CSI: Inferring Mobile ABR Video Adaptation Behavior under HTTPS and QUIC

Shichang Xu
University of Michigan

Subhabrata Sen
AT&T Labs – Research

Z. Morley Mao
University of Michigan

Abstract

Mobile video streaming services have widely adopted Adaptive Bitrate (ABR) streaming to dynamically adapt the streaming quality to variable network conditions. A wide range of third-party entities such as network providers and testing services need to understand such adaptation behavior for purposes such as QoE monitoring and network management. The traditional approach involved conducting test runs and analyzing the HTTP-level information from the associated network traffic to understand the adaptation behavior under different network conditions. However, end-to-end traffic encryption protocols such as HTTPS and QUIC are being increasingly used by streaming services, hindering such traditional traffic analysis approaches.

To address this, we develop *CSI* (Chunk Sequence Inferencer), a general system that enables third-parties to conduct active measurements and infer mobile ABR video adaptation behavior based on packet size and timing information still available in the encrypted traffic. We perform extensive evaluations and demonstrate that *CSI* achieves high inference accuracy for video encodings of popular streaming services covering various ABR system designs. As an illustration, for a popular mobile video service, we show that *CSI* can effectively help understand the video QoE implications of network traffic shaping policies and develop optimized policies, even in the presence of encryption.

ACM Reference Format:
Shichang Xu, Subhabrata Sen, and Z. Morley Mao. 2020. CSI: Inferring Mobile ABR Video Adaptation Behavior under HTTPS and QUIC. In *Fifteenth European Conference on Computer Systems (EuroSys '20), April 27–30, 2020, Heraklion, Greece.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3342195.3387558

1 Introduction

Mobile video streaming is increasingly popular, already accounting for 60% of mobile data traffic, and is predicted to grow to 78% by 2021 [40]. HTTP-based Adaptive Bit Rate (ABR) streaming (predominantly HLS [75] and DASH [31]) has been widely adopted in industry for delivering satisfactory Quality of Experience (QoE) over dynamic cellular network conditions. The server encodes each video into multiple versions with different picture quality levels and encoding bitrates (with higher bitrates for higher-quality encodings) called *tracks*, and splits each track into shorter *chunks*, each representing a few seconds worth of playback content (Figure 1). During streaming, the client downloads a metadata file (called *manifest*) from the server which contains information about all the tracks and their corresponding chunks. Then it downloads (using HTTP) and plays individual chunks in order of increasing playback indexes (ie., playback position in the video). The streaming is adaptive to time-varying network conditions – for each playback index, the client dynamically selects one among the "ladder" of available tracks for download, based on prevailing network conditions and custom complex adaptation algorithms [56, 58, 69, 76, 84].

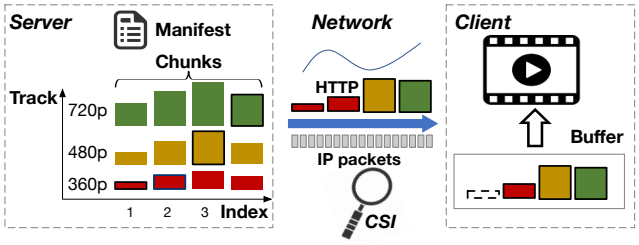


Figure 1. ABR streaming overview

The ABR adaptation logic needs to deal with time-varying network conditions and make complex tradeoffs between (sometimes) competing QoE requirements (*e.g.,* stream high-quality video tracks/chunks that require high network bandwidths, but still minimize stalls that can occur when the network bandwidth drops), and there is no single or simple "optimal" adaptation logic design. Different mobile streaming systems adopt different ABR strategies and tradeoffs. Their clients exhibit substantially different adaptation behaviors even under the same network conditions, and they keep evolving over time [52, 80, 83].

In this work, we develop a novel and general system, CSI (Chunk Sequence Inferencer), that provides the capability to independently conduct active measurements and infer the adaptation behavior and delivered QoE of third party mobile video services, for the increasingly common but challenging use case

EuroSys '20, April 27–30, 2020, Heraklion, Greece

ACM ISBN 978-1-4503-6882-7/20/04.
https://doi.org/10.1145/3342195.3387558

where these services use encrypted (HTTPS/QUIC) communications between the client and the server. Such video streaming systems are typically complex, highly customized and closed source, making it challenging to understand their adaptation designs. To address this, for a specific streaming service and video asset, *CSI* streams the video under specific network conditions of interest (§4). It then analyzes the associated network traffic to infer (1) the *identity* of each downloaded chunk, *i.e.,* the index, the track it belongs to, whether it is an audio or video chunk and (2) the time when each chunk is downloaded. From such information, QoE metrics including displayed video quality and stall occurrences can be further analyzed.

CSI should be particularly useful to a wide range of third-party entities who desire to independently evaluate and understand the adaptation behavior of popular mobile video services. Examples include: (1) mobile network providers desiring to understand whether popular video services are able to deliver satisfactory QoE over their network, and to inform the design of traffic shaping policies that can support delivering good QoE while making efficient use of network resources. (2) video services desiring to conduct comparative analysis of the QoE performance of their service with other video services for calibration purposes and to understand how to improve their adaptation design , and (3) independent testing services and researchers interested in profiling the adaptation behavior of popular video services, to identify potential deficiencies and drive better designs. A common approach in these cases is to perform active measurements: testers stream videos on the target ABR service in certain network conditions (either in the lab or in the wild). They collect and analyze the network traffic trace generated from the testing to study player behavior dynamics and characterize delivered QoE. Without a tool like *CSI*, it is extremely challenging to conduct such measurements and study the adaptation behavior of commercial streaming services from encrypted network traffic.

One key challenge *CSI* addresses is that popular streaming apps [34, 36] are increasingly adopting end-to-end encryption protocols like HTTPS and QUIC and encrypt packet payloads. Existing active measurement approaches depend on being able to extract application level information from the network traffic between the client and server, and determine the identity of each downloaded chunk using information in the corresponding HTTP request URL ([45, 57, 68, 83]). Such approaches are no longer viable in the presence of traffic encryption as all HTTP level information, including the request URL information is encrypted. Even workarounds such as Man-In-The-Middle (MITM [11]) proxies are becoming increasingly less effective (see §8). Machine learning based proposals [52, 70, 72–74, 79] also have various limitations, including requiring labeled QoE data to train models, which is hard to obtain in general (§8).

To address this challenge, *CSI* works by inferring the identities of downloaded chunks from the encrypted network traffic (§3). It leverages the key insight that for commonly used TLS traffic encryption, the data volume sent over the network is similar to the corresponding object size before encryption. Before running the active measurement, *CSI* obtains the sizes of all chunks across all tracks for the target test video. After running the video streaming session, it analyzes the IP-level information still available in the encrypted traffic - specifically the packet sizes and timing information. Conceptually, it analyzes the encrypted traffic to first infer the packets corresponding to client requests for chunks, and then estimates the size of each downloaded chunk based on the traffic downloaded between consecutive requests. It then uses the chunk size as a fingerprint to identify the corresponding playback index and track for each downloaded chunk.

Our key contributions are:

- Foundational insights (§3). We perform extensive measurements and develop two key insights that demonstrate the feasibility of inferring chunk identities from encrypted traffic. (1) Downloaded object sizes can be accurately inferred from associated encrypted packets (§3.2). (2) For the increasingly commonly used Variable Bitrate (VBR) encoding, even with a relatively short sequence of chunk sizes, consisting of a mixture of chunks from different tracks, the identity of each chunk in the sequence can still be identified with high accuracy (§3.3).
- Design of *CSI* (§4). *CSI* enables automated and repeated active measurements for understanding the adaptation behavior and delivered QoE of commercial mobile video streaming under various network conditions. *CSI* automates the measurement process including performing network emulation, player UI instrumentation, data collection and analysis.
- Inference algorithm that is a key component of *CSI* (§5). It is designed to cover a range of common ABR system designs. To efficiently identify the chunk sequence that matches size information from the traffic, *CSI* formulates the matching problem as a shortest path graph search. *CSI* also addresses additional challenges introduced by QUIC's unique properties, such as the stream multiplexing feature [43].
- Evaluation (§6). We perform extensive evaluations and demonstrate that *CSI* achieves high inferencing accuracy (1) across different chunk size variability across 6 popular services (2) across ABR systems with different designs. In addition, the analysis is fast, typically taking only a few seconds to analyze a 10 min long video session.

- Use case (§7). We use Hulu as an example service and illustrate how *CSI* can be used in practice to help understand the QoE implications of parameter settings in token-bucket based traffic shaping policies and derive optimized shaping policies for mobile networks (§7).

CSI is currently being incorporated into a popular open-source mobile video streaming analysis toolkit widely used in the industry. A new version of the toolkit including *CSI* is being prepared for public release.

The design of *CSI* was primarily motivated by the need to analyze complex adaptation behavior of closed-source mobile apps in highly variable network conditions typical of cellular networks. However, *CSI* can also be used for less challenging scenarios such as more stable broadband home networks and web-based ABR streaming.

2 Motivation

In this section, we review existing approaches to monitor adaptation behavior of ABR streaming services and detail the technical challenges introduced by the adoption of traffic encryption. This motivates our effort to develop *CSI*.

Use of active measurements. Due to the closed-source nature of the proprietary mobile streaming apps, third parties typically resort to black-box active measurements to analyze their adaptation behavior and resulting QoE. This can be done either with carefully crafted in-lab network emulation (*e.g.,* replaying varying bandwidth traces collected from real networks *etc.*) or in-the-wild real network testing. In each test, they monitor the network traffic during video playback. Specifically, they analyze across time what chunks are downloaded and the corresponding download times (*e.g.,* the 5th chunk in playback order is selected from the 3rd track and downloaded at a certain time). Using such information, testers can estimate the displayed video quality, measure its variation across time, and analyze stall events during playback.

Challenges introduced by traffic encryption. Existing video streaming analysis techniques [45, 57, 67, 83] rely on parsing HTTP requests information in the network traffic, *e.g.,* URLs, to identify the identity of downloaded chunks. However, with the adoption of encryption protocols, such information is encrypted and no longer available, making existing techniques no longer viable.

In this work, we focus on the two dominant encryption protocols used in video streaming, *i.e.,* HTTPS (*e.g.,* Netflix [34]) and QUIC (*e.g.,* Youtube [41]). QUIC is a UDP-based encrypted transport protocol with feature enhancements designed for better performance [61, 65]. HTTP-over-QUIC is being standardized as HTTP/3 [43] and attracted wide interest from the industry. HTTPS and QUIC cover the vast majority of popular commercial streaming services, and hence we focus on them in this paper.

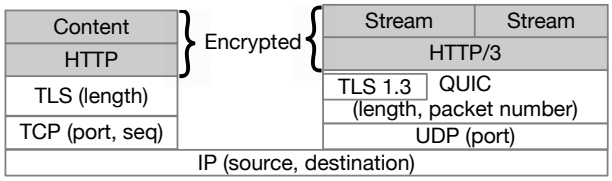


Figure 2. HTTPS/QUIC network stack (and available information)

Symbol	Description
C_i	Chunk corresponding to the i^{th} request
M_i, T_i, I_i, S_i	Media type, track, index and size of C_i
$\tilde{S}_i$	The estimated size of C_i based on traffic
S_{ak}	The size of an audio chunk in the kth audio track

Table 1. The notation used in this paper

As shown in Figure 2, HTTPS and QUIC both use Transport Layer Security(TLS) [3] to encrypt application layer data. Only very limited information can be obtained by monitoring this encrypted traffic, including IP packet timing, IP addresses, TCP/UDP port number, TLS record length in HTTPS and payload length in QUIC. Additionally, during the TLS handshake phase, the server domain name can be obtained from the Server Name Indication (SNI) extension sent by the client. However, all application payload information such as HTTP request URL and response cannot be observed, defeating traditional traffic analysis techniques.

It is worth mentioning that compared to HTTPS, QUIC has some unique properties which make its analysis even more challenging: (1) A retransmitted HTTPS packet can be detected from the SEQ number in the underlying TCP header. But for QUIC, each packet carries a new packet number, even those carrying retransmitted data. This makes it difficult to identify retransmitted QUIC packets and therefore harder to get an accurate estimation of the payload size from the observed network traffic (§3.2). (2) QUIC supports multiplexing multiple streams for multiple objects at the same time within the same connection. This makes it more difficult to infer the sizes of individual objects transmitted on a QUIC connection. We will discuss later in §5 how we address these challenges.

3 CSI Overview: Using Sizes as Fingerprint

In this section, we describe the high-level approach of *CSI* and the key insights underlying it. We will describe practical challenges and more *CSI* design details later in §4 and §5.

In the following, we denote the chunk corresponding to the i^{th} request as C_i, its media type (audio or video[1]), track, index and size as M_i, T_i, I_i and S_i respectively (in Table 1).

[1]Some ABR services multiplex audio and video content together and each chunk contains both video and associated audio content. For such services, we consider all chunks as video chunks.

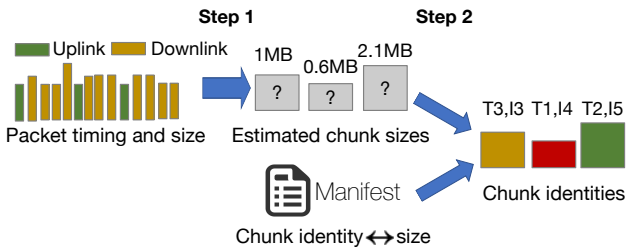


Figure 3. Proposed analysis approach for encrypted traffic (T: track, I: index, Tx,Iy means the y^{th} chunk in the x^{th} track.)

3.1 High-level solution

In advance of running the actual streaming test, *CSI* gathers the sizes of all chunks from all tracks of the test video. Note this only needs to be performed once for each test video. Then *CSI* runs the actual test, streaming the test video on the target service under various network conditions. During the test, *CSI* captures encrypted traffic going through the device. Afterwards, *CSI* infers downloaded chunk identities from the encrypted traffic using a 2 step process.

Step 1. Combining packet sizes and timing information, *CSI* infers (i) packets corresponding to HTTP requests from the player to the server in the upstream direction, and (ii) the set of packets in the downstream direction that correspond to the response (*i.e.,* chunk) for each request. From this, *CSI* estimates the sizes of the downloaded chunks. Assuming a session with n chunk downloads, we denote the estimated chunk sizes as $(\tilde{S}_i)_{i=1}^n$ to distinguish them from the corresponding actual chunk sizes $(S_i)_{i=1}^n$ *CSI* is trying to infer.

Step 2. Given the estimated downloaded chunk sizes $(\tilde{S}_i)_{i=1}^n$, *CSI* identifies the chunk sequence $(C_i)_{i=1}^n$ (where different chunks can belong to different tracks) whose size sequence $(S_i)_{i=1}^n$ most closely matches $(\tilde{S}_i)_{i=1}^n$ as the likely set of chunk downloaded in the session.

The feasibility of such an inference approach depends on two key insights that we derive based on extensive measurements. (1) For encrypted traffic, given a group of packets associated with downloading a chunk, we can estimate chunk sizes with relatively high accuracy. (2) Given the achievable accuracy of chunk size estimation, we can accurately identify the chunk identity based on the estimated size.

Next, we describe our measurement analysis that leads to the above two insights.

3.2 Accuracy of chunk size estimation

We first investigate estimating the chunk size from a set of encrypted packets associated with downloading the chunk.

We try to reduce potential inaccuracies as much as possible. For HTTPS, we remove retransmitted packets based on the SEQ number in the underlying TCP header (§2). We then estimate the chunk size as the sum of the TLS payload lengths in the remaining packets (excluding IP/TCP/TLS headers in Figure 2). For QUIC, we estimate the chunk size as the sum of the QUIC payload lengths in all packets(excluding IP/UDP/QUIC headers).

To evaluate the accuracy of the size estimation, we build an Android app with Cronet [16], an HTTP library that supports both HTTPS and QUIC, and download chunks with sizes ranging from 50KB to 1MB using the two protocols. We capture the associated network traffic and estimate download file sizes following the above steps. We repeat the experiment in different mobile network environments. Each object is downloaded 100 times in total.

We compare the estimated size $\tilde{S}_i$ with the ground truth S_i to measure the estimation accuracy. We find the estimation is quite accurate for both protocols: the maximal error is only 1% and 5% for HTTPS and QUIC respectively. For HTTPS, this error is mainly caused by potential TLS overheads. For QUIC, the error rate is slightly higher since (1) we are unable to identify and eliminate retransmitted QUIC packets based on the packet header information still available (§2), (2) QUIC is built on top of UDP and implements signaling such as congestion control and flow control within the encrypted QUIC payload. We are unable to eliminate the associated traffic overhead of such signaling as well.

Based on the measurements, we have the following relationship between $\tilde{S}_i$ and S_i.

$$S_i \leq \tilde{S}_i \leq (1+k)S_i \qquad \text{(Property (1))}$$

k represents the maximal estimation error (1% for HTTPS and 5% for QUIC).

[49, 55, 59, 63, 71, 77, 78] also show that traffic analysis can be performed to infer HTTPS payload sizes, as it does not use TLS padding [17, 18] to obfuscate payload size, likely due to significant associated data overhead and network resource inefficiencies [53]. Our findings corroborate this observation for HTTPS, and further show that a similar observation also holds for QUIC.

3.3 Accuracy of chunk identification

Next, we explore the chunk size variability of video streaming services to understand whether it is possible to use estimated chunk sizes (given the achievable estimation accuracy in §3.2) to accurately identify downloaded chunks among all encoded chunks.

Streaming services traditionally mainly used Constant Bitrate (CBR) encoding and encoded the video using fixed bitrates for each track. For the CBR case, the track that a downloaded chunk belongs to can be trivially identified based on its size, as the sizes of all chunks in each track are tightly clustered around and different tracks have very different chunk sizes. Recently, services are increasingly adopting VBR encoding [58, 76, 77, 83] due to its higher encoding efficiency [64]. The encoder allocates higher bitrates to encode the chunks corresponding to complex scenes and lower

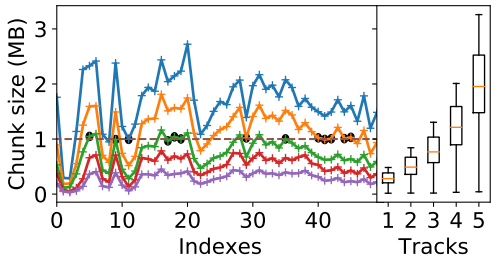


Figure 4. Chunk sizes of a Youtube video (PASR 2.6).

bitrates to chunks corresponding to simpler scenes. This results in size variance even for chunks in the same track. As an example, we plot the chunk sizes of a popular Youtube video (*Adele-Hello*) in Figure 4. We can see that in a given track, chunks at different places in the video exhibits significant size diversity. Such diversity can be helpful in identifying the identity of a chunk based on its size: assuming each chunk has a different size, we can build a unique mapping between the size and chunk identity. However, such VBR encoding also presents challenges in such chunk identification. For example, in Figure 4, some chunks from track 3 can even have similar sizes with chunks from track 5. Such size overlaps make it challenging to identify the track that a downloaded chunk belongs to based on its size information.

To evaluate the feasibility of using chunk sizes as a fingerprint to identify chunks with different VBR encodings, we create videos with different size variabilities for our analysis. We define *PASR* (peak-to-average size ratio) to be the ratio between the 95^{th} percentile chunk size and the average chunk size within a track. We use FFmpeg [4] to encode the commonly used Big Buck Bunny (BBB) test video [1] into 10 different ABR streams (each with a ladder of tracks) with PASR values ranging from 1.1 to 2.0 (increasing at 0.1). For each stream, we encode the video into six tracks with resolutions ranging from 144p to 1080p following the setting suggested in [15]. When encoding the tracks, we follow the three-pass encoding procedure in [50] and configure parameters `-maxrate` and `-b:v` to achieve desired PASR in each setting. We then use MP4Box [12] to split each track into 5-sec chunks.

Q1: Can we uniquely determine the identity of a single chunk given its estimated size?

If chunk sizes can be accurately obtained without any error, two chunks C_i and C_j are indistinguishable based on the size information only when their sizes $S_i = S_j$. When there is potential inaccuracy in size estimation (which is the case for encrypted traffic), assuming the maximum error in size estimation is k, two chunks C_i and C_j are indistinguishable based on the size information if $\frac{S_j}{1+k} \leq S_i \leq (1+k)S_j$, as they can be potentially estimated to have the same size. We define such two chunks to be *similar* with threshold k. Recall that k is 1% for HTTPS and 5% for QUIC. We define a chunk to be *unique* if there is no other chunk similar to it from any track in the video.

We find that even with a k of 1%, all encoded videos have less than 0.1% of unique chunks regardless of the encoding PASR. In other words, 99.9% of chunks have at least 1 other chunk with a similar size. With a relatively low PASR, there is less variability in sizes of chunks in the same track, and therefore multiple chunks in the same track are more likely to have similar sizes. With a relatively high PASR, sizes in the same track span larger ranges, and thus chunk sizes in different tracks are more likely to overlap. In either case, it is hard to guarantee that a single chunk has a unique size among all the chunks across all tracks in the video. Taking the video in Figure 4 as an example, we highlight chunks with size 1MB ($k = 1\%$). We can see that multiple chunks in both the same track and different tracks have similar sizes.

The above analysis shows that starting from a certain estimated size for a single chunk, it is very challenging to uniquely identify the corresponding chunk regardless of the encoding.

Solution. To reduce the ambiguity in chunk identification, we leverage one common property during ABR streaming: the indexes (*i.e.,* playback positions in the track) of the downloaded chunks should grow contiguously.

$$I_i = I_{i-1} + 1 \qquad \text{(Property (2))}$$

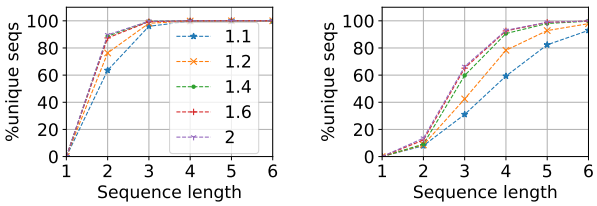
With this constraint, we can combine the estimated size information of multiple consecutive chunks to jointly determine their identities. Note that we do not assume I_1 to be 1 as the playback might not start from the beginning of the video in the test (*e.g.,* resuming from the end point of the previous test).

Q2: Can we uniquely determine the chunk identities given the estimated sizes of multiple consecutive chunks?

We denote a *chunk sequence* as a series of chunks $(C_i)_{i=1}^n$ where the indexes of the chunks grow sequentially (they can be from different tracks). We consider two chunk sequences $(C_{1i})_{i=1}^n$ and $(C_{2i})_{i=1}^n$ to be *similar* if every pair of chunks C_{1i} and C_{2i} in these two sequences are *similar*. A sequence is considered to be *unique* if there is no other sequence *similar* to it.

The total number of chunk sequences increases exponentially when the sequence length increases[2]. However, as shown in Figure 5, for all VBR encodings, the percentage of unique sequences decreases dramatically when the sequence length increases by even small values. Even with a PASR as low as 1.1, 99.9% of 3-chunk sequences are unique with k of 1%, 92.6% of 6-chunk sequences are unique with k of 5%. This implies that for many video services, even those with relatively small chunk size variance in a track, given

[2]Each sequence is uniquely determined by the index of the first chunk and the tracks of all chunks in the sequence



(a) HTTPS $k = 1\%$ **(b)** QUIC $k = 5\%$

Figure 5. Relation between chunk sequence length and percentage of unique sequences for videos with different PASR. Each line represents results for a PASR.

a sequence of only a small number of contiguous chunks, we can uniquely determine the identity of each chunk in the sequence with a high probability.

In summary, our measurement results demonstrate the feasibility of the proposed *CSI* approach. We will perform a similar analysis on the encodings of commercial streaming services later in §6 to validate the generality of the above conclusion.

The robustness of the proposed chunk identification approach derives from its dependence on some enduring behaviors and features of streaming systems and encryption protocols which are robust and difficult to alter in practice. (1) VBR encoding is increasingly adopted by popular services due to its higher encoding efficiency. Our analysis shows that even with a low variance in chunk sizes in a track (*e.g.,* PASR 1.1), with the sizes of 6 consecutive chunks, there is a high probability (*e.g.,* 92.6%) that we can uniquely determine the identity of each chunk in the sequence. (2) In practice, it is challenging for streaming services to adopt extensive padding in the traffic for the purpose of obfuscating downloaded object size information, as it would cause significant overhead in network traffic and cause potential degradation in streaming QoE. (3) If the service adopts CBR encoding, each track will have a distinct chunk size. As discussed earlier, the track of each downloaded chunk will be even simpler to identify based on chunk size information.

4 CSI System Design

Further to the description in §3.1, we present a concrete system design of *CSI* (Chunk Sequence Inferencer) in Figure 6. The key components include the *controller*, the *gateway* and the *mobile device*. The controller automates measurements and analyzes the collected data to infer the streaming behavior of the tested mobile video service running on the mobile device. The gateway performs traffic shaping to emulate different network conditions and collects traffic passing through. *CSI* also leverages *web browsers* on mobile devices to collect required metadata about the test video, *e.g.,* chunk sizes across the tracks. We next detail how to use *CSI* to study the adaptation behavior of proprietary closed-source mobile streaming apps.

4.1 Collecting chunk sizes from all tracks

Recall (§3.1) that, in advance of running the actual streaming experiment, *CSI* needs to gather the sizes of all chunks from all tracks of the test video. Such information is essential for the subsequent step of inferring the identities of downloaded chunks based on the chunk sizes. *CSI* gets such information from the corresponding ABR manifest which specifies information on encoding tracks and chunks (see Figure 1). Clients download the manifest at the beginning of playback to get necessary information to fetch chunks. Many manifests directly specify the sizes of all chunks[76, 83]. *CSI* parses the manifest to get the size information. In other cases, manifests only provide the URLs of all chunks. In such cases, *CSI* sends HTTP HEAD requests to query chunk sizes given the chunk URL information in the manifest. Note this step only needs to be performed once for each test video.

Depending on the streaming service features, *CSI* obtains the manifest using one of the following approaches.

Approach 1: If the streaming service supports browser-based streaming (most commercial services do), *CSI* plays the tested video using a browser instead of mobile apps to get the manifest. The reason is that browsers provide developer tools (*e.g.,* Chrome [2], Firefox [5]) to inspect all network activities including the decrypted payload in encrypted traffic. We find that unlike browsers, mobile apps do not provide such access, but they typically share the server-side setting (*e.g.,* track encoding and chunk sizes) with browser-based players. Thus, even though *CSI* focuses on revealing the adaptation behavior of mobile apps which are the predominant vehicle for consuming video content on mobile devices [38, 42, 44], *CSI* leverages the browser to glean the manifest.

Note that the browser-based streaming here is used just for getting the manifest metadata, and we cannot use analysis of ABR streaming behavior for the browser-based player for understanding the behavior of native mobile apps: different streaming platforms tend to have very different implementations with different client adaptation logic designs (*e.g.,* Java-based player libraries such as ExoPlayer [7] on Android, AV Foundation framework on iOS, and Javascript-based libraries such as Shaka Player [8] in browsers) and thus different performance characteristics. As an illustration, we stream a video on Youtube with a stable network bandwidth of 1 Mbps on and observe the player behavior using information from stats-for-nerds displayed on the screen [28]. We find Youtube on different platforms selects tracks with different encoding resolutions (480p for web and 360p for Android and iOS) and has different maximum buffer durations (90s, 120s and 60s for web, Android and iOS respectively), which could lead to very different performance. Therefore, we only utilize the additional information access offered by developer tool facilities of browsers to obtain the manifest

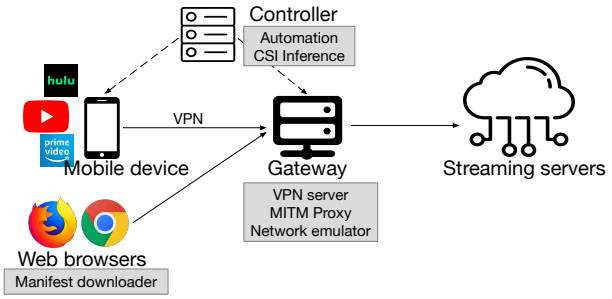


Figure 6. The system architecture of *CSI*

file. The actual testing is still done with the target player platform.

Approach 2: An alternate approach is to glean the manifest from native apps using protocols (or platforms or devices) where intercepting encrypted connections is feasible (see more in §8). For example, there is no known technique to intercept QUIC connections. But when QUIC traffic is blocked, players typically fall back to providing service over HTTPS. Thus, *CSI* blocks QUIC traffic on the gateway and uses Man-In-The-Middle (MITM [11]) approaches to intercept HTTPS connections to get the manifest where feasible. It then uses the manifest information to help analyze the streaming behavior over QUIC, which can have substantially different performance characteristics [65]. Another use case is that newer Android systems no longer trust user-installed certificates and prevent MITM. In such a case, *CSI* could perform MITM interception on older systems to get the manifest, then use it to study the adaptation behavior of newer versions of players on the latest Android systems.

In addition, for platforms such as Youtube, there are publicly available tools (*e.g.,* youtube-dl [30]) to download media files of individual tracks. *CSI* could parse the file to obtain individual chunk sizes within the track.

4.2 Streaming video and collecting data

After per-chunk size information is collected for all tracks, *CSI* next conducts streaming experiments with the target player for which testers desire to understand the adaptation behavior and quantify QoE. *CSI* leverages the UI Automator testing framework [24] to interact with the video player and play the test video. The controller also controls the gateway to perform traffic shaping using the Linux tool *tc* according to the bandwidth trace provided by testers. The traffic from the device is routed through the gateway using a virtual private network (VPN) connection. The gateway captures the transmitting encrypted traffic (and therefore the packet size and timing information).

For some services, *CSI* also collects the identities of displayed chunks by analyzing the device screen. This is optional and helps improve the inference accuracy of *CSI*. We will perform evaluations to explore the inference accuracy with/without such information in §6. The information about displayed chunks can be collected in different approaches. For example, the stats-for-nerds player overlay option in YouTube shows the resolution of the currently displayed track [28]. Netflix has test patterns that encode the track number and frame number as an overlay on the videos [23]. Combining with OCR techniques (*e.g.,* Tesseract OCR [22] and GOCR [6]) to extract such information from the screen, *CSI* gleans information on when and what chunks are displayed on the device.

Note that even when we are able to know the displayed chunks, we need to still infer other information critical to understanding performance, such as the time when each displayed chunk was actually downloaded. Players typically maintain a buffer to absorb network variance and download chunks ahead of the playback time. The network condition when the player downloads a chunk can be totally different from the network condition when the player later displays the chunk. To understand the player adaptation logic, it is therefore essential to identify the time when each displayed chunk was downloaded and associate it with the network condition at that time.

4.3 Performing analysis

Given the collected information, *CSI* infers when and what chunks are downloaded by the client player. It then computes the client buffer occupancy across time and analyzes streaming QoE including video quality and stalls. Combining with information regarding the available network bandwidth across time, testers can develop an understanding of how clients react to different network conditions and gain more insights about the design of the tested service. We detail the *CSI* inference algorithm in the next section.

5 CSI Inference

Earlier in §3, we described the high-level analysis approach of *CSI* to infer the identities of downloaded chunks from the encrypted traffic. In this section, we describe the challenges in developing it into a practical algorithm and present our solution.

5.1 Challenges

To implement the inference into a practical technique, we need to surmount the following challenges.

Challenge 1: Transport multiplexing (MUX). In Step 1 (§3), we desire to identify packets associated with each chunk and then estimate the individual chunk sizes. However, for ABR systems using QUIC and with separate audio tracks, when players send requests to download audio and video chunks, video and audio traffic are multiplexed on the same QUIC connection, making it challenging to separate the corresponding packets. In contrast, on each connection, HTTPS does not send the next chunk request until the current chunk

Design type	Audio track	Protocol	Transport MUX
CH	Combined	HTTPS	N
SH	Separate	HTTPS	N
CQ	Combined	QUIC	N
SQ	Separate	QUIC	Y

Table 2. The ABR streaming system design types

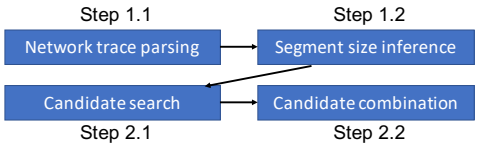


Figure 7. Algorithm overview

is finished downloading, making it easier to separate traffic for different chunks.

Challenge 2: Large search space. In Step 2, we desire to search for likely chunk sequences that closely match with the estimated sizes from the network traffic. Existing traffic analysis work [45, 57, 67, 83] builds fingerprints for each website or app. However, such approaches cannot be directly applied to our problem, as the search space consisting of all possible chunk sequences increases exponentially with the sequence length n. For example, if we stream a 10 min video with 5 video tracks and 5 s second chunk duration, the player downloads 125 chunks and there can be $5^{125} = 7 \times 10^{83}$ potential chunk sequences. Building fingerprints for all combinations and performing exhausted searches quickly become infeasible as the number of chunks grows. It is therefore essential to be able to efficiently hone in on the chunk sequence that matches (as per Property (1)) with the sequence of estimated sizes $\tilde{S}_i$ from Step 1.

5.2 Observations

Before we dive into the detailed algorithm we develop to address the above challenges, we first describe a few observations that motivate our design.

Popular ABR streaming systems show high diversity in various aspects of the system. We categorize all ABR streaming designs into 4 different types based on the choice on the following 2 key factors.

- Combined (**C**) or Separate(**S**) audio/video: whether the server encodes the audio and video content into combined tracks, or encode them as separate tracks.
- HTTPS(**H**) or the QUIC(**Q**) protocol.

This leads to 4 different design types we shall refer to as {S/C}{H/Q} in Table 2. Various popular video streaming services fall into different types, *e.g.,* Amazon Video iOS (SH), Hulu Android(SH) and YouTube Android (both SH and SQ).

We perform measurements on popular streaming services and observe that they share common design practices in the following aspects.

- When separate audio tracks exist, popular services commonly use CBR to encode audio content, resulting in almost fixed-size chunks in each audio track (typically there are only 1 or a few audio tracks). In this paper, for simplicity of exposition, we assume all audio chunks in the kth audio track have a constant size
- On each HTTPS/QUIC connection, players send at most 1 outstanding video chunk request and 1 outstanding audio chunk request at any time. The likely reason for such behavior is that as the congestion control is performed at the connection level, issuing more requests one the same connection does not increase the connection's share of available network bandwidth. Instead, it increases the contention and slows down the download of each chunk. This can increase the potential for stalls and is therefore not preferable. In this paper, we make such an assumption for requests on each connection, in line with our observation of behaviors of popular apps. Note that we do not make assumptions on the number of TCP connections, as apps in practice may open multiple connections and download multiple chunks concurrently.

5.3 Algorithm

We next detail the developed algorithm to identify downloaded chunks from the encrypted network traffic. We first present *CSI* for the 3 design types that do not have transport MUX (*i.e.,* SH, CH and CQ) and are relatively easier to analyze. Then we present *CSI* for the remaining more complex design type that uses transport MUX (*i.e.,* SQ).

5.3.1 ABR designs without transport MUX

Figure 7 presents a more detailed breakdown of the 2 steps mentioned in §3 for ABR design types that do not use transport MUX.

Step 1.1 *CSI* parses the network trace and collects the video streaming related packet information. It identifies video connections using the server hostname from the SNI during the handshake, e.g. "googlevideo.com" for YouTube. When SNI information is missing, CSI infers server hostnames based on DNS queries or server IP addresses.

Step 1.2 In the absence of transport MUX, on each connection, the player does not send the next request until the current chunk is fully downloaded. Thus *CSI* groups downlink traffic between the two consecutive requests and estimates the chunk size as in §3.2. For HTTPS, the request packets can be differentiated from uplink ACK packets using the SEQ number in the TCP packet header. For QUIC traffic, using the instrumentation and setup in §6.2, we find the ACK packets have sizes smaller than 80 bytes, while the request packets are much larger. Thus, *CSI* uses the packet size to differentiate them.

Step 2 Now that *CSI* has the estimated size sequence $(\tilde{S}_i)_{i=1}^n$, it needs to determine the chunk sequence $(C_i)_{i=1}^n$ that satisfies the size constraints in Property (1) for each i and the contiguous index constraint in Property (2). Notice that the search space size of all possible contiguous chunk sequences increases exponentially ($O(T^n)$, T: total number of tracks) with the sequence length. To perform the search efficiently, *CSI* solves it using a two-level hierarchical approach. It first searches for the chunks matching each individual estimated size $\tilde{S}_i$ separately, then combines the chunks for different requests into contiguous sequences by modeling the search into the shortest path problem in a graph. Using the Dijkstra's algorithm [51], the problem can be solved in $O(n^2)$.

We use SH as a representation to explain in detail how *CSI* performs the two-layer search.

Step 2.1 For each $\tilde{S}_i$, *CSI* searches across all video tracks and locates the chunks with actual sizes satisfying Property (1). We denote the m video chunks that match $\tilde{S}_i$ as chunk candidates $\{C_{i1}, ..., C_{im}\}$. As SH has separate audio tracks, C_i could also be an audio chunk. We mark the possibility of C_i to be an audio chunk (A_i) as true if $\tilde{S}_i$ and a certain audio chunk size S_{ak} satisfies Property (1). The actual downloaded chunk C_i will be among all these possible candidates.

Step 2.2 *CSI* combines the candidates for different requests and find the chunk sequence with contiguous indexes satisfying Property (2). The search problem becomes selecting a candidate C_i from $\{C_{im}\}$ for each i, so that the indexes of combined chunk sequence $(C_i)_{i=1}^n$ are contiguous. *CSI* converts these candidates into nodes in a graph and formulates the search as the shortest path problem.

As the example in Figure 9a, for each request, *CSI* identifies multiple chunk candidates (each as a node in the graph) in the previous step. *CSI* adds an edge between candidates corresponding to two consecutive requests if their indexes I_i grow contiguously. For example, an edge is added between C_{11} and C_{21} as I_{11} and I_{21} grows contiguously from 3 to 4. Also, some requests have A_i marked as true and could potentially correspond to audio chunks. In that case, *CSI* also adds edges between video chunk candidates corresponding to requests surrounding this request if their indexes grow contiguously. For example, an edge is added between C_{13} and C_{33} because A_2 is true. Also, I_{13} and I_{33} grow contiguously from 1 to 2. After the edges are added, each connected path represents a chunk sequence with contiguous indexes. To search for the contiguous chunk sequence corresponding to all requests, *CSI* assigns the length of all edges to 0 and uses the Dijkstra's algorithm to find connected paths covering all requests.

Notice that in the search process *CSI* does not add assumptions on the client adaptation algorithms and outputs all possible sequences matching with the traffic. We will evaluate how many sequences the algorithm usually outputs and what are the accuracies of the output in §6. We find that in many cases *CSI* finds a unique sequence. Even in the case of multiple sequence candidates, the multiple sequences typically are similar and result in similar QoE.

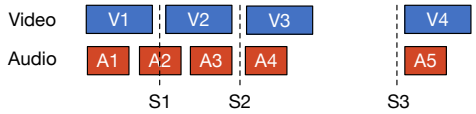


Figure 8. Types of split points: *SP1* (*e.g.,* S3), *SP2* (*e.g.,* S2)

5.3.2 ABR designs with transport MUX

For system types performing transport MUX, i.e. SQ, *CSI* needs to address one more challenge: the traffic corresponding to multiple chunks could be transmitted concurrently on the same connection, making it difficult to analyze individual $\tilde{S}_i$ for each chunk. To address this additional challenge, *CSI* works slightly differently from §5.3.1.

Step 1.2 *CSI* intelligently detects the time points when there is no outstanding request (i.e., all issued requests so far are fully downloaded), and splits traffic into small groups at these time points. With such splitting, each group includes the traffic for a smaller set of complete chunks. For example, in Figure 8, *CSI* splits the traffic at time S2 and S3, resulting in 3 groups containing 5, 2 and 2 chunks.

A key question is *how to find proper split points* for the groups. We hope to make each group as small as possible, as it reduces the search complexity in later Step 2. In the extreme case, assume each group only contains 1 chunk, it is equivalent to the design types without transport MUX. But meanwhile, the splitting needs to make sure all the traffic for the same chunk is in one group and thus cannot be performed arbitrarily. For example, S1 in Figure 8 cannot be used as a split point, as otherwise chunk A2 is split into two groups. *CSI* leverages common properties in video streaming and identifies two types of split points for QUIC traffic.

The first type of split point *SP1* is based on the common ON-OFF traffic pattern that is widely observed in popular players [58, 60, 83]. Due to buffer management, the client typically pauses fetching chunks if the video buffer occupancy is higher than some threshold, and waits until the buffer occupancy drops below another lower threshold, resulting in a periodical ON-OFF pattern in the traffic. Thus *CSI* splits traffic when the OFF period is observed. S3 in Figure 8 is such a split point. In the implementation, the OFF period can be detected using an idle period longer than some threshold. This threshold typically can be set as a few seconds and can also be tuned for each service.

The second type of split point *SP2* is based on the common design that players only download at most 1 video and 1 audio chunk concurrently (§5.2). Thus *CSI* splits the traffic when it observes the player sends out two requests at the same time, as this indicates all previous downloads are finished. S2 in Figure 8 is such a splitting point.

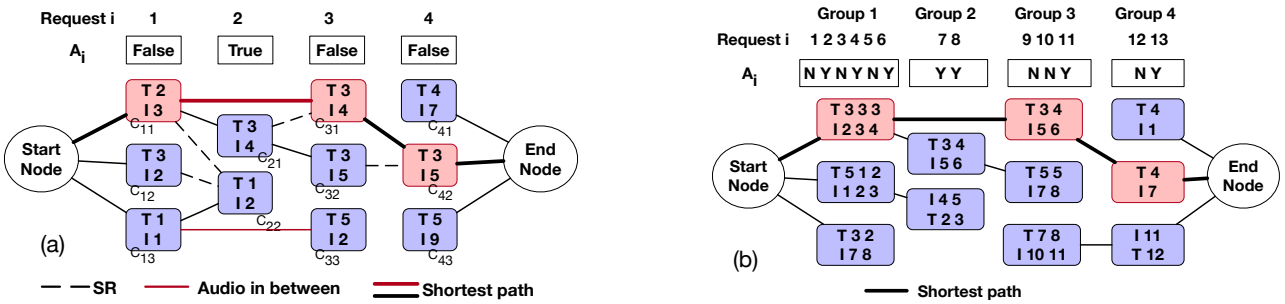


Figure 9. Sample graph for *CSI* inferencing: (a) For 3 types without transport multiplexing. (b) For type SQ. T stands for track, I stands for index.

After the splitting, *CSI* gathers the packets for each traffic group and estimates the total chunk size $\sum_1^n \tilde{S}_i$ for each group in the downlink traffic and the corresponding number of requests n in the uplink traffic.

Step 2 *CSI* performs a two-layer hierarchical search similarly to §5.3.1 to identify chunk sequences matching with the traffic. It first searches short contiguous chunk sequences matching each traffic group, then combines the short sequences for different groups into long contiguous sequences by modeling the search into the shortest path problem.

Step 2.1 For each group *CSI* searches for contiguous chunk sequence candidates given the chunk count and total estimated size constraints. As long as the number of chunks in each group is small, we can practically do an exhaustive search over combinations to find the sequence candidates. We evaluate the splitting in previous Step 1.2 using Youtube with various network bandwidth profiles using the setup in §6. Combining these two types of splitting points, 99.7% of groups are no larger than 10 requests including both video and audio chunks, which can be easily searched.

Step 2.2 Similar to §5.3.1, *CSI* combines the candidates from different groups into a contiguous sequence by formulating the search into the shortest path problem in a graph. The only difference is that each node in the graph is a chunk sequence candidate for a traffic group, instead of a single chunk candidate for a single request. As Figure 9(b) shows, one candidate sequence for traffic group 1, C_{11}, consists of 3 video chunks with index 2,3,4 respectively and 3 audio chunks (omitted in the figure). Another candidate sequence for traffic group 2, C_{21}, consists of 2 video chunks with index 5 and 6. *CSI* adds an edge between them as their chunk indexes grow contiguously. After adding all edges, *CSI* searches for a connected path that covering all requests.

6 System Evaluation

In this section, we first evaluate the generality of *CSI* across encodings of popular streaming services. We then demonstrate that *CSI* achieves high accuracy across different ABR designs.

6.1 Different Encodings

The *CSI* approach builds on the key insights that there is enough variability in chunk sizes and that given estimated sizes (with certain errors) of multiple consecutive chunks, the identities of these chunks can be uniquely determined. We explore the generality of the insights on popular video streaming services.

We analyze a number of ABR videos on 6 popular video streaming services, including Youtube, Facebook Watch, Amazon Video, Vudu, HBO Now and Hulu, and collect the individual chunk sizes across all tracks for each video. For Youtube, we use its data API [29] to query videos with more than 1 million views from different categories. For other services, we analyze videos on their landing page.

We find that for all the video services, there exists significant size variability across chunks (Table 3). For all services, more than half of the videos have a PASR value higher than 1.41. The prevalence of such high PASR values is due to 2 factors: (1) the wide adoption of VBR encoding in the industry. (2) Newer proposed shot-based encoding schemes [14] perform encoding and segmentation on a shot (scene) basis, leading to variable chunk durations and thereby variable chunk sizes. We envision such size variability will remain in the future, considering the industry trend towards more efficient encoding schemes.

We next analyze the feasibility of using chunk sizes as a fingerprint to identify chunks. We find that with k of 1% (see §3.2), for every studied service, in more than half of all the videos, more than 96.9% of 3-segment sequences are unique. When the sequence length increases, higher percentage of sequences have unique sizes. 100% of 6-segment sequences are unique for every studied service. For QUIC where k is 5%, the percentage of unique sequences is relatively lower, but still, more than 90% of 6-segment sequences are unique.

Summary. Our evaluations show that size-based chunk identity inferencing has high accuracy for video encodings across a range of popular streaming services.

Service	#Videos	PASR	% unique sequences (k = 1%)			% unique sequences (k = 5%)		
			1 chunk	3 chunk	6 chunk	1 chunk	3 chunk	6 chunk
Amazon	111	1.35 (1.47)	0.0 (0.0)	96.9 (98.0)	100.0 (100.0)	0.0 (0.0)	16.0 (27.3)	92.8 (96.7)
Facebook	144	1.73 (2.19)	0.0 (0.0)	99.4 (100.0)	100.0 (100.0)	0.0 (0.0)	58.6 (93.9)	99.5 (100.0)
HBO Now	30	1.57 (1.58)	0.0 (0.0)	98.0 (98.4)	100.0 (100.0)	0.0 (0.0)	24.6 (35.5)	97.3 (98.2)
Hulu	30	1.35 (1.44)	0.0 (0.0)	97.3 (98.6)	100.0 (100.0)	0.0 (0.0)	20.9 (32.2)	90.3 (96.4)
Vudu	46	1.52 (1.58)	0.0 (0.0)	99.1 (99.9)	100.0 (100.0)	0.0 (0.0)	45.6 (81.9)	99.1 (100.0)
Youtube	1920	1.94 (2.13)	0.0 (0.0)	99.5 (99.9)	100.0 (100.0)	0.0 (0.0)	68.8 (89.7)	99.8 (100.0)

Table 3. The chunk size variability of popular video services and the percentage of chunk sequences with unique sizes. In cells with format "A(B)", A and B are the median and 95^{th} percentile value across videos.

6.2 Using *CSI* for different ABR designs

We evaluate the accuracy of *CSI* for the 4 ABR designs outlined in §5. We upload 5 videos covering different genres with durations ranging from 13 min to 1 h to Youtube. Youtube encodes each of the uploaded videos into an ABR stream with multiple tracks. We then use a test player built from ExoPlayer, a popular open-source Android media player used by more than 10,000 apps [32, 33, 35, 39], to play the ABR streams from Youtube servers. We use Youtube for our evaluation as it offers a convenient way to encode ABR streams and does not cause Digital Right Management (DRM) related complexities for user-generated content. Note that as shown earlier in §6.1, *CSI* itself works for video encodings across a range of popular services. The default ExoPlayer does not support QUIC. Therefore, to test both HTTPS and QUIC, we added Cronet as the underlying network stack in ExoPlayer and configure it to use either protocol as needed. Two of the four ABR designs involve combined video and audio content in a single track. However, Youtube servers only encode separate audio and video tracks. Hence we explore the combined audio-video cases by creating a manifest file only including the video tracks and excluding the audio tracks.

For repeatability reasons, we used network bandwidth trace-driven replay experiments. Specifically, we ran extensive throughput measurements in operational commercial mobile networks, covering a wide range of scenarios corresponding to different signal strengths and locations, and collected 30 bandwidth traces covering these scenarios. Then, during the experiments, we let *CSI* perform traffic shaping based on these collected bandwidth traces to emulate the measured network conditions. These traces have an average bandwidth ranging from 600kbps to 40Mbps and different bandwidth variability. The very different bandwidth characteristics trigger different adaptation behavior on our ExoPlayer-based client. Note that *CSI* itself neither makes any assumption regarding the track selection logic, nor is it biased towards any specific adaption behavior. To address any experimental noise, we rerun the test 5 times for each video and bandwidth trace combination. In each test run, we stream the video for 10 min. In total we test around 125 hours worth of video playback.

To obtain ground truth on the downloaded chunk identities for measuring the accuracy of the inferred results, we instrument ExoPlayer to log the chunk request timing and URL. As part of our evaluations, we also explore how information on displayed chunks obtained from screen analysis helps improve the accuracy. To get such information, we also add instrumentation in our ExoPlayer client to log the identities of the chunks that are displayed on the screen.

6.2.1 Case: ABR without transport MUX

For the 3 design types without transport MUX (Table 2), we analyze the inference accuracy as follows. For each experiment, the inference might return multiple candidate chunk sequences. We calculate the accuracy of each inferred sequence and get the highest and lowest accuracy across the sequences. The accuracy of an inferred sequence is calculated as the percentage of correctly inferred chunks.

For CQ and CH, *CSI* always find the ground truth as one of the inferred chunk sequences for every run (Table 4). In a small proportion of cases, multiple chunk sequences could have similar sizes and *CSI* would output all of them as candidate solutions. In our evaluations, for 84% of runs, the inferred sequence is unique. Even in the cases where multiple inferred sequences are found, the identified sequence with the most errors still achieves high accuracy: higher than 95% for 95.6% of runs. With additional displayed chunk information, the accuracy can be further improved: *CSI* uniquely infers the ground truth sequence for 92% of all runs.

For SH where audio content is encoded into separate tracks, some requests are for audio chunks, increasing the challenges for the inference. Though video chunks are generally much larger in size than audio chunks, some video chunks from low bitrate tracks or with simple scenes could be encoded with sufficiently low bitrates and have similar sizes with audio chunks. Our evaluation shows that *CSI* still infers the downloaded chunk identities with high accuracy. For SH, the accuracy of the best-inferred sequence is 100% for every run. Even the worst candidate for 91.7% of runs had an accuracy exceeding 95%. With additional displayed chunk information, the accuracy improves: the accuracy of even the worst candidate sequence for 99.4% of runs is higher than 95%.

Case	Without displayed chunk information						With displayed chunk information					
	Best output			Worst output			Best output			Worst output		
	100% match	>95% accuracy	5 pct accuracy	100% match	>95% accuracy	5 pct accuracy	100% match	>95% accuracy	5 pct accuracy	100% match	>95% accuracy	5 pct accuracy
CH	100.0	100.0	100.0	64.3	93.6	86.0	100.0	100.0	100.0	94.9	100.0	99.7
SH	100.0	100.0	100.0	5.0	91.7	85.1	100.0	100.0	100.0	9.4	99.4	95.0
CQ	100.0	100.0	100.0	84.1	95.6	90.5	100.0	100.0	100.0	92.1	100.0	95.2
SQ	98.0	100.0	100.0	4.0	52.8	56.2	98.5	100.0	100.0	91.5	98.0	98.1

Table 4. The evaluation of inference accuracy with ExoPlayer. "100% match" means the percentage of experimental runs with 100% accuracy. ">95% accuracy" means the percentage of runs with accuracy higher than 95%. "5pct" means the 5 percentile of accuracy across the runs.

6.2.2 Case: ABR with transport multiplexing

We next evaluate how well CSI handles the ABR system design that uses transport MUX, *e.g.,* SQ (Table 2). As discussed in §5, transport MUX makes it difficult to estimate the size of each chunk and to infer their identities. Our evaluations show that *CSI* finds multiple matching sequences for 96% of the test runs. For 52.8% of runs even the worst candidate has an accuracy higher than 95% and for 69.8% of experiments with the worst candidate has an accuracy higher than 90%. We also find that by utilizing displayed chunk information, *CSI* can further improve and determine the ground truth as the only output for 91.5% of runs.

6.2.3 Computation time for *CSI*

In terms of computation time, for system designs without transport MUX, *CSI* typically takes a few seconds to analyze a 10 min long trace on a commodity desktop. For system designs with transport MUX, the analysis time can increase up to around a minute due to the larger search space involved. Such short response times are reasonable for active testing, as the analysis is performed offline and does not require real-time. We leave a more efficient implementation of *CSI* to future work.

Summary. Our evaluations show that *CSI* achieves high accuracy for a wide range of system designs, and with reasonable computation times.

7 Demonstrating ABR analysis using CSI

We discussed earlier the need for third-party entities including mobile network operators and app developers to perform active measurement and study the adaptation behavior of commercial mobile streaming systems. In this section, we shall illustrate one such important use case for mobile network operators, *i.e.,* designing traffic management policies. It is worth mentioning that the purpose of this section is to demonstrate the need for using active measurement to derive complex design decisions and how *CSI* helps support such use cases. Determining the optimal traffic management policy itself requires careful considerations of various factors and tradeoffs and is out of the scope of this work.

Due to the associated massive traffic volumes, mobile network providers commonly need to perform traffic management for video traffic (*e.g.,* [21, 62]). A typical approach involves limiting the network bandwidth to a level such that the player is still able to deliver good-quality Standard Definition (SD) content to smartphones. The underlying consideration is that especially given the limited screen size of mobile devices, streaming videos at too high quality and resolutions would at best bring only marginal QoE improvements, while consuming substantially more network data, leading to draining the user's limited data budget much faster, and also potentially significantly increasing the load on the network.

A good traffic shaping policy needs to balance the data usage and delivered QoE. To design such a policy, it is essential to understand the interactions between various parameters in the policy design space and the mobile player's adaptation logic. In the following, as an illustration, we leverage *CSI* to explore designing a token-bucket based shaping policy. The token bucket is widely used for traffic shaping and performs shaping based on the expenditure of tokens [54]. It has 2 key parameters: token generation rate r and bucket size N. Tokens fill the bucket at the rate of r until the bucket is full. When a packet of size s arrives, if there are more than s tokens available, the shaper forwards the traffic and consumes s tokens. Otherwise, it queues the packet until enough tokens are generated. The selection of each of these parameters has to be done carefully, as it can substantially impact the streaming QoE. Next, we use Hulu as an example service to illustrate how *CSI* can be leveraged to determine the QoE impact of different parameters of the token bucket.

We first use *CSI* to gain some basic insights into the adaptation design of Hulu, which is essential for later understanding the QoE impact of various shaping configurations. We select a popular video on Hulu. It has 7 tracks (we refer to these tracks as T1-T7, numbered in the order of increasing bitrate). We perform a series of experiments within each of which we emulate a stable network bandwidth. The bandwidth ranges from 1 Mbps to 4 Mbps across runs. We find that at the beginning of playback, the Hulu client always starts by downloading chunks from T1, *i.e.,* the lowest-bitrate track. After downloading a few chunks, the client switches to a higher-bitrate higher-quality track and continues streaming

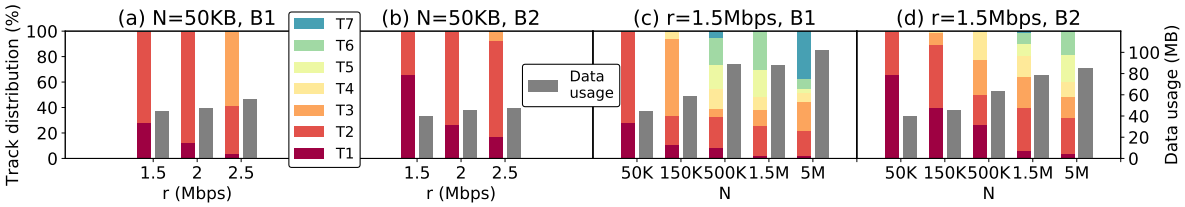


Figure 10. The track distribution and data usage for Hulu with different shaping policy and network conditions.

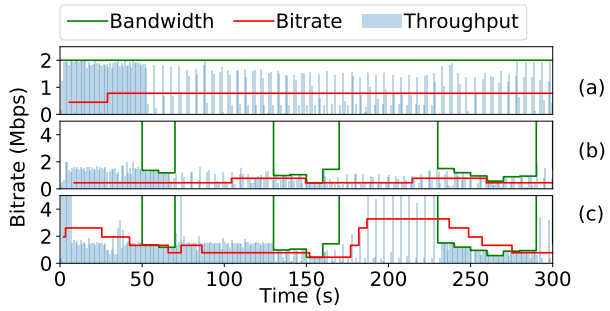


Figure 11. Hulu behavior under (a) 2Mbps, (b) profile B2, r=1.5Mbps, N=50KB, (b) profile B2, r=1.5Mbps, N=5MB.

chunks from that track as long as the network bandwidth remains stable. We illustrate one example run in Figure 11(a) with a stable bandwidth of 2Mbps. Our experiments indicate that the encoding bitrate of the track that the client converges to is at most half of available network bandwidth. We infer buffer occupancy in the client across time using the total duration of downloaded chunks that are not played yet. We find the client pauses downloading the next segment when the buffer occupancy reaches around 145 seconds' worth of playback content, and does not resume until the buffer occupancy drops below this value. This behavior generates a periodic ON-OFF traffic download pattern (after 50s in Figure 11(a)).

We now explore how to properly configure r and N to implement a traffic shaping policy that works well with Hulu. To perform evaluations, we chain a Linux machine at the upstream of the gateway (in Figure 6). On the Linux machine, we configure tbf [25], a Linux *tc* module, to implement traffic shaping policies using a token bucket with rate r and size N. We also use tc on the gateway to emulate cellular bandwidth conditions along the gateway to the mobile device network path. In particular, we test 2 different bandwidth conditions as an illustration: a condition with stable bandwidth 10Mbps (denoted as B1), and a condition with bandwidth 10Mbps in most of the time but occasionally low bandwidth such as 1Mbps (denoted as B2, see Figure 11).

To start with, we fix N to be relatively small (50KB) and test different r values. As shown in Figure 10 (a)-(b), when r increases, the player spends more time streaming better-quality tracks and the fraction of streaming low-quality tracks (*e.g.*, T1) reduces. The data usage also increases accordingly. The reason is that a higher r corresponds to higher available network bandwidth and the client is able to stream tracks with higher bitrates.

We next illustrate how the bucket size N affects app behavior. As an example, we fix r to be 1.5Mbps. As shown in Figure 10 (c)-(d), for both network conditions, with a larger N, the percentage of low-quality tracks reduces and the percentage of high-quality tracks increases. The reason is that the token bucket accumulates tokens at startup or when the cellular network condition is poor. Also, as the Hulu client appears to select tracks whose encoding bitrates are at most half of available bandwidth, its buffer is frequently full and needs to enter "OFF" periods and pause for some time, leading to tokens accumulating in the bucket. With a larger N, the bucket accumulates more tokens and allows for larger bursts when the player resumes downloading chunks. For example, as shown in Figure 11, compared with (b), in (c) where N is higher, the achieved instantaneous throughput is much higher when the player resumes from OFF periods. As a result, the player ramps up from low-quality tracks faster and plays higher-quality tracks. However, a bigger N leads to higher user data usage. The data usage when N is 5MB is 2.2 times of the data usage when N is 50KB (Figure 10(d). A bigger N also leads to potentially more frequent track switches. In Figure 11(b), the player often selects tracks with bitrate much higher than 1.5Mbps and quickly consumes all tokens in the bucket. As a result, the player then is forced to ramp down to a lower-bitrate track T1. Such dramatic switches between high-bitrate high-quality tracks and low-bitrate low-quality tracks are undesirable as they impair user experience, and should therefore be avoided.

As shown above, both parameters r and N jointly have complex interactions with the player adaptation behavior. To create an optimal design, one would need to perform extensive active measurements to test combinations of these parameters under a wide range of cellular network conditions for different videos and streaming services. The optimization effort should carefully consider the tradeoffs involving various QoE metrics as well as data usage, to achieve data savings while delivering a good user QoE. As we illustrated above, in the presence of encrypted traffic, *CSI* can help testers evaluate the impact of different token bucket configurations on the ABR streaming QoE for proprietary commercial services.

8 Related work

To our best knowledge, *CSI* is the first system designed to enable third-party entities to explore and understand the adaptation behavior of closed-source ABR streaming services in the presence of end-end traffic encryption. We next summarize related work and their limitations.

Active measurement with no traffic encryption. Existing works [45, 57, 67, 83] identify downloaded chunks based on HTTP request information (*e.g.,* URLs) from unencrypted traffic. However, with the wide adoption of traffic encryption, such information is no longer available.

Encryption workarounds. There exist some workarounds for testers to decrypt encrypted traffic, but various measures are increasingly adopted by the ecosystem (*e.g.,* device vendors and app developers) to prevent such workarounds for security and privacy considerations. In contrast, we specifically design *CSI* to work without the need to rely on fragile workarounds, which include

- **MITM.** Some workarounds install a self-signed certificate on the device and perform MITM interception using proxies [19, 20, 48]. But this is becoming harder due to (1) more stringent ecosystem trust models, *e.g.,* apps on Android 7 onwards by default no longer trust user-installed certificates [37], (2) stricter protocol implementations, *e.g.,* there are no known working solutions for MITM QUIC traffic, (3) use of *certificate pinning* techniques which hardcode server certificates in the app to prevent third-party interception [46].
- **Rooting.** Some workarounds use a rooted device to intercept encrypted connections (*e.g.,* by disabling certificate checking in the system *etc.*). However, this is increasingly less effective due to (1) smartphone vendors [26, 27] are making it much more difficult or even infeasible to root the devices (*e.g.,* Samsung S5 onwards and iPhone), (2) many video apps such as Google Play Movies, SkyGo and NowTV have logic to block access if the device is detected to be rooted, likely due to security and DRM considerations.
- **App repackaging.** Some approaches propose repackaging client apps to remove encryption. This is technically challenging and hard to generalize due to the closed-source nature of commercial video apps. In addition, user agreement of streaming services (*e.g.,* [9, 10, 13]) may disallow such approaches.

While the above workarounds currently may still be feasible for certain scenarios, we envision them to be increasingly less applicable. We therefore develop *CSI* to avoid such workarounds as much as possible.

Passive monitoring with traffic encryption. Some works train machine learning models to monitor video streaming QoE from encrypted network traffic characteristics such as achieved throughput [52, 70, 72, 73]. Such approaches cannot be directly applied for our use case of inferring ABR adaptation behavior for the following main reasons. (1) They require labeled data on QoE associated with various network characteristics to train application-specific models to classify QoE. Such ground truth QoE data is difficult to obtain without approaches such as *CSI*. Systems like *CSI* can be used to help build models for in-network passive monitoring purposes. (2) They only provide coarse-grained binary classification or qualitative labels on the QoE. In contrast, *CSI* provides fine-grained information on how players adapt to various network conditions and resulting QoE, which is essential for many use cases such as performing QoE diagnosis and deriving better designs.

Traffic analysis (TA). Many works in security/privacy space perform TA to infer certain application-level information from encrypted network traffic. For example, [47, 49, 55, 59, 63, 77, 78, 81, 82] infer details such as which website the user visited, which video is played and which app the user used etc. Different from these, our work is targeted at a very different use case with its unique challenges (see more in §5.1): inferring the identities of downloaded chunks during ABR video streaming. Existing techniques cannot be directly applied to our use case. For example, previous works build fingerprints for each website or app. However, in ABR streaming, the client makes a decision for the track of each chunk. The search space of all possible chunk sequences for a video can be extremely large (10 min's playback could have 10^{83} possible sequences), making existing fingerprinting approaches infeasible. In addition, existing works focus on HTTPS and do not examine QUIC which has its own distinct features and properties like multiplexing that make the traffic analysis task harder. Parallel work [66] uses very specific assumptions on ABR adaptation logic to estimate QoE metrics, but such assumptions do not hold in general.

9 Conclusions and Future Work

We presented *CSI*, a novel scheme for analyzing ABR streaming behaviors of mobile apps in the presence of traffic encryption (HTTPS and QUIC). Extensive evaluations using real videos and network conditions show that *CSI* achieves high inference accuracy and can effectively analyze complex player behaviors, even for very challenging conditions like encrypted transport multiplexing in QUIC. *CSI* is currently being incorporated into a popular mobile video streaming analysis toolkit.

10 Acknowledgements

We express our sincerest gratitude towards the anonymous reviewers who provided valuable feedback to improve this work, and our shepherd, Y. Charlie Hu, for guiding us through the revisions. This material is based upon work partially supported by NSF under grants CNS-1827940, CCF-1628991 and CNS-1629763.

References

[1] [n.d.]. Big Buck Bunny. https://peach.blender.org/.

[2] [n.d.]. Chrome Devtools. https://developers.google.com/web/tools/chrome-devtools/.

[3] [n.d.]. draft-ietf-quic-tls-13 - Using Transport Layer Security (TLS) to Secure QUIC. https://tools.ietf.org/html/draft-ietf-quic-tls-13.

[4] [n.d.]. FFmpeg. http://ffmpeg.org.

[5] [n.d.]. Firefox Developer Tools. https://developer.mozilla.org/en-US/docs/Tools.

[6] [n.d.]. GOCR. http://jocr.sourceforge.net/.

[7] [n.d.]. google/ExoPlayer: An extensible media player for Android. https://github.com/google/exoplayer.

[8] [n.d.]. google/shaka-player: JavaScript player library / DASH client / MSE-EME player. https://github.com/google/shaka-player.

[9] [n.d.]. HBO Now Terms of Use. https://play.hbonow.com/terms.

[10] [n.d.]. Hulu Terms of Use. https://secure.hulu.com/terms.

[11] [n.d.]. Man-in-the-middle attack. https://en.wikipedia.org/wiki/Man-in-the-middle_attack.

[12] [n.d.]. MP4Box: GPAC multimedia packager. https://gpac.wp.imt.fr/mp4box/.

[13] [n.d.]. Netflix Terms of Use. https://help.netflix.com/legal/termsofuse.

[14] [n.d.]. Optimized shot-based encodes: Now Streaming! https://medium.com/netflix-techblog/optimized-shot-based-encodes-now-streaming-4b9464204830.

[15] [n.d.]. Per-Title Encode Optimization. https://medium.com/netflix-techblog/per-title-encode-optimization-7e99442b62a2.

[16] [n.d.]. Quick Start Guide to Using Cronet. https://chromium.googlesource.com/chromium/src/+/master/components/cronet.

[17] [n.d.]. RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. https://tools.ietf.org/html/rfc5246.

[18] [n.d.]. RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3. https://tools.ietf.org/html/rfc8446#section-5.4.

[19] [n.d.]. Squid web proxy SSL bump feature. https://wiki.squid-cache.org/Features/SslBump.

[20] [n.d.]. SSL MITM Proxy. https://crypto.stanford.edu/ssl-mitm/.

[21] [n.d.]. Stream More Video, Use Less Data with Stream Saver - AT&T. https://www.att.com/offers/streamsaver.html.

[22] [n.d.]. Tesseract OCR. https://github.com/tesseract-ocr/tesseract.

[23] [n.d.]. Test patterns | Netflix. https://www.netflix.com/title/80018499.

[24] [n.d.]. UI Automator. https://developer.android.com/training/testing/ui-automator.

[25] [n.d.]. UI Automator. https://linux.die.net/man/8/tc-tbf.

[26] [n.d.]. Why is it so hard to ŕootá smartphone? https://www.androidcentral.com/why-it-so-hard-root-smartphone.

[27] [n.d.]. Why you shouldnt́ root your Android phone. https://www.howtogeek.com/132115/the-case-against-root-why-android-devices-dont-come-rooted/.

[28] [n.d.]. Youtube brings us stats for nerds. http://tubularinsights.com/youtube-stats-for-nerds/.

[29] [n.d.]. Youtube Data API. https://developers.google.com/youtube/v3/.

[30] [n.d.]. youtube-dl: Download videos from YouTube. https://rg3.github.io/youtube-dl/.

[31] 2012. ISO/IEC 23009-1, Information technology - Dynamic adaptive streaming over HTTP (DASH). http://standards.iso.org/ittf/PubliclyAvailableStandards/c057623_ISO_IEC_23009-1_2012.zip.

[32] 2016. ExoPlayer 2 - Why, what and when? https://medium.com/google-exoplayer/exoplayer-2-x-why-what-and-when-74fd9cb139.

[33] 2016. ExoPlayer from the other side. https://medium.com/google-exoplayer/exoplayer-from-the-other-side-5909553abae2.

[34] 2016. Protecting Netflix Viewing Privacy at Scale. https://medium.com/netflix-techblog/protecting-netflix-viewing-privacy-at-scale-39c675d88f45.

[35] 2016. WhatsApp For Android Devices. https://tech.blorge.com/2016/09/23/whatsapp-2-16-274-download-available/-android-devices-new-emojis/155538.

[36] 2016. YouTube's road to HTTPS. https://youtube-eng.googleblog.com/2016/08/youtubes-road-to-https.html.

[37] 2017. Android Network Security Configuration. https://developer.android.com/training/articles/security-config.html.

[38] 2017. App share of total mobile minutes in leading online markets. https://www.statista.com/statistics/692752/app-share-of-mobile-minutes-countries/.

[39] 2017. Building Periscope for Android. http://nerds.airbnb.com/building-periscope-for-android/.

[40] 2017. Cisco Visual Networking Index: Forecast and Methodology, 2016-2021. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html.

[41] 2017. Openwave Mobility Mobile Video Index, Dec 2017. https://owmobility.com/whitepapers/.

[42] 2017. The 2017 U.S. Mobile App Report. https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-2017-US-Mobile-App-Report.

[43] 2018. IETF QUIC working group. https://datatracker.ietf.org/wg/quic.

[44] 2018. Mobile App versus Mobile Website Statistics. https://jmango360.com/wiki/mobile-app-vs-mobile-website-statistics/.

[45] Saamer Akhshabi, Ali C Begen, and Constantine Dovrolis. 2011. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 157–168.

[46] Johanna Amann, Oliver Gasser, Quirin Scheitle, Lexi Brent, Georg Carle, and Ralph Holz. 2017. Mission accomplished?: HTTPS security after diginotar. In *Proceedings of the 2017 Internet Measurement Conference*. ACM, 325–340.

[47] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 605–616.

[48] Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. 2010–. mitmproxy: A free and open source interactive HTTPS proxy. https://mitmproxy.org/ [Version 4.0].

[49] Scott E Coull and Kevin P Dyer. 2014. Traffic analysis of encrypted messaging services: Apple imessage and beyond. *ACM SIGCOMM Computer Communication Review* 44, 5 (2014), 5–11.

[50] Jan De Cock, Aditya Mavlankar, Anush Moorthy, and Anne Aaron. 2016. A large-scale video codec comparison of x264, x265 and libvpx for practical VOD applications. In *Applications of Digital Image Processing XXXIX*, Vol. 9971. International Society for Optics and Photonics, 997116.

[51] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[52] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-Ros, and Konstantina Papagiannaki. 2016. Measuring video QoE from encrypted traffic. In *Proceedings of the 2016 Internet Measurement Conference*. ACM, 513–526.

[53] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 332–346.

[54] Tobias Flach, Pavlos Papageorge, Andreas Terzis, Luis Pedrosa, Yuchung Cheng, Tayeb Karim, Ethan Katz-Bassett, and Ramesh Govindan. 2016. An Internet-wide analysis of traffic policing. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 468–482.

[55] Jiaxi Gu, Jiliang Wang, Zhiwen Yu, and Kele Shen. 2018. Walls Have Ears: Traffic-based Side-channel Attack in Video Streaming. In *INFOCOM 2018-IEEE Conference on Computer Communications, IEEE*. IEEE.

[56] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. 2018. Favor: fine-grained video rate adaptation. In *MMSys*.

[57] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. 2012. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM, 225–238.

[58] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[59] Alfonso Iacovazzi, Andrea Baiocchi, and Ludovico Bettini. 2013. What are you Googling?-Inferring search type information through a statistical classifier. In *Global Communications Conference (GLOBECOM), 2013 IEEE*. IEEE, 747–753.

[60] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2014. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking (TON)* 22, 1 (2014), 326–340.

[61] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. 2017. Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols. In *Proceedings of the 2017 Internet Measurement Conference*. ACM, 290–303.

[62] Arash Molavi Kakhki, Fangfan Li, David Choffnes, Ethan Katz-Bassett, and Alan Mislove. 2016. Bingeon under the microscope: Understanding T-Mobiles zero-rating implementation. In *Proceedings of the 2016 workshop on QoE-based Analysis and Management of Data Communication Networks*. ACM, 43–48.

[63] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. 2015. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *24th USENIX Security Symposium (USENIX Security 15)*.

[64] TV Lakshman, Antonio Ortega, and Amy R Reibman. 1998. VBR video: Tradeoffs and potentials. *Proc. IEEE* (1998).

[65] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 183–196.

[66] Tarun Mangla, Emir Halepovic, Mostafa Ammar, and Ellen Zegura. 2018. eMIMIC: Estimating HTTP-based Video QoE Metrics from Encrypted Network Traffic. In *IEEE/IFIP Conference on Traffic Measurement and Analysis 2018*.

[67] Tarun Mangla, Emir Halepovic, Rittwk Jana, Kyung-Wook Hwang, Marco Platania, and Mostafa Ammar. 2018. VideoNOC: Assessing Video QoE for Network Operators using Passive Measurements. In *Proceedings of ACM Multimedia Systems Conference*. ACM.

[68] Ahmed Mansy, Mostafa Ammar, Jaideep Chandrashekar, and Anmol Sheth. 2014. Characterizing client behavior of commercial mobile video streaming services. In *Proceedings of Workshop on Mobile Video Delivery*. ACM, 8.

[69] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 197–210.

[70] M Hammad Mazhar and Zubair Shafiq. 2018. Real-time Video Quality of Experience Monitoring for HTTPS and QUIC. In *INFOCOM 2018-IEEE Conference on Computer Communications, IEEE*. IEEE.

[71] Brad Miller, Ling Huang, Anthony D Joseph, and J Doug Tygar. 2014. I know why you went to the clinic: Risks and realization of https traffic analysis. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 143–163.

[72] Irena Orsolic, Dario Pevec, Mirko Suznjevic, and Lea Skorin-Kapov. 2016. Youtube QoE estimation based on the analysis of encrypted network traffic using machine learning. In *Globecom Workshops (GC Wkshps), 2016 IEEE*. IEEE, 1–6.

[73] Irena Orsolic, Dario Pevec, Mirko Suznjevic, and Lea Skorin-Kapov. 2017. A machine learning approach to classifying YouTube QoE based on encrypted network traffic. *Multimedia tools and applications* 76, 21 (2017), 22267–22301.

[74] Wubin Pan, Gaung Cheng, Hua Wu, and Yongning Tang. 2016. Towards QoE assessment of encrypted YouTube adaptive video streaming in mobile networks. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–6.

[75] Roger Pantos and William May. 2016. HTTP live streaming. (2016).

[76] Yanyuan Qin, Shuai Hao, KR Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. 2018. ABR streaming of VBR-encoded videos: characterization, challenges, and solutions. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. ACM, 366–378.

[77] Andrew Reed and Benjamin Klimkowski. 2016. Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11 n connections. In *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*. IEEE, 1107–1112.

[78] Andrew Reed and Michael Kranch. 2017. Identifying HTTPS-protected Netflix videos in real-time. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. ACM, 361–368.

[79] Paul Schmitt, Francesco Bronzino, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2020. Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* (2020).

[80] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: near-optimal bitrate adaptation for online videos. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 1–9.

[81] Qixiang Sun, Daniel R Simon, Yi-Min Wang, Wilf Russell, Venkata N Padmanabhan, and Lili Qiu. 2002. Statistical identification of encrypted web browsing traffic. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, 19–30.

[82] Andrew M White, Austin R Matthews, Kevin Z Snow, and Fabian Monrose. 2011. Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks. In *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE, 3–18.

[83] Shichang Xu, Subhabrata Sen, Z Morley Mao, and Yunhan Jia. 2017. Dissecting VOD services for cellular: performance, root causes and best practices. In *Proceedings of the 2017 Internet Measurement Conference*. ACM, 220–234.

[84] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 325–338.