Silhouette – Identifying YouTube Video Flows from Encrypted Traffic

Feng Li
Verizon Labs
Waltham, MA, 02145
feng.li@verizon.com

Jae Won Chung
Verizon Labs
Waltham, MA, 02145
jae.won.chung@verizon.com

Mark Claypool
Worcester Polytechnic Institute
Worcester, MA, 01609
claypool@cs.wpi.edu

ABSTRACT
Video streaming traffic often dominates mobile wireless networks, forcing Internet Service Providers (ISPs) to deploy video shaping to identify and then manage traffic during congested periods. Unfortunately, the increasing use of end-to-end encryption (e.g., TSL/SSL) makes it difficult to identify video flows even with deep packet inspection. As an alternative, this paper presents Silhouette—a real-time, lightweight video classification method suitable for ISP middle-boxes. Silhouette uses only flow statistics (i.e., “shape”) for video identification making it payload-agnostic, effective for identifying video flow even when encrypted. Preliminary results with pre-classified YouTube traffic shows the promise of the Silhouette approach, yielding high identification accuracy over a range of video content and encoding qualities.

CCS CONCEPTS
• Information systems → Multimedia streaming; • Networks → Packet classification;

KEYWORDS
YouTube, Service Classification, HTTP Adaptive Streaming, QUIC

ACM Reference Format:

1 INTRODUCTION
Video traffic is the king of the hill [10], accounting for around 50% of traffic volume over a U.S. tier-1 mobile Internet service provider (ISP), and is projected to increase 11x by 2020, accounting for 75% of all mobile data traffic [3].

As a response to the deluge of video, mobile Internet service providers deploy traffic shaping or policing solutions to mitigate the impact of video traffic. For example, T-Mobile’s BingeOn plan throttled video bitrates to 1.5 Mb/s for unlimited plan customers [6], and other U.S. mobile ISPs soon followed suit, limiting video rates to lower radio link utilizations. However, given the self-imposed rate limits of video by encoding and the latency resilience of video from buffering, there is the potential to smooth video rather than just limit its rate, easing congestion without significantly impacting video quality [11].

But if such video shaping potential is to be realized, ISPs need real-time, accurate identification of video traffic. Blindly pacing or throttling flows that may or may not be video can impair application performance in general. Currently, in an attempt to identify video flows, ISPs often deploy deep packet inspection (DPI) engines in network middle-boxes. These DPI engines typically rely on packet content, such as host headers and Transport Layer Security (TLS), but may also use Server Name Indications (SNIs) in order to selectively apply shaping rules to flows [12]. However, DPI-based video detection has several drawbacks:

Easy to spoof. Kakihara et al. demonstrated a simple spoofing method for BingeOn with an HTTP proxy only several months after the service became publicly available [6]. Li et al. [12] developed a library that exposed traffic classification policies used inside mobile ISP’s middle-boxes, making it even easier to circumvent traffic classification rules.

Ineffective with encryption. End-to-end encryption, such as TLS/SSL, renders content-based DPI engines ineffective. Unfortunately, over mobile networks today, nearly 70% of traffic is protected by TLS or SSL [10]. Thus, unless video end points are unencrypted or are terminated at ISP proxies, payload-based identification alone will fail.

Ineffective when source unknown. Many DPI engines build a traffic signature from training data sets where video sources are known a priori. If the pre-trained DPI engine is presented with video from a new source, it is unable to properly classify the traffic as video.

This paper presents Silhouette,1 a video classification method that uses only the transport layer flow characteristics (the “outline” or “shape” of the flow), not needing to inspect transport layer sequence numbers or application layer payloads. Silhouette delimits flows with the classic 5-tuple (source & destination IP addresses, source & destination ports, and protocol type), but then records only average transport layer payload size and data rate to decide whether a given flow is video or not. The Silhouette algorithms are generic enough to work with HTTPS/TCP and QUIC/UDP-based streaming and does not require prior collection and training, nor run-time machine learning.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

NOSSDAV’18, June 12–15, 2018, Amsterdam, Netherlands
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5772-2/18/06... $15.00
https://doi.org/10.1145/3210445.3210448

---

1 A silhouette is a shape/outline of something visible against the background.
Evaluation of Silhouette with 660 video traces – 66 video clips with five different resolutions (240p, 360p, 480p, 720p and 1080p) and two different protocols (TCP and QUIC) – yields an accuracy of 88%, with most of the unidentified videos those with low encoding rates.

The rest of the paper is organized as follows: Section 2 summarizes related work; Section 3 provides background on YouTube streaming; Section 4 presents Silhouette and an illustrative example; Section 5 evaluates Silhouette with a broad range of YouTube videos; and Section 6 summarizes our conclusions and presents possible future work.

2 RELATED WORK

Identifying video flows amid encrypted traffic is not a new research area, spanning at least a dozen of machine learning (ML)-based proposed solutions over the last two decades [8, 14, 15, 18]. Unfortunately, while promising, ML techniques require a priori training and are unusable for new, as yet unclassified applications and can be processing-intensive if run in real-time. Deep packet inspection (DPI)-based approaches can achieve high accuracy [13], but the high computational complexity and ineffectiveness for encrypted payloads make the DPI-based approach impractical for real-time classification.

Other research has focused on measuring video QoE for encrypted traffic by passively monitoring the network and application statistics [7]. For example, Dimopoulos et al. [5] infer stalling, average video quality and quality variations by observing network statistics. However, their method requires a data “cleaning” phase to filter out traffic from domain names not related to the services. Thus, although they infer the QoE of previously-identified encrypted video sessions, they do not address the problem of identifying video flows from encrypted traffic.

Reed et al. [16] identify specific Netflix videos using only the information provided by TCP/IP headers. However, their approach requires a “finger print” database built from a large number of video clips, making it ineffective for systems such as YouTube where around 400 hours worth of new video are uploaded per minute. Also, since their approach uses TCP sequence numbers, it is ineffective on video protocols with negative acknowledgments, such as QUIC [1]. Lastly, ISPs typically do not need to know the exact video content for treatment, but rather just need to correctly identify a flow as video.

Other work has proposed treatment-based traffic classification [4, 11], where video traffic, once identified, is paced without significantly degrading user QoE. However, their video detection is designed for the real-time streaming protocol (RTSP) and may not be accurate for modern video traffic which predominantly streams over HTTP.

3 YOUTUBE VIDEO STREAMING

For the past decade or so, most content providers have used HTTP for video streaming. HTTP streaming combines the advantages of firewall penetration and easy network address translation [5, 9] with TCP’s reliable packet delivery and congestion control. Even QUIC over UDP is similar for video streaming with TCP + TLS + HTTP/2 [1]. YouTube HTTP video streaming has two Transport Layer implementations: QUIC-enabled HTTP streaming and TCP-enabled HTTP streaming. While Bhat et al. [2] find QUIC does not necessarily improve performance over TCP-based streaming, from the application layer perspective, QUIC-based YouTube streaming should behave similarly to TCP-based YouTube Streaming. Although layers under the application may impact some flow statistics (e.g., retransmission behaviors for QUIC and TCP), given the application-layer commonality, the transport layer statistics should appear similar. This motivates our search for a general classification method to detect HTTP video traffic for both HTTPS/TCP and QUIC/UDP.

YouTube video streaming supports two different approaches [5, 9]: progressive downloading for low quality videos (240p, 360p, and 480p) and HTTP adaptive streaming (HAS) for high definition (HD) videos (720p and 1080p).

For streaming with progressive downloading, each video session consists of two phases: startup and steady state. In startup, the video session downloads rapidly to fill the client player’s playout buffer as fast as possible. Once the buffer is filled, the video session enters steady state and pauses downloading, resuming only when the playout buffer depletes.

For HAS, videos are split on the server into multiple segments, each corresponding to 2 to 10 seconds of playback time. Each video segment is encoded into a range of different qualities [9]. The client determines which segment to download as a function of the throughput observed while downloading the previous segments and the available seconds of playback in its playout buffer [5]. In both progressive downloading and HAS, the video player sends an HTTP request to the server to request the next video segment. In HAS, the next segment may have a different quality encoding than the previous segment, while in progressive downloading, all segments have the same quality. Therefore, for HTTP YouTube streaming, we expect to observe multiple HTTP requests in the uplink direction.

Figure 1 shows the first 20 seconds of network behavior for a QUIC/UDP, 1080p, 24 fps YouTube video, the official movie trailer for Alpha 2018, using a Chrome browser on Linux. The x-axis is the elapsed packet capture time and the y-axis is the transport layer payload size – excluding UDP/IP headers. The red plus (+) symbols are downlink packets and the blue circles are uplink packets.

The session is bursty and has large downlink packets but mostly small uplink packets. However, some uplink packets are larger than 500 bytes, and these precede a series of large downlink packets.

Figure 1: YouTube Streaming with QUIC (Alpha Trailer)

1https://www.youtube.com/yt/about/press/
Based on YouTube streaming behavior described above, the large uplink requests are likely video retrieval requests for the next video segments – Silhouette leverages this in the next section.

4 SILHOUETTE

Inspired by our observations of YouTube video streaming in Section 3, we propose Silhouette, a real-time heuristic method to detect video streaming flows based on Application Data Units (ADUs) and network statistics. This section describes Silhouette, which consists of two algorithms (Section 4.1), and shows an example with a YouTube streaming video (Section 4.2). A Python implementation of Silhouette is available online.5

4.1 Video Flow Identification

A key aspect of Silhouette is identifying Application Data Units (ADUs) (i.e., a video segment), depicted in Figure 1. When an observer (e.g., a traffic control middle-box) sees a packet sent from the client (e.g., a video player) carrying a payload larger than a fixed threshold $L_{adu}$, the observer makes a note of the time as the start of a new ADU. Subsequently, the observer sums up the sizes of the packet payloads in the opposite direction until the next large request in the uplink direction. The sum of the payload sizes between the two requests is the size of the ADU ($adu$). The transport layer packets without any payload (e.g., pure TCP ACKs) do not carry any application layer information [11, 17] and are not used to decide ADU boundaries or video segment sizes.

4.1.1 Feature Selection. Silhouette uses video segment size and inter-request time as major video classification features. Unlike other ADU approaches [16, 17], Silhouette ADU size calculation does not require segment sequence number analysis – this is advantageous in the presence of QUIC which supports multiplexing and uses Negative Acknowledgments (NACK).

Silhouette also uses other network layer statistics common to both HTTP/TCP and QUIC/UDP streaming to improve video identification accuracy. These metrics include:

- **average downlink payload size ($\bar{L}$):** total downlink payload size divided by the number of non-empty downlink packets. Average payload size has been effective for traffic differentiation in previous work [4, 11].

- **data rate ($R$):** cumulative transport layer payload size divided by flow elapsed time. Data rate is used to differentiate video from audio, which has a considerably lower data rate (less than 192 kb/s).

4.1.2 Video Identification with ADU. The Silhouette algorithm consists of two parts: detecting an ADU (Algorithm 1) and classifying a video (Algorithm 2). Table 1 summarizes the default threshold values used by Silhouette.

In HTTP/TCP streaming, the flow uplink data consists of requests for video segments and TCP ACKs. In QUIC/UDP, the flow uplink data is more varied since QUIC sends statistics data with its NACK/ACK packets in the uplink. Silhouette uses $L_{req}$ as a minimum threshold to filter out small packets which are unlikely to carry video segment requests (Line 8 in Algorithm 1). The $L_{req}$ default threshold is 500 bytes, tuned from observing hundreds of YouTube video sessions for both HTTPS/TCP and QUIC/UDP.

ADUs alone are often not sufficient to differentiate video flows from audio flows, especially for QUIC/UDP where video and audio can be multiplexed in a single connection. Audio segments can even be requested before the complete transfer of a video segment [9]. Thus, Silhouette uses $T_r$ as a threshold for video ADU size to differentiate smaller audio. The $T_r$ default threshold is 100 KB, approximately 2 seconds of 240p video encoded at 400 kb/s.6

As discussed in Section 4.1.1, Silhouette uses average downlink payload length ($\bar{L}$) and data rate ($R$) to differentiate audio and video traffic. Silhouette uses $L_a = 900$ bytes as the default payload length threshold for video ($\frac{\bar{L}}{a}$ a typical MTU without network and transport headers), and $L_a = 450$ bytes as the default payload length threshold for audio ($\frac{\bar{L}}{a}$ a typical MTU without network and transport headers). Silhouette uses $R_a = 300$ kb/s as the default data rate threshold for video, and $R_a = 192$ kb/s as the default data rate threshold for audio.

Algorithm 1 consists of two parts: detecting an ADU (Algorithm 1) and classifying a video (Algorithm 2). Table 1 summarizes the default threshold values used by Silhouette.

### Algorithm 1 Application Data Unit Detection

1. **variables**
2. adu: application data unit length.
3. $\bar{L}$: transport layer payload length of pkt $p$.
4. $C_{adu}$: number of ADUs detected.

5. **end variables**

6. for each uplink packet $p$ in flow id do
7. $l \leftarrow payload\_len(p)$
8. if $l < L_{req}$ then
9. return
10. end if
11. if $adu \geq T_r$ then
12. $C_{adu} \leftarrow C_{adu} + 1$
13. end if
14. $adu \leftarrow 0$
15. end for

<table>
<thead>
<tr>
<th>Thresh.</th>
<th>Description</th>
<th>Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_c$</td>
<td>segment length threshold for video ADU</td>
<td>100 KB</td>
</tr>
<tr>
<td>$R_c$</td>
<td>rate threshold for video</td>
<td>300 kb/s</td>
</tr>
<tr>
<td>$R_a$</td>
<td>rate threshold for non-video</td>
<td>192 kb/s</td>
</tr>
<tr>
<td>$L_a$</td>
<td>pkt length threshold for video</td>
<td>900 B</td>
</tr>
<tr>
<td>$L_{req}$</td>
<td>pkt length threshold for non-video</td>
<td>450 B</td>
</tr>
<tr>
<td>$T_{adu}$</td>
<td>number of ADUs threshold for video</td>
<td>3</td>
</tr>
</tbody>
</table>

Algorithm 2 consists of two parts: updating the statistics information for the flow and the current ADU (Lines 10 to 18), and identifying if the flow is a video based on the statistics. Line 19 in Algorithm 2 does the video identification. Only if the flow meets all three conditions is it classified as video: 1) data rate ($R$) is greater than the minimum video rate threshold ($R_v$), 2) average payload length ($\bar{L}$) is greater than the video payload length threshold ($L_c$), and 3) at least 3 video ADUs ($T_{adu}$). Note, this last criteria is because HTTP streaming always has at least three uplink requests:

5http://perform.wpi.edu/downloads/#silhouette

6YouTube recommends encoding 240p video at 400 kb/s.
Algorithm 2 Video Flow Identification

1. variables
2. \( c \): number of pkts with transport layer payload.
3. \( l \): transport layer payload length of pkt \( p \).
4. \( t_e \): elapsed time from 1st pkt.
5. \( L \): average length of transport layer payload.
6. \( S \): cumulative length of transport layer payload.
7. \( R \): application layer throughput.

2. end variables
3. for each downlink packet \( p \) in flow id do
4. \( l \leftarrow \text{payload}_l(p) \)
5. if \( l = 0 \) then \( \triangleright \) no application level data
6. return current_type(id)
7. end if
8. \( \text{adu} \leftarrow \text{adu} + 1 \)
9. \( c \leftarrow c + 1 \)
10. \( S \leftarrow S + 1 \)
11. \( L \leftarrow L/c \) \( \triangleright \) avg payload length
12. \( R \leftarrow S/t_e \)
13. if \((R \geq R_a) \cap (L \geq L_a) \cap (C_{\text{adu}} \geq T_{\text{adu}})\) then
14. return \text{true} \( \triangleright \) likely video flow
15. else if \((S < T_p) \cup (R < R_a) \cup (L < L_a)\) then
16. return \text{false} \( \triangleright \) unlikely video flow
17. else
18. return \text{maybe} \( \triangleright \) maybe video flow
19. end if
20. end for

one to retrieve the manifest file, one to retrieve the audio segment, and one to fetch video segments.

Line 19 in Algorithm 2 identifies a non-video flow. A flow is marked as non-video if any of the following three conditions is met: 1) the cumulative payload length \( (S) \) is less than one video ADU \( (L_{\text{adu}}) \); 2) the data rate \( (R) \) is less than the audio rate threshold \( (R_a) \); and 3) the average payload length \( (L) \) is less than the audio payload length threshold \( (L_a) \).

If a flow fails to be identified as either video or non-video, it is marked as \text{maybe}, since it may be video, but it may not be. Usually, some file transfers fall into the maybe category (e.g., downloading a modest-sided picture, such as a thumbnail) since they have only one, large ADU.

4.2 Video Classification Example

Table 3 shows an example of Silhouette in action, using a YouTube QUIC/UDP streaming session of the movie trailer (Alpha). The video is 159 second long, encoded at 1080p with 24 f/s. Before the trailer begins, YouTube streams a 30 second advertisement. In order to validate the classification results, we manually enabled the "stats for nerds" option in the player.

The video format id is 248 (webm video at 1080p), and the audio format id is 250 (opus at 70 kb/s). For validation, we retrieved these two clips using the YouTube Downloader tool, and confirm that the volume of Flow\#20 and Flow\#10 match with the downloaded media file size.

From Table 2, even the single video streaming session opens 24 connections. From the server name indications (SNIs), most of the connections are not related to streaming; some are used for account management, some are used for the advertisement, and some are used to collect statistics. The flows with SNIs of "---sn---googlevideo.com might carry media content, with the exception of Flow\#21 because its duration and volume are too small. The advertisement was provided from the server r1---sn---9xp1w-cvne, and the Alpha trailer was provided from the server r6---sn---8xgp1w-cvne. Silhouette correctly identifies Flow\#13 and Flow\#20 as video without checking their SNIs. Silhouette marks Flow\#10 as \text{maybe} because it does not have a valid video ADU and its volume and duration are too small.

5 EVALUATION

Building upon the single-session classification example from Section 4.2, we conduct a large scale evaluation of Silhouette by: 1) gathering a set of traces with a known number of video flows (Section 5.1), 2) analyzing the (in-)effectiveness of SNI-based classification (Section 5.2), and 3) evaluating the effectiveness of Silhouette (Section 5.3).

5.1 Ground Truth Collection

In order to assess Silhouette more broadly, we gathered traces from 66 YouTube videos, repeatedly streamed over a range of encodings and protocols. This provides the "ground truth" — real traces known \text{a priori} to be video. We have made these traces available online for other researchers.

Before streaming, we set up a testbed to collect packet level traces from a Linux desktop with a dual core Intel i5-4460 CPU, 32 GB RAM, running the 4.11.0-rc2 kernel. The computer connected to Internet with a broadband connection (150 Mb/s) through an IEEE 802.11 a/b/g/n router at a carrier frequency of 2.4 GHz. The desktop computer was close enough to the wireless router to achieve a maximum TCP downlink throughput of 50 Mb/s, which is more than fast enough to stream 1080p video, the highest encoding rate tested. The computer uses Google Chrome (v63.0.3239.84) for streaming. A custom Python script automatically plays each video through the Chrome browser, using tcpdump to collect full packet traces.

To cover a range of typical video flows, we stream all 66 videos from the "2018 Movie Trailers" YouTube play list, with five different encoding qualities (240p, 360p, 480p, 720p, and 1080p) and two protocols (HTTP/1.0 and QUIC/UDP) – in total 660 video sessions. We manually enable the "stats for nerds" option in the YouTube player for each session to record the video server name and the format id for validation purposes. In order to avoid possible browser caching, we clear all history data after every session and restart the browser.

5.2 Server Name Indication

The server name indication (SNI) can be used to infer the possible role/content of each flow, a method used by major U.S. mobile ISPs [12]. Although our scripts initiated only 660 streaming sessions, they spawned over 15,000 flows. YouTube uses the additional flows for other tasks such as managing accounts (e.g., https://youtube/google.com), third party advertising (e.g., adservice.doubleclick.com), collecting

---

3https://rg3.github.io/youtube-dl

4Play list id: PLh2QScbhApnwDiPFT66hek$j4kky1fp_v1h, January 2018
In summary, SNIs alone are ineffective for differentiating video/non-video flows.

5.3 Silhouette

Table 3 summarizes the Silhouette classification results for our ground-truth dataset. The videos are broken down by streaming protocol (“HTTPS/TCP” and “QUIC/UDP”, the main columns) and then by video encoding (the rows). For each protocol, the classification results for the 66 video flows are tabulated: video (video correctly classified as video), maybe (video not classified as video, but not mis-classified as non-video), and non-video (video mis-classified as non-video). The “%” column depicts the percentage of video flows correctly classified as video. The final row tabulates the results for all video encoding types.

Figure 3 depicts the accuracy of Silhouette for the ground-truth dataset, with the x-axis the video encoding and the y-axis the accuracy percent. The overall accuracy for Silhouette is 87.9%, and for video qualities over 480p, accuracy reaches 95%. Silhouette accuracy is lowest for HTTPS/TCP 240p video (24%) where 39 out of the 66 videos are marked as maybe. Looking more closely, for 240p and 360p videos, HTTPS/TCP-based streaming experiences long idle durations that bring down average flow rates below the video rate threshold (R_t = 300 kb/s).

Although there are multiple connections in one single YouTube session (as the example in Table 2 shows), Silhouette does not classify any flows outside of SNI r’−sn*.googlevideo.com as video — i.e., Silhouette has a zero false positive rate.

6 CONCLUSIONS

Accurate, real-time classification of encrypted video is becoming increasingly important given the dominance of HTTP-streaming video and the growth in end-to-end encryption.
Table 3: Video Detection Results

<table>
<thead>
<tr>
<th>Video Resolution</th>
<th>HTTPS/TCP</th>
<th>QUIC/UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vid.</td>
<td>maybe</td>
</tr>
<tr>
<td>240p</td>
<td>16</td>
<td>39</td>
</tr>
<tr>
<td>360p</td>
<td>49</td>
<td>14</td>
</tr>
<tr>
<td>480p</td>
<td>63</td>
<td>3</td>
</tr>
<tr>
<td>720p</td>
<td>64</td>
<td>3</td>
</tr>
<tr>
<td>1080p</td>
<td>66</td>
<td>0</td>
</tr>
<tr>
<td>All</td>
<td>256</td>
<td>99</td>
</tr>
</tbody>
</table>

Figure 2: Flow Statistics in "Ground Truth" Data Set

Figure 3: Silhouette Accuracy

This paper presents Silhouette, a heuristic video identification method suitable for YouTube streaming. Silhouette classifies based only on the flow "shape" in terms of data rates and payload lengths, making it suitable for both HTTPS/TCP-based and QUIC/UDP-based streaming. Since Silhouette does not require any application layer information nor server name indication (SNID), it is effective even for encrypted videos. Moreover, because Silhouette does not classify based on packet inter-arrival times, it should be robust for all link types. Silhouette is light-weight—it does not require any high-CPU intensive processing (e.g., deep packet inspection (DPI)) nor external cross layer signaling, making it suitable for ISP middleboxes that need to classify and then treat video flows.

Evaluation with a "ground truth" dataset of 660 YouTube videos covering a range of content and encoding rates shows classification accuracy near 90% and closer to 100% for high-definition videos. While Silhouette should be effective regardless of provider (e.g., Netflix versus YouTube), future work includes evaluation over more content providers. Future work also includes a real-time implementation using a Linux Traffic Control (TC) service, and deployment into middle-boxes at an ISP to detect and treat video flows.

REFERENCES