, Philadelphia)
16. Real Time Streaming Protocol (RTSP). https://tools.ietf.org/html/rfc2326
17. Chromium Blog. https://blog.chromium.org/2020/05/celebrating-10-years-of-webm-and-webrtc.html
18. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540. https://httpwg.org/specs/rfc7540.html
19. MPEG-DASH: dynamic adaptive streaming over HTTP. https://mpeg.chiariglione.org/standards/mpeg-dash
20. Mueller C (2018) Low latency streaming: what is it and how can it be solved?. https://bitmovin.com/cmaf-low-latency-streaming/
21. Lim M, Akcay MN, Bentaleb A, Begen AC, Zimmermann R (2020) When they go high, we go low: low-latency live streaming in dash.js with lol. Proceedings of the 11th ACM Multimedia Systems Conference, MMSys '20, 321–326 (ACM). https://doi.org/10.1145/3339825.3397043
22. Taraghi B, Hellwagner H, Timmerer C (2023) LLL-CAdViSE: live low-latency cloud-based adaptive video streaming evaluation framework. IEEE Access 11:25723–25734. https://doi.org/10.1109/ACCESS.2023.3257099
23. Erfanian A, Amirpour H, Tashtarian F, Timmerer C, Hellwagner H (2021) Lwte-live: light-weight transcoding at the edge for live streaming. Proceedings of the Workshop on Design, Deployment, and Evaluation of Network-Assisted Video Streaming, VisNEXT'21, 22–28 (ACM). https://doi.org/10.1145/3488662.3493829

24. Bentaleb A, Taani B, Begen AC, Timmerer C, Zimmermann R (2018) A survey on bitrate adaptation schemes for streaming media over http. IEEE Commun Surv Tutorials 21(1):562–585
25. Bishop M (2021) Hypertext Transfer Protocol Version 3 (HTTP/3). Internet-Draft draft-ietf-quic-http-34, Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34
26. Seufert M, Schatz R, Wehner N, Casas P (2019) QUICker or not? -an empirical analysis of QUIC vs TCP for video streaming QoE provisioning. 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 7–12 (Paris)
27. Palmer M, Krüger T, Chandrasekaran B, Feldmann A (2018) The QUIC fix for optimal video streaming. Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC, EPIQ'18, 43–49 (ACM)
28. Nguyen M, Amirpour H, Timmerer C, Hellwagner H (2020) Scalable high efficiency video coding based http adaptive streaming over quic. Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC, EPIQ '20, 28–34 (ACM). https://doi.org/10.1145/3405796.3405829
29. Makhijani K, Li R, Boukary HE (2019) Using big packet protocol framework to support low latency based large scale networks. ICNS 2019 (Athens)
30. Dong L, Li R (2019) Distributed mechanism for computation offloading task routing in mobile edge cloud network. International Conference on Computing, Networking and Communications (ICNC), 630–636 (Honolulu, HI, USA)
31. Albalawi A, Yousefi H, Westphal C, Makhijani K, Garcia-Luna-Aceves J (2020) Enhancing end-to-end transport with packet trimming. GLOBECOM 2020 - 2020 IEEE Global Comms Conf, 1–7
32. Popa A et al. (2022) Implementing packet trimming support in hardware. arXiv:2207.04967
33. Cao K, Liu Y, Meng G, Sun Q (2020) An overview on edge computing research. IEEE Access 8:85714–85728
34. Clayman S, Sayıt M (2022) The effects of packet wash on SVC video in limited bandwidth environments. IEEE 23rd Intl. Conf. on High Performance Switching and Routing (HPSR) (Jiangsu, China)
35. Open Network Operating System (ONOS®). https://opennetworking.org/onos/
36. Lederer S, Müller C, Timmerer C (2012) Dynamic adaptive streaming over http dataset. Proceedings of the 3rd Multimedia Systems Conference, MMSys '12, 89–94 (ACM)
37. Lantz B, Heller B, McKeown N (2010) A network in a laptop: rapid prototyping for software-defined networks. Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Association for Computing Machinery)
38. Yin X, Jindal A, Sekar V, Sinopoli B (2015) A control-theoretic approach for dynamic adaptive video streaming over http. SIGCOMM Comput Commun Rev 45(4):325–338. https://doi.org/10.1145/2829988.2787486
39. Seufert M et al (2015) A survey on quality of experience of HTTP adaptive streaming. IEEE Comms Surv Tutorials 17(1):469–492. https://doi.org/10.1109/COMST.2014.2360940

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.