Optimising DASH over AQM-enabled Gateways Using Intra-chunk Parallel Retrieval (Chunklets)

Jonathan Kua, Grenville Armitage
Internet For Things (I4T) Research Laboratory
Swinburne University of Technology
Melbourne, Australia
{jtkua, garmitage}@swin.edu.au

***Abstract*—Multimedia streaming is a significant source of Internet traffic, with Netflix and YouTube accounting for more than 50% of North American fixed network peak download traffic in 2016. Dynamic Adaptive Streaming over HTTP (DASH) is a recent standard for live and on-demand video streaming services, where clients adapt the video quality on-the-fly to match the network capacity by requesting multi-rate video chunk-by-chunk. Emerging Active Queue Management (AQM) schemes such as PIE and FQ-CoDel are being progressively deployed either at the ISP-end and/or home gateway to counter *bufferbloat* and will impact consumer DASH streams. We propose using intra-chunk parallel connections (*chunklet*s) to retrieve DASH content when bottlenecks implement AQMs. We experimentally evaluate and characterise the impact of using chunklets over traditional FIFO, symmetric/asymmetric PIE and FQ-CoDel AQM bottlenecks. We show FQ-CoDel's flow isolation and fair capacity sharing ability enables DASH chunklets to attain the best throughput multiplication effect, hence translating to better user experience in the presence of competing elastic flows.**

***Index Terms*—DASH, chunklets, TCP, AQM, PIE, FQ-CoDel**

I. Introduction

With the proliferation of various Internet-enabled devices in the home, diverse application flows have to compete for the last-mile ISP bottleneck link or home gateway that connects home users to the Internet. Many of these applications rely on the Transmission Control Protocol (TCP) to ensure reliable transport of data and effective use of available bandwidth. However, TCP is known to fill bottleneck buffers until packet losses occur, causing cyclical queue filling and draining and inflation of end-to-end Round Trip Time (RTT) delays. Oversized bottleneck buffers lead to the well-known *bufferbloat* [1] phenomenon.

In recent years, various new Active Queue Management (AQM) schemes have been developed by the Internet Engineering Task Force (IETF) to combine the burst-tolerating attributes of large buffers with low long-term queuing delays. These schemes include Proportional Integral controller Enhanced (PIE) [2], Controlled Delay (CoDel) [3] and FlowQueue-CoDel (FQ-CoDel) [4]. A variant of PIE has been integrated into DOCSIS 3.1 [5] and FQ-CoDel has been highly recommended for embedded Linux gateways.

Video streams represent a significant portion of inbound traffic to the home environment, with Netflix and YouTube accounting for more than 50% of North American fixed network peak download traffic in 2016 [6]. Dynamic Adaptive Streaming over HTTP (DASH) is a recent standard for live and on-demand video streaming services, where clients adapt the video quality on-the-fly to match the network capacity by requesting multi-rate video on a chunk-by-chunk basis by using a single TCP connection [7]. Since DASH traffic pattern differs from long-lived traffic flows, it has complex implications when interacting with bottlenecks that implements AQMs schemes.

For example, FQ-CoDel isolates individual traffic flows into sub-queues then serves each sub-queue with a Deficit Round Robin (DRR) scheduler. The result is relatively even capacity sharing, which may actually be detrimental to a DASH flow (often a single, persistent TCP connection) that is competing with multiple other concurrent TCP flows.

We consider the new case where a DASH client might use multiple concurrent TCP connections to retrieve different parts of a video chunk (which we refer to as *chunklets*) in parallel over a network path dominated by an AQM bottleneck. We show that DASH chunklets can leverage the flow isolation and capacity sharing of an FQ-CoDel bottleneck to achieve a larger share of the bottleneck bandwidth when competing with long-lived, elastic bulk transfer flows. This translates to improved Quality of Experience (QoE) for the end-user. A chunklet strategy sits underneath, essentially invisible to, the chunk-level adaptive bitrate strategies available to (and implemented by) existing DASH clients.

In this paper, we make the following key contributions:

- Experimentally evaluate and characterise the impact of using intra-chunk parallel retrieval on a DASH client's performance in AQM environments.
- Show that FQ-CoDel can be leveraged by DASH chunklets to achieve a higher bandwidth share in the presence of cross traffic.
- Experimentally evaluate scenarios where ISP-end of the last-mile has upgraded to using AQMs while the home user continue to use traditional FIFO, and vice versa.

The rest of this paper is structured as follows. Section II provides background on DASH, AQMs and a summary of related work. Section III describes our use case model, experimental testbed setup, which includes our video client/server, dataset and *chunkleting* proxy. In Section IV, we present an analysis and discussion of our results. We outline potential future work in Section V and offer concluding remarks in Section VI.

II. Background and Related Work

In this section we present an overview of DASH architecture, summarise PIE, CoDel, and FQ-CoDel AQMs and review related work.

A. Dynamic Adaptive Streaming over HTTP (DASH)

Modern Internet video streaming platforms have adopted DASH-based content delivery techniques. In DASH streaming systems [7], video content is encoded into multiple versions at different discrete bitrates (quality). Each encoded video is then segmented into small video segments or chunks, each containing a few seconds of video. Chunks from one bitrate are aligned in the video time line to chunks from other bitrates so that the client can smoothly switch bitrates, if necessary, at the chunk boundary. The server provides a corresponding Media Presentation Description (MPD) file which describes the information of the available content and associated encoding bitrates or *Representation Rates* (RR). Video content and MPDs are served by standard HTTP servers.

DASH is layered on top of HTTP/TCP, hence it does not control the content transmission rate directly. Instead it relies on the underlying TCP algorithm to regulate the content transmission rate, which is determined by congestion feedback from the client-server network path. To begin a streaming session, the client requests an MPD file from the content server and then starts requesting video chunks as fast as possible to fill the playout buffer (typically back-to-back requests in sequence using one persistent TCP connection). Once the playout buffer is full, the player enters a steady state phase of periodically downloading new chunks as previous chunks are consumed and rendered as audio/video content.

In steady state, the ON period represents DASH client downloading a chunk, and the OFF state represents otherwise. The time between the start of two consecutive ON periods is typically the chunk size – amount of video content within each chunk – in seconds (*aka* cycle time). The client typically keeps a few chunks in the buffer to maintain adequate playback.

TCP estimates the number of bytes that can be 'in flight' and unacknowledged at any given time with its congestion window (`cwnd`). `cwnd` starts low[1], grows as packets are received and acknowledged by the client, and shrinks when packets are lost or the if connection has gone idle for too long [8]. A DASH client's chunk retrieval process means TCP sends repeated bursts of packets followed by some periods of inactivity. If `cwnd` is too low, or fails to grow quickly enough, the DASH client experiences low per-chunk *Achieved Rate* (AR)[2]. If `cwnd` grows beyond the path's bandwidth-delay product (BDP) it starts filling bottleneck queues and inflicting additional queuing delays on all flows sharing the bottleneck.

DASH clients use adaptive bitrate (ABR) algorithm to adapt to fluctuating network conditions. ABRs use various feedback signals observed for each chunk (such as recent AR estimates and/or playout buffer occupancy) to select a suitable RR for the next chunk to be downloaded. Consider an example when DASH client uses AR as its ABR feedback signal. If AR is high, ABR should select a higher RR. On the other hand, if AR decreases, ABR should dynamically switch to a lower RR level to avoid playout buffer under-run. A good ABR algorithm will strike a delicate balance between reacting to network conditions and adapting video RRs smoothly [9].

[1]The default was 2 packets, but some recent OSes now start TCP connections with a `cwnd` of 10 packets long.

[2]Video chunk size divided by the time taken to receive it (essentially an estimate of per-chunk TCP throughput).

B. PIE, CoDel and FQ-CoDel

Unlike earlier AQM schemes, PIE, CoDel and FQ-CoDel are modern approaches that aim to keep long-term queuing delays low instead of merely controlling the queue occupancy.

1) Single queue PIE and CoDel: PIE and CoDel operate on single queues and keep queuing delays low by dropping packets when queuing delays persistently exceed a target delay, T_{target} (PIE: officially 15ms [2], although the Linux implementation currently defaults to 20ms [10]; CoDel: 5ms).

PIE introduces a burst tolerance parameter which (by default) allows packets arriving within the first 150ms of an empty queue to pass successfully.[3] After this, when a packet arrives, it is randomly dropped with a certain probability. This probability is periodically updated, based on how much the current queuing delay (estimated from the queue length and the dequeue rate) differs from $T_{target} = 15ms$ and whether the queuing delay is currently going up or down. Packets of ECN-enabled flows will be marked instead of being dropped when the dropping probability is $<10\%$.

CoDel [3] tracks the (local) minimum queuing delay experienced by packets in a certain interval (initially 100ms). When the minimum queuing delay is less than $T_{target} = 5ms$ or the buffer size is less than one full-size packet, packets are neither dropped nor ECN marked. When the minimum queuing delay exceeds T_{target}, CoDel enters the drop state where a packet is dropped and the next drop time is set. The next drop time decreases in inverse proportion to the square root of the number of drops since the dropping state was entered. When the minimum queuing delay is below T_{target} again, CoDel exits the drop state.

2) Multi-queue FQ-CoDel: FQ-CoDel [4] classifies flows into one of 1024 (by default) different queues by hashing the 5-tuple of IP protocol number and source and destination IP and port numbers. Each queue is separately managed by the CoDel algorithm. A modified Deficit Round Robin (DRR) scheduler services these queues, in which each queue can dequeue up to a quantum of bytes (one MTU by default) per iteration. This scheme gives priority to queues with packets from new flows or from "sparse" flows with packet arrival rate small enough so that a new queue is assigned to them upon packet arrival. A bottleneck managed by FQ-CoDel can achieve low latency (due to per-queue CoDel), relatively even capacity sharing (due to the fixed hashing function) and priority for low-rate or transactional traffic (such as DNS and VoIP traffic).

[3]Earlier versions of PIE stipulated 100ms burst tolerance [11], later increased to 150ms in the final published RFC [2].

Due to the FQ and DRR scheduling behaviours, we expect to see DASH achieving a higher aggregate throughput when using multiple concurrent connections for chunk retrieval in the presence of other competing flows.

C. Related Work and Motivation

Evaluations of DASH-based streaming over modern AQM schemes are only beginning to emerge. A recent experimental study of DASH traffic over AQMs showed a single DASH stream with no cross traffic benefits from PIE's higher burst tolerance and queuing delay targets [12]. However, in the presence of cross-traffic, the FlowQueue DRR scheduler provides better flow isolation and capacity sharing, hence protecting DASH flows from collateral damage in both upstream and downstream cases.

Huang et al. [13] use traffic traces from major streaming companies to understand the interactions between TCP and DASH-like traffic. They identified a vicious cycle – the *"downward spiral"*. During multi-second OFF periods, TCP resets `cwnd` to its initial value due to idle periods longer than the current minimum TCP retransmission timeout (around 200ms in their experiments). Consequently `cwnd` ramps up in slow start mode to retrieve subsequent chunks. The client still selects the highest sustainable RR when there are no cross traffic. However, when cross traffic starts to fill the bottleneck buffer during OFF periods, packet losses start to occur, resulting in low video throughput for subsequent chunks. As throughput decreases further, the client requests chunks of lower RR and smaller size. As chunk size decreases, TCP has less time to reach its fair share before it finishes downloading each chunk, leading to further underestimation of available capacity, causing the client to be trapped in the downward spiral.

Our motivation evolves from the use of parallel connections to accelerate file transfers (present in applications such as GridFTP [14], download accelerators and web browsers). Most related work in the context of video streaming uses multiple connections to download different chunks (inter-chunk/full chunk concurrency) throughout a streaming session [15], [16], [17], [18], [19]. The public DASH client, dash.js prior to version 1.4 [20], also included a prototype for parallel downloading of video chunks.

A chunklets approach differs in that we divide individual range-based chunk requests into a set of intra-chunk range requests *sent in parallel* over long-lived connections to the content server. The underlying idea has previously appeared in [21] (without giving it a particular name), and been discussed among dash.js developers and several companies but not yet implemented in dash.js. The authors of [21] evaluated the potential for this approach to improve DASH performance over wireless environments. They measured the performance with forty 15-sec chunks, varied the number of connection(s) from one to seven and tested them against three different packet loss rates, RTT values, bandwidths and four different chunk sizes. They also did an analysis on the 100-packet FIFO queue occupancy with simulations.

To the best of our knowledge, we are the first to experimentally evaluate and characterise the impact of intra-chunk parallel retrieval (chunklets) in AQM environments by using real equipment with a well-known and well-supported DASH client. Our experiments also collect more chunk samples than in previous work and consider greedy/elastic bulk transfers as competing traffic flows.

III. Evaluation Methodology

Here we describe our use case model, experiment testbed setup and performance measures.

A. Consumer home network environment

Figure 1 illustrates our use case model – a home network that is connected to local or international video streaming and other Internet-based services over a last-mile broadband link. This link typically offers asymmetric bandwidth. The *downstream* bottleneck (for traffic flowing into the home) is at the ISP-end while the *upstream* bottleneck (traffic flowing away from home) is at the home gateway.

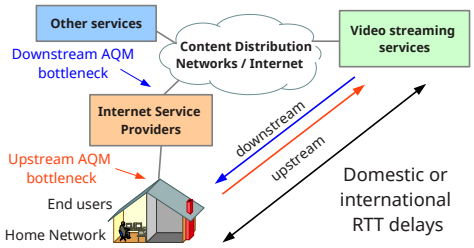


Fig. 1. Use case model: A home network connected to the Internet with AQM enabled at the ISP-end or/and home gateway

AQMs are likely to replace traditional FIFO in modern and affordable home gateways, with ISPs progressively upgrading their equipment to use AQMs. We establish baseline results by considering DASH and elastic TCP traffic flows being bottlenecked by PIE and FQ-CoDel AQMs on both ends. Then we explore scenarios where AQMs are either deployed at the ISP-end or at the home gateway, representing likely scenarios where ISP has upgraded to using AQM while the consumer still use traditional FIFO modem, and vice versa.

1) Sharing the last-mile bottleneck: Due to its chunked video retrieval, DASH generates a periodic ON-OFF traffic pattern in the steady state. Interesting challenges arise when DASH's OFF period overlaps with other downstream traffic. As noted in Section II-C, competition with other bulk (elastic) data transfers can lead the DASH client into a "downward spiral" of progressively lower rate estimations.

Upstream bulk transfers compete with a DASH connection's TCP acknowledgement (ACK) packets. When the upstream link is congested, the DASH connection's ACKs suffer additional queuing delays (and potentially loss), indirectly limiting the DASH client's estimate of available network capacity.

Examples of competing traffic in the home include file downloads/uploads, VoIP calls, video conference calls, online gaming, mobile devices downloading updates or synchronising/back-up data to the 'cloud', PCs downloading

OS updates and so on. All of these activities have impact on latency-sensitive applications such as DASH video streams.

2) Representative network conditions: Video streaming providers typically try to install content servers or caches close to their customers, to minimise delays caused by a path's base RTT and reduce the cost of transiting intermediate network providers. So we will emulate relatively low RTTs.

Home broadband services vary around the world. We choose to emulate a last-mile offering 12Mbps downstream / 1Mbps upstream (12/1 Mbps), such as would be achieved by a medium-performance ADSL2+ connection. The 12Mbps downstream significantly exceeds the highest representation rate of our video dataset (Table I), ensuring our DASH client is able to retrieve at the highest RR during baseline experiments. We also explore a last-mile offering 25/5 Mbps service, covering the case where a single High Definition (HD) RR is being retrieved throughout the streaming session and ensure that chunklets are not afflicted by the "downward spiral".

B. Experimental Setup

Figure 2 shows our TEACUP-based [22] testbed. The router runs 64-bit FreeBSD 10.1-RELEASE to provide a configurable bottleneck (bandwidth, delays, queuing disciplines, buffer sizes) between client(s) and server(s) on either side of the router. DASH client, chunkleting proxy and lighttpd (http://www.lighttpd.net/) server run on 64-bit FreeBSD 10.1-RELEASE whereas iperf (https://iperf.fr/) client and server run on 64-bit Linux openSUSE 12.3 (kernel 3.17.4).

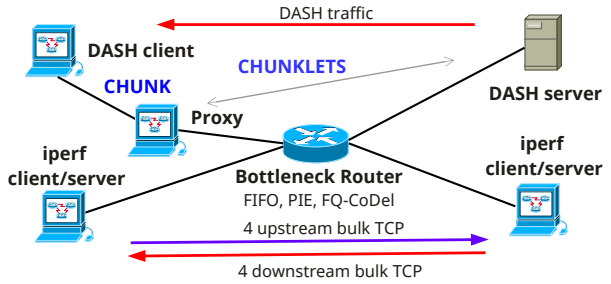


Fig. 2. Experimental setup: DASH client requests and receives *chunk* via proxy, proxy initiates N requests and reassembles N *chunklets* responses before returning *chunk* to client. *Chunklet* flows compete with elastic TCP flows over the same FIFO/AQM bottleneck.

1) Emulating network conditions: We use FreeBSD's dummynet/ipfw [23] to provide FIFO, PIE and FQ-CoDel queue management schemes and emulate specific last-mile rate limits and base RTTs. Packets sit in a configurable bottleneck queue while being rate-shaped to the bottleneck bandwidth, and then sit in a 1000-packet buffer while being delayed. The bottleneck queue is 100 packets for FIFO experiments (to exceed each path's bandwidth delay product) and 1000 packets for PIE and FQ-CoDel experiments. We configure the upstream and downstream queues separately, hence we can apply different queuing disciplines and buffer sizes in each direction.

2) Creating DASH and competing flows: We use dash.js version 2.4.1 (https://github.com/Dash-Industry-Forum/dash.js/wiki) as our DASH client, and our DASH server is a lighttpd

TABLE I
REPRESENTATION RATES AVAILABLE IN 2-SEC DATASET

Resolution	Encoding Level	Representation Rates
320x240	1 - 3	46, 89, 131kbps
480x360	4 - 8	178, 222, 263, 334, 396kbps
854x480	9 - 10	533, 595kbps
1280x720	11 - 14	0.8, 1.0, 1.2, 1.5Mbps
1920x1080	15 - 20	2.1, 2.5, 3.1, 3.5, 3.8, 4.2Mbps

version 1.4.35 web server (with persistent HTTP connections and range-requests support enabled). Our dash.js client uses a throughput-based ABR strategy, where the next RR is based on the smoothed average ARs of the previous three chunks.

We use FreeBSD and TCP NewReno on our DASH server and DASH client.[4] We use Linux and TCP CUBIC to generate long-lived competing flows with iperf, as many online services run on Linux and CUBIC is known for its effective bandwidth utilisation (which provides a good stress test for DASH streams). All hosts disable Explicit Congestion Notification (ECN) and enable both TCP window scaling and receive buffer auto-tuning.

We use the ITEC-DASH BigBuckBunny dataset[5] [24] and select content encoded in 2-sec chunks of video (*cf.* Netflix using 2-sec chunks). Chunks are available at 20 different encoding RRs ranging from approximately 46kbps to 4.2Mbps as shown in Table I. We ran experiments for 400 seconds to ensure sufficient 2-sec samples were collected.

3) Chunkleting proxy: Rather than modifying dash.js to 'do' chunklets, we adapted an open-source HTTP proxy [25] to transparently turn chunk requests into chunklet requests, then stitch the replies back together into chunks. This allows us to run experiments using a stock-standard dash.js client.

Specifically, our proxy intercepts each HTTP GET request from the dash.js client and parses any range-requests to identify the chunk being requested. The proxy calculates N consecutive sub-ranges of bytes to represent the N chunklets, and sends N simultaneous HTTP GET requests (one for each chunklet's sub-range) over N parallel persistent connections to the DASH server.

As responses arrive from the DASH server, the proxy reassembles and concatenates N chunklet responses, recreating the full chunk requested by the dash.js client. The proxy's actions are thus transparent to both the client and the server. Hence, chunkleting with or without proxy does not make any material difference to the underlying network characteristics.

For implementation simplicity, a chunk of Y bytes results in $(N-1)$ chunklets of $int(Y/N)$ bytes and a final chunklet carrying between $[int(Y/N)]$ and $[int(Y/N)+(N-1)]$ bytes.

C. Performance indicators

Traffic was captured using tcpdump on all host and router interfaces. This data was then processed by Synthetic Packet

[4]Inspired by FreeBSD in Netflix's OpenConnect platform (https://openconnect.netflix.com/software) and NewReno being similar to the default TCP in Apple's OSX and older Microsoft servers.

[5]Full-length video sequences encoded at different bitrates, resolution and quality can be downloaded at: http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/BigBuckBunny

Pairs (SPP) [26] to construct per-packet network-layer RTT measurements, RRs were extracted by parsing the client's HTTP GET requests, and per-chunk ARs were calculated using payload lengths extracted from HTTP response headers and the time taken to transfer packets making up each chunk. We calculated *Smoothed AR* (SA) – the average of the AR from the last three chunks – to mimic the signal used by dash.js to select the next chunk's RR.

Studies have shown that users are sensitive to frequent and significant RR switches [27], [28], so we use the extracted RRs to derive the number of RR transitions and calculate the following *instability index* as defined in [29]:

$$\frac{\sum_{d=0}^{k-1} |b_{x,t-d} - b_{x,t-d-1}|.w(d)}{\sum_{d=1}^{k} b_{x,t-d}.w(d)}$$

Our instability index is obtained by calculating the weighted sum of the number of RR level (Table I) transitions within the last k = 10 video chunks (where $b_{x,t}$ is the encoding level retrieved at time t), which is then normalised to a value between 0 and 1 by dividing it by the weighted sum of all RR levels observed in the last 10 chunks (corresponding to 20 seconds of video). The weight function $[w(d) = k - d]$ is applied to add a linear penalty to the more recent RR switches. The window is then slided in steps of one, and the instability index is re-calculated for each window. Although an instability index close to zero indicates a more stable system, the instability index cannot be used alone to determine a DASH flow's likely QoE. It must be considered in the context of corresponding RR distributions, with users potentially valuing one over the other. For instance, a DASH flow experiencing high instability with higher median RR might provide better experience than a DASH flow with low instability at lower median RR. A DASH flow that self-limits itself to a lower range of RRs might present with low instability index but it is in fact a "consistently bad" experience for the user.

In order to ensure their relevance to long-term viewing, we calculate SA and RR statistics after the playout buffer is pre-filled (*i.e.* during DASH ON-OFF steady state).

IV. Results and Analysis

In this section we present our experimental results and analysis. We first illustrate how chunklets can improve DASH performance, then we consider scenarios where DASH can benefit from chunklets when competing with either downstream or upstream elastic flows. Lastly, we consider the impact of chunklets under deployment scenarios where the bottleneck is AQM-aysmmetric.

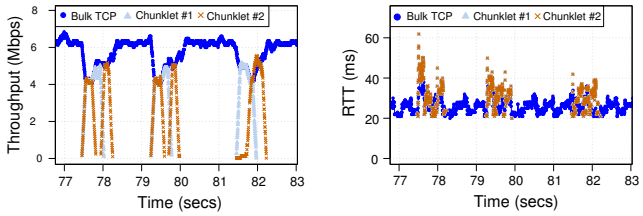
A. The Achieved Rate Multiplication Effect

Modern AQMs aim to keep long term queuing delays low by providing negative feedback with packet losses to reduce the transmission rate of the sender. This also indirectly help flows sharing the bottleneck to attain a relatively fair share of the bandwidth. In particular, when FQ-CoDel is used, the bottleneck capacity is shared evenly among all flows.

With regular DASH, a chunk is retrieved and transferred every cycle time sequentially. When competing with x number of flows, its effective per-chunk achieved rate (AR_1) is $1/x$ of the bottleneck bandwidth (C) when the bottleneck is managed by FQ-CoDel, hence the DASH client can only retrieve RR that is sustainable by AR_1. However, when using N concurrent chunklets, each chunklet flow is perceived as a unique flow to FQ-CoDel, hence DASH's AR sees a multiplication effect with an increased rate of $AR_N = (N/x) * C$, hence allowing the client to retrieve RR sustainable up to AR_N.

As an illustrative example, Figure 3 shows the throughput, induced RTT and `cwnd` evolution of two chunklets overlapping in time and their competing flows[6]. In this case, the DASH client is able to attain an average of $2/6th$ of the 25Mbps bottleneck capacity (~8.3Mbps) whereas each bulk TCP flow can only achieve $1/6th$ of the capacity (~4.2Mbps) when DASH is in the ON period (Figure 3a). Chunklets may not be fully overlapping in time because ACK spacing and `cwnd` growth can slightly differ in time for each chunklet flow. An AR multiplier effect will only be observed during periods where chunklets are overlapping in time.

Nevertheless, it is important to note that chunklets cannot be too small in size so as to allow the number of HTTP headers to dominate the transmission at the chunk level[7]. A good chunkleting algorithm will calculate and determine a safety threshold for *N* and adapt *N* if necessary so that a certain "chunk efficiency" can be maintained. Our current work do not consider adaptive chunkleting, however we have verified that the overhead percentages of our *N* values are minimal and do not adversely affect DASH client's chunk retrieval process.



(a) Throughput vs time (b) RTT vs time

(c) `cwnd` vs time

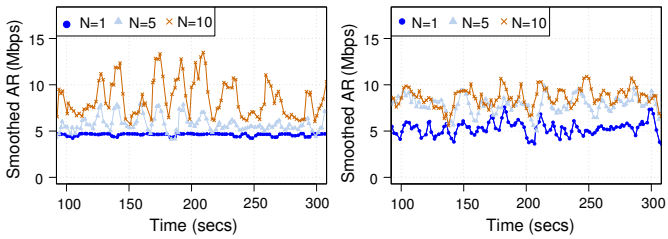
Fig. 3. Two DASH chunklets competing with four elastic TCP flows over {25/5Mbps, 20ms RTT, FQ-CoDel} bottleneck

Figure 4 compares Smoothed AR (SA) of FQ-CoDel and PIE when *N* = 1 (regular chunk, baseline), 5, 10 chunklets

[6]We only included one bulk TCP flow in these graphs for clarity, all the other flows track these curves closely.

[7]*N* HTTP headers are required to transmit *N* chunklets. Hence, simplistically, the HTTP overhead ratio is $(N * H)/[(N * H) + V]$, where *H* is the HTTP header size and *V* is the video chunk size. The overhead ratio increases as *N* increases and chunklets will start to become inefficient.

in a 200-second time window. SA across FQ-CoDel is more stable and higher than PIE when N is high, with a peak of ~14Mbps when N = 10, as compared to ~10Mbps with PIE. This is because FQ-CoDel enables chunklet flows to achieve a fairer share of the bandwidth and overlapping more in time (in the long term).



(a) FQ-CoDel (b) PIE

Fig. 4. Smoothed AR multiplier in steady state for N = 1, 5, 10 chunk(lets)

We demonstrate the effect of AR multiplication as N increases by running experiments where the bottleneck capacity (25/5 Mbps, 20ms RTT) is well in excess for a *single* chunk to achieve at least the AR above the supplied 3.5Mbps RR (albeit without chunkleting) when competing with four downstream bulk TCP flows. Figure 5 shows the SA distribution as N increases, when using FIFO, PIE and FQ-CoDel bottleneck. FIFO shows lower SA but still manage to retrieve the 3.5Mbps RR content; both PIE and FQ-CoDel allows DASH to experience an increase in SA when N increases. PIE shows a quicker rise in SA than FQ-CoDel because it has a higher target delay, allowing the queue to grow longer and TCP `cwnd` to grow further, hence sending and allowing more packets in flight. FQ-CoDel shows a wider distribution as N gets higher, due to the variation in chunklet overlapping percentages causing different AR multipliers as FQ scheduler services the queues.

B. Chunklets over FIFO and symmetric AQM bottlenecks

Here we consider the impact of downstream or upstream elastic TCP flows on DASH chunklets, and how AQMs can help mitigate the adverse effects presence in traditional FIFO.

1) Combating downstream elastic flows: In scenarios where DASH is competing with downstream elastic TCP flows, we consider cases where both DASH and TCP flows have the same base RTT (20ms – indicating both services originates from local providers; 60ms – indicating services originating from international providers). We also consider the case where video streams are retrieved from local content servers (20ms RTT) and compete with TCP flows originating from a farther location (emulating users downloading files from overseas).

Figure 6 shows SA increases for all FIFO, PIE and FQ-CoDel as N increases. Instability indices are generally low (< 0.02 in all cases). However, larger SA variations are present in FIFO, leading to higher variations in retrieved RR, hence a higher median instability index, indicating frequent and significant RR switches. When $N < 6$, single queue PIE has wider spread in both SA and RR than FQ-CoDel. Although it has higher RR values, its instability index is also higher than FQ-CoDel. FQ-CoDel has a wider spread when $N > 6$ possibly due to the increased probability of high overlapping percentage as N increases. Both PIE and FQ-CoDel see higher instability index at $N > 6$.

When both DASH flow and bulk TCP flows have the base RTT of 60ms (Figure 7), we see a similar behaviour but with a lower average SA due to the fact that AQMs emulates a small effective buffer (less than the paths' BDP on average), causing DASH client to retrieve lower RRs. FQ-CoDel shows lower instability index when $N < 7$.

On the other hand, when DASH flow originates from a closer location (20ms RTT) than elastic TCP flows (60ms RTT), its average SA increases. Comparing with Figure 6 when both DASH and elastic flows are competing from 20ms RTT away, this scenario sees PIE allowing DASH to achieve higher SA and relatively high RRs as N increases. Bulk transfer flows are less aggressive and begin to cede capacity as path RTT increases due to the small buffer size emulated by AQMs. There tend to be no significant SA increase as N increases because all chunklets are served in a single queue, and there is no FQ-induced AR multiplication effect as observed in FQ-CoDel.

2) Protecting chunklets' ACKs in congested uplinks: Since DASH's transmission is regulated by its ACKs, hence protecting upstream ACK streams in scenarios where the uplink is congested will indirectly allow DASH to achieve better ARs. We evaluate this scenario by generating four upstream TCP flows to compete with DASH chunklet flows. Figure 9 shows FIFO performs the worst with ~0.1Mbps SA and selecting very low RRs despite having zero instability index (as discussed in Section III). This is because DASH's ACK streams are incapacitated by the aggressive TCP flows, hence unable to regulate downstream DASH flows. By applying PIE and FQ-CoDel in the upstream direction allows DASH to achieve much higher SA, because both schemes controls queuing delays by inducing packet losses on the elastic flows, signaling their sources to backoff and reduce their sending rate. This then allows low-rate ACK streams to traverse the upstream bottleneck and regulate DASH adequately. Our results show that chunkleting is not ideal when competing with upstream flows. In both AQM cases, SA trends downwards as N increases and stablise around N = 6. In the presence of chunklets, FQ-CoDel results show better RRs values and less spreading (low instability index) than PIE. The flow protection and capacity sharing ability of FQ-CoDel in the upstream assists in ensuring chunklets' ACKs streams are protected so that DASH transmission can occur in a timely fashion.

C. Chunklets over asymmetric AQM bottlenecks

As AQMs gradually replaces FIFO queues in ISP/consumer grade equipment, there may well exists scenarios with "asymmetric bottlenecks" – where one end of the bottleneck implements AQM while the other still uses FIFO. This will result in interesting implications on DASH flows when competing with elastic traffic. We evaluate these potential scenarios by applying AQMs or 100-packet FIFO at either the upstream or downstream bottleneck, along a 12/1Mbps, 20ms RTT network

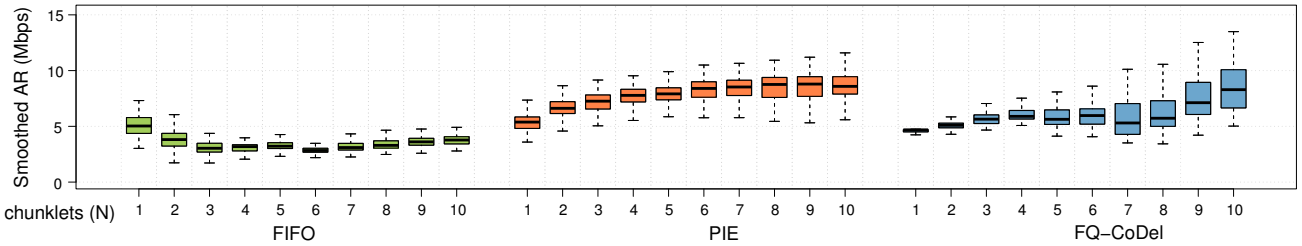


Fig. 5. Smoothed AR *vs N* for FIFO, PIE and FQ-CoDel @ {25/5Mbps, 20ms RTT} path, fixed 3.5Mbps RR, 4 downstream elastic flows

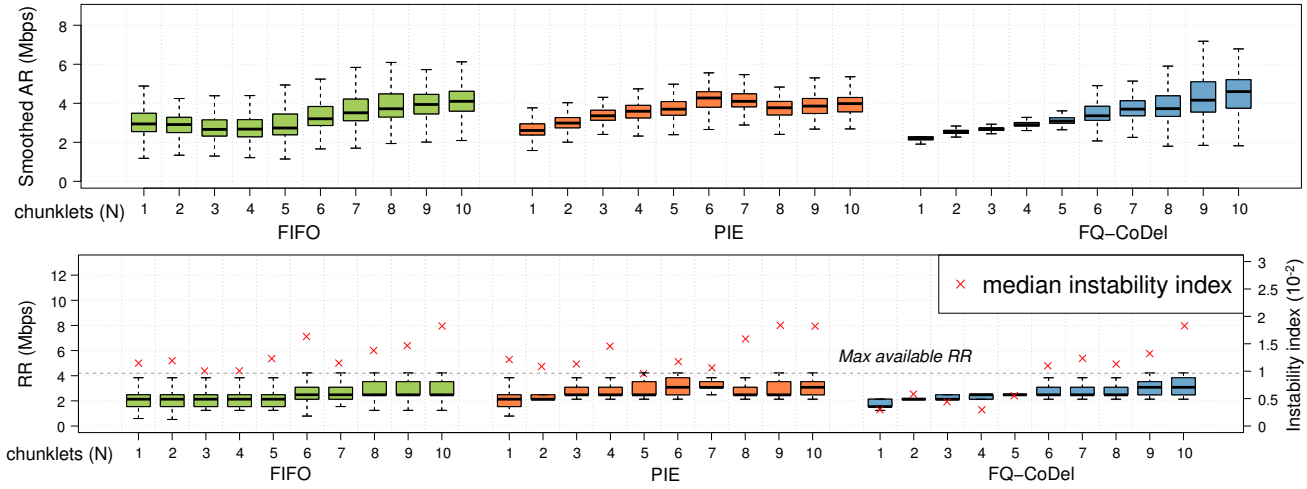


Fig. 6. Smoothed AR, RR and instability index *vs N* for FIFO, PIE and FQ-CoDel @ {12/1Mbps, 20ms RTT} path, 4 downstream elastic flows

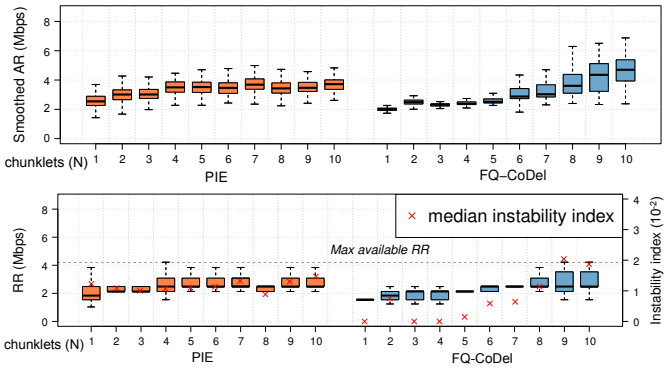


Fig. 7. Smoothed AR, RR and instability index *vs N* for PIE and FQ-CoDel @ {12/1Mbps, 60ms RTT} path, 4 downstream elastic flows

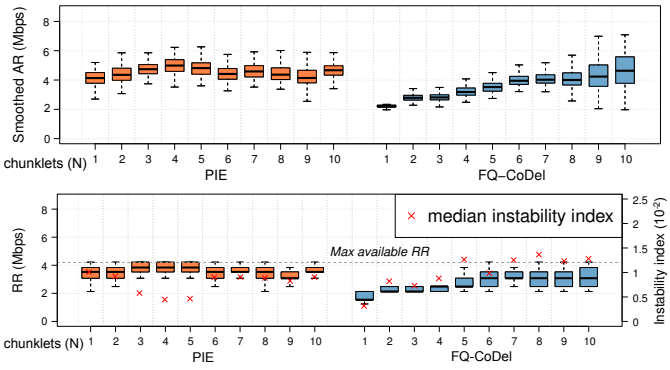


Fig. 8. Smoothed AR, RR and instability index *vs N* for PIE and FQ-CoDel – 12/1Mbps, DASH @ 20ms RTT, 4 downstream elastic flows @ 60ms RTT

path. Competing elastic flows are generated in the direction where AQMs are applied.

1) Downstream AQM: Figure 10 shows that both PIE and FQ-CoDel see the usual improvements when DASH chunklet flows increases. FQ-CoDel performs better than PIE as it manage to achieve higher SA when as *N* increases and it has lower RR instability index than PIE. Since FIFO only manages low-rate ACK streams in the upstream direction of all flows at a decent rate (1Mbps), it is not detrimental to the DASH chunklet flows.

Our experiment results validate the value of having AQMs in at downstream bottleneck (most likely ISP last-mile) to improve inter-flow fairness in the downstream direction as majority of home users are pulling/downloading content into the home; and most users are more inclined to continue using their current home gateways.

2) Upstream AQM: DASH (on top of TCP) performance heavily relies on the timely arrival of ACK packets. We explore the impact of chunkleting over AQMs when DASH ACKs are competing with upstream elastic flows in the same direction. Figure 11 shows a similar SA distribution curve to Figure 9. Although chunkleting seems to reduce the SA, FQ-CoDel provides significant advantages when it is applied to isolate low-rate DASH ACK streams from the competing TCP flows' aggressive data streams and share the limited upstream capacity equally. By protecting DASH's ACK streams from the

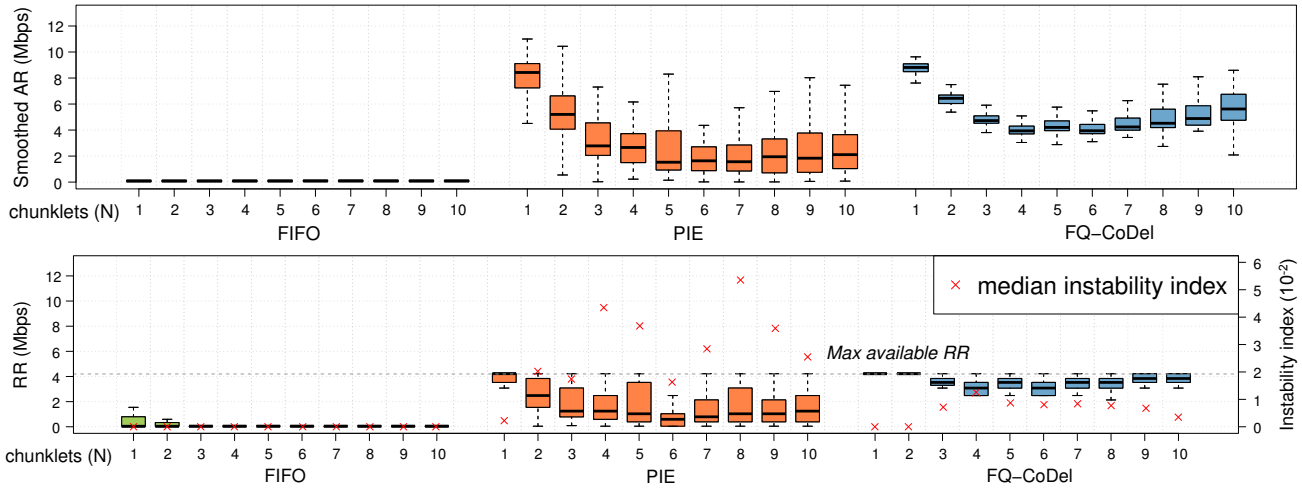


Fig. 9. Smoothed AR, RR and instability index *vs N* for FIFO, PIE and FQ-CoDel @ {12/1Mbps, 20ms RTT} path, 4 upstream elastic flows

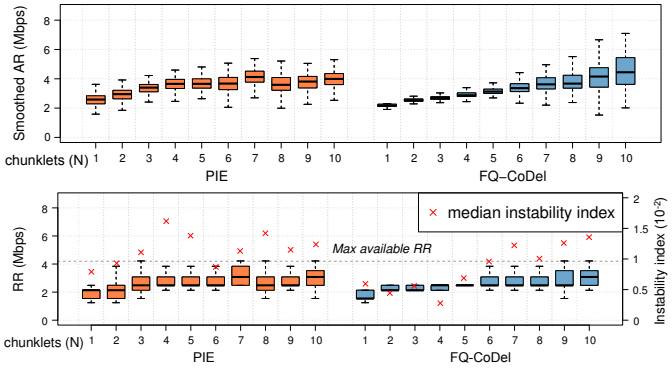


Fig. 10. Smoothed AR, RR and instability index *vs N* for PIE, FQ-CoDel @ {12/1Mbps, 20ms RTT} path, downstream AQM, 4 downstream elastic flows

harm of TCP flows, FQ-CoDel enables DASH streams to be regulated favourably, hence achieving maximum RRs and zero instability index, indicating an ideal QoE where the maximum video quality is retrieved throughout a video streaming session.

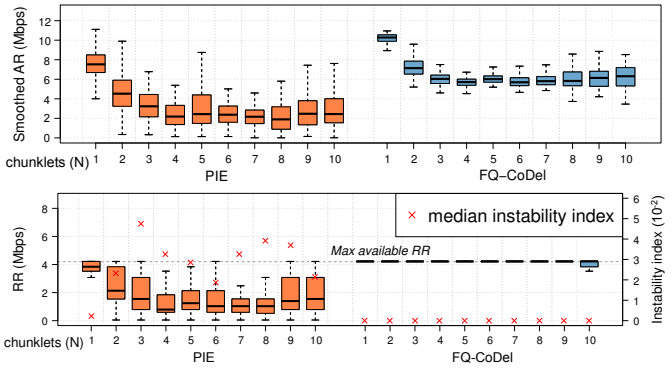


Fig. 11. Smoothed AR, RR and instability index *vs N* for PIE and FQ-CoDel @ {12/1Mbps, 20ms RTT} path, upstream AQM, 4 upstream elastic flows

V. Open Issues and Future Work

A number of open issues and future work items emerge from our experimental evaluation of chunklets for improved retrieval of DASH content over AQM-enabled gateways.

As noted in Section IV-A, the AR multiplier effect depends on a number of factors. As we increase N (the number of chunklets per chunk) we have more parallel TCP connections (potentially increasing the overall share of a bottleneck's capacity) but smaller chunklets (meaning more overhead, such as HTTP headers, per byte of content transferred). And the share of bottleneck bandwidth depends on the degree of overlap between chunklets of a given chunk – we see lower AR multiplier effect if chunklets only partially overlap in time. Exploring the optimal trade-off between N, overheads and chunklet overlap in time is a complex piece of future work.

A DASH client learns the byte ranges for video chunks at different RR levels from the MPD. Hence, a *safety threshold* (in terms of upper bound for N or minimum chunklet size) can be pre-calculated to ensure that chunklets are not too small so as to become inefficient. Future work will involve developing an analytical model that predicts the AR multiplier effect given a specific content type (chunk size distributions vary with content/genres) and chunklet overlapping probabilities to derive an optimal N. We envisage the native integration of chunklets into DASH clients, which will then allow for chunklet state information to be fed into ABRs, providing another dimension for rate adaptation. A chunklet-aware client can couple per-chunklet ARs with throughput/buffer-based feedback signals and use an *adaptive chunkleting* mechanism – intelligently varying N on-the-fly, so that chunklets can be efficient yet inter-flow friendly when sharing the bottleneck.

Since the AR multiplier gain heavily depends on the degree of chunklet overlapping, it is crucial that the HTTP server serves chunklets concurrently when responding to simultaneous requests. Serialisation of chunklets will result in no performance gain. The server will also need to keep the TCP connections alive throughout the content playback duration to avoid unnecessary costs (such as TCP connection setup/teardown overhead, time taken for congestion window to grow in Slow Start mode). Server-side measurements and

optimisation techniques can be explored as part of future work.

Other future work include a comparison study of chunklets (intra-chunk) and inter-chunk techniques, performance evaluation of single/multi-DASH scenarios with a more comprehensive set of QoE metrics, the effect of chunklets on other competing traffic types (such as latency-sensitive applications), and how adaptive chunkleting clients might detect bottleneck type (FIFO or AQM) and adapt N in order to minimise induced queuing delay spikes (particularly through FIFO bottlenecks).

VI. Conclusions

With the surge of online streaming services adopting DASH-like technologies and modern AQMs being progressively rolled out in ISP equipment and consumer grade devices, we have identified a potential technique for optimising DASH performance over AQM-enabled gateways from the client-end – which we call *chunklets*. Chunklets use parallel TCP connections to perform concurrent intra-chunk video retrieval, leveraging the propensity of AQMs to improve inter-flow fairness and capacity sharing. When competing with downstream elastic TCP flows, we showed that chunklets can provide better performance over AQMs than regular DASH over FIFO or AQMs. We demonstrated the significant benefits of using FQ-CoDel when DASH chunk(lets) ACK streams are competing with aggressive upstream TCP flows for limited capacity. The FlowQueue scheduler protects DASH's ACKs, hence enabling DASH transmission to occur in a timely manner. We also explored asymmetric-AQM scenarios and showed the merits of upgrading to AQMs at either the ISP-end or home gateways.

Acknowledgements

This work was enabled in part by PhD stipend support from Netflix, Inc.

References

[1] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Queue*, vol. 9, no. 11, pp. 40:40–40:54, Nov. 2011. [Online]. Available: http://doi.acm.org/10.1145/2063166.2071893

[2] R. Pan, P. Natarajan, F. Baker, and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem," RFC 8033, Internet Engineering Task Force, Feb. 2017. [Online]. Available: https://tools.ietf.org/html/rfc8033

[3] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," IETF Draft, December 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-aqm-codel-06

[4] T. Høeiland-Jørgensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "FlowQueue-Codel," IETF Draft, draft-ietf-aqm-fq-codel-06, March 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-aqm-fq-codel-06

[5] G. White and R. Pan, "Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems," RFC 8034, Internet Engineering Task Force, Feb. 2017. [Online]. Available: https://tools.ietf.org/html/rfc8034

[6] Sandvine, "Sandvine Global Internet Phenomena Report," 2016. [Online]. Available: https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america-and-north-america.pdf

[7] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP: Standards and Design Principles," in *Proceedings of the 2nd Annual ACM Conference on Multimedia Systems*, ser. MMSys '11, 2011, pp. 133–144.

[8] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681, Internet Engineering Task Force, Sep. 2009. [Online]. Available: https://tools.ietf.org/html/rfc5681

[9] G. Tian and Y. Liu, "Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12, 2012, pp. 109–120.

[10] Linux, "Linux tc-pie man page: PIE - Proportional Integral controller-Enhanced AQM algorithm," January 2014. [Online]. Available: http://man7.org/linux/man-pages/man8/tc-pie.8.html

[11] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem," IETF Draft, draft-pan-aqm-pie-02, September 2014. [Online]. Available: https://tools.ietf.org/html/draft-pan-aqm-pie-02

[12] J. Kua, G. Armitage, and P. Branch, "The Impact of Active Queue Management on DASH-Based Content Delivery," in *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, Nov 2016, pp. 121–128. [Online]. Available: https://doi.org/10.1109/LCN.2016.24

[13] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, ser. IMC '12, 2012, pp. 225–238.

[14] Globus, "GridFTP - Globus Toolkit," January 2017. [Online]. Available: http://toolkit.globus.org/toolkit/docs/latest-stable/gridftp/

[15] R. Kuschnig, I. Kofler, and H. Hellwagner, "Improving Internet Video Streaming Performance by Parallel TCP-Based Request-Response Streams," in *2010 7th IEEE Consumer Communications and Networking Conference*, Jan 2010, pp. 1–5.

[16] C. Liu, I. Bouazizi, and M. Gabbouj, "Parallel Adaptive HTTP Media Streaming," in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, July 2011, pp. 1–6.

[17] S. Lederer, C. Mueller, B. Rainer, C. Timmerer, and H. Hellwagner, "Adaptive streaming over Content Centric Networks in mobile networks using multiple links," in *2013 IEEE International Conference on Communications Workshops (ICC)*, June 2013, pp. 677–681.

[18] R. Major and M. Hurst, "Apparatus, system, and method for adaptive-rate shifting of streaming content," Oct. 21 2014, uS Patent 8,868,772. [Online]. Available: https://www.google.com/patents/US8868772

[19] S. Smanchat, K. Sangkul, and J. Y. Tham, "Enabling Parallel Streaming of Multiple Video Sections by Segment Scheduling," in *Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia*, ser. MoMM 2015, 2015, pp. 221–226.

[20] W. Law, "Multiple HTTP connections per video segment," https://github.com/Dash-Industry-Forum/dash.js/issues/1029, 2016.

[21] M. Ansari and M. Ghaderi, "Parallel HTTP for Video Streaming in Wireless Networks," in *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sept 2016, pp. 337–342.

[22] S. Zander and G. Armitage, "TEACUP v1.0 - A System for Automated TCP Testbed Experiments," http://caia.swin.edu.au/reports/150529A/CAIA-TR-150529A.pdf, Melbourne, Australia, 29 May 2015.

[23] R. Al-Saadi and G. Armitage, "Dummynet AQM v0.2 – CoDel, FQ-CoDel, PIE and FQ-PIE for FreeBSD's ipfw/dummynet framework," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 160418A, 18 April 2016. [Online]. Available: http://caia.swin.edu.au/reports/160418A/CAIA-TR-160418A.pdf

[24] S. Lederer, C. Müller, and C. Timmerer, "Dynamic Adaptive Streaming over HTTP Dataset," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12, 2012, pp. 89–94.

[25] proxy2, "HTTP/HTTPS proxy in a single python script," 2016. [Online]. Available: https://github.com/inaz2/proxy2

[26] S. Zander and G. Armitage, "Minimally-Intrusive Frequent Round Trip Time Measurements Using Synthetic Packet Pairs," in *The 38th IEEE Conference on Local Computer Networks (LCN 2013)*, October 2013.

[27] R. K. Mok, E. W. Chan, X. Luo, and R. K. Chang, "Inferring the QoE of HTTP Video Streaming from User-viewing Activities," in *Proceedings of the First ACM SIGCOMM Workshop on Measurements Up the Stack*, ser. W-MUST '11, 2011, pp. 31–36.

[28] N. Cranley, P. Perry, and L. Murphy, "User Perception of Adapting Video Quality," *International Journal of Human-Computer Studies*, vol. 64, no. 8, pp. 637–647, Aug. 2006.

[29] J. Jiang, V. Sekar, and H. Zhang, "Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With FESTIVE," *IEEE/ACM Trans. Netw.*, vol. 22, no. 1, pp. 326–340, Feb. 2014.