Enhancing Packet Trimming Behaviour for Programmable Protocols

Stuart Clayman*, Müge Sayıt† David Griffin*, Miguel Rio*,
*Dept. of Electronic and Electrical Engineering, University College London, London, UK
†Dept. of Comp. Sci. and Elec. Eng., University of Essex, Colchester, UK

***Abstract*—One of the recent methods for effectively utilising the available bandwidth of the network as conditions change is in-network packet trimming. It has advantages for applications which rely on low-latency and low-loss, such as video streaming. Packet trimming allows a network node to reduce the size of packets as they travel across the network, by removing some of the content, during the journey of each packet. In our previous work, we presented a system for layered SVC video streaming, and showed how the trimming process can be successfully implemented and utilised as a virtualized edge deployed function. If trimming is needed, there is a task which has to unpack all the video content and meta-data, including associated significance values, then determine which data to remove, and finally repack and rebuilding a new packet. Conversely, a simple scheme for packet trimming, in the form of a basic buffer cutting process, has been implemented in switches. This process keeps the first bytes of a packet, and drops the rest. Here we investigate if a buffer cutting process can be used for layered video, and describe the issues arising and the design aspects needed to adapt from packet rebuilding to packet cutting.**

***Index Terms*—Programmable Protocols, In-Network Packet Trimming, Layered Video**

I. Introduction

One of the recent methods for effectively utilising the available bandwidth of the network as conditions change is *in-network packet trimming* [1]. It is a promising approach that is advantageous for applications which rely on low-latency, low-loss behaviour, such as video streaming. The utilization of the packet trimming technique allows a network node to reduce the size of packets as they travel across the network, by removing some of the content, during the journey of each packet. In our previous work, we showed how this can be implemented and utilised [2]. In particular, when sending video data and using packet trimming, the packets can be modified if there are some bandwidth limitations. In our scheme, the payload is split into data chunks, each having its own significance value as an indicator of its priority within the data. These data chunks, within the payload, can be removed in transit if necessary. Doing the trimming avoids most packet loss as it reduces the packet size, instead of dropping the packet during network congestion situations. As a consequence, it significantly decreases the need for retransmissions, which reduces the end-to-end latency.

In our previous work on packet trimming, we utilised one of the new Programmable Network Protocols, called Big Packet Protocol (BPP) [1]. Using this protocol, we have directly shown that packet trimming is ideal for providing low-latency, low-loss delivery of packets, particularly for video traffic [2]. We described an embedded algorithm for Packet Trimming, whereby the algorithm utilizes a Bandwidth Utilization Evaluator to calculate the current bandwidth available versus the number of bytes that have been sent in the time period [3]. The algorithm works out if the current packet should be trimmed by having some content chunks removed, when the number of bytes being sent is more than the available bandwidth.

We have discovered that packet trimming not only works well for fixed bandwidths, but as it is highly adaptable at a per-packet granularity, it also works well with dynamic bandwidth changes. These facilities are essential for enhanced next generation applications, making it ideal for both 5G+ and 6G scenarios, where trimming functions can be placed at the edge. Packet trimming can be deployed at the edge, by using specialised virtual edge nodes which do the trimming [4], and we have demonstrated that this works for 6G scenarios [5].

In the scheme outlined, the chunks have different significance values, and the trimming process must be able to keep the most significant chunks, and remove those chunks with less significance. The sender is responsible for such labelling, as it knows the context and requirements of the applications. The network node is application agnostic, and is not required to know anything application specific. The node processes the chunks and their significance, as the packets journey via the network node. If trimming is needed, the task has to unpack and split all the chunks and the meta-data, including the significance values, then determine which chunks to remove, and finally repack the chunks and rebuilding a new packet.

Conversely, simple packet trimming, in the form of a basic buffer cutting process, has been implemented in switches in data centers [6]. There, the switches trim the packets to just headers under congestion conditions. In [7], packet trimming is utilized in the context of ML Training. Again, a basic cut is undertaken at a specific position, pre-configured in the switch for all packets. This cutting method is used in both cases, and although it has no mechanism for evaluating different content chunks and no way to deal with different significance within the data, it is fast and easy to implement in hardware.

One of the big questions that arises for our approach to packet trimming is: *can we make packet trimming which utilizes a number of data chunks, each having it's own significance value, match with the trimming design of switches* ***and*** *simple buffer cutting?*

In this paper we present a method to enhanced In-network

Packet Trimming behaviour in edge-based network nodes, using an approach that uses buffer cutting rather than the current packet splitting and rebuilding. Overall, we can say that our trimming process is non-trivial, however it can deal with the different chunks and the different significance values.

Changing over to a cutting process does raise a number of issues and consequential effects which need to be resolved. Firstly, we need to update the algorithm for trimming in order to cut the buffer at specific point, and not rebuilding. Secondly, as there are no updates into the packet content, we notice that with cutting the meta-data related to the chunks does not get modified by the node. This means that the client is responsible to double check that the meta-data is correct for the content chunks. Thirdly, the network node has no guarantee that the significance values for the chunks are organised and sorted in an order that is useful for a cutting process. Our process requires that the most significant chunks are kept, but if there is just cutting, how to we maintain this requirement ?

Another aspect we need to address is that although packet trimming approach has been successful in many ways, we have observed trimming limitations when we get close to the bandwidth limit. In [8], we presented an approach which overcame some of these issues by removing extra content from the packets when network conditions were restricted. However, we have observed there are still some situations where even that approach was not enough, and here we present an enhancement where we trim more aggressively, independent of the meta-data priorities and significance set in the packet.

The contributions of this paper are: (i) a trimming method that uses cutting rather than rebuilding as way to trim in a faster, less intensive approach at the node. The side-effect being that the client needs to do more work to validate the packet; (ii) a mechanism deployed in the sender to reorder the chunks in the packet, by the significance value not by the position. Another side-effect is that the client has to reorder the chunks by position when the packet arrives to create valid structures; and (iii) add an approach for more aggressive trimming in the network node for when we get close to the bandwidth limit. This overrides the meta-data in the packet.

We will present these methods and designs in section III, an evaluation in the section IV, plus our conclusions and future work in section V of the paper.

II. Background

A. Packet Trimming

Packet trimming is a technique that reduces packet size while packets are in transit through the network [1], [9]. Under congestion situations, conventional mechanisms drop packets when buffers overflow. In contrast, packet trimming selectively removes part of a packet's payload instead of discarding the entire packet. This approach requires network nodes to maintain more detailed information regarding link conditions and active flows. Thereby, packet trimming enables more controlled use of available bandwidth and improves effective bandwidth utilization.

Packet trimming can significantly reduce packet loss and can make more efficient use of network capacity by shrinking packets rather than dropping them. However, achieving good performance depends on fast hardware capable of modifying packets at transfer rate. When congestion arises, packets are trimmed at the bottleneck link according to the available capacity, avoiding packet drops which would otherwise occur.

Interest in packet trimming has increased in recent years as advances in programmable and high-speed network hardware have made in-network packet modification feasible. Handley et al. investigated packet trimming in data center environments, presenting the design of the NDP architecture [6]. NDP achieves near-optimal completion times for short flows while sustaining high throughput across diverse scenarios. In NDP, switches trim packets to just headers, which are then forwarded using high-priority queues. This ensures rapid notification to the receiver that the packet payload has been removed. Upon receiving a trimmed header, the receiver can generate a negative acknowledgment (NACK), also transmitted via high-priority queues, prompting the sender to retransmit the original packet. As noted by the authors, packet trimming makes actual packet loss extremely rare within the network.

In [7], the authors utilize packet trimming for dealing with ML training data when there is network congestion. They use a technique they call Just-in-Time Gradient Compression, which uses Packet Trimming to reduce packet size, but still deliver ML data. They note that the capability of dealing with temporal congestion via packet trimming provides an ideal opportunity to provide just-in-time gradient compression. They presented an approach where packets with gradients are specifically created to be compressible via packet trimming. This avoids retransmission delays, which is a common problem with congested ML training environments. Also their preliminary evaluations show that trimmable encoding has low computational overhead. One of the limits they observed is that *the position in which a packet is trimmed is fixed* for all packets. They say that if a trimmed packet contains 45 bytes of compressed payload, and there is a 42-byte standard header (for Ethernet, IP, UDP), then the switch should be configured to trim packets at 87 bytes upon congestion.

There are currently no standardized transport protocols which support packet trimming as a fundamental operation. Big Packet Protocol (BPP) has been specified as a *Programmable Protocol* that uses trimming as a part of its programmability features [1]. BPP has been specifically designed for use with applications that require both low latency and high reliability. BPP allows applications to set application specific meta-data into the packets, and get specific behavior with direct support from suitably enhanced network nodes, at the granularity of a flow or an individual packet. This is done by utilizing functionality built into enhanced network nodes [9]. BPP packets have fields for meta-data which are processed by switches and routers and acted on during the transmission of the packet. The payload of a BPP packet is partitioned into a header and some chunks. Some of these chunks can be dropped during their journey over the network, depending

on the values set in the header and on the current load of the network links. The purpose of having trimming is that packets are transferred from the sender to the client without needing any retransmission. The task of trimming the packet content in the network (called *Packet Wash* in BPP), has an impact on the application at the client as not all the data arrives. These applications have to adapt to differences in the sent payload verses the received payload.

B. Emergent Video Streaming Technologies and Methods

Recent work in video streaming generally focuses on achieving low-latency delivery while maintaining high Quality of Experience (QoE). As one of the most prevalent streaming technologies, Dynamic Adaptive Streaming over HTTP (DASH) has received significant attention, particularly in the context of Low-Latency DASH (LL-DASH) [10]. Low latency in LL-DASH is enabled through chunked encoding and chunked delivery mechanisms [11].

With the emergence of Multi-access Edge Computing (MEC) as a key architectural component of Next Generation Networks, content and processing capabilities can be deployed closer to end users, enabling further latency reduction. Several studies have investigated the use of MEC to enhance QoE and reduce latency in video streaming scenarios. These works primarily explore mechanisms such as edge caching [12], [13], QoE-aware decision making and prediction [14], and Quality of Service (QoS) provisioning for 5G networks [15].

In addition, to support smooth video playback with low latency and minimal packet loss, the authors in [16] examine the challenges and opportunities of deploying Low Latency Low Loss Scalable Throughput (L4S) over 5G networks. L4S leverages Explicit Congestion Notification (ECN) to provide early congestion feedback, allowing senders to react before excessive queuing delays build up in the network.

While a broad range of studies aim to improve either the latency or the QoE for video streaming applications, existing solutions largely rely on network provisioning, caching, or adaptive bitrate strategies. In contrast, few approaches exploit emerging transport-layer mechanisms or real-time congestion mitigation techniques.

Our approach follows a different design principle: instead of reacting to congestion after it occurs, the outlined packet trimming mechanism proactively prevents or minimizes congestion, thereby enabling low-latency and low-loss video delivery. With BPP, packets are specified with a number of content *chunks*, any of which can be *trimmed* during transmission by a network node. In our work, we use these separate chunks for the individual layers of the layered video – a 3 layer video will use 3 chunks in a packet. In our previous work [3], [17], we presented an implementation of BPP and demonstrated the benefits of packet trimming for video streaming. In [18], Mohammadreza et al. demonstrated that our previous work is suitable for streaming of 2K game flows. Their scenarios involve high-resolution streaming of three genre-diverse games, with experiments having a sudden bandwidth drops of up to 50%. They conclude that the packet trimming approach preserves visual quality with negligible degradation even during sudden bandwidth drops.

In this paper, we extend our previous work by focusing on accelerating the packet trimming process at network nodes and improving scalability by shifting part of the processing complexity from the network to the client side. We further investigate the impact of this design on QoE in low-latency video streaming scenarios.

III. Design Aspects

In this section we present the approaches and design aspects required as we adapt from packet trimming using rebuilding to packet trimming using buffer cutting. We are investigating whether a buffer cutting process can be used for trimming packets, and for layered SVC video in particular. We describe the issues arising and the solutions needed to address these.

When using a buffer cutting approach as opposed to a packet unpacking, splitting and rebuilding approach, we would expect this to be a method to trim in a faster, less CPU intensive approach at the network node. To update our system to do cutting, we need to modify the algorithm for trimming in order to cut the buffer at specific point, and not do the packet rebuilding. As there are no updates which set new values into the packet content when doing cutting, we notice that the meta-data related to the chunks does not get modified by the node. A side-effect of this scheme is that the client needs to do more work, as the network node does not update any fields in the meta-data. This means that the client is responsible to double check that the meta-data is correct for the content chunks.

A. Split and Rebuild

The approach we have used previously, has used a split, unpack, trim, and rebuild method. The packet, whose encoding is presented in [2], is split and unpacked from the bitwise encoding into internal objects. The algorithm does a bandwidth calculation and potential trimming occurs. Those objects can be manipulated by the algorithm to select the chunks to keep and the chunks to trim. Finally the objects are turned back into a bitwise encoding, with the correct meta-data, and the relevant data to forward is created with a packet rebuild process from the object parts.

In Fig. 1 we show how the packet is split into objects, and if a chunk is trimmed then within the rebuilt packet the meta-data block is modified, the trimmed chunk is eliminated.

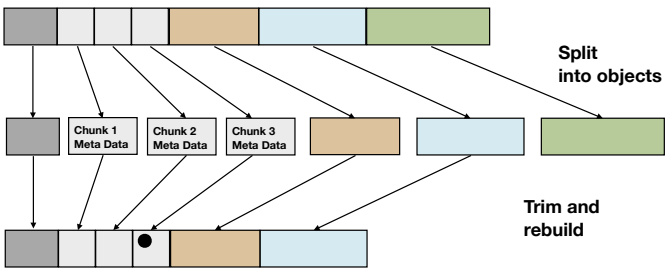


Fig. 1: Splitting a packet into objects and rebuilding it. 3rd chunk is trimmed – 3rd meta-data is modified

B. Calculate and Cut

One of the big questions we are addressing is whether we can make packet trimming, which utilizes a number of data chunks, match with a trimming design which uses simple buffer cutting. We need to update the algorithm for trimming in order to cut the buffer at specific point, and not do the splitting and rebuilding process.

For the buffer cutting approach, the packet is not deconstructed and split into objects. Rather, the trimming calculations are done and a trim position is evaluated, instead of removing a chunk object. At the end of evaluating all of the chunks, the byte buffer of the payload is cut at the calculated position. With this approach no bytes in the payload are updated, and the only effect is that the byte buffer may be smaller, if trimming occurs. The resulting bytes are forwarded onwards. As the meta-data for trimmed chunks is not updated, the client has to cross-check their status and evaluate the validity of the data before using the contents.

In Fig. 2 we show how the packet is cut. The position is calculated and the buffer is cut in order to eliminate specific chunks. As it is just a cut, no meta-data updates are done.

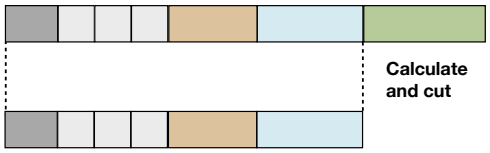


Fig. 2: Calculating position and Cutting it.
3rd chunk is cut – no meta-data updates

Here we present the changes required to support the buffer cutting process, by considering the sender, the network node, and the client.

1) Sender: At the sender things are the same. The packets are created in the same way as the *Split and Rebuild* approach.

2) Node: At the network node, the process needs to calculate where each chunk ends so it knows the cutting position. As the trim is only done at the chunk boundary position, knowing the chunk sizes is important. The trimming process still visits each chunk to determine if it should be trimmed. Instead of removing a chunk as in the *Split and Rebuild* approach, it just updates a cutting position and then considers the next chunk. After visiting all the chunks, the byte buffer will be cut. It does do not have rebuild a packet or update meta-data blocks, it just sends a new packet with the trimmed payload.

3) Client: At the client, it has to inspect all the incoming packets and recalculate the meta-data state and the chunk values from incoming packet contents. It first evaluates the sizes of the content from the meta-data, and then the size of all the received chunks. It checks if the sizes are the same. If so, the packet arrived with no cutting. If the sizes are different, the client knows that something was cut by a network node. As there are no updates into the packet content by the network node, we know that with cutting, the meta-data related to the chunks does not get modified. From here, client can re-evaluate the correct meta-data and validate chunks, before it passes the chunks upward to the application.

C. Chunk Reordering by Significance Value

Another aspect that needs to be considered is that the network node has no guarantee that the significance values for the chunks are organised and sorted in an order that is useful for a cutting process. Our process requires that the most significant chunks are kept, but if there is just cutting, how to we maintain this requirement ? In the BPP protocol, the significance value of any chunk is set by the sender, who knows the importance of the chunks to the whole data set.

For the video examples we have set a higher priority for layer 0 (the base layer) of the video, a higher significance for layer 1, and the highest significance for layer 2. This allows the trimming process to eliminate less significant video data in the network. In Fig. 3, where chunk 1 has significance 1, chunk 2 has significance 5, chunk 3 has significance 9, we see how the significance value increases for each chunk position, such that when the packet is trimmed at the network node, the resulting packet is correct.

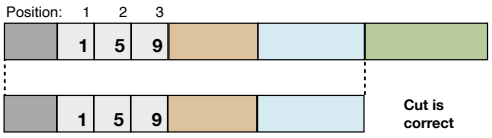


Fig. 3: Significance of chunks is: 1, 5, 9.
Trim by cutting is correct

It is possible that the significance value of the chunks can be in any order. Only the applications know the priorities of the chunks, and these are labelled by the sender and processed by the client. So in a packet, the chunks can be created in a specific order, but their significance can be any value. For example, the last chunk significance could be higher than the first chunk, or the middle chunk could have the lowest significance value. In our design, significance values range from 1 to 15, where a value of 1 denotes the most significant chunk and 15 denotes the least significant chunk [3]. In Fig. 4, we see how the significance value of the chunk 2, namely 9, is less significant than chunk 1 and chunk 3. The trimming process should keep chunks 1 and 3, but trim chunk 2. With buffer cutting we do not have the split and rebuild process which can drop any chunk, there is only a simple cut. As we see, the cutting produces the wrong result in the packet. It can be compared to the expected result.

A solution to this problem is for the sender to reorder the chunks by their significance value, and keeping track of the original position before the chunks are reordered. We have seen what happens if we do not consider the significance value of the chunks, but still do cutting – namely we cut the wrong chunk. Reordering at the sender prevents this problem. In Fig. 5, we see the structure of the packet after the sender has reordered the chunks. They are now sorted by significance value, so the chunk order went from [1, 2, 3] to [1, 3, 2] and the significance values went from [1, 9, 5] to [1, 5, 9].

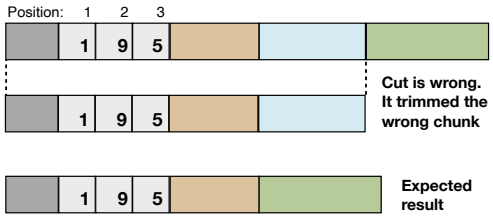


Fig. 4: Significance of chunks is: 1, 9, 5.
Trim by cutting is wrong

When the network node cuts the packet, the correct chunks are kept, but the changed order is forwarded to the client.

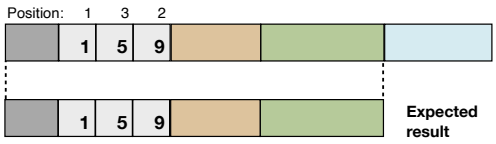


Fig. 5: Chunks are reordered by significance value.
Trim by cutting is correct

By reordering chunks by the significance value, and not sending by the original chunk position order, means the client has to do pre-processing on the packet. The client has to check the position of each chunk and reorder these in order to create a valid structure. In Fig. 6, we see the corrected order. This version is sent up to the application.

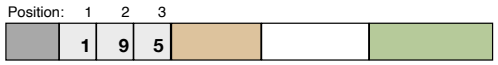


Fig. 6: Client has to reorder chunks by chunk position.
2nd chunk been trimmed

Here we present the changes required to support the reordering process, by again considering the sender, the network node, and the client.

1) Updated Packet: To support ordering the chunks by significance value, and then reordering them again at the client by chunk position, we need to keep track of the original chunk position when the packet is created at the sender. Currently, the meta-data for each chunk has a field for the significance value, but the chunk position was not explicitly held. It is calculated by going through a loop. As such we needed to add a field in the meta-data to keep the chunk position.

The current meta-data block in the packet contains 6 bytes (48 bits) for each chunk. Nearly all the bits are utilised, but there was one padding field of 5 bits. This is ideal for our use, and we have repurposed it to hold the chunk position. 5 bits allows us to keep track of 32 chunks, which should be adequate for most applications.

2) Sender: At the sender, the application will create data objects in chunk position order. The sender then has to create packets which are sorted into significance value order. As the bitwise encoding of the data is mapped into a packet, the sender first labels a chunk position field into the meta-data, and then reorders the chunks by significance value. The packets are sent onto the network once this ordering has taken place.

3) Node: At the network node the buffer cutting approach of section III-B is used and the trimming is done at chunk boundary position. As cutting is not able to split and rebuild a packet or update meta-data blocks, the node just cuts the reordered buffer, as in Fig. 5, providing the expected result.

4) Client: At the client, it has to inspect the incoming packet, and determine if the chunks are in the correct order, or check if they were reordered by the sender just before transmission. If they are reordered, it has to reorder the chunks by position order. It also has to rebuild the meta-data state and ensure valid chunks are passed up to the application.

D. Policy based Enhanced Trimming

Another behavioural aspect we need to address is that although the packet trimming approach has been successful in many ways, we have observed trimming limitations when we get close to the bandwidth limit. In [8], we presented an approach which overcame some of these issues by removing extra content from the packets when network conditions were restricted. However, we have observed there are still some situations where even that approach was not enough.

When examining some of the videoes used in [18], and the PES2019 video in particular, we have seen issues with the specific encoding options used in the JSVM tool for these large videos. This generates huge video frames, which will then create a large number of packets for a single frame, and as a consequence means the trimming doesn't always operate as ideally as we wish. In the PES2019 video, which is encoded at 32 Mbps, there are multiple frames of over 1 megabyte, and so the spikes in the volume of encoded video are more than the bandwidth utilization evaluator determines is available at the network node. The default trimming approach is unable to reduce the bandwidth as required.

The *basic* bandwidth utilization evaluator algorithm, which we presented in [3], works reasonably well for most situations. The main approach of the algorithm is to estimate the traffic volume which can be forwarded onwards based on the available bandwidth of the output link plus the number of milliseconds into the current second. If needed, it will trim chunks. The *Trim By Significance Value* approach has the following operation:

- if we need to trim *AND* the significance value of a chunk > threshold set in the packet, *THEN* trim the chunk

It has shown good results, although it is not perfect. Our enhancement is to have policies / rules so that we trim more aggressively, independent of the meta-data priorities and the significance values set in the packet.

In the following algorithm we have extended the `trimContent()` function, to go beyond the standard trimming approach which inpects the significance value for each chunk, and added 2 new policies:

- if the bandwidth used is >= 90% of the bandwidth, *THEN* trim all the chunks except the first chunk,

- if the bandwidth used is >= 80% of the bandwidth, *THEN* trim just the last chunk,

These policies do not inspect the significance values for any chunk, but the forcibly trim the chunks. The following algorithm shows how these are combined.

```
                                  ▷ Main processing loop to trim content
function trimContent(packetTrimLevel):
    // Visit chunks from last to first
    for c ← chunkCount to 1 do
        if (bandwidthUsed >= 0.9) then
                              ▷ Always trim top chunks – keep chunk 1
            if (c > 1) then
                TrimChunk (c)
        else if (bandwidthUsed >= 0.8) then
                                          ▷ Always trim top chunk
            if (c == chunkCount) then
                TrimChunk (c)
        else
                                    ▷ Standard trim by significance value
                                  ▷ No guarantee chunk will be trimmed
            TrimBySignificance(c, significance(c))
```

All of the design aspects needed to enhance the packet trimming behaviour for an end-to-end system have been presented. We now show an evaluation of these approaches.

IV. Performance Evaluations

In this section, we evaluate the performance of the proposed approach from two complementary perspectives. First, we analyze its impact on the computational complexity at the network node. Second, we assess the resulting effects on the QoE. We need to determine if buffer cutting provides a working and effective solution for packet trimming, and produces accurate video files. Also, we need to determine if the extra trimming provided by the policies is also effective.

A. Experimental Setup

To evaluate the performance of the proposed system, we conducted experiments using the well-known open-source animated film, Big Buck Bunny. The raw source material was obtained from the Xiph.org Foundation media repository [19].

The video sequence was encoded into a SVC format using the Joint Scalable Video Model (JSVM) reference software [20]. We configured the encoder to produce a 3-layer temporal and quality scalable stream. The encoding parameters were selected to maintain high fidelity while demonstrating significant bitrate differentials between the base and enhancement layers. The resulting experimental H.264/SVC video file has a total size of 53.32 MB, encompassing 12.5 seconds of footage.

The bitstream structure consists of 300 frames, comprising 900 Video Coding Layer (VCL) Network Abstraction Layer (NAL) units and 607 non-VCL NAL units. The specific video details and cumulative bitrates for the resulting layers are detailed in Table I. We use the *Even Split* packing strategy at the sender, described in [2], for collecting the video layers and structuring the data, which transmits 69144 packets.

For the evaluation of the end-to-end system we have 4 different strategies:

- **Rebuild** – this is the approach we have used previously, and uses a split, unpack, trim, and rebuild method for processing packets explained in Sec. III-A.
- **Rebuild + Policy** – this is an enhancement of the **Rebuild** approach, with the addition of the policies for extra trimming explained in Sec. III-D.
- **Cut** – this is an approach using the newly devise cut method, explained in Sec. III-B.
- **Cut + Policy** – this is an enhancement of the **Cut** approach, again with the addition of the policies for extra trimming explained in Sec. III-D.

B. Performance of Different Approaches in the Network Node

As the **Cut** strategy is expected to reduce the processing load on the network node, we first measure the CPU utilization associated with the rebuild and cut strategies. Fig. 7 presents the CPU load as a function of time for both approaches. From the figure, it can be observed that the **Rebuild** strategy imposes a significantly higher computational burden on the network node. In contrast, the **Cut** strategy results in consistently lower CPU utilization. These results indicate that, while an increasing number of flows may introduce serious scalability issues, the cut strategy is a promising approach for video streaming with packet trimming.

In Fig 8 we compare the amount of bandwidth used by the **Cut** strategy and the amount used by the **Cut + Policy** strategy. Given our design goals, ourlined in Sec. III-D we expect to see that Cut + Policy is more aggressive in trimming chunks from the video. For this evaluation we specifically used the PES2019 video created for [18] because of its large frame sizes. As stated, this video is encoded at 32 Mbps, but there are multiple frames of over 1 megabyte, and so the spikes in the volume of encoded video are more than is available at the network node. With the Cut line, shown in pink in the figure, we see that the system attempts to use more than 100% of the available bandwidth. The Cut + Policy line, shown in green in the figure, we see that the extra trimming means the system does not attempt to use more than 100% of the available bandwidth. It shows Cut + Policy really does trim more content and help us avoid sending more than the available bandwidth.

TABLE I: The Characteristics of the Video File

Video type	H.264	**PSNR**	44.68 dB
Resolution	1280 x 720	**File Size**	53.32 GB
Frame No	300	**L0 bitrate**	20.1 Mbps
Frame rate	24 fps	**L0+L1 bitrate**	30.2 Mbps
Duration	12.5 sec	**L0+L1+L2 bitrate**	34.9 Mbps

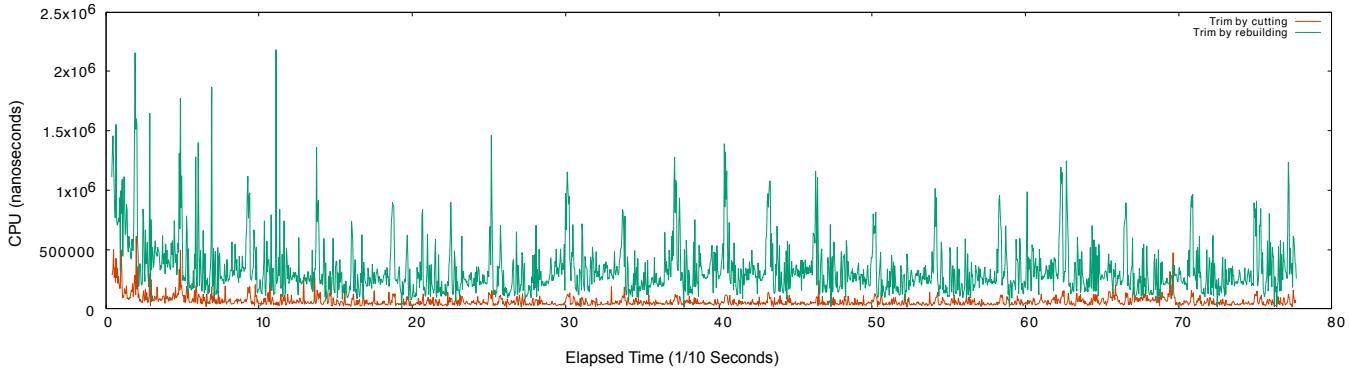


Fig. 7: Network Node Performance of Rebuilding (Rebuild approach) vs Cutting (Cut approach)

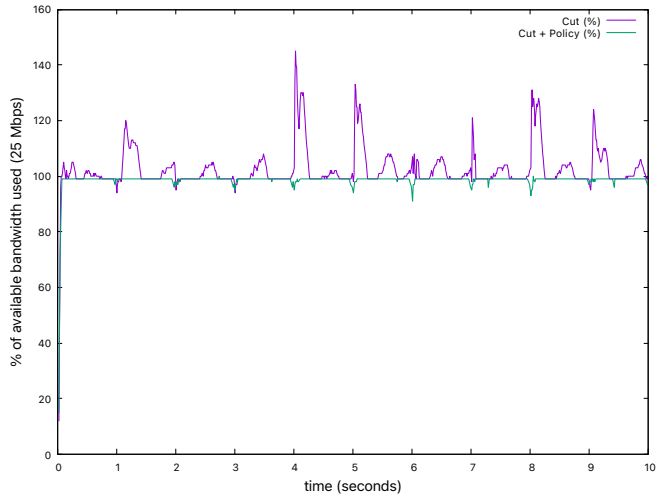


Fig. 8: Bandwidth utilization at Network Node – Cut vs Cut + Policy (First 10 seconds)

C. Performance of Different Approaches at the Client

To evaluate the impact of different delivery approaches on client-side performance and QoE, we measured network latency and Peak Signal-to-Noise Ratio (PSNR). To mitigate latency and prevent playback stalls, a pre-buffering strategy was implemented prior to stream initialization. Figure 9a illustrates the startup latency for each configuration; notably, these values correspond to the minimum buffering thresholds required to guarantee continuous, stall-free playback.

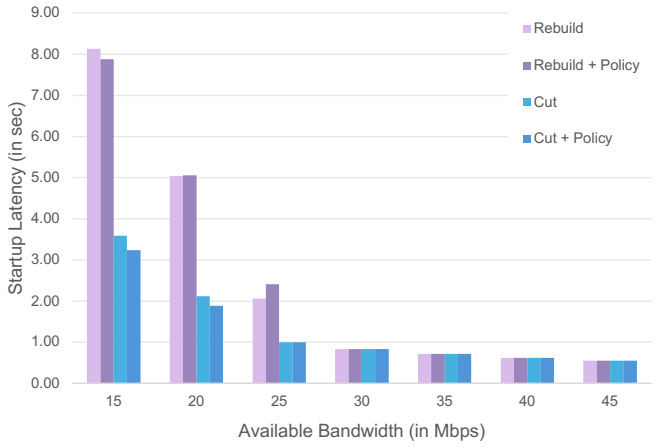
The results reveal significant differences in startup latency across the evaluated approaches, particularly under constrained bandwidth conditions. With the rebuilding strategy, the startup latency is approximately 8 sec and 5 sec for available bandwidths of 15 Mbps and 20 Mbps, respectively. By contrast, the proposed cut-based approach reduces the startup latency by nearly 50%. When the bandwidth increases to 25 Mbps, the startup latency for both the cut and cut-policy approaches decreases to approximately 1 sec. This residual latency is primarily attributed to the transmission of the first video frame from the server to the client. Playback begins immediately upon reception of the first I-frame, after which the stream is delivered at the playback rate. This is enabled by the adaptive packet trimming mechanism, which allows smooth playback even when the encoded video bitrate (35 Mbps) exceeds the available network bandwidth (25 Mbps).

The PSNR value of the streamed video is 44.68 dB. Fig. 9b presents the PSNR values obtained across the different experimental configurations, after trimming operations implemented. The results indicate that the *new policy* introduces a slight degradation in visual quality compared to the rebuilding strategy, with an average PSNR reduction of approximately 0.3 dB. This effect is mainly due to more aggressive trimming operations triggered when operating close to the bandwidth limit. In contrast, the cut strategy exhibits almost no negative impact on PSNR. The client achieves PSNR levels comparable to those of the rebuilding strategy, demonstrating that latency reductions can be obtained without compromising visual quality.

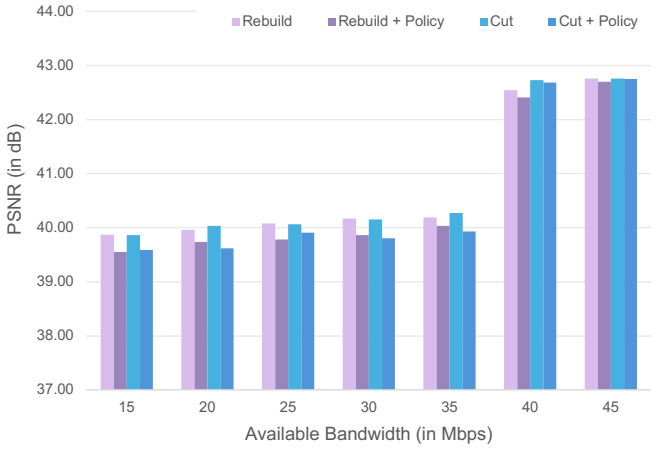
Table II reports the total size of the received video data (in bytes) for each strategy. The client may actually collect more bytes, as it can get packets with unusable data, but it has a *cleaning* mechanism to deal with inconsistent NALs [2]. If needed, some NALs which have lost their dependent NALs can be cleaned away, providing a valid stream for the decoder. The results show that, particularly under constrained bandwidth conditions, the policy-based strategies trim a larger amount of data compared to their non-policy counterparts, as expected. Nevertheless, as indicated by the PSNR results, this additional trimming does not lead to any noticeable degradation in visual quality. On the contrary, it yields a positive impact on startup

TABLE II: The Size of Received Video Files (in GB)

Available Bandwidth	Rebuild	Rebuild + Policy	Cut	Cut + Policy
15	9.72	7.21	9.72	7.23
20	9.79	7.37	9.86	7.25
25	9.94	7.43	9.90	7.65
30	10.10	7.55	10.06	7.45
35	10.14	7.92	10.31	7.69
40	27.47	24.96	28.81	28.39
45	29.08	28.60	29.08	28.94



(a) Startup Latency



(b) PSNR

Fig. 9: Client-side performance and QoE – Latency & PSNR

(a) Frame 206 of 300 from the original video

(b) Frame 206 of 300 after Cut at 15 Mbps

Fig. 10: Frame 206 of 300 - Comparison

latency: by reducing the volume of transmitted data, the overall latency is significantly decreased.

In Fig. 10a and 10b we show 2 screenshots from videos, one being the original H264 video at the sender, the other being the received H264 video which has been collected by the client after using the Cut approach in the network node and an available bandwidth setting of 15 Mbps. Both videos were decoded using the JSVM decoder.

In both examples, we see frame 206 from a total of 300 frames. We observe how good the quality of the received image is. There are few visual artifacts given the large amound of trimming, given we passed a 35 Mbps video via a 15 Mbps link. Overall, we observe this is a successful transmission mechanism for layered video as the approach has been effective in managing limited network bandwidth with low delay and with minimal quality degradation.

The results here confirm that *buffer cutting* works well as a packet trimming mechanism, the *chunk reordering* operates as described, and that the addition of *policies for trimming* extra content in bandwidth limited situations also works well.

V. Conclusions

We have successfully adapted our trimming approach from packet trimming using rebuilding to packet trimming using buffer cutting. We presented the approaches and design aspects required, and highlighted: (i) a trimming method that uses buffer cutting rather than rebuilding as way to trim in a faster, less intensive approach at the node; (ii) a mechanism deployed in the sender to reorder the chunks in the packet, by the significance value not by the position; and (iii) an approach for more aggressive trimming in the network node, by using some policies, for when we get close to the bandwidth limit, rather than using the meta-data in the packet.

One of the most significant outcomes of the proposed enhancements is the successful implementation of a fully functional end-to-end system using a buffer cutting approach. A major challenge in this context is ensuring the generation of decodable video files at the client side, as the network nodes typically prioritize minimizing network load rather than preserving video decoding characteristics. We have validated

the correctness of the system by successfully decoding the client-side video files using the H264 video decoder that is part of the JSVM software. We have answered our initial question in a positive way, and demonstrated that the design aspects and the updates are working.

For the buffer cutting, there is less work to do in the network node, as it does not have to do the split, unpack, trim, and rebuild approach utilised in our other work. However, the side-effect is that the client needs to do more work to validate the packet. Another side-effect is that the sender has to do some extra work to reorder the chunks in significance value order, and again, that the client has to do more work as it has to reorder the chunks by position when the packet arrives. This is done to ensure correctness and allow the upper layers of the software to create valid structures.

A simplistic viewpoint on this approach is that we have just moved some computation from the network node to the sender, and even more computation to the clients. However, in the network node all the work is concentrated in one place, whereas at all the sender and clients this work is distributed. Any scheme that can reduce load in the network, will allow more capacity and flows to be handled is beneficial.

In future work we will consider more aspects of the policy approach to trimming chunks when approaching the bandwidth limit. For the policies in this work, we have hard-coded some bandwidth values that trigger the policy, however it is likely these policies and associated bandwith values would be set by a network operator. We will investigate more configurable and flexible policy specification. Our system uses a virtualized edge function, and in future we will investigate if it is possible to use a programmable network card for the implementation of embedded packet trimming.

REFERENCES

[1] R. Li, A. Clemm, U. Chunduri, L. Dong, and K. Makhijani, "A new framework and protocol for future networking applications," in *NEAT 2018 - Proc. of ACM Workshop on Networking for Emerging Applications and Technologies*, August 2018.

[2] S. Clayman and M. Sayıt, "Low Latency Low Loss Media Delivery Utilizing In-Network Packet Wash," *Journal of Network and Systems Management*, vol. 31, no. 1, p. 29, 2023.

[3] M. Tüker, E. Karakış, M. Sayıt, and S. Clayman, "Using Packet Trimming at the Edge for In-Network Video Quality Adaption," *Annals of Telecommunications*, 2023, Springer Nature.

[4] S. Clayman, E. Karakış, M. Tüker, E. Ak, B. Canberk, and M. Sayıt, "Dynamic Packet Content Construction and Processing for End-to-End Streaming in 6G," in *2023 IEEE 28th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2023.

[5] S. Clayman, M. Sayit, D. Griffin, and M. Rio, "Using edge-based packet trimming for effective bandwidth utilization in 6g," in *2024 3rd International Conference on 6G Networking (6GNet)*, 2024, pp. 54–62.

[6] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. Moore, G. Antichi, and M. Wójcik, "Re-Architecting Datacenter Networks and Stacks for Low Latency and High Performance," in *ACM SIGCOMM*, Los Angeles, CA, USA, Aug 2017, pp. 29–42.

[7] X. Chen, S. Vargaftik, and R. B. Basat, "When ml training cuts through congestion: Just-in-time gradient compression via packet trimming," in *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, ser. HotNets '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 177–185. [Online]. Available: https://doi.org/10.1145/3696348.3696880

[8] S. Clayman, M. Sayıt, D. Griffin, and M. Rio, "Enhancing bandwidth utilization for video streaming with packet trimming in 6g," in *2024 20th International Conference on Network and Service Management (CNSM)*, 2024, pp. 1–5.

[9] R. Li *et al.*, "A Framework for Qualitative Communications Using Big Packet Protocol," in *NEAT 2019: Proc. of ACM Workshop on Networking for Emerging Applications and Technologies*, August 2019, pp. 22–28.

[10] A. Bentaleb, M. Lim, M. N. Akcay, A. C. Begen, S. Hammoudi, and R. Zimmermann, "Toward One-Second Latency: Evolution of Live Media Streaming," *IEEE Communications Surveys and Tutorials*, vol. 28, pp. 2418–2456, 2026.

[11] DASH-IF/DVB, "Report on low-latency live service with DASH," DASH Industry Forum and DVB Project, Technical Report, 2017, accessed: January, 2026. [Online]. Available: https://dashif.org/docs/Report%20on%20Low%20Latency%20DASH.pdf

[12] J. Aguilar-Armijo, C. Timmerer, and H. Hellwagner, "SPACE: Segment Prefetching and Caching at the Edge for Adaptive Video Streaming," *IEEE Access*, vol. 11, pp. 21 783–21 798, 2023.

[13] J. Yang, A. K. Bashir, Z. Guo, K. Yu, and M. Guizani, "Intelligent cache and buffer optimization for mobile vr adaptive transmission in 5g edge computing networks," *Digital Communications and Networks*, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864823001190

[14] H.-W. Kao and E. H.-K. Wu, "QoE Sustainability on 5G and Beyond 5G Networks," *IEEE Wireless Communications*, vol. 30, no. 1, pp. 118–125, 2023.

[15] K. Horita, N. Fukumoto, and A. Nakao, "Improving QoS of 5G Video Streaming Through Network Exposure Function," in *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, 2023, pp. 375–380.

[16] D. Brunello, I. Johansson S, M. Ozger, and C. Cavdar, "Low latency low loss scalable throughput in 5g networks," in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, 2021, pp. 1–7.

[17] L. Dong and R. Li, "Distributed Mechanism for Computation Offloading Task Routing in Mobile Edge Cloud Network," in *International Conference on Computing, Networking and Communications (ICNC)*, Honolulu, HI, USA, 2019, pp. 630–636.

[18] M. Ghafari, T. Cholez, and O. Festor, "Evaluation of Packet Wash for Low-Latency High-Bitrate Game Streaming," in *Proceedings of the 3rd Workshop on Emerging Multimedia Systems*, ser. EMS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 7–12. [Online]. Available: https://doi.org/10.1145/3746441.3748228

[19] Xiph, "Media Xiph," 2026. [Online]. Available: https://media.xiph.org/

[20] Fraunhofer, "JSVM – H.264/AVC scalable video coding (SVC) extension JSVM reference software," 2011. [Online]. Available: http://vcgit.hhi.fraunhofer.de/jvet/jsvm