Performance Characterization of a Commercial Video Streaming Service

Mojgan Ghasemi
Princeton University

Partha Kanuparthy
Yahoo Research*

Ahmed Mansy
Yahoo

Theophilus Benson
Duke University

Jennifer Rexford
Princeton University

Abstract

Despite the growing popularity of video streaming over the Internet, problems such as re-buffering and high startup latency continue to plague users. In this paper, we present an end-to-end characterization of Yahoo's video streaming service, analyzing over 500 million video chunks downloaded over a two-week period. We gain unique visibility into the causes of performance degradation by instrumenting both the CDN server and the client player at the chunk level, while also collecting frequent snapshots of TCP variables from the server network stack. We uncover a range of performance issues, including an asynchronous disk-read timer and cache misses at the server, high latency and latency variability in the network, and buffering delays and dropped frames at the client. Looking across chunks in the same session, or destined to the same IP prefix, we see how some performance problems are relatively persistent, depending on the video's popularity, the distance between the client and server, and the client's operating system, browser, and Flash runtime.

1. INTRODUCTION

Internet users watch hundreds of millions of videos per day [6], and video streams represent more than 70% of North America's downstream traffic during peak hours [5]. A video streaming session, however, may suffer from problems such as long startup delay, re-buffering events, and low video quality that negatively impact user experience and the content provider's revenue [25, 14]. Content providers strive to improve performance through a variety of optimizations, such as placing servers closer to clients, content caching, effective peering and routing decisions, and splitting the video session (i.e., the HTTP session carrying the video traffic) into fixed-length *chunks* in multiple bitrates [9, 37, 20, 23, 32]. Multiple bitrates enable adaptive bitrate algorithms (ABR) in the player to adjust video quality to available resources.

Despite these optimizations, performance problems can arise anywhere along the end-to-end delivery path shown in Figure 1. The poor performance can stem from a variety of root causes. For example, the *backend service* may increase the chunk download latency on a cache miss. The *CDN* servers can introduce high latency when accessing data from disk. The *network* can introduce congestion or random packet losses. The *client's download stack* may handle data inefficiently (e.g., slow copying of data from OS to the player via the browser and Flash runtime) and the *client's rendering path* may drop frames due to high CPU load.

*Work done at Yahoo. Current affiliation: Amazon Web Services.

IMC 2016, November 14-16, 2016, Santa Monica, CA, USA
© 2016 ACM. ISBN 978-1-4503-4526-2/16/11...$15.00
DOI: http://dx.doi.org/10.1145/2987443.2987481

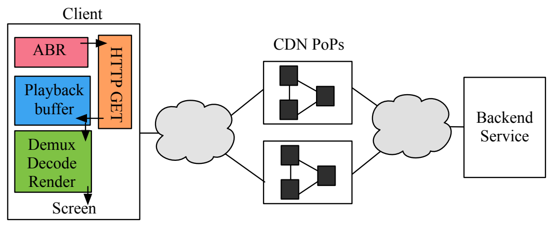


Figure 1: End-to-End video delivery components.

While ABR algorithms can adapt to performance problems (e.g., lower the bitrate when throughput is low), understanding the *location* and *root causes* of performance problems enables content providers to take the right corrective (or even proactive) actions, such as directing client requests to different servers, adopting a different cache-replacement algorithm, or further optimizing the player software. In some cases, knowing the bottleneck can help the content provider decide *not* to act, because the root cause is beyond the provider's control—for example, it lies in the client's browser, operating system, or access link. The content provider could detect the existence of performance problems by collecting Quality of Experience (QoE) metrics at the player, but this does not go far enough to identify the underlying cause. In addition, the buffer at the player can (temporarily) mask underlying performance problems, leading to delays in detecting significant problems based solely on QoE metrics.

Instead, we adopt a *performance-driven* approach for uncovering performance problems. Collecting data at the client or the CDN alone is not enough. Client-side measurements, while crucial for uncovering problems in the download stack (e.g., a slow browser) or rendering path (e.g., slow decoder), cannot isolate network and provider-side bottlenecks. Moreover, a content provider cannot collect OS-level logs or measure the network stack at the client; even adding small extensions to the browsers or plugins would complicate deployment. Server-side logging can fill in the gaps [38], with care to ensure that the measurements are sufficiently lightweight in production.

In this paper, we instrument the CDN servers and the video player of a Web-scale commercial video streaming service, and join the measurement data to construct an end-to-end view of session performance. We measure per-*chunk* milestones at the player, which

Location	Findings
CDN	1. Asynchronous disk reads increase server-side delay.
	2. Cache misses increase CDN latency by order of magnitude.
	3. Persistent cache-miss and slow reads for unpopular videos.
	4. Higher server latency even on lightly loaded machines.
Network	1. Persistent delay due to physical distance or enterprise paths.
	2. Higher latency variation for users in enterprise networks.
	3. Packet losses early in a session have a bigger impact.
	4. Bad performance caused more by throughput than latency.
Client	1. Buffering in client download stack can cause re-buffering.
	2. First chunk of a session has higher download stack latency.
	3. Less popular browsers drop more frames while rendering.
	4. Avoiding frame drops needs min of $1.5\frac{sec}{sec}$ download rate.
	5. Videos at lower bitrates have *more* dropped frames.

Table 1: Summary of key findings.

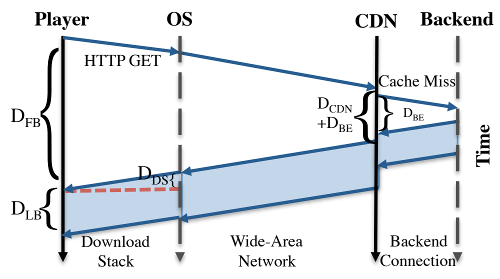


Figure 2: Time diagram of chunk delivery. Solid lines are instrumentation while dashed lines are estimates.

runs on top of Flash (e.g., the time to get the chunk's first and last bytes, and the number of dropped frames during rendering), and the CDN server (e.g., server and backend latency), as well as kernel-space TCP variables (e.g., congestion window and round-trip time) from the server host. Direct measurement of the main system components help us avoid relying on inference or tomography techniques that would limit the accuracy; or requiring other source of "ground truth" to label the data for machine learning [13]. In this paper, we make the following contributions:

1. A large-scale instrumentation of *both sides* of the video delivery path in a commercial video streaming service over a two-week period, studying more than 523 million chunks and 65 million on-demand video sessions.

2. End-to-end instrumentation that allows us to characterize the player, network path, and the CDN components of session performance across multiple layers of the stack, *per-chunk*. We show an example of how partial instrumentation (e.g., player-side alone) would lead to incorrect conclusions about performance problems. Such conclusions could cause the ABR algorithm to make wrong decisions.

3. We characterize transient and persistent problems in the end-to-end path that have not been studied before; in particular the client's *download stack* and *rendering path*, and show their impact on QoE.

4. We offer a *comprehensive characterization* of performance problems for Internet video, and our key findings are listed in Table 1. Based on these findings, we offer insights for video content providers and Internet providers to improve video QoE.

2. CHUNK PERFORMANCE MONITORING

Model. We model a video session as an ordered sequence of HTTP(S)[1] requests and responses over a single TCP connection between the player and the CDN server—after the player has been assigned to a server. The session starts with the player requesting the manifest, which contains a list of chunks in available bitrates (upon errors and user events such as seeks, manifest is requested again). The ABR algorithm — tuned and tested in the wild to balance between low startup delay, low re-buffering rate, high quality and smoothness — chooses a bitrate for each chunk to be requested from the CDN server. The CDN service maintains a FIFO queue of arrived requests and maintains a thread pool to serve the queue. The CDN uses a multi-level distributed cache (between machines, and the main memory and disk on each machine) to cache chunks with an LRU replacement policy. Upon a cache miss, the CDN server makes a corresponding request to the backend service.

The client host includes two independent execution paths that share host resources. The *download* path "moves" chunks from the NIC to the player, by writing them to the playback buffer. The *rendering* path reads from the playback buffer, de-muxes (audio from video), decodes and renders the pixels on the screen—this path could use either the GPU or the CPU. Note that there is a stack below the player: the player executes on top of a Javascript and Flash runtime, which in turn is run by the browser on top of the OS.

2.1 Chunk Instrumentation

We collect *chunk*-level measurements because: (1) most decisions affecting performance are taken per-chunk (e.g., caching at the CDN, and bitrate selection at the player), although some metrics are chosen once per session (e.g., the CDN server), (2) sub-chunk measurements would increase CPU load on client, at the expense of rendering performance (Section 4.4), and (3) client-side handling of data within a chunk can vary across streaming technologies, and is often neither visible nor controllable. For example, players implemented on top of Flash use a progress event that delivers data to the player, and the buffer size or frequency of this event may vary across browsers or versions.

We capture the following milestones per chunk at the player and the CDN service: (1) When the chunk's HTTP GET request is sent, (2) CDN latency in serving the chunk, in addition to backend latency for cache misses, and (3) the time to download the first and last bytes of the chunk. We denote the player-side *first-byte delay* D_{FB} and *last-byte delay* D_{LB}. Figure 2 summarizes our notation. We divide a chunk's lifetime into the three phases: fetch, download, and playout.

Fetch Phase. The fetch process starts with the player sending an HTTP request to the CDN for a chunk at a specified bitrate until the first byte arrives at the player. The byte transmission and delivery traverse the host stack (player, Flash runtime, browser, userspace to kernel space and the NIC)—contributing to the *download stack latency*. If the content is cached at the CDN server, the first byte is sent after a delay of D_{CDN} (the cache lookup and load delay); otherwise, the backend request for that chunk incurs an additional delay of D_{BE}. Note that the backend and delivery are always pipelined. The first-byte delay D_{FB} includes network round-trip time (rtt_0), CDN service latency, backend latency (if any), and client download stack latency:

$$D_{FB} = D_{CDN} + D_{BE} + D_{DS} + rtt_0 \qquad (1)$$

[1]Both HTTP and HTTPS protocols are supported at Yahoo; for simplicity, we use HTTP instead of HTTPS in the rest of the paper.

We measure D_{FB} for each chunk at the player. At the CDN service, we measure D_{CDN} and its constituent parts: (1) D_{wait}: the time the HTTP request waits in the queue until the request headers are read by the server, (2) D_{open}: after the request headers are read until the server first attempts to open the file, regardless of cache status, and (3) D_{read}: time to read the chunk's first byte and write it to the socket, including the delay to read from local disk or backend. The backend latency (D_{BE}) is measured at the CDN service and includes network delay. Characterizing backend service problems is out of scope for this work; we found that such problems are relatively rare.

A key limitation of player-side instrumentation is that application layer metrics capture the mix of download stack latency, network latency, and server-side latency. To isolate network performance from end-host performance, we measure the end-to-end network path at the CDN host kernel's TCP stack. Since kernel-space latencies are relatively very low, it is reasonable to consider this view as representative of the network path performance. Specifically, the CDN service snapshots the Linux kernel's `tcp_info` structure for the player TCP connection (along with context of the chunk being served). The structure includes TCP state such as smoothed RTT, RTT variability, retransmission counts, and sender congestion window. We sample the path performance periodically every 500ms[2]; this allows us to observe changes in path performance.

Download Phase. The download phase is the window between arrivals of the first and the last bytes of the chunk at the player, i.e., the last-byte delay, D_{LB}. It depends on the chunk size, which depends on chunk bitrate and duration. To identify chunks suffering from low throughput, on the client side we record the requested *bitrate* and the *last-byte delay*. To understand the network path performance and its impact on TCP, we snapshot TCP variables from the CDN host kernel at least once per-chunk (as described above).

Playout Phase. As a chunk is downloaded, it is added to the playback buffer. If the playback buffer does not contain enough data, the player pauses and waits for sufficient data; in case of an already playing video, this causes a rebuffering event. We instrument the player to measure the number (buf_{count}) and duration of rebuffering events (buf_{dur}) per-chunk played.

Each chunk must be decoded and rendered at the client. In the absence of hardware rendering (i.e., GPU), chunk frames are decoded and rendered by the CPU, which makes video quality sensitive to CPU utilization. A slow rendering process drops frames to keep up with the encoded frame rate. To characterize rendering path problems, we instrument the Flash player to collect the average rendered frame rate per chunk (avg_{fr}) and the number of dropped frames per chunk ($drop_{fr}$). A low rendering rate, however, is not always indicative of bad performance; for example, when the player is in a hidden tab or a minimized window, video frames are dropped to reduce CPU load [14]. To identify these scenarios, the player collects a variable (vis) that records if the player is visible when the chunk is displayed. Table 2 summarizes the metrics collected for each chunk at the player and CDN.

2.2 Per-session Instrumentation

In addition to per-chunk milestones, we collect session metadata; see Table 3. A key to end-to-end analysis is to *trace* session performance from the player through the CDN (at the granularity of chunks). We implement tracing by using a globally unique session ID and per-session chunk IDs.

[2]The frequency is chosen to keep overhead low in production.

Location	Statistics
Player (Delivery)	sessionID, chunkID, D_{FB}, D_{LB}, bitrate
Player (Rendering)	buf_{dur}, buf_{count}, vis, avg_{fr}, $drop_{fr}$
CDN (App layer)	sessionID, chunkID, D_{CDN} (wait, open, and read), D_{BE}, cache status, chunk size
CDN (TCP layer)	CWND, SRTT, SRTTVAR, retx, MSS

Table 2: Per-chunk instrumentation at player and CDN.

Location	Statistics
Player	sessionID, user IP, user agent, video length
CDN	sessionID, user IP, user agent, CDN PoP, CDN server, AS, ISP, connection type, location

Table 3: Per-session instrumentation at player and CDN.

3. MEASUREMENT DATASET

We study 65 million VoD sessions (523m chunks) with Yahoo, collected over a period of 18 days in September 2015. These sessions were served by a random subset of 85 CDN servers across the US. Our dataset predominantly consists of clients in North America (over 93%).

Figure 3(a) shows the cumulative distribution of the length of the videos. All chunks in our dataset contain six seconds of video (except, perhaps, the last chunk).

We focus on desktop and laptop sessions with Flash-based players. The browser distribution is as follows: 43% Chrome, 37% Firefox, 13% Internet Explorer, 6% Safari, and about 2% other browsers; the two major OS distributions in the data are Windows (88.5% of sessions) and OS X (9.38%). We do not consider cellular users in this paper since the presence of ISP proxies affects the accuracy of our findings.

The video viewership and popularity of videos is heavily skewed towards popular content; see Figure 3(b). We find that top 10% of most popular videos receive about 66% of all playbacks.

Data preprocessing to filter proxies. A possible pitfall in our analysis is the existence of enterprise or ISP HTTP proxies [35], since the CDN server's TCP connection would terminate at the proxy, leading to network measurements (e.g., RTT) reflecting the server-proxy path instead of the client. We filter sessions using a proxy when: (i) we see different client IP addresses or user agents [34] between HTTP requests and client-side beacons[3], or (ii) the client IP address appears in a very large number of sessions (e.g., more more minutes of video per day than there are minutes in a day). After filtering proxies, our dataset consists of 77% of sessions.

Ethical considerations: Our instrumentation methodology is based on logs/metrics about the traffic, without looking at packet payload or video content. For privacy reasons, we do not track users (through logging) hence we cannot study access patterns of individual users. Our analysis uses client IP addresses internally to identify proxies and perform coarse-grained geo-location; after that, we use opaque session IDs to study the dataset.

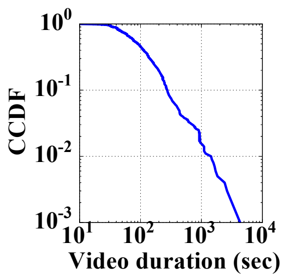
4. CHARACTERIZING PERFORMANCE

In this section, we characterize the performance of each component of the end-to-end path, and show the impact on QoE. Prior work has shown that important stream-related factors affect the QoE: startup delay, rebuffering ratio, video quality (average bi-

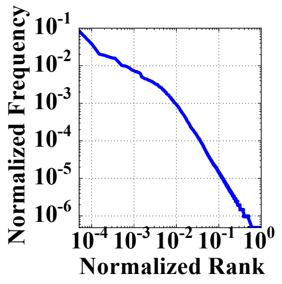
[3]A beacon is a message sent back from the client to the analytic servers, carrying information.

Latency	Description
D_{FB}	Time to fetch the first byte
D_{LB}	Time to download the chunk (first to last byte)
D_{CDN}	CDN latency ($= D_{wait} + D_{open} + D_{read}$)
D_{BE}	Backend latency in cache miss
D_{DS}	Client's download stack latency
rtt_0	Network round-trip time during the first-byte exchange

Table 4: Latency notations and their description



(a) CCDF of video lengths (one month)



(b) Rank vs. popularity (one day)

Figure 3: Length and popularity of videos in the dataset.

trate), and the rendering quality [14, 37]. They have developed models for estimating QoE scores of videos by assigning weights to each of these stream metrics to estimate a user behavior metric such as abandonment rate.

We favor looking at the impact on individual QoE factors instead of a single QoE score to assess the significance of performance problems. This is primarily because of the impact of content on user behavior (and hence, QoE). First, user behavior may be different for long-duration content such as Netflix videos (e.g., users may be more *patient* with a longer startup delay) than short-duration content (our case). Second, the type of content being viewed impacts user behavior (and hence the weights of QoE factors). For example, the startup delay for a news video (e.g., "breaking news") may be more important to users than the stream quality; while for sports videos, the quality may be very important. Given the variety of Yahoo videos, we cannot use a one-size-fits-all set of weights for a QoE model. Moreover, the results would not generalize to all Internet videos. Instead, we show the impact of each problem directly on the QoE factors.

4.1 Server-side Performance Problems

Yahoo uses the Apache Traffic Server (ATS), a popular caching proxy server [2], to serve HTTP requests. The traffic engineering system maps clients to CDN nodes using a function of geography, latency, load, cache likelihood, etc. In other words, the system tries to route clients to a server that is likely to have a hot cache. The server first checks the main memory cache, then tries the disk, and finally sends a request to a backend server if needed.

Server latencies are relatively low, since the CDN and the backend are well-provisioned. About 5% of sessions, however, experience a QoE problem due to the server, and the problems can be *persistent* as we show below. Figure 4 shows the impact of the server-side latency for the first chunk on the startup delay (time to play) at the player.

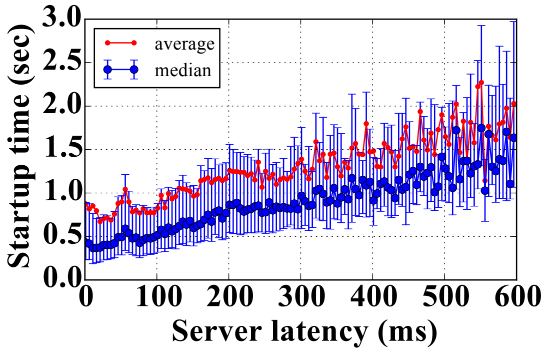


Figure 4: Impact of server latency on QoE (startup time), error bars show the interquartile range (IQR).

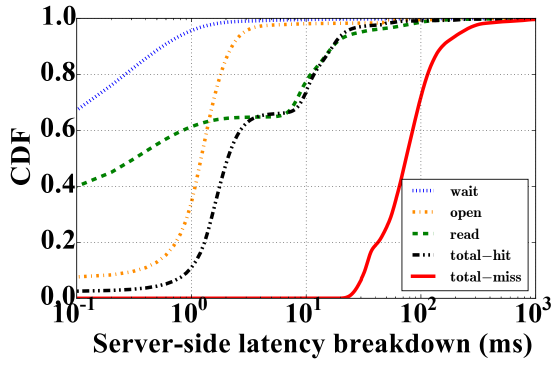


Figure 5: CDN latency breakdown across all chunks.

1. Asynchronous disk read timer and cache misses cause high server latency. Figure 5 shows the distribution of each component of CDN latency across chunks; it also includes the distribution of total server latency for chunks broken by cache hit and miss. Most of the chunks have a negligible waiting delay ($D_{wait} < 1ms$) and open delay. However, the D_{read} distribution has two nearly identical parts, separated by about 10ms. The root cause is that ATS executes an asynchronous read to read the requested files in the background. When the first attempt in opening the cache is not immediately returned (due to content not being in memory), ATS retries to open the file (either from the disk or from backend service) using a 10ms timer [4].

On a cache miss, the backend latency significantly affects the serving latency according to Figure 5. The median server latency among chunks experiencing a cache hit is 2ms, while the median server latency for cache misses is 40 times higher at 80ms. The average and 95^{th} percentile of server latency in case of cache misses increases tenfold. In addition, cache misses are the main contributor when server latency has a higher contribution to D_{FB} than the network RTT: for 95% of chunks, network latency is higher than server latency; however, among the remaining 5%, the cache miss ratio is 40%, compared to an average cache miss rate of 2% across session chunks.

Take-away: Cache misses impact serving latency, and hence QoE (e.g., startup time) significantly. An interesting direction to explore is to alter the LRU cache eviction policy to offer better cache hit rates. For example, policies for popular-heavy workloads, such as GD-size or perfect-LFU [11].

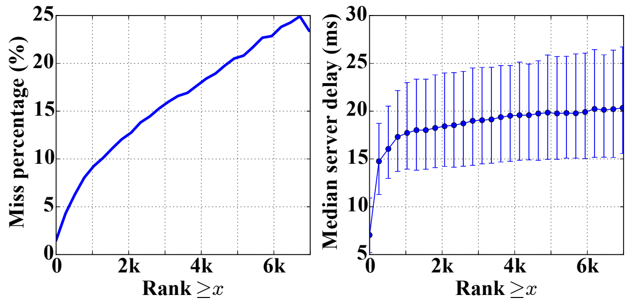


Figure 6: Performance vs popularity: (a) miss rate vs rank, (b) CDN latency (excluding cache misses) vs rank.

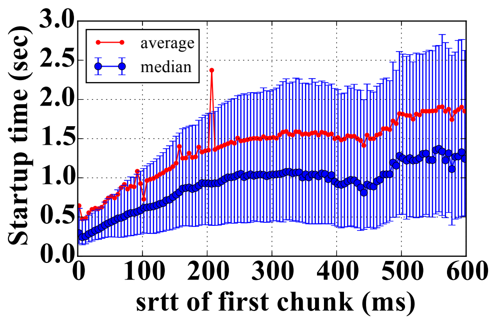


Figure 7: Average and median startup delay vs. network latency, error bars show the interquartile range (IQR).

2. Less popular videos have persistent high cache miss rate and high latency. We observed that a small fraction of sessions experience performance problems that are *persistent*. Once a session has a cache miss on one chunk, the chance of further cache misses increases significantly; the mean cache miss ratio among sessions with at least one cache miss is 60% (median of 67%). Also, once a session has at least one chunk with a high latency ($> 10ms$), the chance of future read delays increases; the mean ratio of high-latency chunks in sessions with at least one such chunk is 60% (median of 60%).

One possible cause for persistent latency, even when the cache hit ratio is high, is a highly loaded server that causes high serving latency; however, our analysis shows that server latency is not correlated with load[4]. This is because the CDN servers are well provisioned to handle the load.

Instead, the *unpopularity* of the content is a major cause of the persistent server-side problems. For less popular videos, the chunks often need to come from disk, or worse yet, the backend server. Figure 6(a) shows the cache miss percentage versus video rank (most popular video is ranked first) using data from one day. The cache miss ratio drastically increases for unpopular videos. Even on a cache hit, unpopular videos experience higher server delay, as shown in Figure 6(b). The figure shows mean server latency after removing cache misses (i.e., no backend communication). The unpopular content generally experiences a higher latency due to higher read (seek) latency from disk.

Take-away. The persistence of cache misses could be addressed by pre-fetching the subsequent chunks of a video session after the first miss. Pre-fetching of subsequent chunks would particularly help with unpopular videos since backend latency makes up a significant part of their overall latency and could be avoided.

When an object cannot be served from local cache, the request will be sent to the backend server. For a popular object, many concurrent requests may overwhelm the backend service; thus, the ATS retry timer is used to reduce the load on the backend servers; the timer introduces extra delay for cases where the content is available on local disk.

3. Load vs. performance due to cache-focused client mapping. We have observed that more heavily loaded servers offer *lower* CDN latency (note that CDN latency does not include the network latency, but only the time a server takes to start serving the file). This result was initially surprising since we expect busier servers to have worse performance; however, this can be explained by the *cache-focused mapping* CDN feature: As a result of cache-based assignment of clients to CDN servers, servers with less popular content have more chunks with either higher read latency as the content is not fresh in memory (and the ATS retry-timer), or worse yet, need to be requested from backend due to cache-misses.

While unpopular content leads to lower performance, because of lower demand it also produces fewer requests, hence servers that serve less popular content seem to have worse performance at a lower load than the servers with a higher load.

Take-away. An interesting direction to achieve better utilization of servers and load balancing is to actively partition popular content among servers (on top of cache-focused routing). For example, given that the top 10% of videos make up 66% of requests, distributing only the top 10% of popular videos across servers can balance the load.

4.2 Network Performance Problems

Network problems can manifest themselves in the form of increased packet loss, reordering, high latency, high variation in latency, and low throughput. Each can be persistent (e.g., far away clients from a server have persistent high latency) or transient (e.g., spike in latency caused by congestion). In this section, we characterize these problems.

Distinguishing between a transient and a persistent problem matters because although a good ABR may *adapt* to temporary problems (e.g., by lowering bitrate), it cannot avoid bad quality caused by persistent problems (e.g., when a peering point is heavily congested, even the lowest bitrate may see re-buffering). Instead, persistent problems require *corrective actions* taken by the video provider (e.g., placement of new CDN PoPs) or ISPs (e.g., additional peering).

We characterize the impact of loss and latency on QoE. To characterize long-term problems, we aggregate sessions into /24 IP prefixes since most allocated blocks and BGP prefixes are /24 prefixes [29, 16]. Figure 7 shows the effect of network latency during the first chunk on video QoE, specifically, startup delay, across sessions. High latency in a session could be caused by a persistently high baseline (i.e., high $srtt_{min}$)[5], or variation in latency as a result of transient problems (i.e., high variation, σ_{srtt}). Figure 8 shows the distribution of both of these metrics across sessions. We see that both of these problems exist among sessions; we characterize each of these next.

[4]We estimated load as of number of parallel HTTP requests, sessions, or bytes served per second.

[5]Note that TCP's estimate of RTT, SRTT, is an EWMA average; hence $srtt_{min}$ is higher than the minimum RTT seen by TCP. The bias of this estimator, however, is not expected to be significant for our study since it is averaged.

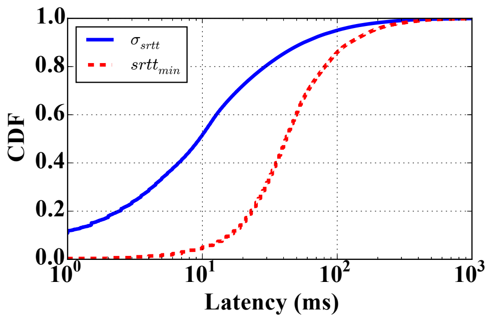


Figure 8: CDF of baseline ($srtt_{min}$) and variation in latency (σ_{srtt}) among sessions.

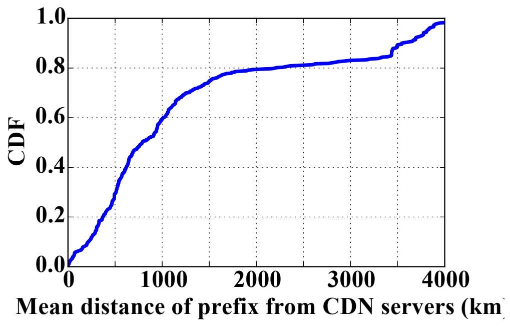


Figure 9: Mean distance (km) of US prefixes in the tail latency from CDN servers.

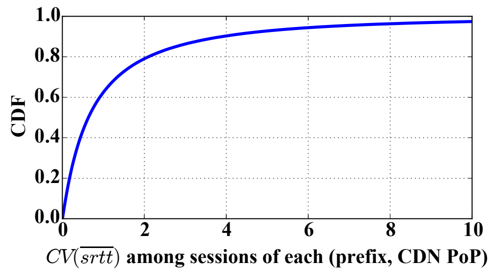


Figure 10: CDF of path latency variation: CV of latency per path, a path is defined by a (prefix, PoP) pair.

1. Persistent high latency caused by distance or enterprise path problems. In Figure 8, we see that some sessions have a high minimum RTT. To analyze the minimum latency, it is important to note that the SRTT samples are taken after 500ms from the beginning of the chunk's transmission; hence, if a chunk has self-loading [21], the SRTT sample may reflect the additional queuing delay and not just the baseline latency. To filter out chunks whose SRTT has grown while downloading, we use an estimate of the initial network round-trip time (rtt_0) per-chunk. Equation 1 shows that $D_{FB} - (D_{CDN} + D_{BE})$ can be used as an upper-bound estimate of rtt_0. We take the minimum of SRTT and rtt_0 per-chunk as the baseline sample. Next, to find the minimum RTT in a session or prefix, we take the minimum among all these per-chunk baseline samples in the session or prefix.

In order to find the underlying cause of persistently high latency, we aggregate sessions into /24 client prefixes. The aggregation overcomes client last-mile problems, which may increase the latency for one session, but are not persistent problems. A prefix has more RTT samples than a session; hence, congestion is less likely to inflate all samples.

We focus our analysis on prefixes in the 90^{th} percentile latency, where $srtt_{min} > 100ms$; which is a high latency for cable/broadband connections (note that our CDN and client footprint is largely within North America). To ensure that a temporary congestion or routing change has not affected samples of a prefix, and to understand the persistent problems in poor prefixes, we repeat this analysis every day in our dataset and calculate the recurrence frequency, $\frac{\#\text{days prefix in tail}}{\#days}$. We take the top 10% of prefixes with highest reoccurrence frequency as prefixes with a persistent latency problem. This set includes $57k$ prefixes.

In these $57k$ prefixes, 75% are located outside the US and are spread across 96 different countries. These non-US clients are often limited by geographical distance and propagation delay. However, among the 25% of prefixes located in the US, *the majority are close to CDN nodes.* Since IP geolocation packages may not be accurate outside US, in particular favoring the US with 45% of entries [29], we focus our geo-specific analysis to US clients. Figure 9 shows the relationship between the $srtt_{min}$ and geographical distance of these prefixes in the US. If a prefix is spread over several cities, we use the average of their distances to the CDN server. Among high-latency prefixes inside the US within a 4km distance, only about 10% are served by residential ISPs, while the remaining 90% of prefixes originate from corporations and private enterprises.

Take-away: Finding clients that suffer from persistent high latency due to geographical distance helps video content providers in better placement of new CDN servers and traffic engineering. It is equally important to look at close-by clients suffering from high latency to (1) avoid over-provisioning servers in those geographics and wasting resources, and, (2) identify the IP prefixes with known persistent problems and adjust the ABR algorithm accordingly, for example, to start the streaming with a more conservative initial bitrate.

2. Residential networks have lower latency variation than enterprises. To measure RTT variation, we calculate the coefficient of variation (CV) of SRTT in each session, which is defined as the standard deviation over the mean of SRTT. Sessions with low variability have $CV < 1$ and sessions with high SRTT variability have $CV > 1$. For each ISP and organization, we measure the ratio of sessions with $CV > 1$ to all sessions. We limit the result to ISPs/organizations that have least 50 video streaming sessions to provide enough evidence of persistence. Table 5 shows the top ISPs/organizations with highest ratio. Enterprises networks make up most of the list. To compare this with residential ISPs, we analyzed five major residential ISPs and found that about 1% of sessions have $CV > 1$.

In addition to per-session variation in latency, we characterize the variation of latency in prefixes as shown in Figure 10. We use the average $srtt$ of each session as the sample latency. To find the coefficient of variance among all source-destination paths, we group sessions based on their prefix and the CDN PoP. We see that 40% of (prefix, PoP) pairs belong to paths with high latency variation ($CV > 1$).

Take-away: Recognizing which clients are more likely to suffer from latency variation is valuable for content providers because it helps them make informed decisions about QoE. In particular, the player bitrate adaptation and CDN traffic engineering algorithms can use this information to optimize streaming quality under high

isp/organization	#sessions with $CV > 1$	#all sessions	Percentage
Enterprise#1	30	69	43.4%
Enterprise#2	4,836	11,731	41.2%
Enterprise#3	1,634	4,084	40.0%
Enterprise#4	83	208	39.9%
Enterprise#5	81	203	39.9%

Table 5: ISP/Organizations with highest percentage of sessions with $CV(SRTT) > 1$.

latency variation. For example, the player can make more conservative bitrate choices, lower the inter-chunk wait time (i.e., request chunks sooner), and increase the buffer size to deal with variability.

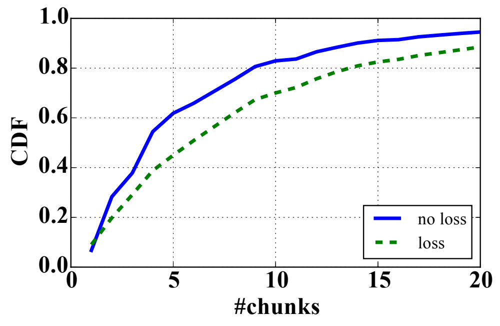
3. Earlier packet losses have higher impact on QoE. We use the retransmission count to study the effect of packet losses. A majority of the sessions ($> 90\%$) have a retransmission rate of less than 10%, with 40% of sessions experiencing no loss. While 10% can severely impact TCP throughput, not every retransmission is caused by an actual loss (e.g., due to early retransmit optimizations, underestimating RTO, etc.). Figure 11 shows the differences between sessions with and without loss in three aspects: (a) number of chunks (are these sessions shorter?), (b) bitrate (similar quality?), and (c) re-buffering. We see that the session length and bitrate distributions are almost similar between the two groups; however, re-buffering difference is significant and sessions without loss have better QoE.

While higher loss rates generally indicate higher re-buffering (Figure 12), the loss rate of a TCP connection does not necessarily correlate with the video QoE; the timing of the loss matters too. Figure 13 shows two example sessions (case-1 and case-2) where both sessions have 10 chunks with similar bitrates, cache status, and SRTT distributions. Case-1 has a retransmission rate of 0.75% compared to 22% in case-2; but it experienced dropped frames and re-buffering despite the lower loss rate. As Figure 13 shows, the majority of losses in case-1 happen in the first chunk, while case-2 has no loss during the first four chunks, building up its buffer to 29.8 seconds before a loss happens and successfully avoids re-buffering.

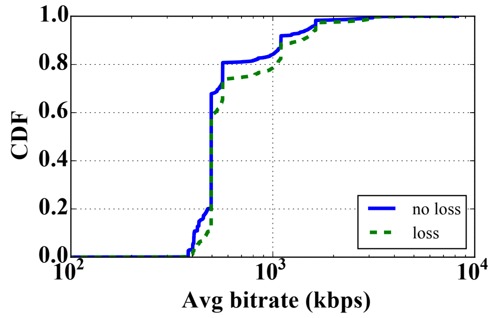
Because the buffer can hide the effect of subsequent loss, we believe that it is important to not only measure loss rate in video sessions, but also *the chunk ID that experiences loss*. Loss during earlier chunks has more impact on QoE because the playback buffer would hold less data for earlier chunks. We expect losses during the first chunk to have the highest effect on re-buffering. Figure 14 shows two examples: (1) $P(\textit{rebuf at chunk} = X)$, which is the percentage of chunks with that chunk ID seeing a re-buffering event; and (2) $P(\textit{rebuf at chunk} = X|\textit{loss at chunk} = X)$, which is the same probability conditioned on occurrence of a loss during the chunk. While occurrence of a loss in any chunk increases the likelihood of a re-buffering event, the increase is more significant for the first chunk.

We observe that losses are more likely to happen on the first chunk: Figure 15 shows the average per-chunk retransmission rate. The bursty nature of TCP losses towards the end of slow start [7] could be the cause of higher loss rates during the first chunk, which TCP avoids in subsequent chunks when transitioning into congestion avoidance state.

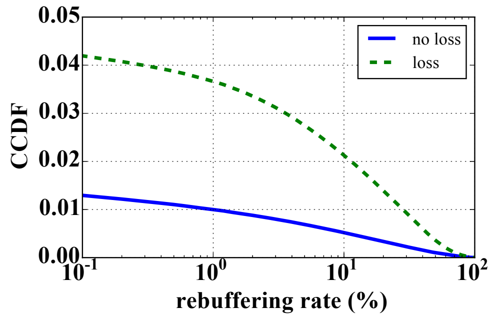
Take-aways: Due to the existence of a buffer in video streaming clients, the session loss rate does not necessarily correlate with QoE. The temporal location of loss in the session matters as well:



(a) CDF of session length with and without loss



(b) CDF of Average bitrate with and without loss



(c) CCDF (1-CDF) of Re-buffering rate with and without loss

Figure 11: Differences in session length, quality, and re-buffering with and without loss.

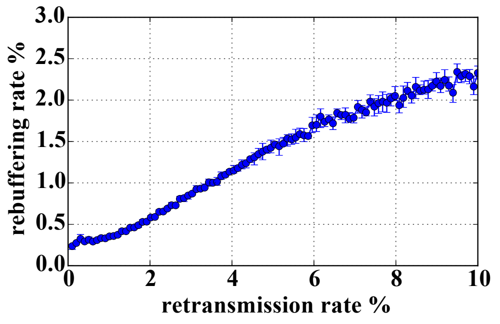


Figure 12: Rebuffering vs retransmission rate in sessions.

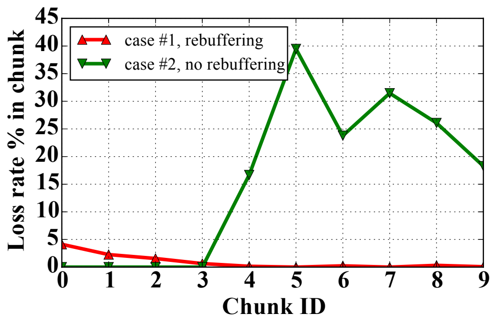


Figure 13: Example case for loss vs QoE.

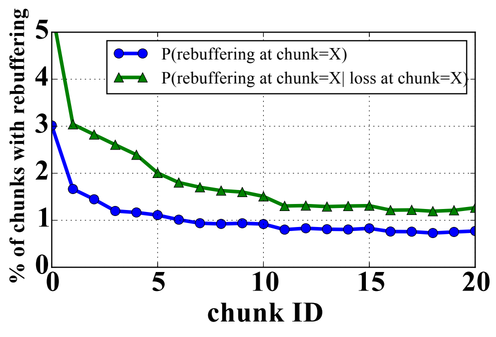


Figure 14: Re-buffering frequency per chunkID, Re-bufffering frequency given loss per chunkID.

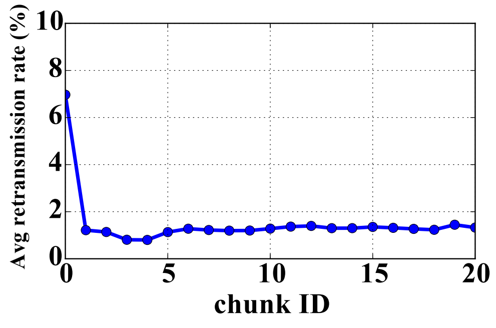


Figure 15: Average per-chunk retransmission rate.

earlier losses impact QoE more, with the first chunk having the biggest impact.

Due to the bursty nature of packet losses in TCP slow start caused by the exponential growth, the first chunk may have the highest per-chunk retransmission rate. Prior work showed a possible solution to work around a related issue using server-side pacing [19].

4. Throughput is a bigger problem than latency. To separate chunks based on performance, we use the following intuition: the playback buffer decreases when it takes longer to download a chunk than there are seconds of video in the chunk. With τ as the chunk duration, we tag chunks with bad performance when the following score is less than one:

$$perf_{score} = \frac{\tau}{D_{FB} + D_{LB}} \qquad (2)$$

We use D_{LB} as a "measure" of throughput. Both latency (D_{FB}) and throughput (D_{LB}) play a role in this score. We define the latency share in performance by $\frac{D_{FB}}{D_{FB}+D_{LB}}$ and the throughput share by $\frac{D_{LB}}{D_{FB}+D_{LB}}$. We show that while the chunks with bad performance generally have higher latency and lower throughput than chunks with good performance, throughput is a more "dominant" metric in terms of impact on the performance of the chunk. Figure 16(a) shows that chunks with good performance generally have higher share of latency and lower share of throughput than chunks with bad performance. Figure 16(b) shows the difference in absolute values of D_{FB}, and Figure 16(c) shows the difference in absolute values of D_{LB}.

While chunks with bad performance generally have higher first and last byte delays, the difference in D_{FB} is negligible compared to that of D_{LB}. We can see that most chunks with bad performance are limited by throughout and have a higher throughput share.

Take-away: Our findings could be good news for ISPs because throughput can be an easier problem to fix (e.g., establish more peering points) than latency [3].

4.3 Client's Download Stack

1. Some chunks have significant download stack latency. Video packets traversing the client's download stack (OS, browser, and the Flash plugin) may be delayed due to buffered delivery. In the extreme case, all the chunk bytes could be buffered and delivered late and all at once to the player[6], resulting in a significant increase in D_{FB}. Since the buffered data is delivered at once or in short time windows, the *instantaneous throughput* ($TP_{inst} = \frac{\textit{chunk size}}{D_{LB}}$) will be much higher at the player than the arrival rate of the chunk bytes from the network. We use TCP variables to estimate the download throughout per-chunk:

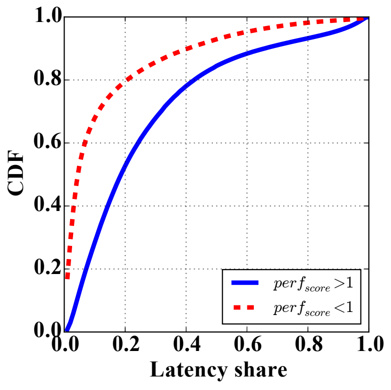
$$\textit{throughput} = MSS \times \frac{CWND}{SRTT} \qquad (3)$$

To detect chunks with this issue, we detect outliers using standard deviation: when a chunk is buffered in the download stack, its D_{FB} is much higher than that of the other chunks — more than $2 \cdot \sigma$ greater than the mean — despite other similar latency metrics (i.e., network and server-side latency are within one σ of the mean). Also, its TP_{inst} is much higher — more than $2 \cdot \sigma$ greater than the mean — due to the buffered data being delivered in a shorter time, while the estimated throughput from server side (using CWND and SRTT) does not explain the increase in throughput. Equations 4 summarize the detection conditions:

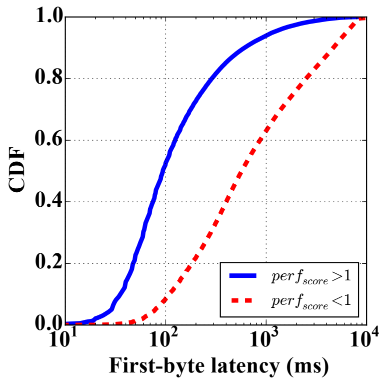
$$\begin{aligned} D_{FB_i} &> \mu_{D_{FB}} + 2 \cdot \sigma_{D_{FB}} \\ TP_{inst_i} &> \mu_{TP_{inst}} + 2 \cdot \sigma_{TP_{inst}} \\ SRTT, D_{server}, CWND &< \mu + \sigma \end{aligned} \qquad (4)$$

Figure 17 shows an example session that experiences the download stack problem (DS) taken from our dataset; our algorithm detected chunk 7 with much higher D_{FB} and TP_{inst} than the mean. Figure 17(a) shows D_{FB} of chunks and its constituents parts. We see that the increase in chunk 7's D_{FB} is not caused by a latency increase in backend, CDN, or network RTT. Figure 17(b) shows that this chunk also has an abnormally high throughput that seems impossible based on the estimated network throughput (Equation 3) at the server-side. The presence of both observations in the same chunk suggests that the chunk was buffered inside the client's stack

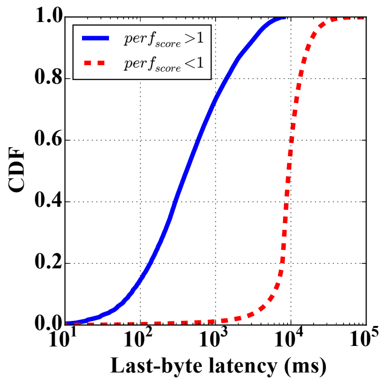
[6]Note that the delay is not caused by a full playback buffer, since the player will not request a new chunk when the buffer is full.



(a)

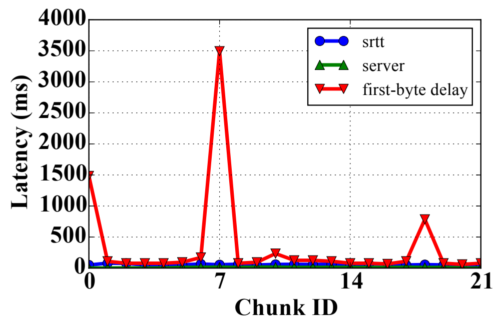


(b)

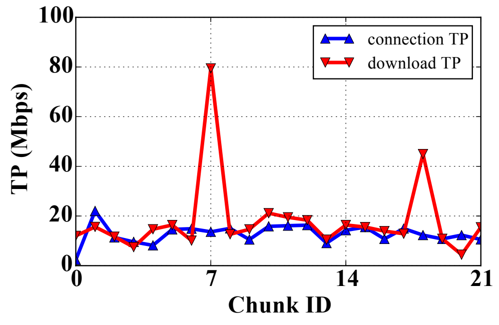


(c)

Figure 16: Latency vs throughput: (a) Latency share ($\frac{D_{FB}}{D_{FB}+D_{LB}}$), (b) D_{FB}, and (c) D_{LB} vs performance score.



(a) First-byte delay and its constituents in the session



(b) Network throughput vs instantaneous download throughput of the chunk

Figure 17: A case study showing the effects of client download stack (chunk#7).

and delivered late to the player. The buffered data was delivered almost instantaneously, since it mainly involves a kernel to userspace copy.

We have detected 1.7m chunks (0.32% of all chunks) using this method, demonstrating how often the client's download stack can buffer the data and hurt performance. About 1.6m video sessions have at least one such chunk (3.1% of sessions).

Take-aways: The download stack problem is an example where looking at one-side of measurements (CDN or client) alone would lead to wrong conclusions, and where both sides may blame the network. It is only with end-to-end instrumentation that this problem can be localized correctly. Failure in correctly recognizing such an effect on latency may lead to the following problems:

Over-shooting: Some ABR algorithms use player-level throughput (i.e., the instantanous throughput) in the bitrate selection process (e.g., a moving average of previous N chunks' throughput). Buffered delivery can lead to overestimation of end-to-end TCP throughput.

Under-shooting: If the ABR algorithms are either latency-sensitive, or use the average throughput (as opposed to the instantaneous throughput), the affected chunks may cause underestimation of the connection's throughput.

Incorrect actions: When the TCP throughput is low, content providers may initiate a corrective action, such as re-routing the client. If the download stack latency is not diagnosed, clients may be falsely re-routed.

While designing ABR algorithms that rely on throughput or latency measurements, using server-side data (CWND and SRTT) enables the player to estimate the state of the network more accurately than client-side measurements alone. This could be done by the CDN in an HTTP header with the next chunk. When it is not possible to incorporate server-side measurements, the current ABR algorithms that rely on client-side measurements should detect and exclude outliers in their throughput/latency input.

2. Persistent download-stack problems. The underlying assumption in the above method is that the majority of chunks will not be buffered by download stack, hence we can *detect the outlier* chunks. However, when a persistent problem in client's download stack affects all or most chunks, this method cannot detect the problem. If we could directly observe the network RTT, rtt_0, we can estimate D_{DS} using Equation 1 per-chunk. The current vanilla Linux kernel does not expose individual RTT samples via the `tcp_info` structure, and kernel changes or collecting packet traces may be infeasible in production settings.

To work around this limitation, we use a conservative estimate of rtt_0 as the TCP retransmission timer (RTO)[7]. RTO is how long the sender waits for a packet's acknowledgment before deciding it is lost; hence, RTO can be considered as a conservative estimate of

[7] $RTO = 200ms + SRTT + 4 \times SRTTVAR$, according to RFC 2988 [27].

	Safari on Linux	*Safari on Windows*	*Firefox on Windows*	*Other on Windows*	*Firefox on Mac*
mean DS(ms)	1041	1028	283	281	275

Table 6: OS/browser with highest D_{DS}.

rtt_0. We use RTO to estimate a lower bound of the client download stack latency per-chunk:

$$D_{DS} \geq D_{FB} - D_{CDN} - D_{BE} - RTO \qquad (5)$$

Using this method, we see that 17.6% of all chunks experience a positive download stack latency. In 84% of these chunks, download stack latency share in D_{FB} is higher than network and server latencies, making it the bottleneck in D_{FB}. Table 6 shows the top OS/browser combinations with highest persistent download stack latency. We see that among major browsers, Safari on non-OS X environments has the highest average download stack latency. In the "other" category, we find that less-popular browsers on Windows, in particular, Yandex and SeaMonkey, have high download stack latencies.

Take-aways and QoE impact: Download stack problems are worse for sessions with re-buffering: among sessions with no re-buffering, the average D_{DS} is less than 100ms. In sessions with up to 10% re-buffering ratios, the average D_{DS} grows up to 250ms, and in sessions with more than 10% re-buffering ratios, the average D_{DS} is more than 500ms. Although the download stack latency is not a frequent problem, it is important to note that when it is an issue, it is often the major bottleneck in latency. Any adaptation mechanisms at the client should detect the outliers to improve QoE.

It is important to know that some client setups (e.g., Yandex or Safari on Windows) are more likely to have persistent download stack problems. Recognizing the lasting effect of client's machine on QoE helps content providers avoid actions caused by wrong diagnosis (e.g., re-routing clients due to *seemingly* high network latency when problem is in download stack).

3. First chunks have higher download stack latency. We find that the distribution of D_{FB} in first chunks is higher than other chunks: the median D_{FB} among first chunks is $300ms$ higher than other chunks. Using packet traces and developer tools on browsers, we confirmed that this effect is not visible in OS or browser timestamps. We believe that the difference is due to higher download stack latency of first chunk. To test our hypothesis, we select a set of *performance-equivalent* chunks with the following conditions: (1) no packet loss, (2) CWND $>$ ICWND, (3) no queuing delay and similar SRTT (we use $60ms < SRTT < 65ms$ for presentation), and (4) $D_{CDN} < 5ms$, and cache-hit chunks.

Figure 18 shows the distribution of D_{FB} among the equivalent set for first versus other chunks. We see that despite similar performance conditions, first chunks experience higher D_{FB}. The root cause appears to be the processing time spent in initialization of Flash events and data path setup (using the progressEvent in Flash) at the player, which can increase D_{FB} of first chunk.[8]

Take-away: First chunks experience a higher latency than other chunks. Video providers could eliminate other sources of performance problems at startup and reduce the startup delay by methods such as caching the first chunk of video titles [30], or by assigning higher cache priorities for first chunks.

4.4 Client's Rendering Stack

[8]We can only see Flash as a blackbox, hence, we cannot confirm this. However, a similar issue about ProgressEvent has been reported [1].

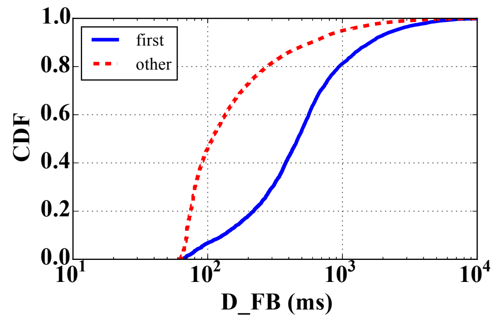


Figure 18: D_{FB} (ms) of first vs other chunks in equivalent performance conditions.

1. Avoiding dropped frames requires at least $1.5\frac{sec}{sec}$ download rate. In a typical video session, video chunks include multiplexed and encoded audio and video. They need to be de-multiplexed, decoded, and rendered on the client's machine, which takes processing time. Figure 19 shows the fraction of dropped frames versus average download rate of chunks. We define the average download rate of a chunk as video length (in seconds) over total download time ($\frac{\tau}{D_{FB}+D_{LB}}$). A download rate of $1\frac{sec}{sec}$ is barely enough: after receiving the frames, more processing is needed to decode frames for rendering. Increasing the download rate to $1.5\frac{sec}{sec}$ enhances the framerate; however, increasing the rate beyond this does not improve the framerate.

To see if this observation can explain the rendering quality, we look at the framerate as a function of chunk download rate: 85.5% of chunks have low framerate ($>$ 30% drop) when the download rate is below $1.5\frac{sec}{sec}$ and good framerate when download rate is at least $1.5\frac{sec}{sec}$. About 5.7% of chunks have low rates but good rendering, which can be explained by the buffered video frames that hide the effect of low rates. Finally, 6.9% of chunks have low framerate despite a minimum download rate of $1.5\frac{sec}{sec}$, not confirming the hypothesis. However, this could be explained as follows: First, the average download rate does not reflect instantaneous throughput. In particular, earlier chunks are more sensitive to changes in throughput, since fewer frames are buffered at the player. Second, when the CPU on the client machine is overloaded, software rendering can be inefficient irrespective of the chunk arrival rate.

Figure 20 shows a simple controlled experiment, where a player is running in the Firefox browser on OS X with eight CPU cores, connected to the server using a 1Gpbs Ethernet link. The first bar represents the per-chunk dropped rate while using GPU decoding and rendering. Next, we turned off hardware rendering; we see increase in frame drop rate with background processes using CPU cores.

2. Higher bitrates have better rendered framerate. Higher bitrates contain more data per frame, thus imposing a higher load on the CPU for decoding and rendering in time. We expect chunks with higher bitrates to have more dropped frames as a result. We did not observe this in our data. However, we observed the following trends in the data: (1) higher bitrates are often requested in connections with lower RTT variation: SRTTVAR across sessions with bitrates higher than 1Mbps is $5ms$ lower than the rest. Less variation may result in fewer frames delivered late. (2) higher bitrates are often requested in connections with lower retransmission rate: the retransmission rate among sessions with bitrates higher

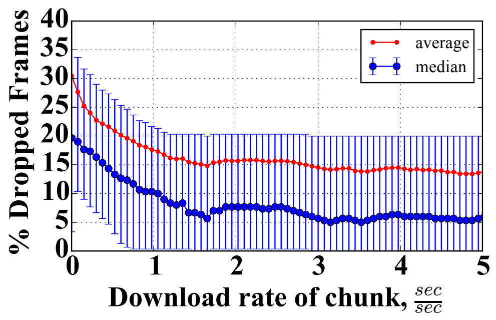


Figure 19: %Dropped frames vs chunk download rate, first bar represents hardware rendering.

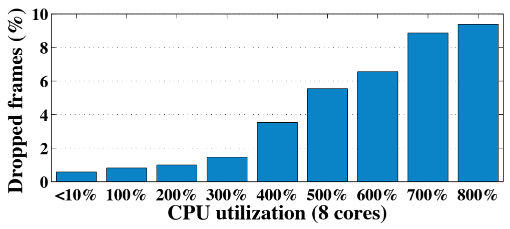


Figure 20: Dropped frames per CPU load in a controlled experiment.

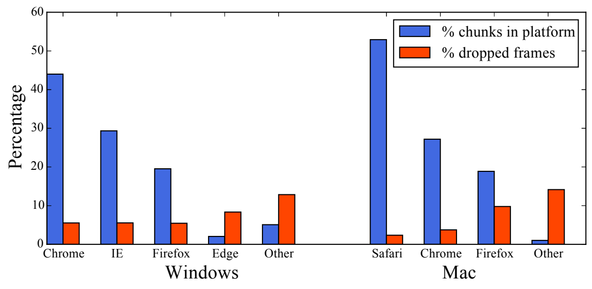


Figure 21: Browser popularity and rendering quality in the two major platforms: Windows vs Mac.

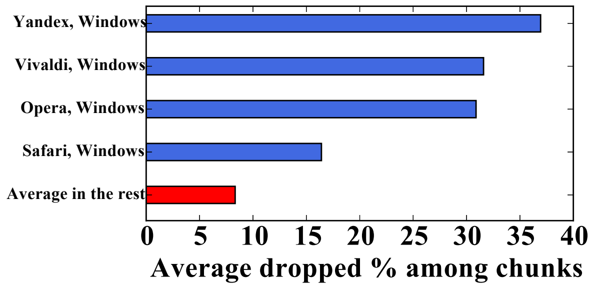


Figure 22: Dropped % of (browser, OS), $rate \geq 1.5\frac{sec}{sec}$**,** $vis = True$**.**

than 1Mbps is 1% lower than the rest. Lower packet loss rate results in less frames dropped or arrived late.

3. Less popular browsers have low rendering quality. If we limit our analysis to chunks with good performance ($rate > 1.5\frac{sec}{sec}$) where the player is visible (i.e., $vis = true$), the rendering quality can still be bad due to inefficiencies in client's rendering path. Since we cannot measure the client host environment in production, we only characterize the clients based on their OS and browser.

Figure 21 shows the fraction of chunks requested from browsers on OS X and Windows platforms (each platform is normalized to 100%), as well as the average fraction of dropped frames among chunks served by that browser. Browsers with internal Flash (e.g., Chrome) and native HLS support (Safari on OS X) outperform other browsers (some of which may run Flash as a process, e.g., Firefox's protected mode). Also, the unpopular browsers (grouped as Other) have the lowest performance. We further break them down as shown in Figure 22. We restrict to browsers that have processed at least 500 chunks. Yandex, Vivaldi, Opera or Safari on Windows have low rendered framerate compared to other browsers.
Take-aways: De-multiplexing, decoding and rendering video chunks could be resource-heavy on the client machine. In absence of hardware (GPU) rendering, the burden falls on CPU to process frames efficiently; however, the resource demands from other applications on the host can affect the rendering quality. We found that video rendering requires processing time, and that a video arrival rate of $1.5\frac{sec}{sec}$ could be used as a rule-of-thumb for achieving good rendering quality. Similar to download stack problems, rendering quality differs based on OS and browser. In particular, we found unpopular browsers to have lower rendering quality.

5. DISCUSSION

Monitoring and diagnosis is a challenging problem for large-scale content providers due to insufficient instrumentation or measurement overhead limitations. In particular, (1) sub-chunk events such as bursty losses will not be captured in per-chunk measurements; capturing them will impact player's performance, (2) SRTT does not reflect the value of round-trip time at the time of measurement, rather is a smoothed average; vanilla Linux kernels only export SRTTs to userspace today. To work with this limitation, we use methods discussed in Section 4.2, (3) the characterization of the rendering path could improve by capturing the underlying resource utilization and environment (e.g., CPU load, existence of GPU), and (4) in-network measurements help further localization. For example, further characterization of network problems (e.g., is bandwidth limited at the core or the edge?) would have been possible using active probes (e.g., traceroute or ping) or in-network measurements from ISPs (e.g., link utilization). Some of these measurements may not be feasible at Web-scale.

6. RELATED WORK

Video streaming characterization: There is a rich area of related work in characterizing video-streaming quality. [28] uses ISP packet traces to characterize video while [36] uses CDN-side data to study content and Live vs VoD access patterns. Client-side data and a clustering approach is used in [22] to find critical problems related to user's ISP, CDN, or content provider. Popularity in user-generated content video system has been characterized in [12]. Our work differs from previous work by collecting and joining fine-grained per-chunk measurements from both sides and direct instrumentation of the video delivery path, including the client's download stack and rendering path.

QoE models: Studies such as [14] have shown correlations between video quality metrics and user engagement. [25] shows the impact of video quality on user behavior using quasi experiments.

Network data from commercial IPTV is used in [31] to learn performance indicators for users QoE, where [8] uses in-network measurements to estimate QoE for mobile users. We have used the prior work done on QoE models to extract QoE metrics that matter more to clients (e.g., the re-buffering and startup delay) to study the impact of performance problems on them.

ABR algorithms: The bitrate adaptation algorithms have been studied well, [15] studies the interactions between HTTP and TCP, while [9] compares different algorithms in sustainability and adaptation. Different algorithms have been suggested to optimize video quality, in particular [23, 32] offer rate-based adaptation algorithms, where [20] suggests a buffer-based approach, and [37] aims to optimize quality using a hybrid model. Our work is complementary to these works, because while an optimized ABR is necessary for good streaming quality, we showed problems where a good ABR algorithm is not enough and corrective actions from the content provider are needed.

Optimizing video quality by CDN selection: Previous work suggests different methods for CDN selection to optimize video quality, for example [33] studies policies and methods used for server selection in Youtube, while [24] studies causes of inflated latency for better CDN placement. Some studies [26, 18, 17] make the case for centralized video control planes to dynamically optimize the video delivery based on a global view while [10] makes the case for federated and P2P CDNs based on content, regional, and temporal shift in user behavior.

7. CONCLUSION

In this paper, we presented the first Web-scale end-to-end measurement study of Internet video streaming to characterize problems located at a large content provider's CDN, Internet, and the client's download and rendering paths. Instrumenting the end-to-end path gives us a unique opportunity to look at multiple components together during a session, at per-chunk granularity, and to discover transient and persistent problems that affect the video streaming experience. We characterize several important characteristics of video streaming services, including causes for persistent problems at CDN servers such as unpopularity, sources of persistent high network latency, and persistent rendering problems caused by browsers. We draw insights into the client's download stack latency (possible at scale only via end-to-end instrumentation); and we showed that the download stack can impact the QoE and feed incorrect information into the ABR algorithm. We discussed the implications of our findings for content providers (e.g., pre-fetching subsequent chunks), ISPs (establishing better peering points), and the ABR logic (e.g., using apriori observations about client prefixes).

Acknowledgments

This work benefited from a large number of engineers from the Video Platforms Engineering group at Yahoo. We thank the group for helping us with instrumentation and patiently answering many of our questions (and hearing our performance optimization observations). Thanks to P.P.S. Narayan for supporting this project at Yahoo. This work was in part supported by the National Science Foundation grant CNS-1162112.

8. REFERENCES

[1] ActionScript 3.0 reference for the Adobe Flash. http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/FileReference.html.

[2] Apache Traffic Server. http://trafficserver.apache.org.

[3] It's latency, stupid. https://rescomp.stanford.edu/~cheshire/rants/Latency.html.

[4] Open read retry timer. https://docs.trafficserver.apache.org/en/4.2.x/admin/http-proxy-caching.en.html#open-read-retry-timeout.

[5] Sandvine: Global Internet phenomena report 2015. https://www.sandvine.com/trends/global-internet-phenomena/.

[6] Youtube statistics. https://www.youtube.com/yt/press/statistics.html.

[7] AGGARWAL, A., SAVAGE, S., AND ANDERSON, T. Understanding the performance of TCP pacing. In *IEEE INFOCOM* (2000), pp. 1157–1165.

[8] AGGARWAL, V., HALEPOVIC, E., PANG, J., VENKATARAMAN, S., AND YAN, H. Prometheus: Toward quality-of-experience estimation for mobile apps from passive network measurements. In *Workshop on Mobile Computing Systems and Applications* (2014), pp. 18:1–18:6.

[9] AKHSHABI, S., BEGEN, A. C., AND DOVROLIS, C. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *ACM Conference on Multimedia Systems* (2011), pp. 157–168.

[10] BALACHANDRAN, A., SEKAR, V., AKELLA, A., AND SESHAN, S. Analyzing the potential benefits of CDN augmentation strategies for internet video workloads. In *IMC* (2013), pp. 43–56.

[11] BRESLAU, L., CAO, P., FAN, L., PHILLIPS, G., AND SHENKER, S. Web caching and Zipf-like distributions: Evidence and implications. In *IEEE INFOCOM* (1999), pp. 126–134.

[12] CHA, M., KWAK, H., RODRIGUEZ, P., AHN, Y.-Y., AND MOON, S. I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system. In *IMC* (2007), pp. 1–14.

[13] DIMOPOULOS, G., LEONTIADIS, I., BARLET-ROS, P., PAPAGIANNAKI, K., AND STEENKISTE, P. Identifying the root cause of video streaming issues on mobile devices. In *CoNext* (2015).

[14] DOBRIAN, F., SEKAR, V., AWAN, A., STOICA, I., JOSEPH, D., GANJAM, A., ZHAN, J., AND ZHANG, H. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM* (2011), pp. 362–373.

[15] ESTEBAN, J., BENNO, S. A., BECK, A., GUO, Y., HILT, V., AND RIMAC, I. Interactions between HTTP adaptive streaming and TCP. In *Workshop on Network and Operating System Support for Digital Audio and Video* (2012), pp. 21–26.

[16] FREEDMAN, M. J., VUTUKURU, M., FEAMSTER, N., AND BALAKRISHNAN, H. Geographic locality of ip prefixes. In *IMC* (2005), pp. 13–13.

[17] GANJAM, A., JIANG, J., LIU, X., SEKAR, V., SIDDIQI, F., STOICA, I., ZHAN, J., AND ZHANG, H. C3: Internet-scale control plane for video quality optimization. In *USENIX NSDI* (2015), pp. 131–144.

[18] GEORGOPOULOS, P., ELKHATIB, Y., BROADBENT, M., MU, M., AND RACE, N. Towards network-wide QoE fairness using OpenFlow-assisted adaptive video streaming. In *ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking* (2013), pp. 15–20.

[19] GHOBADI, M., CHENG, Y., JAIN, A., AND MATHIS, M. Trickle: Rate limiting YouTube video streaming. In *USENIX Annual Technical Conference* (2012), pp. 17–17.

[20] Huang, T.-Y., Johari, R., McKeown, N., Trunnell, M., and Watson, M. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *ACM SIGCOMM* (2014), pp. 187–198.

[21] Jain, M., and Dovrolis, C. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. In *ACM SIGCOMM* (2002), pp. 295–308.

[22] Jiang, J., Sekar, V., Stoica, I., and Zhang, H. Shedding light on the structure of internet video quality problems in the wild. In *CoNext* (2013), pp. 357–368.

[23] Jiang, J., Sekar, V., and Zhang, H. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE. In *CoNext* (2012), pp. 97–108.

[24] Krishnan, R., Madhyastha, H. V., Srinivasan, S., Jain, S., Krishnamurthy, A., Anderson, T., and Gao, J. Moving beyond end-to-end path information to optimize CDN performance. In *IMC* (2009), pp. 190–201.

[25] Krishnan, S. S., and Sitaraman, R. K. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. In *IMC* (2012), pp. 211–224.

[26] Liu, X., Dobrian, F., Milner, H., Jiang, J., Sekar, V., Stoica, I., and Zhang, H. A case for a coordinated Internet video control plane. In *ACM SIGCOMM* (2012), pp. 359–370.

[27] Paxson, V., and Allman, M. Computing TCP's Retransmission Timer. RFC 2988 (Proposed Standard), 2000. Obsoleted by RFC 6298.

[28] Plissonneau, L., and Biersack, E. A longitudinal view of HTTP video streaming performance. In *Multimedia Systems Conference* (2012), pp. 203–214.

[29] Poese, I., Uhlig, S., Kaafar, M. A., Donnet, B., and Gueye, B. IP geolocation databases: Unreliable? *SIGCOMM Computer Communications Review*, 2 (2011), 53–56.

[30] Sen, S., Rexford, J., and Towsley, D. Proxy prefix caching for multimedia streams. In *IEEE INFOCOM* (1999), pp. 1310–1319.

[31] Song, H. H., Ge, Z., Mahimkar, A., Wang, J., Yates, J., Zhang, Y., Basso, A., and Chen, M. Q-score: Proactive service quality assessment in a large IPTV system. In *IMC* (2011), pp. 195–208.

[32] Tian, G., and Liu, Y. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *CoNext* (2012), pp. 109–120.

[33] Torres, R., Finamore, A., Kim, J. R., Mellia, M., Munafo, M. M., and Rao, S. Dissecting video server selection strategies in the YouTube CDN. In *International Conference on Distributed Computing Systems* (2011), pp. 248–257.

[34] Weaver, N., Kreibich, C., Dam, M., and Paxson, V. Here be web proxies. In *PAM* (2014), pp. 183–192.

[35] Xu, X., Jiang, Y., Flach, T., Katz-Bassett, E., Choffnes, D., and Govindan, R. Investigating Transparent Web Proxies in Cellular Networks. In *Proc. of PAM* (2015).

[36] Yin, H., Liu, X., Qiu, F., Xia, N., Lin, C., Zhang, H., Sekar, V., and Min, G. Inside the bird's nest: Measurements of large-scale live VoD from the 2008 olympics. In *IMC* (2009), pp. 442–455.

[37] Yin, X., Jindal, A., Sekar, V., and Sinopoli, B. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM* (2015), pp. 325–338.

[38] Yu, M., Greenberg, A., Maltz, D., Rexford, J., Yuan, L., Kandula, S., and Kim, C. Profiling network performance for multi-tier data center applications. In *USENIX NSDI* (2011), pp. 57–70.