Exploring the interplay between CDN caching and video streaming performance

Ehab Ghabashneh and Sanjay Rao
Purdue University

***Abstract*—Content Delivery Networks (CDNs) are critical for optimizing Internet video delivery. In this paper, we characterize how CDNs serve video content, and the implications for video performance, especially emerging 4K video streaming. Our work is based on measurements of multiple well known video publishers served by top CDNs. Our results show that (i) video chunks in a session often see heterogeneous behavior in terms of whether they hit in the CDN, and which layer they are served from; and (ii) application throughput can vary significantly even across chunks in a session based on where they are served from. The differences while sensitive to client location and CDN can sometimes be significant enough to impact the viability of 4K streaming. We consider the implications for Adaptive bit rate (ABR) algorithms given they rely on throughput prediction, and are agnostic to whether objects hit in the CDN and where. We evaluate the potential benefits of exposing where a video chunk is served from to the client ABR algorithm in the context of the widely studied model predictive control (MPC) algorithm. Emulation experiments show the approach has the potential to reduce prediction inaccuracies, and enhance video streaming performance.**

I. Introduction

Internet video traffic has grown tremendously over the past decade. According to reports from Cisco, video is projected to account for over 82% of the Internet video traffic in 2022, up from 18% in 2006. Ensuring that video can be delivered to users with high quality is a challenge. With the advent of new technologies such as 4K streaming (involving 40 Mbps or more), these challenges are likely to remain in the future.

A key factor essential to delivering high quality Internet video is the use of Content Delivery Networks (CDNs) that cache content close to the network edge. CDNs themselves are hierarchical with multiple levels of caches, and different latency [22]. While the use of CDNs is pervasive for delivering video, there is fairly limited understanding of the extent to which CDNs successfully serve video content from the network edge, and the impact on client video performance.

In this paper, we take a step towards understanding the interplay between CDNs, and client video performance through end-to-end measurements. Our work is conducted in the context of HTTP-based video streaming, where video content is split into chunks that are a few seconds in duration, and encoded at different bit rates. Our measurements are conducted from well provisioned Internet hosts, and using publicly visible HTTP headers that show the distribution of objects among the various levels in the CDN hierarchy. The experiments involve two popular video publishers, that between them use three different well known CDNs. The measurements include 4K videos given our goal of understanding performance implications for such videos. For each chunk in a video session, our measurements characterize where in the CDN hierarchy (or origin) the chunk is served from, the application-perceived throughput, the chunk size, and the latency incurred before each chunk is received.

Our results show that it is common for chunks in a video session to be served from different layers in the CDN hierarchy or the origin. Further, while there is a higher tendency for chunks at the start of a session to be served from edge locations, even chunks later in the session may be served from multiple locations. Interestingly, some sessions may exhibit significant switching in terms of the location from which consecutive chunks are served.

We next measure the impact of where chunks are served from on video client performance. While the impact on latency is clear, our results indicate the impact on application-perceived throughput is more nuanced, and sensitive to the client location, and CDN. Serving content from the edge offers higher application-perceived throughput when the latency savings in receiving the first byte are significant, when certain optimizations for large file transfer are not enabled by the CDN, or when the transfer is limited by the throughput from higher levels of the CDN hierarchy (or origin) to the edge. Further, for higher bit rate video, the differences in throughput may be significant enough to impact the video streaming experience.

An implication of our findings is that when streaming high bit rate video, the throughput perceived by the client may depend on where in the CDN hierarchy the object is served from. While several Adaptive Bit Rate (ABR) algorithms for video streaming have emerged [29], [37], [25], a challenge common to many of them is the need to predict throughput. Errors in predicting throughput can lead to poor bit rate adaptation decisions. ABR algorithms today are agnostic as to whether an object hits or misses in the CDN, and where in the hierarchy they are served from, which can potentially lead to prediction inaccuracies. We evaluate the potential benefits of CDN-aware schemes that expose information about where the next video chunk is served from to the client by suitably altering MPC, a representative, and well studied ABR algorithm [37]. Experiments on an emulation testbed show that incorporating CDN awareness (i) reduces prediction error for 81.7% of sessions by 17.16% on average; and improves a composite video delivery metric by more than 10% for 20% of the sessions.

Overall, our results lead to new understanding of the interplay between CDN caching and Internet video performance, and new insights on how to optimize video performance by exploiting

such knowledge.

II. Methodology and Data collection

Background. Much Internet video delivery today involves chunk-based streaming over HTTP. A video is split into chunks (typically a few seconds in duration), and encoded at multiple bit rates. Client video players select which bit rates to access each chunk at using Adaptive Bit Rate (ABR) algorithms [29], [37], [25] in a manner that considers the amount of video content in the client buffer, and predictions regarding network bandwidth. Each video chunk is requested using an HTTP request and response interaction.

Video content is typically served from Content Delivery Networks (CDNs). The CDNs themselves are typically organized as a hierarchy of caches [22]. Requests from the user go to a CDN edge server. The edge server directly serves the request if the object is in its cache. Alternately, it may forward the request to a CDN server higher in the hierarchy, and ultimately to the origin. The fetched object is then streamed to the client.

The performance of video streaming applications critically depends on the application throughput, (or the throughput perceived by the video player), which we define more precisely later. The application throughput itself may depend on whether a chunk is served from the CDN, or the origin, and which layer of the CDN the chunk is served from. To sustain video at a particular bit rate, the application throughput must be sufficiently high. Further, application throughput perceived by chunks downloaded in the past is used by ABR algorithms to predict throughput for future chunks.

Motivation. Our work is motivated by several questions:

• Where in a CDN hierarchy are objects in a video session served from? Is the treatment of video chunks with a session homogeneous, or are there variations within a session? How do these policies vary across CDNs?

• What is the impact of where objects are served from by a CDN on video streaming performance?

• Given that higher bit rate video such as 4K video content is starting to emerge, what are the implications of CDN architecture on 4K video performance?

Measurement methodology. To explore these questions, we conducted measurement studies by streaming videos from 2 different popular video publishers (Twitch and Vimeo) from 4 different locations in the United States. We chose these publishers because (i) they are popular (within Alexa top 100 US rank [16]); (ii) they provide videos that can be viewed without subscription fees, yet do not disable CDN caching; and (iii) they use CDNs such as Akamai [1], Fastly [9] and CloudFront [2] that support special HTTP headers which allow us to identify which layer in the hierarchy a video chunk was a hit (§II-A). All client locations were well provisioned network environments, including 2 home locations, one university site, and another business location, and corresponded to 3 different geographical regions. We refer to these locations as USHome1, USHome2, USUniv, and USBus respectively.

We collected two sets of data:

• The ABR-Set, which involved downloading videos with the client running ABR algorithm as normal. Both publishers (Twitch and Vimeo) have a category for popular/trending videos, and we picked our videos from this category randomly.

• The 4K-Set, where we selected videos encoded at 4K rates, and disabled client ABR, so only 4K video was streamed. We conducted such experiments for two reasons. First, we wished to explore the sustainability of streaming 4K video in typical well provisioned client network environments today. Second, some of our performance measures (especially application perceived throughput) are sensitive to chunk size as we will discuss later. Disabling ABR and focusing on 4K allow us to minimize the impact (though not completely eliminate chunk size variation given the variable bit rate encoding used). We focused this data collection on Vimeo as Twitch does not stream 4K videos. We picked the videos by filtering Vimeo to list only 4K quality videos, then we picked the ones with high number of views.

All downloaded videos in both data-sets were between 5 and 15 minutes long. The ABR-Set included data about 104 videos downloaded from Twitch, and 50 from Vimeo, from USHome1. The 4K-Set included 100 video sessions from USHome1, 70 sessions from USUniv, 44 sessions from USBus, and 24 sessions from USHome2. The videos from USHome1 were downloaded over a 7 day period in October 2018, while data was collected from other locations in July 2019.

Measurement setup and data collected. We conduct our study by automating Google Chrome to run and play real videos from both video publishers. Each video is downloaded as a series of HTTP request response interactions. We collected the HTTP headers associated with the responses which provided valuable information for our analysis. We collect headers using the browsermob proxy [5], an open source proxy, and binding it to Chrome using the '–proxy-server' flag. The proxy monitors all HTTP requests and responses and archives relevant header fields when the 'captureHeaders' option is set to 'True'. Table I summarizes the header fields most relevant to our measurement. As the proxy tracks all HTML objects, we use the Content-Type header to distinguish video chunks from other objects. The video publishers in our study typically use "video/MP2T" or "video/mp4" for the Content-Type header.

Field Name	Definition
startedDateTime	The requests' issue time
wait time (TTFB)	Time from sending the request until first byte of response received
receive time	Time from receiving the first byte of the response until the last byte
Content-Type	Type of the HTTP object
Content-Length	Total response's length
X-Cache	Headers added by CDN (more details later)
X-Cache-Remote	

Table I: Headers fields retrieved by browsermob proxy.

CDN	Required header
Akamai	pragma:akamai-x-cache-on pragma:akamai-x-cache-remote-on
Fastly	Fastly-Debug:1
CloudFront	No header required

Table II: Required HTTP header for each CDN.

A. Classifying where objects are served from

A key part of our measurement methodology is classifying a video chunk based on which CDN layer the object is served from. To achieve this, we exploit the fact that many CDNs support HTTP headers for debugging purposes. Specifically, we add special headers in the outgoing HTTP requests sent (summarized in Table II), and interpret the values of the appropriate headers in the HTTP responses (summarized in Table III). We elaborate for each CDN below:

CloudFront. For the publishers we considered, every HTTP response for an object served from CloudFront includes an X-Cache header. We classify an object as being served from a Cloudfront server if the header has a value "Hit from cloudfront", and from an origin server if the value is "Miss from cloudfront" as summarized in Table III following [14].

Fastly. We add the "Fastly-Debug:1" header to all HTTP requests to identify whether the request was served from Fastly or not [8], [15]. A response that contains "Fastly-Debug-Path" or "Fastly-Debug-TTL" indicates that the object was served from Fastly. Further, a HTTP response in such a case has an X-Cache header which may take 6 different values as shown in Table III. A response with a single HIT/MISS indicates a single Fastly CDN layer (called Shield) was encountered. A response with two such values indicates two Fastly CDN layers were encountered (referred to as the Fastly Edge, and the Fastly Shield). For example, a "HIT, MISS" indicates that the object missed in the Fastly Edge and hit in Fastly Shield, whereas a "MISS, MISS" indicates the object missed at both layers. Note that the Edge retains the last cache status of Shield for any request it forwarded to Shield. For example, "MISS, HIT" indicates that the Edge served the request, with the first MISS corresponding to a stale cache status for the Shield obtained in a previous request. The various cases and the resulting classification are shown in Table III, based on [15].

Akamai. We add the pragmas "akamai-x-cache-on" and "akamai-x-cache-remote-on" to all HTTP requests [17]. The first contacted CDN server appends an X-Cache header to its HTTP response. If a second server is involved, it appends an X-Cache-Remote header. If the X-Cache header has a value of "TCP_HIT" or "TCP_MEM_HIT", it indicates the object was a hit at first edge server (Akamai L1). If it has a "TCP_MISS", and the X-Cache-Remote value contains "HIT", then object was hit at second edge (Akamai L2). If both headers have a value of "TCP_MISS", it is possible that the object was served by a further CDN server, or the origin server. In either case we classify the object as being served from origin.

The response headers might also have values that indicate a refresh hit. For instance, with Cloudfront, in a small number of cases, the response header had a value "RefreshHit from cloudfront". Here, although the object hit in the Cloudfront cache, the Cloudfront server confirmed with the origin server that the object is not stale before serving the object. We classify such an object as served by Cloudfront. We adopted a similar approach with other CDNs.

CDN	HTTP response headers	Value	Server
Akamai	X-Cache X-Cache-Remote	TCP_HIT or TCP_MEM_HIT (Any response)	Akamai L1 Akamai L2
		TCP_MISS (All responses)	Origin
Fastly	X-Cache	HIT,HIT	Edge
		MISS,HIT	
		HIT	Shield
		HIT,MISS	
		MISS	Origin
		MISS,MISS	
CloudFront	X-Cache	Hit from cloudfront	Cloudfront
		Miss from cloudfront	Origin

Table III: HTTP response and its corresponding value. The last column shows our classification of where the object is served from.

B. Performance metrics

We characterize the performance obtained when objects are served from different layers using the following metrics:

• *Time to first byte (TTFB):* This is the time from when a HTTP request for a chunk is made to when the first byte of a response is received and is directly collected for each chunk using the HTTP headers (Table I).

• *Network throughput:* This is measured as the ratio of the chunk size to the receive time, where the receive time is the time from when the first byte is received to when the last byte is received. Both the chunk size and the receive time are obtained from the HTTP response headers (Table I).

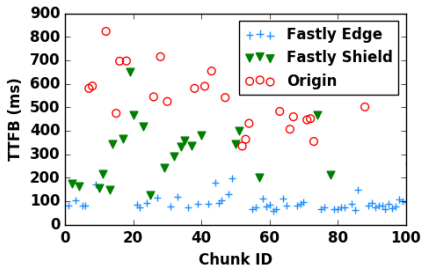
• *Application throughput:* This refers to the throughput perceived by the video player, which includes the impact of both TTFB, and network throughput. Given a chunk of size S_i, with TTFB T_i, and receive time R_i, the application throughput is written as $\frac{S_i}{R_i+T_i}$. If the network throughput is denoted by N_i, then, the application throughput may be equivalently written as $\frac{1}{1/N_i+T_i/S_i}$. The application throughput is the primary metric that the video player cares about. When the chunk size is large, and the TTFB T_i is small, it is close to the network throughput, but lower otherwise.

III. Measurement Results

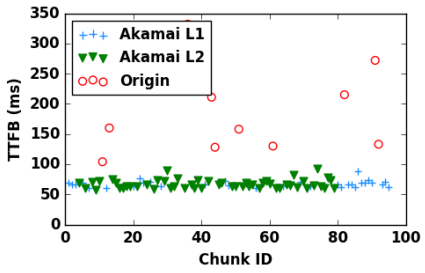
We analyze where video chunks in a session are served from (the origin, or a particular CDN layer), and the impact on video streaming performance. We begin by presenting results for data collected from USHome1, and the ABR-Set. We then present results for the 4K-Set, and for other US locations.

A. Where are objects served from

For the primary location considered, we observed that the video sessions from Vimeo either used Fastly, or Akamai as a CDN. Figure 1(a) presents an example video session from Vimeo served by Fastly. The X-Axis represents different chunks in the session, and the Y-Axis represents the TTFB of that chunk. The different colors and shapes correspond to the object served from different locations (Fastly Edge, Fastly Shield, or the origin). The figure shows that the chunks could be served from any of the three locations, and the locations are fairly interspersed. Further, the TTFB of the chunks from the Edge is



(a) Vimeo Fastly.



(b) Vimeo Akamai.

Figure 1: Example video sessions served from origin and different CDN layers.

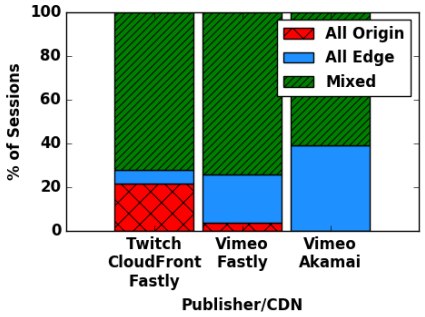
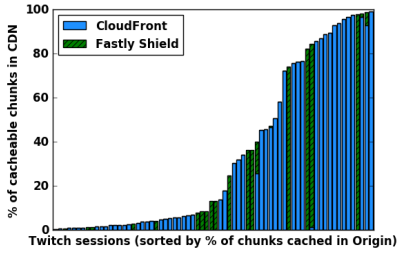
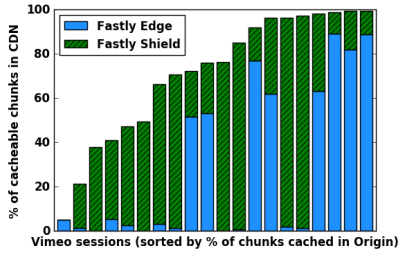


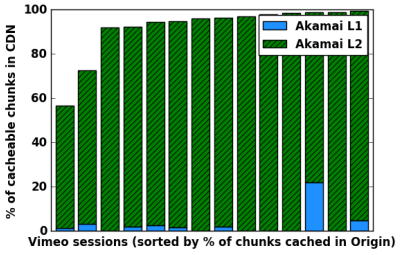
Figure 2: Percentage of sessions that are served fully by edge, origin, or mixed.



(a) Twitch sessions served by CloudFront/ Fastly CDN (merged for convenience).



(b) Vimeo sessions served by Fastly CDN.



(c) Vimeo sessions served by Akamai CDN.

Figure 3: Percentage of cached video chunks per publisher per CDN.

much smaller than Shield, and in turn smaller than the origin. Figure 1(b) shows similar results for a session from Akamai. Similar trends hold – chunks are served from different layers, and the layers exhibit different TTFB.

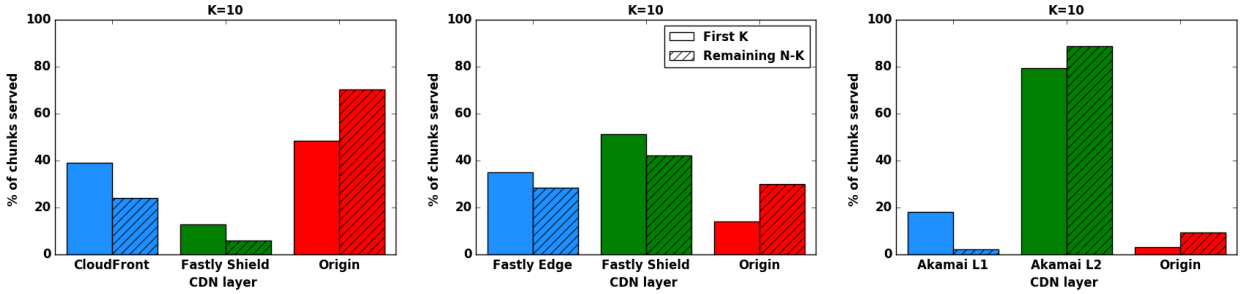
With Twitch, we observed that the video sessions either used Cloudfront or Fastly, though only a single level of each CDN. For a small number of sessions (7.14%), different chunks in the session used different CDNs.

Figure 2 presents a classification of sessions for each publisher and CDN combination. The sessions are classified as (i) All Origin, if all chunks in a session are served by the origin; (ii) All Edge, if all chunks are served by a CDN layer (irrespective of which layer); and (iii) Mixed, if some chunks are served by the origin, and others by the CDN. The figure shows that 72% of Twitch sessions, 74% of Vimeo Fastly (FS), and 61% of Vimeo Akamai (AK) sessions are mixed. This indicates that a majority of sessions involve chunks served by both the origin, and the CDN. A smaller portion of the sessions are served entirely by the origin, or entirely by the CDN. In the rest of the analysis we ignore these sessions, and focus on the mixed sessions.

Figure 3 analyzes the mixed sessions (i.e., sessions with chunks served from the edge and the origin) further, and decomposes sessions based on which CDN layer the objects are served from. Each bar corresponds to a session, and indicates the percentage of chunks within a session that are served by a CDN layer. Note that each bar does not add up to 100% since this represents objects served from the origin. For instance, the right most bar of Figure 3(b) indicates that for that session, 88.8% of chunks are served from Fastly Edge (blue), 10.6% are served from Fastly Shield (green), and the rest from origin.

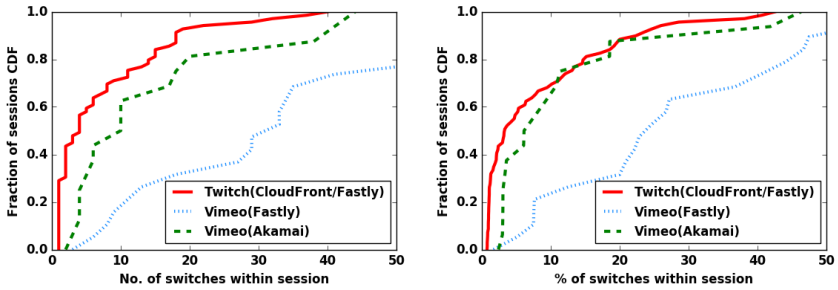
For Vimeo and Fastly (middle graph), 80% of the mixed sessions have objects served by the origin, and the two CDN layers (Fastly Edge and Shield). Further inspection shows that on average 28.74%, 29.41%, and 41.85% of the chunks of these sessions are served from the origin, Fastly Edge, and Fastly Shield respectively. For Vimeo and Akamai (right most) graph, 57.14% of the sessions have objects served by the origin and two different CDN layers. The vast majority of chunks are served by Akamai L2 (89%), with 8.24% served from the origin, and 2.74% served from Akamai L1. For Twitch, we combine sessions served by Fastly, and Cloudfront in the same graph for convenience, and observe that most sessions see a mix of CDN (Fastly or Cloudfront), and origin.

Session start vs. later phases. We next consider plausible explanations for why chunks within the same video session are served from different layers in the hierarchy. One potential hypothesis is that chunks earlier in the session are served from layers closer to the user. This is because users are known to watch the first few minutes of a video, before abandoning it for a newer video. Thus, it is possible that initial chunks are more popular and cached closer to the users. To explore this further, we split all chunks in all sessions into two categories: (i) the first K chunks; and (ii) the remaining chunks. We choose $K = 10$ which corresponds to several tens of seconds, which corresponds to the time-frames that a user may watch a video before moving on (33% of users abandon video sessions within 30 seconds if not interested [33]). Figure 4(a) shows the fraction of chunks that are served from each CDN layer and the origin for the first K, and the next $N - K$ chunks for



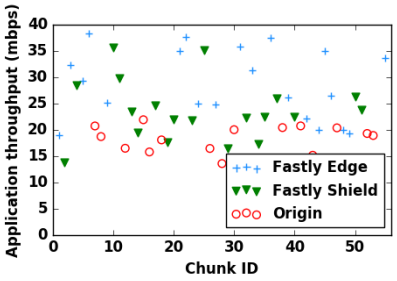
(a) Twitch: CloudFront/Fastly. (b) Vimeo: Fastly. (c) Vimeo: Akamai.

Figure 4: Percentage of First-K, and N-K chunks served by CDN layer.



(a) Number of switches per session. (b) Percentage of switches per session.

Figure 5: Percentage and number of chunks that experience a location switch for different CDN and publisher combination.



(a) Vimeo Fastly.

Figure 6: Difference in application throughput based on serving location for an example session.

Twitch served from CloudFront and Fastly. The results show that while the CDN (CloudFront and Fastly Shield) account for a larger fraction of the first K chunks (e.g., Cloudfront accounts for 39% of the first K chunks but only 24% of the remaining chunks), the origin serves a larger fraction of the last $N-K$ chunks. Similar results are seen in Figure 4(b) for Vimeo sessions served through Fastly. The results are even more striking for Vimeo sessions served by Akamai (Figure 4(c)) – the Akamai L1 cache accounts for nearly 20% of the first K chunks, but only 2% of the remaining chunks, though the fraction of chunks served by the Akamai L2 cache, and the origin increases for the rest of the session compared to the initial chunks. While the above results indicate that the initial phases see more requests from the edge, they also indicate that a substantial fraction of chunks even in the later part of the session are served from different locations.

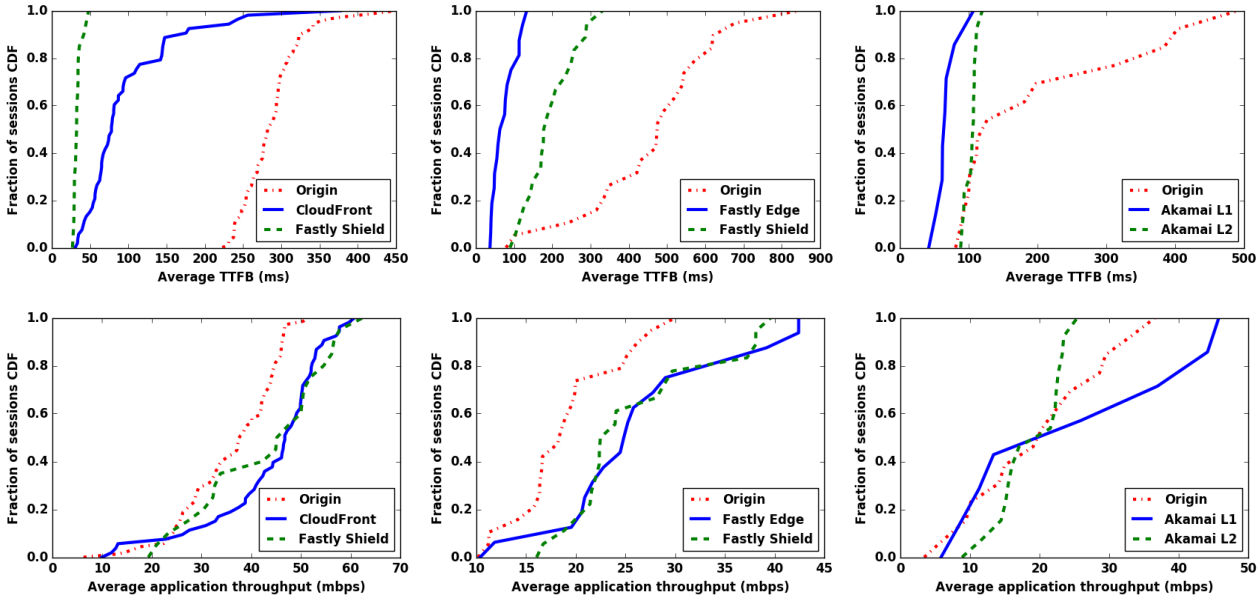
How often are consecutive chunks served from different locations? We next consider whether consecutive chunks are typically served from the same location, or if there is often a variation in the location from which consecutive chunks are served. To analyze this, we consider each instance of chunk i being served from a different location as chunk $i-1$ (e.g., for Vimeo and Akamai, the locations are Akamai L1, L2 and origin) as a *serving location switch*. Figure 5(a) shows the number of serving location switches in the session for different publisher and CDN combinations, while Figure 5(b) shows the percentage of chunks in a session that experienced a serving location switch. Figure 5(a) shows that for Twitch (Cloudfront/Fastly) and Vimeo (Akamai) 20% of sessions had more than 14 switches, while for Vimeo (Fastly) 72% of session experienced at least 14 switches. Figure 5(b) shows that for Twitch (Cloudfront/Fastly), and Vimeo (Akamai), about 25% of the sessions see more than 11% of the chunks experience a serving location switch. In contrast, for Vimeo (Fastly), nearly 80% of the sessions experience a serving location switch, and for the median session, 23.71% of chunks experience a serving location switch. Regardless, the number of switches is significant in many sessions, and this has implications for the performance of ABR algorithms, as we will see in §IV.

B. Impact on performance

So far, we have seen that objects within a video session may be served from different CDN layers or the origin. We next consider the impact on performance.

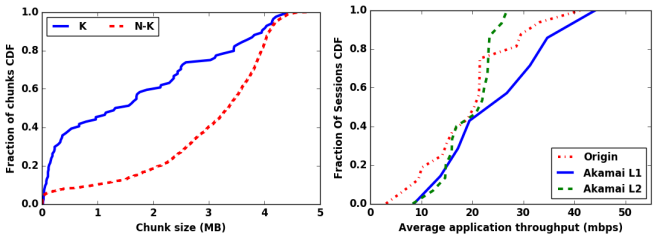
Figure 6 presents an example of a video session from Vimeo served by Fastly. The X-Axis represents different chunks in the session, and the Y-Axis represents the application throughput of that chunk. The figure shows that the application throughput for each chunk varies significantly depending on where it is served from, with the average application throughput for chunks served from the origin, Fastly Shield, and Fastly Edge being 18.04 Mbps, 23.57 Mbps, and 28.64 Mbps respectively.

We next study performance using data collected across



(a) Twitch: CloudFront/Fastly. (b) Vimeo:Fastly. (c) Vimeo:Akamai.

Figure 7: Average TTFB and application throughput per publisher per CDN.



(a) Chunk size of first K and remaining N-K chunks (K=10).

(b) Average application throughput excluding first K chunks of each session.

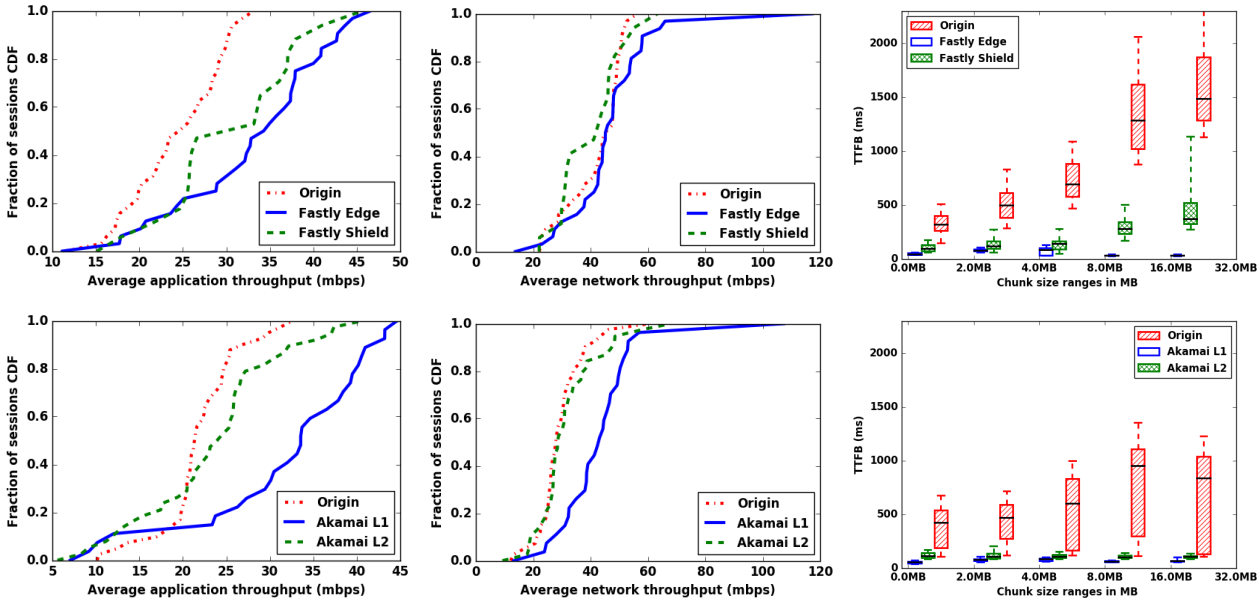
Figure 8: Vimeo Akamai. Explaining impact of initial chunks on application throughput.

sessions. For each video session, and for each layer, we computed the average TTFB and application throughput of all chunks served by that layer. The top row of Figure 7 shows the cumulative distribution of the average TTFB across sessions with each graph corresponding to a different combination of publisher and CDN, and each curve corresponding to objects served from a given layer. The figure is as expected, with TTFB increasing for layers further from the users, and the highest TTFB for the origin.

The bottom row of Figure 7 presents the distribution of average application throughput across sessions for chunks served by each layer. The figures show that the average application throughput of objects served from the CDN layer is generally higher than the throughput of objects served from the origin. An exception is Vimeo and Akamai (bottom right). To explain the discrepancy, note that when chunk sizes are small, application throughput tends to be biased to be small. Recall from Figure 4(c) that the Akamai L1 cache tends to serve objects predominantly in the start of the session. These chunks tends to be of smaller size since they are downloaded during the slow start phase. To illustrate this further, consider Figure 8(a) which shows a CDF of the sizes of the first K chunks across all sessions, and the remaining $N - K$ chunks. The initial chunks are clearly smaller (with a median size of 1.32 MB), while the median size of the remaining chunks is 3.35MB. Figure 8(b) shows the average application throughput after eliminating the impact of the first 10 chunks for Vimeo and Akamai. Figure 8(b) shows clear improvement in throughput compared to Figure 7(c) for the L1 layer.

Impact on 4K streaming. While the results above were using the ABR-Set, we next present results with the 4K-Set to understand the sustainability of streaming 4K video (§II). Figure 9 summarizes results for sessions corresponding to the 4K-Set from USHome1. The sessions were split nearly evenly between Fastly and Akamai with 79% and 89.3% of their sessions being mixed (objects served from both CDN and origin) respectively. Figure 9(a) shows that while the throughput from the origin is too low to sustain 4K rates, the application throughput is clearly better for CDN layers closer to the user with both Fastly and Akamai. Further inspection indicated the contributing factors were different for the two CDNs. Specifically, we noticed that the TTFB values were really high with Fastly, and much lower with Akamai. Figure 9(c) explores this further by showing boxplots that depict the variation of TTFB for chunks in different size ranges, The figure shows that for Fastly, the TTFB grows significantly with larger chunk sizes for both the origin (with a median exceeding 1.5 seconds for the largest sizes), and Fastly Shield (with a median of about 400 ms for the largest sizes). In contrast, for Akamai, the TTFB does not grow with chunk size for L2, and grows more slowly for the origin.

We hypothesize that this is because an optimization for large

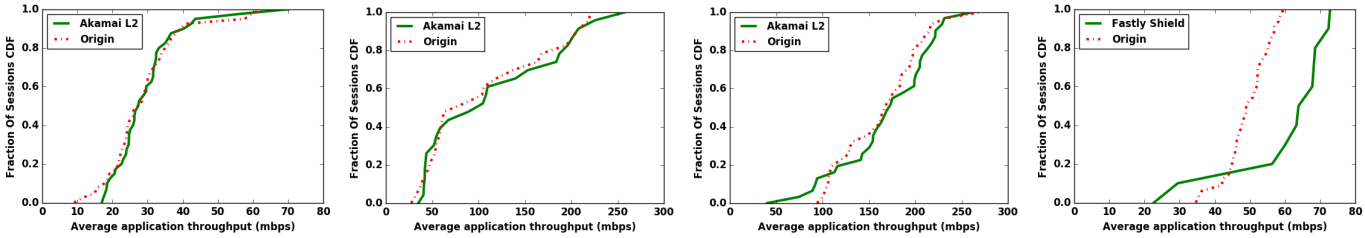


(a) Average application throughput of sessions for each CDN.

(b) Average network throughput of sessions for each CDN.

(c) Distribution of TTFB over different ranges of chunk sizes for each CDN.

Figure 9: Performance with 4K-Set collected from USHome1.



(a) USBus: Akamai.

(b) USHome2: Akamai.

(c) USUniv: Akamai.

(d) USUniv: Fastly.

Figure 10: Performance with 4K-Set collected from other locations. Each graph shows a CDF of the average application throughput across sessions for each location and CDN combination.

file transfer was not turned on with Vimeo and Fastly. As per Fastly documentation [10], a feature called streaming miss must be explicitly configured to ensure that when fetching an object from the origin, the response is streamed back to the client immediately without waiting for the full object to arrive at the edge to reduce first-byte latency. In contrast, we believe an equivalent optimization had been enabled for Akamai.

As further evidence, Figure 9(b) plots the network throughput (which only considers time from first to last byte and does not include TTFB effects, see §II-B). Consider that the streaming miss optimization is turned off. Then, irrespective of whether the object missed entirely in the CDN, or which layer it hit in, we would expect the network throughput to be N_{EU} which denotes the throughput from the edge server to the user. This is because the object must be completely shipped to the edge before being sent to the user. In contrast, if the streaming miss optimization were turned on, and an object hit in a higher layer C, rather than the edge E, we would expect the network throughput seen by the client to be $\min(N_{CE}, N_{EU})$ where N_{CE} is the throughput from the higher layer server to the edge. Indeed, Figure 9(b) shows that the network throughput is similar for all layers with Fastly, but Akamai L1 has distinctly higher network throughput than other layers.

Figure 10 presents results with the 4K-Set for other client locations. All sessions collected from USBus and USHome2 were served by Akamai, while the sessions from USUniv were split equally between Akamai and Fastly. In all cases, practically all sessions had chunks served by both the origin and the CDN. Figure 10 shows the average application throughput of sessions for all the CDN and location combinations. We omit Akamai L1 and Fastly Edge since we observed they primarily served the first few chunks of a session. First, for the only Fastly case, Figure 10(d) shows that Fastly Shield has significantly higher average application throughput than the origin for nearly 80% of sessions which we confirmed is due to high origin TTFB for similar reasons described above. In the remaining cases, Akamai L2 performs slightly better than the origin for USHome2 and USUniv, though almost

indistinguishable for USBus. Further inspection shows that with Akamai, the origin TTFB is small even for the largest file sizes indicating an optimization similar to streaming miss is in place. We hypothesize the limited benefits are because the network throughput from the origin to the CDN is comparable to the network throughput from the CDN edge to the user. Overall, these results show that CDN caching does not always result in a win in throughput (since the performance from the cache server to the client may be the limiting factor), and the benefits are sensitive to client location and CDN.

IV. Implications for ABR algorithms

Our results so far have shown that video chunks within a session may be served from different CDN layers, or the origin, with the potential for different application throughput. Many ABR algorithms [29], [37], [25] use a prediction of future throughput in guiding their bit rate decisions, with the prediction itself informed by past throughput observations. Yet, current ABR algorithms are agnostic as to whether a chunk hits or misses in the CDN, or which CDN layer it is served from. In this section, we evaluate the potential improvements in prediction accuracies, and ABR algorithm performance if CDN awareness is incorporated in ABR algorithms. We conduct this study with RobustMPC [37] (that we henceforth refer to as just MPC), a representative and well-known ABR algorithm.

MPC background. MPC selects bit rates so as to optimize a metric referred to as `QoE-linear` (defined later) over a look-ahead window of the next k chunks ($k = 5$), while taking current client buffer occupancy, and predicted network throughput into account. MPC predicts the future throughput by calculating the harmonic mean of the application throughput observed over the past m (m=5 by default), and discounts the prediction by the maximum prediction error seen over those chunks.

Incorporating CDN awareness in MPC. MPC like most ABR algorithms is agnostic of whether chunks are served from the origin, or the CDN, which could lead to inaccuracies in throughput prediction. We modified MPC to separately maintain the past throughput and prediction errors for chunks served from the CDN and origin. Second, we include a hint that indicates whether the next chunk is served from the CDN or the origin, which results in different throughput predictions. We consider a two-level scheme for simplicity, though it is possible to maintain separate estimates for each CDN layer. We henceforth refer to this scheme as MPC-CDN. We only consider a hint regarding the immediate next chunk, though better accuracies can potentially be obtained if hints were provided regarding the next k chunks. In practice, such hints can be provided by the CDN as part of the metadata when a previous chunk is served.

Performance metrics. In delivering video, it is desirable to ensure the average bit rates of chunks is high, keep the rebuffering seen by clients low, and minimize variation in bit rates across chunks. We measure the performance of a video session using the `QoE-linear` metric (proposed in [37]), which is a linear combination of the above metrics:

$$\texttt{QoE-linear} = \sum_{k=1}^{N} B_k - \tau \sum_{k=1}^{N} T_k - \delta \sum_{k=1}^{N-1} |B_{k+1} - B_k| \quad (1)$$

Here, N is the number of chunks in the video, B_i is the bit rate selected for chunk i (in Mbps), T_i is the rebuffering time associated with chunk i. Further, τ and δ are the penalties associated with rebuffering and bit rate variation respectively. We used $\tau = 52.142$ which indicates that a rebuffering time of 1 second would incur as much penalty as the reward obtained if a chunk were downloaded at the maximum bit rate in mbps (following previous works [29], [37], [20]), and $\delta = 1$ (default value).

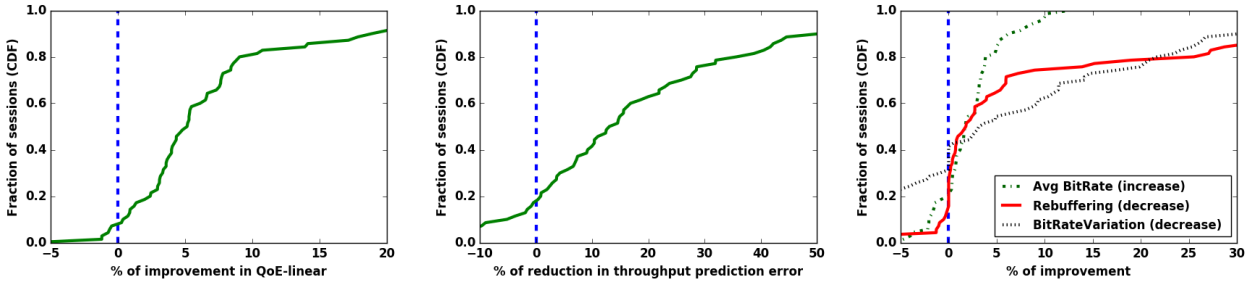
Implementation and evaluation setup. We incorporated our changes in the MPC source code distributed with [29]. We used Chromium (Google's open-source web browser [6]) and dash.js (version 2.4) as the video player [7]. By default, dash.js sends all its HTTP requests to a single server. We modified the dash.js code to send requests to multiple servers depending on where the chunk is served from (CDN or origin). We used a buffer size of 60 seconds, and the "Big Buck Bunny" video [4] which has a total length of 635 seconds. We used ffmpeg [11] to encode the video into the following bit rates [9194,15501,25243,52142] kbps which correspond to [720p, 1080p, 1440p, 2160p] video qualities following Google's encoding recommendations [18]. We used MP4BOX [13] to split the encoded videos to 4-second chunks, resulting in 159 chunks. To handle large 2160p (4K) chunks, we increased the default video memory limit (default of 150MB) in Chromium's code [30], and rebuilt it.

We used 71 traces where the percentage of origin-served chunks in a session is between 20% and 80% since these traces have a reasonable number of chunks served from both the CDN and the origin. We created two separate traces, one each for the CDN and the origin. Each trace has the throughput of the corresponding chunks (obtained from our measurements in §III). We interpolate the throughput at any intermediate point in time with the average of the throughput of the previous and next chunks served from the appropriate location.

We used Mahimahi [31] as our network emulator. We used the node.js http-server [12] to run both video servers (CDN and origin). We run two mahimahi shells, one to emulate the CDN trace, and the other emulated the origin trace. MPC-CDN reads a hint map at the start which indicates whether each chunk is served from the origin or the CDN. For experiments with each trace, we use as many chunks as in the trace, or in our video, whichever is smaller.

Our experiments were run using a gaming laptop with 3.5GHz quad-core processor, 16GB of RAM, and NVIDIA GeForce GTX 960M GPU, sufficient to play multiple 4K videos without stuttering. We run the Chromium client, the ABR server (MPC-CDN or MPC), and two HTTP servers (CDN and origin) on the same machine.

Results. Figure 11(b) shows a CDF of the reduction in the throughput prediction error across sessions with MPC-CDN relative to MPC. For each scheme and each session, the



(a) Improvement in `QoE-linear`. Points to the right of the Y-Axis indicates improvement, and to the left indicates a reduction.

(b) Reduction in throughput prediction error.

(c) Increase in average bit rate , and decrease in rebuffering and bit rate variation.

Figure 11: Emulation results showing the benefits of incorporating CDN-awareness in MPC.

throughput prediction error is the mean of the (normalized) prediction error of each chunk in the session. Figure 11(b) shows that MPC-CDN reduced the throughput prediction error for 81.7% of sessions (points right of the Y-Axis) with 50% of the sessions seeing a reduction in error of at least 13.1%, and 20% of the sessions seeing a reduction of at least 35.7%.

Figure 11(a) shows a CDF of the % increase in `QoE-linear` with MPC-CDN relative to MPC. `QoE-linear` improved for 91.5% of sessions, with an improvement of at least 10% for 20% of the sessions. Figure 11(c) shows the benefits for the individual video delivery metrics, i.e., the increase in average bit rate , the reduction in rebuffering ratio, and the reduction in bit rate variation. For instance, 73.23% of sessions experienced a reduction in rebuffering, with the reduction exceeding 25.5% for 20% of the sessions. While some sessions did see an increase in rebuffering, the increase was modest, and less than 5% for all sessions. Overall these results show the potential benefits of incorporating CDN awareness.

V. Related Work

Researchers [32], [36], [19], [28], [24], [21] have proposed new caching policies to improve the hit rates at edge, and eventually improve video QoE. In contrast our work treats caching policy as blackbox, and sheds light on how the chunks are served from existing CDNs using an end-to-end measurement approach. We also show that caching at the edge may not always improve application throughput, and the results are sensitive to the CDN and client location. Furthermore, we also redesign ABRs to incorporate CDN awareness.

Prior work [27], [23] showed how centralized control planes can allow clients to dynamically switch CDNs to improve video performance. Others [26] pointed out the potential for oscillations if lower video bitrate are cached, and higher bitrates are served from the origin, and suggested throttling bandwidth from the cache to prevent oscillations. In contrast, we conduct measurements to uncover real-world evidence that chunks in the same session could be served from different CDN layers (or origin), and find more complex patterns (e.g., chunks of the same bitrate could be served from different locations). Further, throttling at the cache [26] forgoes the opportunity of obtaining better video quality. We instead look at exposing information to the client to improve ABR algorithm performance.

A similar methodology as ours was used to identify where objects in a web page were served from for one CDN [34]. Our methodology generalizes to multiple CDNs, and our focus is on video streaming. Researchers [35] have developed better throughput prediction approaches for video streaming based on Hidden Markov Models. However, these models do not separate the past throughput measures of the origin and various CDN layers which can improve the prediction accuracy. Finally, others (e.g., [3]) have explored coordination for bulk transfer traffic between ISPs and CDNs. In contrast, we look at how exposing information from the CDN to the client can improve ABR algorithm performance.

VI. Conclusions

In this paper, we have presented a methodology for analyzing in an end-to-end fashion how chunks in a video session are served by a CDN, and the implications for video streaming performance, especially emerging 4K video streaming. Our measurements of popular video publishers and three different CDNs show that chunks of the same session are often served by different CDN layers, or miss altogether in the CDN. Further, the differences in application throughput based on where chunks are served from can differ significantly enough to impact whether 4K streaming can be sustained even in well provisioned network environments. However, the extent of differences is sensitive to client location, and CDN. We considered the implication of our findings on ABR algorithms. Evaluations on an emulation testbed show that incorporating CDN awareness in ABR algorithms can lead to better throughput prediction accuracies and ABR algorithm performance – e.g., reducing rebuffering ratios by over 25.5% for 20% of the video sessions.

Acknowledgments. We thank Humberto La Roche for his helpful feedback throughout this project. This work is supported in part by an NSF ICE-T:RC Award (Number 1836889), and a Cisco Research Award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or Cisco.

References

[1] Akamai. https://www.akamai.com/.

[2] Amazon cloudfront. https://aws.amazon.com/cloudfront/?nc=sn&loc=0.

[3] Application-layer traffic optimization (ALTO) problem statement. Available at http://tools.ietf.org/html/rfc5693.

[4] Big buck bunny video. https://peach.blender.org/.

[5] Browsermob proxy. https://bmp.lightbody.net/.

[6] Chromium. https://www.chromium.org/Home.

[7] Dash industry forum: Dash.js. http://dashif.org/reference/players/javascript/1.4.0/samples/dash-if-reference-player/.

[8] Deciphering fastly-debug header. https://support.fastly.com/hc/en-us/community/posts/360040167211-Deciphering-Fastly-Debug-header.

[9] Fastly. https://www.fastly.com/.

[10] Fastly:streaming miss. https://docs.fastly.com/guides/performance-tuning/streaming-miss/.

[11] FFmpeg. https://ffmpeg.org/.

[12] http-server: a command-line http server. https://github.com/http-party/http-server#readme.

[13] Mp4box. https://gpac.wp.imt.fr/mp4box/.

[14] Reduce CloudFront Latency "X-Cache: Miss from cloudfront. https://aws.amazon.com/premiumsupport/knowledge-center/cloudfront-latency-xcache/.

[15] Understanding cache hit and miss headers with shielded service. https://docs.fastly.com/guides/performance-tuning/understanding-cache-hit-and-miss-headers-with-shielded-services.

[16] US Alexa Rank. https://www.alexa.com/topsites/countries/US.

[17] Using akamai pragma headers to investigate or troubleshoot akamai content delivery. https://community.akamai.com/customers/s/article/Using-Akamai-Pragma-headers-to-investigate-or-troubleshoot-Akamai-content-delivery?language=en_US.

[18] Youtube recommended upload encoding settings. https://support.google.com/youtube/answer/1722171?hl=en.

[19] Hasti Ahlehagh and Sujit Dey. Video-aware scheduling and caching in the radio access network. *IEEE/ACM Transactions on Networking (TON)*, 22(5):1444–1462, 2014.

[20] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: Auto-tuning video abr algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 44–58, New York, NY, USA, 2018. ACM.

[21] Divyashri Bhat, Amr Rizk, Michael Zink, and Ralf Steinmetz. Network assisted content distribution for adaptive bitrate video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 62–75. ACM, 2017.

[22] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical internet object cache. In *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, ATEC '96, pages 13–13, Berkeley, CA, USA, 1996. USENIX Association.

[23] Aditya Ganjam, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar, and Hui Zhang. C3: Internet-scale control plane for video quality optimization. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 131–144, Oakland, CA, May 2015. USENIX Association.

[24] Chang Ge, Ning Wang, Severin Skillman, Gerry Foster, and Yue Cao. Qoe-driven dash video caching and adaptation at 5g mobile edge. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 237–242. ACM, 2016.

[25] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 97–108, New York, NY, USA, 2012. ACM.

[26] Danny H. Lee, Constantine Dovrolis, and Ali C. Begen. Caching in http adaptive streaming: Friend or foe? In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, NOSSDAV '14, pages 31:31–31:36, New York, NY, USA, 2014. ACM.

[27] Xi Liu, Florin Dobrian, Henry Milner, Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. A case for a coordinated internet video control plane. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 359–370. ACM, 2012.

[28] Ge Ma, Zhi Wang, Miao Zhang, Jiahui Ye, Minghua Chen, and Wenwu Zhu. Understanding performance of edge content caching for mobile video streaming. *IEEE Journal on Selected Areas in Communications*, 35(5):1076–1089, 2017.

[29] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 197–210, New York, NY, USA, 2017. ACM.

[30] Joseph Medley. Exceeding the buffering quota. https://developers.google.com/web/updates/2017/10/quotaexceedederror.

[31] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '15, pages 417–429, Berkeley, CA, USA, 2015. USENIX Association.

[32] Hasti A Pedersen and Sujit Dey. Enhancing mobile video capacity and quality using rate adaptation, ran caching and processing. *IEEE/ACM Transactions on Networking (TON)*, 24(2):996–1010, 2016.

[33] Mary Pedersen. Best practices: What is the optimal length for video content? https://adage.com/article/digitalnext/optimal-length-video-content/299386.

[34] Shankaranarayanan Puzhavakath Narayanan, Yun Seong Nam, Ashiwan Sivakumar, Balakrishnan Chandrasekaran, Bruce Maggs, and Sanjay Rao. Reducing latency through page-aware management of web objects by content delivery networks. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, SIGMETRICS '16, pages 89–100, New York, NY, USA, 2016. ACM.

[35] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 272–285, New York, NY, USA, 2016. ACM.

[36] Tuyen X Tran, Abolfazl Hajisami, Parul Pandey, and Dario Pompili. Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges. *IEEE Communications Magazine*, 55(4):54–61, 2017.

[37] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 325–338, New York, NY, USA, 2015. ACM.