# Slow but Steady: Cap-based Client-Network Interaction for Improved Streaming Experience

Vengatanathan Krishnamoorthi
Linköping University, Sweden

Niklas Carlsson
Linköping University, Sweden

Emir Halepovic
AT&T Labs – Research, USA

***Abstract*—Due to widespread popularity of streaming services, many streaming clients typically compete over bottleneck links for their own bandwidth share. However, in such environments, the rate adaptation algorithms used by modern streaming clients often result in instability and unfairness, which negatively affects the playback experience. In addition, mobile clients often waste bandwidth by trying to stream excessively high video bitrates. We present and evaluate a cap-based framework in which the network and clients cooperate to improve the overall Quality of Experience (QoE). First, to motivate the framework, we conduct a comprehensive study using the lab setup showing that a fixed rate cap comes with both benefits (e.g., data savings, improved stability and fairness) and drawbacks (e.g., higher startup times and slower recovery after stalls). To address the drawbacks while keeping the benefits, we then introduce and evaluate a framework that includes (i) buffer-aware rate caps in which the network temporarily boosts the rate cap of clients during video startup and under low buffer conditions, and (ii) boost-aware client-side adaptation algorithms that optimize the bitrate selection during the boost periods. Combined with information sharing between the network and clients, these mechanisms are shown to improve QoE, while reducing wasted bandwidth.**

## I. Introduction

HTTP Adaptive Streaming (HAS) has become the dominant approach for video delivery due to its ability to use the existing web infrastructure and to adapt to diverse network conditions for a variety of clients. Adaptation algorithms in HAS clients determine quality levels requested by clients to maintain high Quality of Experience (QoE). While QoE is a subjective measure, it is well understood that playback stalls, long startup delay and fluctuating quality levels significantly affect QoE [1], [2], [3], [4], and that QoE can be represented by objective metrics, such as *visual quality* (e.g., expressed as a *bitrate*), *stalls, quality switching, and startup delay* [5], [6].

While variability in link capacity can lead to stalls and unstable quality, such issues can occur even on stable links when multiple adaptive players compete for bandwidth, and some or all of them attempt video quality (bitrate) above the sustainable level [7]. In addition, some streaming services ignore client device context, such as screen size, and attempt to stream video at too high bitrates to add further utility to small screen users. This wastes bandwidth, causes unfairness and eats into users' data on metered links, such as cellular.

To alleviate these issues, we envision a network that co-operates with the streaming clients to improve the overall QoE, while reducing wasted bandwidth. As a first step in this direction, we explore the performance tradeoffs of data rate caps that limit the maximum bandwidth of each video stream to tune the maximum quality (and video bitrate) requested by clients. Rate cap is probably the most simple, effective and scalable mechanism commonly applied in networks to control bandwidth demand [8], [9]. To fully understand the impact of rate caps on HAS video, we conduct a comprehensive study using controlled testbed experiments and simulations, with and without caps.

Our findings highlight the main benefits and drawbacks of *fixed* rate cap when serving HAS clients. On the positive side, fixed rate caps are shown to significantly improve fairness and stability (e.g., reduced number of quality switches, in some cases reduce the number of stalls and their durations, and reduce the bandwidth wasted on video bitrates that either exceed the sustainable playback rate or provide no additional utility on small-size screens. The main drawbacks observed are increased startup times and slower stall recovery.

Motivated by the above observations, we design a cap-based framework with boosting that leverages the advantages of caps while simultaneously alleviating the drawbacks. First, the network *temporarily boosts clients* during their startup phase, after a stall, and when they are under low-buffer conditions. Second, these network-side boost periods are complemented with *boost-aware client-side adaptation* based on optimization of QoE-related metrics. We show that temporary bandwidth boosting combined with boost-aware client-side adaptation helps *reduce the startup times*, *speed up buffer recovery* after the stall, and *improve buffer occupancy*.

We then focus on the impact and value of the information sharing between clients and the network. Through evaluation of several policies that differ in the *degree of information sharing*, we show that there are significant performance benefits from *timely information sharing and cooperation*. Ideally, the network should have good knowledge of clients under low-buffer conditions, while clients should have (explicit or implicit) knowledge about the caps and the boost periods, so they could optimize the use of the extra bandwidth. Our results particularly highlight the importance of clients acquiring and using information regarding caps when being boosted; e.g., as with our boost-aware adaptation algorithms.

In summary, the main contributions are: (i) a comprehensive study highlighting the benefits and drawbacks of using a fixed

This is the authors version of the work (as accepted). It is posted here by permission of IEEE for your personal use. Not for redistribution. The definitive version is published in
*Proc. IEEE/ACM International Symposium on Quality of Service (IEEE/ACM IWQoS)*, Banff, Canada, June 2018, and is available at IEEE Xplore Digital Library via http://dx.doi.org/[doi].

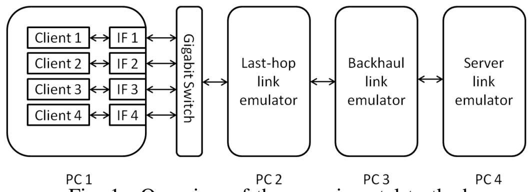


Fig. 1. Overview of the experimental testbed.

cap, (ii) a cap-based boost framework that consists of network-side boosting and boost-aware client-side rate adaptation algorithms, and (iii) experimental insights from implementing and testing these mechanisms under a wide range of scenarios.

The remainder of the paper is organized as follows. Section II outlines the methodology. Section III highlights the benefits and drawbacks of using a fixed rate cap. Section IV presents our boosting framework. Sections V and VI present experimental and simulation-based evaluation. Section VII discusses related works and Section VIII concludes the paper.

## II. Methodology and testbed

For our evaluation, we use a combination of experiments and simulations. Simulations are used to achieve scale and experiments are used to validate results for scenarios capturing the characteristics of scenarios with different bottleneck links (e.g., server-side, backhaul network, last-hop), access technologies (e.g., WiFi or LTE), bandwidth variations (e.g., using competing players, or real-world traces), video encodings (e.g., use of different videos), and players (e.g., instrumentation and experiments using both OSMF and dash.js). Through evaluation across a wide range of scenarios and technologies, we show that the conclusions are generally applicable.

### *A. Experimental testbed*

To capture the characteristics of different links and access technologies along the end-to-end delivery path, we set up an experimental testbed consisting of four workstations. Referring to Figure 1, PC 1 hosts four competing streaming clients and the other three machines (PCs 2, 3, and 4) are responsible for emulating the network conditions in each part of the delivery path. *Dummynet* [10] is used to emulate the link characteristics (e.g., bandwidth, packet delay and loss) at each link. We also configure each PC to reside in a different IP subnet.

PC 4 runs the video server, and emulates the path properties along the links connecting the server to the operator's network. PC 3 emulates the backhaul link of the operator and implements per-client rate caps. PC 2 emulates the last-hop. To capture some key differences in modern access technologies, we apply different scheduling policies at this link. A FIFO queue is used to capture the behavior of most home WiFi routers. Individual queues for each client are used to capture the implementation by a typical LTE base station (eNodeB).

Finally, PC 1 is configured with four gigabit Ethernet cards, 8 gigabytes of RAM, and a quad-core Intel Xeon CPU. We use the IP Network Namespaces (*ip-netns*) package to associate each interface with its own logical namespace and run HAS clients on top of these namespaces. This allows us to have multiple non-interfering client instances residing in a single machine. To avoid interfering with experiments, additional interfaces are used for administrative and logging purposes.

**Clients and servers:** We use two open-source players: OpenSource Media Framework (OSMF)[1] and dash.js[2], to show broader applicability of our framework. The dash.js source code is used in its unmodified form (implementing a hybrid of BOLA [11] and rate-based adaptation). In contrast, motivated by prior research suggesting the use of larger buffers, we set the OSMF player to use min/max buffer thresholds of 30/40 seconds. As a proof of concept implementation, we modify the OSMF player to share information with the network and implement our optimized boost-aware adaptation algorithms.

In the OSMF setup, we run Adobe Media Server and the client code is modified to log parameters of interest to file. While we run experiments with different videos, the default one is 10 minutes long and is available in the following average encoding bitrates: 144, 268, 625, 1124, 2217, 4198 kbit/s. Since dash.js uses a client-side JavaScript implementation, it has limited access to the client's file system. To collect client-side metrics, we instrument the web browser and export metrics to an external database.

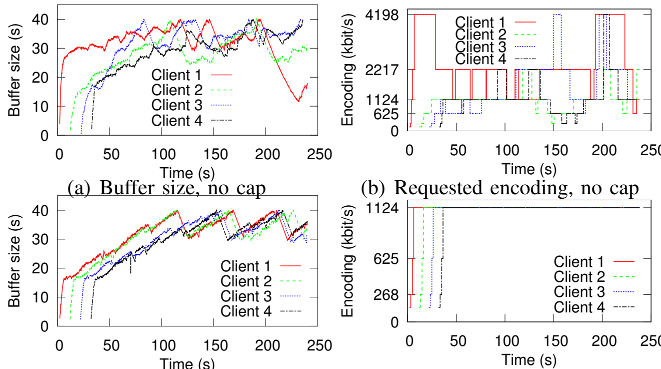
**Limitations:** Our WiFi and LTE scenarios do not capture all differences between WiFi and LTE seen in practice. For example, we do not emulate details associated with heterogeneous and time varying signal-to-noise (SNR) ratios, and how this can impact scheduling in LTE. Rather than trying to capture the low-level details of the physical environment we simply emulate the bandwidth conditions of the clients using real bandwidth traces and differentiate the scenarios based on how queuing is done at the last hop (single FIFO queue for WiFi and individual queues for the LTE-based scenario). We argue that the trace-based per-client throughput and simplified edge abstraction allows for a reasonable comparison of the impact of using per-client rate caps on application-level performance.

### *B. Simulation setup*

We developed a simulator in C++ that captures the clients' rate adaptation, bandwidth variations, individual caps, temporary boosting, and boost-aware client-side policies. In multi-client scenarios we assume a shared bottleneck link and use max-min fairness at all times, taking into account each client's individual cap and whether each client is currently active or not (based on the on-off periods of chunk-based delivery). The default bitrate adaptation logic is kept simple. Each client uses an Exponentially Weighted Moving Average (EWMA) with $\alpha = 0.4$ to compute a historic per-chunk throughput estimate. The client then selects the highest available encoding below 80% of this estimate. Buffer thresholds are selected and used to make the same decisions that the OSMF player would make.

**Bandwidth and chunk-size variations:** For many of our experiments (both with testbed and simulator) competing clients generate most of the bandwidth variations. However, we also perform experiments where we vary the bandwidth

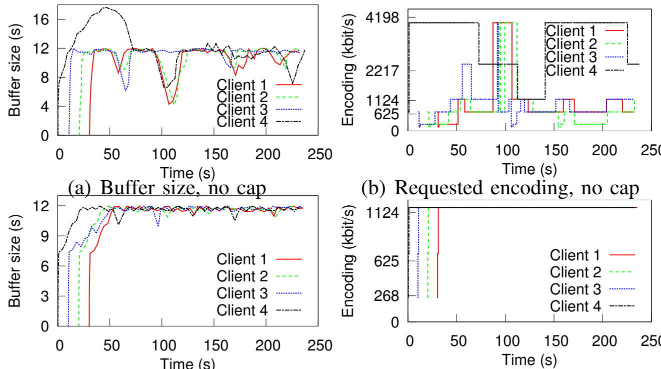
1. http://www.adobe.com/devnet/video/articles/osmf_overview.html
2. https://github.com/Dash-Industry-Forum/dash.js/wiki



(a) Buffer size, no cap (b) Requested encoding, no cap

(c) Buffer size, with caps (d) Requested encoding, with caps

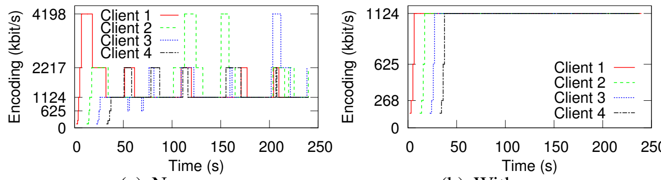
Fig. 2. Example with four competing OSMF clients.



(a) Buffer size, no cap (b) Requested encoding, no cap

(c) Buffer size, with caps (d) Requested encoding, with caps

Fig. 3. Example with four competing DASH.js clients.



(a) No cap (b) With caps

Fig. 4. Encoding rates in LTE-based scenario (OSMF).

available for a client based on traces. In these cases, the clients' maximum available bandwidth is equal to the minimum of (i) their individual rate cap and (ii) the bandwidth value in the trace. For the experiments, this value is then used to constrain each client's maximum rate, while for the simulations, this value is used to calculate each clients (fair) bandwidth share.

The chunk-size variability within each video associated with Variable BitRate (VBR) encoding can cause chunk downloads to take longer than expected. To capture that not all chunks are the same size (despite having the same playback duration and encoding quality), we extracted the chunk sizes of 50 random YouTube videos and used the obtained chunk-size sequences (one for each quality level and video) in our simulations.

## III. Impact of a Fixed Rate Cap

This section highlights the main benefits, drawbacks, and tradeoffs when using a *fixed* rate cap. Throughout the section we do not modify the clients; we only consider the impact of fixed caps.

### A. Experimental scenarios: Benefits and drawbacks

Figures 2 and 3 show example results with four competing OSMF and dash.js players, respectively. In both cases, the top row ((a) and (b)) shows results for the no-cap case and the bottom row ((c) and (d)) shows results with a fixed cap. Furthermore, sub-figures (a) and (c) show the buffer occupancy over time, while sub-figures (b) and (d) show the encoding rate of the played video chunks. In these experiments, clients share a 6000 kbit/s bottleneck, and start times of sessions are staggered by 10 seconds. We compare cases without caps and with individual caps of 1500 kbit/s (on the backhaul link). This cap corresponds to the equal share of bandwidth that each client would theoretically obtain using TCP.

*1) Performance benefits of caps:* Several positive observations stand out. First, the use of individual caps *improves playback stability*. Requested bitrates highly fluctuate in the no-cap case. In contrast, the capped clients quickly reach a stable playback quality and use this quality steadily throughout the playback session. Variations like in the no-cap cases can have significant negative impact on QoE [1], [2], [3] and have previously been reported by others [7], [12].

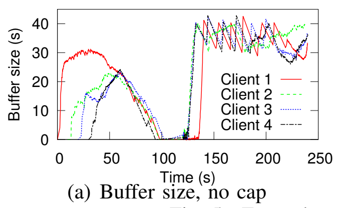
Consequently, the *buffer conditions are more stable*, as clients do not try to fetch unsustainable encodings, and the need to buffer more video is alleviated. The larger initial buffer size of the first arriving dash.js client (Figure 3(a)) can be explained by lack of competition and the way the unmodified dash.js client uses throughput measurements for dynamic buffer sizing. When throughput is high, the player aims for a larger target buffer occupancy ($\approx 40$ seconds), while when the throughput is lower the target is 12 seconds. Here, the first client initially aims for the larger target, but must fall back to the smaller target once other clients start playback.

Second, the use of individual caps *improves fairness among competing clients*. For example, the first arriving client in the non-capped case unfairly obtains higher throughput and encoding quality than the other clients (e.g., more of the session at highest quality, as seen in Figure 3(b)). This type of unfairness among competing HAS players is known [3], [13], and is due to one client starting out with a larger bandwidth share combined with the conservativeness of TCP and HAS adaptation. This has a compounding (negative) effect for clients that start with the smaller bandwidth share.

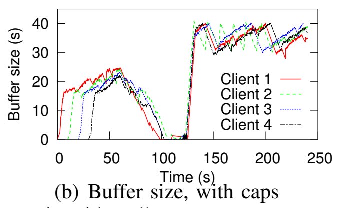
Third, without caps, bandwidth variations cause a lot of wasted user data and bandwidth on downloading encodings well above clients' fair share (1500 kbit/s) or utility threshold. This is often observed in mobile clients with small screen sizes, whose utility starts to diminish with encoding rates above 1000 kbit/s (c.f. Figure 7 and [14]).

We can conclude that with caps, stability in playback quality and buffer conditions combined with fairness *improves overall user QoE*. Even when bandwidth is equally shared among non-capped clients, instability causes them to play more poor quality chunks, reducing their utility. For example, in the non-capped case 23.2% of the chunks have encodings below 1124 kbit/s, compared to 6.25% with the 1500 kbit/s cap.

The above observations also hold true for other scenarios. Figure 4 shows the encoding levels observed for the corresponding LTE-based scenario, when using OSMF. The buffer



(a) Buffer size, no cap



(b) Buffer size, with caps

Fig. 5. Example scenario with stalls.

TABLE I

RELATIVE INCREASE IN STARTUP TIMES WITH CAP.

| | | | Client arrival | | | |
|---|---|---|---|---|---|---|
| | | Encoding | First | Second | Third | Fourth |
| Shared bottleneck | 6000 kbit/s | Unmodified | +26.4% | +30.8% | +13.2% | +27.4% |
| | | 144 kbit/s | +21.1% | +23.0% | +27.1% | +25.9% |
| | | 268 kbit/s | +33.9% | +36.0% | +18.9% | +32.9% |
| | | 625 kbit/s | +92.0% | +80.6% | +5.5% | +52.7% |
| | | 1124 kbit/s | +130.5% | +84.5% | +113.3% | +51.7% |
| | 12000 kbit/s | Unmodified | +20.6% | +20.5% | +17.5% | +19.0% |
| | | 144 kbit/s | +21.5% | +21.8% | +19.5% | +14.6% |
| | | 268 kbit/s | +33.5% | +32.5% | +27.8% | +34.5% |
| | | 625 kbit/s | +96.0% | +92.6% | +77.2% | +85.0% |
| | | 1124 kbit/s | +190.1% | +184.9% | +112.6% | +111.5% |

conditions (omitted) are very similar in the WiFi case.

Fourth, caps can reduce stall occurrences and (sometimes) stall durations. While (as intuition suggests) the use of caps extends the minimum time over which clients can recover from a specific stall event, we have found that use of individual caps also can help reduce stalls. One reason is that the high variability in both encodings and buffer occupancy for non-capped clients increases their likelihood to stall in the first place. Then, stalls that occur when a client downloads a large chunk (encoded at a high rate) often takes longer to recover from. This problem is illustrated by the first client (red) in Figure 5(a), who stalls for a longer period than other clients due to downloading a higher-quality chunk at the time of the stall. In this example scenario, we forced a stall by temporarily reducing the bottleneck capacity to 100 kbit/s between 60-120 seconds (instead of the default 6000 kbit/s), and the clients do not try to abort and replace the stalled chunk download with a lower quality download.

*2) Performance drawbacks of caps:* The primary drawbacks of fixed caps are increased startup times and slower stall recovery, caused by reduced peak download rates. We focus on startup times and note that the startup times provide insights into the stall recovery times after outages or when replacing stalled chunk downloads with lower encoding downloads.

Table I shows the average increase in startup times when using fixed caps compared to the non-capped case. We again use four competing clients with a shared server-side bottleneck of 6000 or 12000 kbit/s, and an individual cap of 1500 kbit/s. To understand the impact of competition, we stagger arrivals by 20 seconds and report the average startup times for each client over 10 runs. Results are shown both for the unmodified OSMF client and when using a pre-determined encoding quality for the initial chunk. In both cases, the player starts playback in a default fashion, after having obtained the first chunk (with a four second playback duration). Some players use a manifest-specified encoding to start (e.g., HLS), while some decide based on pre-set initial bandwidth (e.g. ExoPlayer). There is also an option to maintain history of previous sessions to drive this decision.

Results for the first arriving client in Table I show the base increase in startup delay for capped clients that have no competition. The main insight is that even medium initial bitrate of 625 kbit/s can experience near doubling of startup time. Attempting higher encoding results in further significant increase. The unmodified client fares much better due to its conservative initial selection. We also note that overall increases in startup time are not as high as it might be theoretically expected. This can be explained by the combination of TCP dynamics (not all clients reaching their fair share quickly) and the ON-OFF nature of HAS allowing for a new client (often 4th) to join when an existing client is in the OFF phase.

## B. Impact of fixed vs. fair-share caps

We next examine the quantitative impact of two example cap policies and a no-cap baseline on the observed benefits.

- *Fair-share cap:* Given $m$ clients, each client has an individual cap equal to $1/m^{\text{th}}$ of the bottleneck bandwidth.
- *Fixed cap:* Each client is given an individual cap equal to 1500 kbit/s, regardless of the bottleneck bandwidth.
- *No cap:* There are no individual caps.

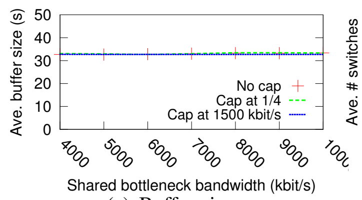
While variations of these policies can be implemented in networks, here we use them to study the impact of selecting different sized caps. For example, with $m$=4, when the shared bottleneck is smaller (larger) than 6000 kbit/s, the *fixed cap* of 1500 kbit/s is above (below) the fair share. Similarly, when the shared bottleneck is smaller (larger) than 6000 kbit/s, the *fair-share cap* is below (above) 1500 kbit/s.

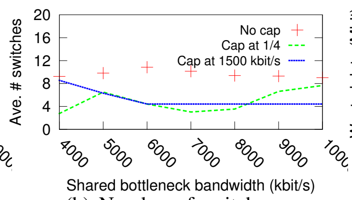
The *fair-share cap* policy avoids bandwidth waste on downloading chunks of higher than sustainable encoding rates when clients share bandwidth fairly. This bandwidth is better used to reduce the number of poor quality chunks and to maintain stability. Motivated by this observation, we introduce the *wasted fair-share volume* metric, which measures the fraction of bytes delivered at encoding rate higher than the fair-share.

The *fixed cap* policy is motivated by the diminishing utility observed by small screen clients (e.g., mobile phones) with encodings above 1000 kbit/s (c.f. Figure 7 or [14]). Motivated by this limitation of the human eye, we also introduce the *wasted small-screen volume* metric, which measures the fraction of bytes delivered with encodings above 1500 kbit/s.

Figure 6 shows the average (a) buffer size, (b) number of switches, (c) wasted fair-share and small-screen volume, across different bandwidth levels. The results represent averages over 50 simulation runs (each with different videos). We note that the *fair-share cap* and *fixed cap* are identical when the shared bottleneck is 6000 kbit/s. On the left-hand-side of this point, the *fair share* is more restrictive and on the right-hand-side, the *fixed cap* is more restrictive.

To maintain similar average buffers across policies and conditions (Figure 6(a)), the capped clients require significantly fewer encoding switches (Figure 6(b)) and the number of switches is typically the lowest for the more restrictive of the two policies. Again, the use of caps helps improve stability. The lack of cap (i.e., *no cap* policy curves) can result in a lot of wasted bandwidth (Figure 6(c)). We note that these


(a) Buffer size


(b) Number of switches

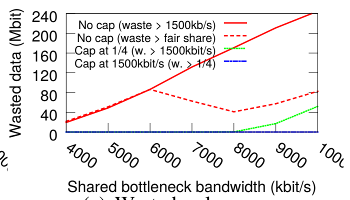

(c) Wasted volume

Fig. 6. Comparison of playback metrics with different rate cap policies.

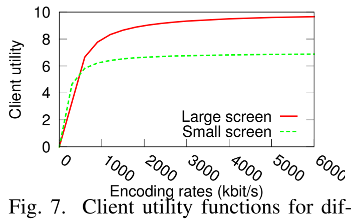
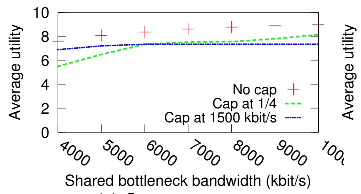


Fig. 7. Client utility functions for different screen sizes (from [14]).


(a) Large screen

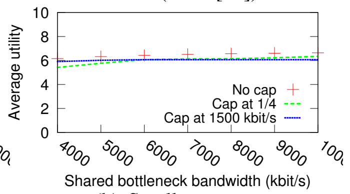

(b) Small screen

Fig. 8. Rate-based (only) utility for different screen sizes.

values are conservative. For example, in the default case (with 6000 kbit/s), we compute a 22.1% saving if all chunks with encodings higher than 1500 kbit/s instead were to be delivered using 1500 kbit/s encodings. This corresponds to our *wasted* data. However, in our test case, the highest encoding rate below 1500 kbit/s is 1124 kbit/s. Taking this into account, in practice, 30.2% of the bandwidth could actually have been saved if simply selecting these chunks at the 1124 kbit/s encoding. To put the savings into further perspective, we note that the chunks encoded at rates larger than the fair share (here, equal to 1500 kbit/s) make up 54.2% of the total delivered data.

When discussing users' QoE, multiple dimensions must be taken into account, including: (i) stall metrics such as stall occurrences and durations, (ii) playback utility metrics based on encoding rates of individual chunks, and (iii) playback quality fluctuations. In the above experiments we did not observe stalls, and since we thus far in the paper primarily have focused on (i) and (iii), we now turn our attention to (ii). When discussing playback utility, we associate each chunk with an individual utility, using the utility function proposed by Vleeschauwer et al. [14]:

$$f(q) = \beta\left((q/\theta)^{1-\alpha} - 1\right)/(1-\alpha), \quad (1)$$

where $\alpha$>1, $\beta$>0 and $\theta$>0 are screen dependent parameters. Figure 7 shows the example utility functions used in our evaluation for (i) small screen clients ($\alpha$=2, $\beta$=7, $\theta$=0.1 Mbps), and (ii) large screen clients ($\alpha$=2, $\beta$=10, $\theta$=0.2 Mbps). Clearly, there are diminishing returns to downloading higher encoding rates $q$, and low encoding rates should be avoided to maintain a high minimum utility. Again, caps help significantly here, as (indirectly) implied by fewer encoding switches (Figure 6(b)) and less wasted data (Figure 6(c)). Another way to measure the encoding-related utility is the average utility across all played chunks (ignoring encoding switches), shown in Figure 8. In general, the encoding-related utility differences between the policies are small, especially for the small screen case, and in practice, these small utility gains are typically substantially outweighed by the added benefits observed by capped clients.

### C. Impact of cap location

We have also evaluated the use of fixed caps at different bottleneck links using our experimental testbed as well as simulations. While these results are omitted due to space, we have found that (i) the *impact of the cap location is relatively minor*, and (ii) the simulation results align well with our testbed experiments, despite not capturing all the subtleties of TCP behavior, and therefore resulting in less extreme instabilities and unfairness for the non-capped clients.

### D. Real-world LTE experiments

We validate the data saving and stability on real commercial apps, from two VoD and two Live services, by capping per-client throughput at 1500 kbit/s. Four mobile devices (Samsung J7 running Android 7.0) are set up to use a Squid proxy over a cellular LTE network. The clients are started staggered 10-20 seconds, and then restarted periodically to create more startup measurements. The proxy applies the individual rate caps, limits the overall link capacity at 6 Mbit/s, and collects traces of the devices' HTTP traffic. Metrics for startup times, bitrates, and rebuffer ratios are extracted as outlined by Mangla et al. [15]. All apps use HLS with audio and video muxed into the same stream, but use different bitrates and chunk durations. Live1 is a major TV channel, Live2 is a major TV broadcaster, VoD1 is a major sports channel, and VoD2 is a major TV broadcaster. Table II summarizes these results.

We find that the cap prevents higher bitrates (above 1500 kbit/s) from being delivered, as expected. This yields savings of 9-32% on data compared to the un-capped case with 6 Mbit/s. By removing the highest bitrates and competition between players, switching improves by 33-97% for three of the services (with one service increasing the switching by 2.5%). This qualitatively confirms our testbed findings on data savings and stability. Furthermore, for the three services with substantial stability improvements, the cap reduces the number of stalls by 78-100% and the rebuffering ratio by 86-100%, at the expense of increased startup times (by 11-77%).

Large-scale benefits of rate caps on data savings can also be derived from recent work [15], using the observation that some video services internally cap bitrates delivered to clients, while others do not. Services that cap bitrates on a cellular network can have up to 50% lower bandwidth usage [15], without losing the visual quality on small screens [14].

## IV. Bandwidth Boosting

Thus far, we have shown that rate caps come with several benefits (e.g., improved stability, fairness, less wasted bandwidth, sometimes fewer and shorter stalls), but also some drawbacks (e.g., increased startup times and slower stall recovery). To address these drawbacks while further

TABLE II
REAL-WORLD MEASUREMENTS WITH AND WITHOUT CAPS.

| Service | Cap vs. no cap | Play time | Ave. bitrate | Switches /min | Startup time | Stalls | Rebuff. ratio |
|---|---|---|---|---|---|---|---|
| Live1 | No cap | 6,744 | 1,347 | 0.28 | 7.1 | 0.50 | 0.54% |
|  | Capped | 3,944 | 982 | 0.28 | 7.9 | 0.00 | 0.00% |
| Live2 | No cap | 2,640 | 1,055 | 1.17 | 7.6 | 1.00 | 3.36% |
|  | Capped | 3,310 | 843 | 0.04 | 13.4 | 0.22 | 0.46% |
| VoD1 | No cap | 5,150 | 1,170 | 0.58 | 23.3 | 1.24 | 4.94% |
|  | Capped | 4,920 | 1,061 | 0.16 | 27.8 | 0.14 | 0.43% |
| VoD2 | No cap | 9,300 | 1,192 | 0.19 | 4.8 | 0.69 | 0.49% |
|  | Capped | 10,716 | 812 | 0.13 | 4.6 | 1.11 | 0.51% |

TABLE III
SUMMARY OF NOTATION AND INFORMATION.

|  | Notation | Information |
|---|---|---|
| Client | $B_0$ | Current buffer occupancy |
|  | $\overline{B}$ | Maximum buffer threshold |
|  | $B^*$ | Target buffer |
|  | $Q_l$ | Encoding with quality level $l$ ($1 \leq l \leq L$) |
|  | $q_0$ | Most recently requested chunk's encoding rate |
|  | $X$ | Current estimated throughput |
|  | $T_c$ | Chunk duration |
| Network | $C$ | Default rate cap |
|  | $C^+$ | Boosted rate cap |
|  | $\delta$ | Remaining boost duration |

reducing stall occurrences, we introduce two mechanisms: (i) temporary boosting, and (ii) boost-aware client-side rate adaptation (*boost-aware adaptation*). The idea of temporary boosting is simple. When a client starts viewing a video or otherwise drains its buffer, the network temporarily increases its individual cap. The boost-aware adaptation modifies the player's encoding selections to optimize the use of extra bandwidth during the boost period. Together, these mechanisms help reduce startup times, improve stall recovery times, and provide healthier buffer conditions (through faster buffer fill), effectively also improving the protection against stalls.

## *A. Information sharing vs. inference*

For efficient implementation of the above mechanisms the client and network need to share (or in other ways extract) information regarding buffer conditions and boost periods so that (i) the network can determine appropriate times and magnitudes to boost clients, and (ii) the clients have the necessary information to optimize use of this extra bandwidth.

**Player-to-network sharing:** To correctly apply boosting, the network needs to "know" when clients have low buffers. Ideally, clients would share this information with the network using existing [16] or custom-made protocols. Using BUFFEST [17] and similar systems [18], [19], [20], [21], this information can also be inferred from network traffic, even when encrypted, with reduced but acceptable accuracy. These systems can extract low-buffer signals on their own or verify the signals submitted by players claiming low buffers.

**Network-to-player sharing:** Clients can benefit substantially from information that the network can provide. For example, without knowing their individual caps, clients may download encodings higher than their regular caps (during boost periods), hurting long-term performance. Naturally, any protocol that allows the network to communicate with clients can be extended to include this information. In our implementation of boost-aware adaptation (Section IV-B), we assume that the network explicitly notifies clients about when, by how much, and for how long they are boosted. We also note that HAS clients are in fact well-equipped to recognize caps and boost periods via their historical throughput measurements, especially if they are consistently applied. We next evaluate potential benefits of boosting with different levels of cooperation, but leave the mechanisms and implementations by which this cooperation is achieved for future work.

## *B. Client-side behavior with boosting*

To take full advantage of the extra bandwidth during boost periods, clients should perform careful encoding selection, leveraging both local information (e.g., "Client" rows in Table III) and network information, either shared or inferred (e.g., "Network" rows in Table III).

First, the client should limit buffer filling to the maximum buffer threshold $\overline{B}$ (unless still boosted when reaching this threshold). Assuming that the current buffer is $B_0$ and each chunk has a play duration of $T_c$, a bound on the number of chunks $n$ that the client should aim to download during the remaining boost time $\delta$ can then be calculated as:

$$n \leq (\overline{B} - B_0 + \delta)/T_c. \tag{2}$$

Second, the client should not waste bandwidth on encodings greater than the encoding that the (regular) rate cap allows. Therefore, assuming that the cap is $C$ and the next $n$ chunk requests are enumerated 1 through $n$, we have:

$$q_i \leq \max_l(Q_l | Q_l \leq C), \qquad 1 \leq i \leq n. \tag{3}$$

Third, since it is important that clients quickly recover from low buffer conditions and frequent encoding fluctuations should be avoided, it is advantageous to plan for monotonically non-decreasing encodings during the boost period:

$$q_1 \leq q_2 \leq ... \leq q_n \leq \max_l(Q_l | Q_l \leq C). \tag{4}$$

This allows the clients to quickly build up a buffer safety margin, while also satisfying constraint (3).

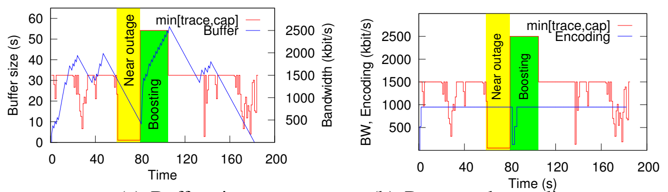
Fourth, a client should not plan to exceed the current estimated throughput $X$ (estimated as the boosted cap $C^+$ when no chunk has been downloaded during the boost period) during the remaining boost period $\delta$:

$$\sum_{i=1}^{n} q_i T_c \leq \delta X. \tag{5}$$

Given the above constraints, in the following, we describe three candidate policies that a client can apply when selecting the encoding of the next chunk to download.

**Greedy fixed quality:** The client first determines the largest possible number of chunks $n$ that satisfies constraint (2) and is feasible according to the constraint (5) using some minimum quality $Q_1$. Then, given this $n$, the client picks the largest possible encoding level $l$ that satisfies constraint (4) and is feasible within the remaining download budget, as determined by constraint (5). The solution to this problem can be computed in $\mathcal{O}(\log(L))$ using a binary search on feasible $n$ and $l$, where $L$ is the number of available encodings.

**Maximum encoding quality:** Similar to the first policy, we first greedily determine the maximum number of chunks to download during the remaining boost duration, which satisfies constraint (2). Second, we solve the following optimization problem: maximize $\sum_{i=1}^{n} q_i$, subject to constraints (4) and (5).



(a) Buffer size (b) Requested encoding rates

Fig. 9. Near-outage example with client using *multi-objective quality*.

**Multi-objective quality:** Finally, we consider a more generalized formulation that weighs different objectives into a combined objective function:

$$\text{maximize} \sum_{i=1}^{n} \Big( \alpha B_i + (1-\alpha) f(q_i) \qquad (6)$$

$$- \beta |f(q_n) - f(\max_l (Q_l | Q_l \leq C))| - \gamma |B_n - B^*| \Big).$$

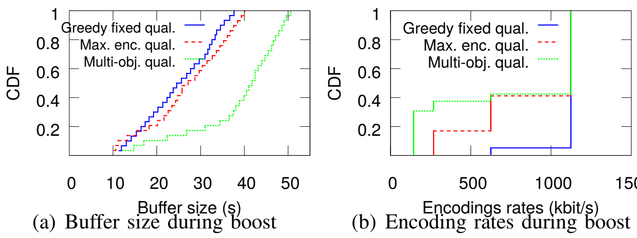
Here, $B_i = B_0 + iT_C - \sum_{j=1}^{i} \frac{q_j T_c}{X}$ is the estimated (predicted) buffer level after chunk $i$ has been downloaded, and we use a parameter $\alpha$ to weigh the relative importance of maximizing the average buffer size $\frac{1}{n}\sum_i B_i$ and the average encoding utility $\frac{1}{n}\sum_i f(q_i)$, where $f(q_i)$ is the utility function (1). Parameters $\beta$ and $\gamma$ give more/less (relative) weight to the secondary objectives of ramping up the encodings so to be as close to the ideal steady-state encoding $\max_l(Q_l | Q_l \leq C)$, when operating under the cap $C$, and to fill the buffer near target buffer size $B^*$. In our experiments, this is typically equal to the upper buffer threshold $\overline{B}$ in OSMF, or the average expected buffer in the steady state; i.e., $(\underline{B} + \overline{B})/2$, where $\underline{B}$ is the lower buffer threshold in OSMF.

## V. Experimental evaluation of boosting

### A. *Illustrative example of the boosting framework*

We have implemented the above policies in OSMF. As the first validation, we use a simple "near-outage" scenario. Here, we used real bandwidth traces [22] combined with an individual cap of 1500 kbit/s. We add a 20-second near-outage between 60 and 80 seconds, during which we set the client's cap to 50 kbit/s, followed by a $\delta$=25 second boost period, where the individual cap is temporarily increased to $C^+$=2500 kbit/s. Figure 9 illustrates this scenario for an example client using *multi-objective quality*. Throughout the evaluation, unless stated otherwise, we run this policy with $\alpha$=0.5, $\beta$=1, $\gamma$=1, and $B^*$=$\overline{B}$. As per design, the client selects encodings such that it quickly refills the buffer and uses increasingly higher encodings, aiming to fill the buffer at the end of the boost period (in this case somewhat over-shooting).

The three boost-aware policies (as per design) make somewhat different tradeoffs between the buffer size progression during the boost period and the encoding selection. As desired, the *multi-objective quality* policy is typically the fastest to ramp up the buffer size and therefore spends the least amount of time with a small buffer (e.g., below 10 or 20 seconds). To achieve this, it typically uses lower encodings at the start of the boost period. The other policies prioritize average encodings (in their objective functions) at the expense of slower buffer



(a) Buffer size during boost (b) Encoding rates during boost

Fig. 10. Example comparisons of boost-aware client-side adaptation policies.

TABLE IV
NORMALIZED RELATIVE STARTUP REDUCTION WITH BOOST.

| | | | Client arrival | | | |
|---|---|---|---|---|---|---|
| | | Boost | First | Second | Third | Fourth |
| Shared bottleneck | 6 Mb/s (vs.625) | +625 kbit/s | -94.8% | -84.8% | -42.4% | -71.9% |
| | | +1250 kbit/s | -96.2% | -87.6% | -47.7% | -75.7% |
| | | +2500 kbit/s | -102.0% | -90.2% | -46.7% | -78.9% |
| | 6 Mb/s (vs.144) | +625 kbit/s | -12.0% | -13.7% | -13.2% | -17.0% |
| | | +1250 kbit/s | -13.6% | -16.9% | -22.8% | -22.1% |
| | | +2500 kbit/s | -20.2% | -19.9% | -21.0% | -26.4% |
| | 12 Mb/s (vs.144) | +625 kbit/s | -11.8% | -13.1% | -9.8% | -5.6% |
| | | +1250 kbit/s | -16.4% | -18.2% | -18.0% | -11.5% |
| | | +2500 kbit/s | -21.2% | -21.6% | -17.3% | -13.3% |

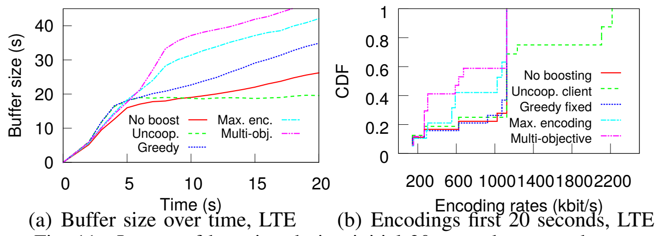
progression (and hence also lower buffer levels). These relative differences are illustrated in Figure 10. Here, we show the CDFs of the buffer size and selected encodings during the boost period for each of the three policies, as observed over 10 runs using different bandwidth traces (but the same near-outage and boost period as in the original example).

### B. *Startup improvement with boosting*

As desired, boosting improves the video startup times that increased under the fixed cap. Table IV summarizes the normalized reduction in startup times when using *multi-objective quality*. Here, four clients with 1500 kbit/s caps start playback staggered by 20 seconds and each client is boosted for 20 seconds during the startup phase.

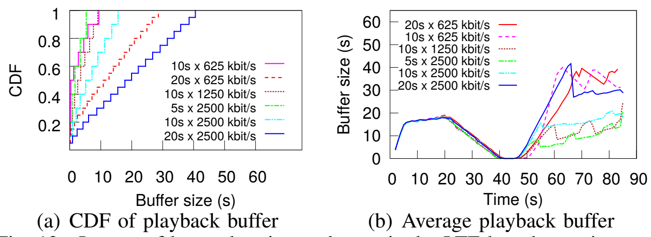
The *normalized reduction in startup times* allows for easier comparison to the increases due to fixed caps (shown in Table I), and we calculate it as follows. We take the absolute increase in startup time under a fixed cap for a certain encoding rate of initial chunks (e.g., 144 or 625 kbit/s), and divide it by the startup times in the corresponding non-capped case. With this normalization, we can directly compare the decreases in Table IV (emphasized with negative signs) with the increases in Table I (emphasized with plus signs). Specifically, the change for boosted clients relative to non-capped clients is equal to the sum of the values extracted from the two tables. For the boost-aware adaptation policies, the initial chunk requests may differ between sessions. Therefore, the 144 kbit/s values provide a pessimistic lower bound on the improvements.

Overall, we find that *boosting helps recover most of the increase in startup delays*, and sometimes even improves over the non-capped case. These results clearly show that boosting can substantially help, even when there is competition (as for the third and fourth arriving clients). For example, with a 625 kbit/s boost and a 6000 kbit/s shared bottleneck, the first and fourth arriving clients see on average a reduction by -94.8% and -71.9%, respectively, compared to the capped clients with



(a) Buffer size over time, LTE (b) Encodings first 20 seconds, LTE

Fig. 11. Impact of boosting during initial 20 second startup phase.



(a) CDF of playback buffer (b) Average playback buffer

Fig. 12. Impact of boost duration and magnitude; LTE-based experiments.

625 kbit/s encoding for the initial chunks. Comparing to average increases of +92.0% and +52.7% with cap, only moderate boosting is needed for the boost-aware system to outperform the non-capped clients. While additional boosting is needed to recover most of the increases for capped clients that always start from 144 kbit/s, a boost magnitude of +2500 kbit/s provides very similar startup times to the corresponding non-capped client. For example, the first and fourth client recover -20.2% of +21.1% and -26.4% of +25.9%, respectively. The LTE and WiFi results are consistent, with LTE experiments having ≈2.9% longer startup times than WiFi, on average.

Looking beyond the initial chunk(s), temporary boosting combined with boost-aware adaptation can further help build up a fair sized buffer relatively quickly without sacrificing video quality. Figures 11(a) and 11(b) show the evolution of the average buffer size over time and the encoding CDF for an LTE client during the first 20 seconds. The corresponding results for WiFi (omitted) are similar.

Here, we compare the three boost-aware client-side adaptation policies head-to-head, but also include results for two additional baseline policies. In the *uncooperative client* case, the client completely ignores all cap or boost information and instead downloads chunks as it normally would according to the HAS player's default adaptation algorithm, including the boost period. In the *no boosting* case, no boosting is provided by the network and the client simply goes back to using the HAS player's default adaptation algorithm. The healthiest buffer levels are observed when boosting is combined with the *multi-objective quality* policy. As expected, non-boosted and uncooperative clients perform the worst, showing that boosting is mostly wasted if clients do not cooperate. This strengthens the argument that cooperation between the clients and network is needed to achieve the best possible performance.

## C. Boost settings: duration and magnitude

The boost duration and magnitude impact the boost's effectiveness. Figure 12 presents buffer size results for example boost durations and (extra) boost magnitudes in LTE experiments. WiFi results (omitted) are similar. Here, Figure 12(a) shows CDFs of the buffer occupancy during the boost periods and Figure 12(b) shows the average buffer values as a function of time. In all experiments, there are four competing clients with a shared bottleneck of 6000 kbit/s and individual caps of 1500 kbit/s. Furthermore, for one of the clients we introduce a 25 second near-outage (with 50 kbit/s) at the 20 second mark and then boost the client at the 45 second mark, when the buffer typically has fallen to (or roughly below) 10 seconds. We again present results only for *multi-objective quality*.

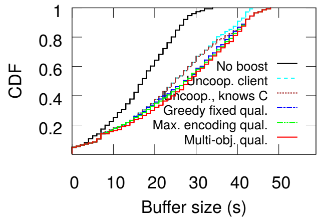
To allow comparisons of the impact of boost parameters, we carefully picked six example cases. Three of them have the same (extra) boost magnitude (measured as the difference between the boosted and non-boosted cap; i.e., $C^+ - C$) of 2500 kbit/s, three have the same duration of 10 seconds, and three have the same time-aggregate boost (as calculated over the boost period; i.e., $\delta(C^+ - C)$) of 12500 kbit.

The boost magnitude is important for fast stall recovery. For example, the 2500 kbit/s boosts always provide faster buffer buildup during the first 5 seconds of the boosts than the 1250 kbit/s and 625 kbit/s boosts. In fact, with the 2500 kbit/s boost, we are able to get back to the original buffer level within 10 seconds. With smaller boosts it takes longer. While the boost duration (e.g., when keeping the extra magnitude equal to 2500 kbit/s) does not impact the initial speed with which the buffer grows (Figure 12(b)), we have found that longer duration boosts can be valuable if aiming for a large buffer at the end of the boost (e.g., right-most points in Figure 12(a)). Furthermore, comparing the three 2500 kbit/s curve after 20 seconds (at the 65 second mark), when only one of the 2500 kbit/s boosts are still active, we see a clear ordering of the corresponding buffer sizes, with the 20 seconds boost providing by far the largest buffer of these three. The observations are consistent across both WiFi (omitted) and LTE.

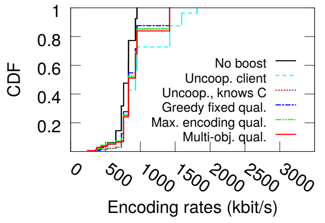
Finally, we note that we have achieved further speedups in the buffer recovery (for a given boost) with the help of a scheduler that temporarily gives more weight to the boosted client's video flow. However, this type of boosting is highly speculative since commodity base stations currently cannot distinguish application layer characteristics in traffic. These results are therefore omitted due to lack of space. Yet, we argue that this framework could be a good platform for such optimization if similar functionality is made available in 5G and future WiFi base stations.

## VI. Simulation-based Evaluation of Boosting

To evaluate the long-term dynamics of cap-based boosting we use simulations with four active clients. Each client plays videos back-to-back for a total of 150 viewings per client. When a client completes the playback of one video, it randomly picks another from a set of 50 videos. For each video, we keep track of the exact chunk sizes (extracted from 50 YouTube videos), playback position, and playback duration. All clients are capped at 1500 kbit/s with a shared bottleneck of 6000 kbit/s. For one in every four viewings, we then introduce a 20 second (complete) outage at a random time during the session (forcing low buffer conditions). At times

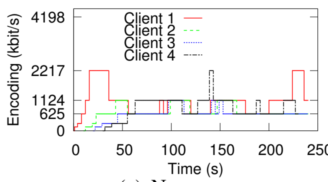


(a) CDF of buffer size

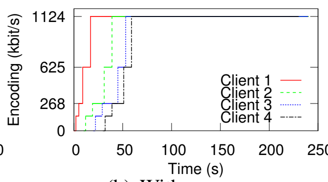


(b) CDF of requested encoding rate

Fig. 13. Comparison of different policies.



(a) No cap



(b) With caps

Fig. 14. Encoding rates in simulated scenario.

when the buffer falls below 10 seconds, the network boosts the client to 2500 kbit/s for 25 seconds, and the client uses the boost-aware adaptation algorithms, when applicable.

Figures 13(a) and 13(b) show the CDFs of the buffer size and the requested encodings observed during the boost period, respectively, for each of the above policies. In addition to the *uncooperative client* and *no boosting* baselines, we also include baseline results with an *uncooperative client that knows* $C$ and therefore restrains itself from downloading chunks at encoding higher than $\max_l(Q_l|Q_l \leq C)$, but otherwise applies the HAS player's default adaptation algorithm. When interpreting these results it is important to note that the simulations do not capture all the subtle interactions between TCP and (competing) HAS players, and that our fixed-cap validation results (mostly omitted) have shown that the simulator provides conservative estimates of the improvements of fixed rate caps. To support the claim that the differences observed with simulations are conservative, we include simulation results for the requested encoding rates (Figure 14) for the same scenario as our original OSMF experiments (Figure 2). Similar reasons explains the smaller differences between the boost-aware policies observed here vs. testbed. Yet, the simulation validate the relative performance tradeoffs between the policies. For example, while the differences are smaller between the policies than in the more realistic testbed experiments, the relative performance tradeoffs are the same and the largest buffer improvements (e.g., relative to non-boosted clients) are observed with the boost-aware policies.

The boost-aware policies all perform relatively similar, with only smaller variations in their relative tradeoffs. Throughout the 150 viewings per policy, we only observed a single stall per policy. Similar observations can be made from the average buffer conditions as a function of the time since the boost period begun, shown in Figure 15. This figure also shows how *multi-objective quality*, as per design, most quickly builds up the buffer during the initial part of the boost period. Quickly filling a buffer as a safety margin is important, as it can reduce stall duration and speed up recovery. Finally, we note that selecting a large $\gamma$ (compared to $\beta$) with *multi-objective quality*, further stresses the importance to quickly ramp up the buffer size (Figure 16), at the expense of a few extra low-encoding chunks at the start.

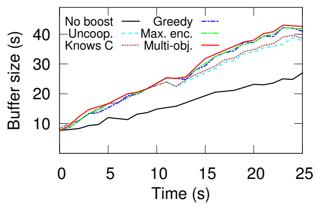


Fig. 15. Average buffer size over time, after boosting.

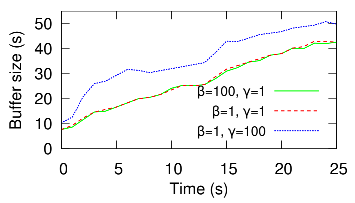


Fig. 16. Buffer size. *Multi-objective quality* with different $\beta$,$\gamma$ ($\alpha$=0.5).

## VII. Related work

Unstable quality and unfair sharing of bottleneck bandwidth are known to affect HAS clients, commonly occurring when HAS flows compete with each other or with background traffic [7], [12]. Prior works include client-, network-, and server-based approaches to address these problems. The *client-based* techniques attempt to address the issue using robust quality adaptation algorithms [3], [23], [11] and better bandwidth estimation [24]. Neither of these fully solve the instability at scale. Network-based and server-based solutions include server-based shaping [7], cooperative proxy-caches [25], [26], network-assisted prioritization [27], and network-assisted quality selection [28]. *Server-based* approaches are complex to implement in today's CDN-based HAS delivery, where edge nodes would need to maintain shaping profiles for many clients, since they typically serve the same content across different networks (Tier 1-3 ISPs) and last mile technologies (WiFi, LTE, Ethernet).

Closely related to our paper are works focusing on *network-side* solutions. The centralized control architecture of SDN is proposed to facilitate dynamic shaping of flows [29], [30], [31], [32]. However, these solutions require converting current carrier networks to an SDN architecture and adding new HAS-specific network elements, which is not realizable in the short term. While solutions implemented in gateway modems and routers can solve issues related to stability and fairness, such solutions do not address competition with other clients in upstream links [12]. Several network-based approaches optimize the rate adaptation across multiple parallel clients [33], [14] or the bandwidth share given to each client [34]. However, they lack experimental evaluation with real clients and live network deployments. While some of them suggest bandwidth shaping at cellular base stations [14], [34], the actual impact on real traffic is unknown. The practicality of such approach is also very low, since current standards do not include visibility into transport or higher layers by cellular base stations, nor do current implementations have such capabilities.

In contrast to these works, we first measure the impact of simple rate caps on HAS traffic in a real cellular network and report their implications at scale, finding both benefits and drawbacks. To the best of our knowledge we are the first to propose and study the performance of rate boosting in a cooperative cap-based system. Our solution proposes improvements suitable for current traffic management practices, reducing the step to actual deployment, but is also applicable to future design with potential additions of new network elements.

## VIII. Conclusion

This paper proposes client-network cooperative boosting as a framework for improving HAS video streaming under rate caps, as commonly deployed in today's networks. We first conduct a comprehensive study of the rate cap impact on HAS videos, which spans testbed experiments using multiple players, trace-driven simulations, and passive measurements from a real-world test deployment. We identify that the impact of fixed caps in a cellular network is largely positive, with improved playback and buffer stability, as well as data savings, but the drawbacks include slower startup and stall recovery, as well as slow buffer fill. To address the key shortcomings, we develop a cooperative rate-boosting approach, where the network boosts the cap when clients need it. Using experiments and simulations, we demonstrate the increasing benefit of boosting with increased information exchange between the client and network. We also propose a client-side rate adaptation algorithm to optimize the benefit of boosting, which recovers all startup impairments and improves user experience, while eliminating wasted bandwidth and hence reducing cost for both users and the network.

## Acknowledgements

This work was funded in part by the Swedish Research Council (VR) and the Swedish National Graduate School in Computer Science (CUGS) at Linköping University.

## References

[1] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang., "Understanding the impact of video quality on user engagement," in *Proc. ACM SIGCOMM*, 2011.

[2] S. Krishnan and R. Sitaraman, "Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs," in *Proc. IMC*, 2012.

[3] T. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari., "Confused, timid, and unstable: Picking a video streaming rate is hard," in *Proc. ACM IMC*, 2012.

[4] T. Hossfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia, "Identifying QoE optimal adaptation of HTTP adaptive streaming based on subjective studies," *Computer Networks*, 2015.

[5] "VQEG: Objective video quality assessment," 2018. [Online]. Available: https://www.its.bldrdoc.gov/vqeg/projects/audiovisual-hd.aspx

[6] "ITU-T P.1203: Objective video QoE standard," 2018. [Online]. Available: https://www.itu.int/rec/T-REC-P.1203

[7] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen., "Server-based traffic shaping for stabilizing oscillating adaptive streaming players," in *Proc. ACM NOSSDAV*, 2013.

[8] A. M. Kakhki, F. Li., D. Choffnes, A. Mislove, and E. K. Bassett, "BingeOn Under the Microscope: Understanding T-Mobile's Zero-Rating Implementation," in *Proc. Internet-QoE Workshop*, 2016.

[9] T. Flach, P. Papageorge, A. Terzis, L. Pedrosa, Y. Cheng, T. Karim, E. Katz-Bassett, and R. Govindan, "An Internet-Wide Analysis of Traffic Policing," in *Proc. ACM SIGCOMM*, 2016.

[10] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 12–20, 2010.

[11] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: near-optimal bitrate adaptation for online videos," in *Proc. IEEE INFOCOM*, 2016.

[12] R. Houdaille and S. Gouache, "Shaping http adaptive streams for a better user experience," in *Proc. ACM MMSys*, 2012.

[13] A. Rao, A. Legout, Y. Lim, D. Towsley, C. Barakat, and W. Dabbous, "Network characteristics of video streaming traffic," in *Proc. ACM CoNEXT*, 2011.

[14] D. D. Vleeschauwer, H. Viswanathan, A. Beck, S. Benno, G. Li, and R. Miller, "Optimization of http adaptive streaming over mobile cellular networks," in *Proc. IEEE INFOCOM*, 2013.

[15] T. Mangla, E. Halepovic, R. Jana, K. Hwang, M. Platania, M. Ammar, and E. Zegura, "Videonoc: Assessing video qoe for network operators using passive measurements," in *Proc. MMSys (to appear)*, 2018.

[16] "ISO/IEC FDIS 23009-5:2017: Information Technology – Dynamic adaptive streaming over HTTP (DASH) – Part 5: Server and network assisted DASH (SAND)," 2017.

[17] V. Krishnamoorthi, N. Carlsson, E. Halepovic, and E. Petajan, "Buffest: Predicting buffer conditions and real-time requirements of http(s) adaptive streaming clients," in *Proc. ACM MMSys*, 2017.

[18] T. Mangla, E. Halepovic, M. Ammar, and E. Zegura, "MIMIC: Using passive network measurements to estimate HTTP-based adaptive video QoE metrics," in *Proc. IEEE/IFIP TMA*, 2017.

[19] P. Casas, M. Seufert, and R. Schatz, "YOUQMON: A system for online monitoring of YouTube QoE in operational 3G networks," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 41, pp. 44–46, 2013.

[20] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki, "Measuring video QoE from encrypted traffic," in *Proc. IMC*, 2016.

[21] T. Wu, S. Petrangeli, R. Huysegems, T. Bostoen, and F. D. Turck, "Network-based video freeze detection and prediction in HTTP adaptive streaming," *Comp. Comm.*, vol. 99, pp. 37–47, 2017.

[22] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: Analysis and applications," in *Proc. ACM MMSys*, 2013.

[23] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proc ACM SIGCOMM*, 2016.

[24] L. Zhi, Z. Xiaoqing, J. Gahm, P. Rong, H. Hao, A. Begen, and D. Oran, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE Journal on Selected Areas in Communications*, 2014.

[25] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri., "Helping hand or hidden hurdle: Proxy-assisted HTTP-based adaptive streaming performance," in *Proc. IEEE MASCOTS*, 2013.

[26] S. Benno, J. O. Esteban, and I. Rimac., "Adaptive streaming: The network HAS to help," *Bell Lab. Tech. Journal*, 2011.

[27] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. D. Turck, "Software-defined network-based prioritization to avoid video freezes in HTTP adaptive streaming," *Int. Journ. of Netw. Management*, 2016.

[28] N. Bouten, R. de Schmidt, J. Famaey, S. Latre, A. Pras, and F. D. Turck, "QoE-driven in-network optimization for adaptive video streaming based on packet sampling measurements," *Computer Networks*, 2015.

[29] J. Kleinrouweler, S. Cabrero, and P. Cesar, "Delivering stable high-quality video: An SDN architecture with DASH assisting network elements," in *Proc. ACM MMSyS*, 2016.

[30] A. Bentaleb, A. C. Begen, and R. Zimmermann, "SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking," in *Proc. ACM MMSyS*, 2016.

[31] D. Bhat, A. Rizk, M. Zink, and R. Steinmetz, "Network assisted content distribution for adaptive bitrate video streaming," in *Proc. MMSys*, 2017.

[32] G. Cofano, L. D. Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, and S. Mascolo, "Design and experimental evaluation of network-assisted strategies for HTTP adaptive streaming," in *Proc. ACM MMSyS*, 2016.

[33] N. Bouten, S. Latré, J. Famaey, W. Van Leekwijck, and F. De Turck, "In-network quality optimization for adaptive video streaming services," *IEEE Trans. on Multimedia*, vol. 16, no. 8, pp. 2281–2293, 2014.

[34] A. H. Zahran, J. J. Quinlan, K. K. Ramakrishnan, and C. J. Sreenan, "SAP: Stall-aware pacing for improved DASH video experience in cellular networks," in *Proc. ACM MMSys*, 2017.