Characterizing User Platforms for Video Streaming in Broadband Networks

Yifan Wang
University of New South Wales
Sydney, NSW, Australia
wangyifan.frank@student.unsw.edu.au

Minzhao Lyu
University of New South Wales
Sydney, NSW, Australia
minzhao.lyu@unsw.edu.au

Vijay Sivaraman
University of New South Wales
Sydney, NSW, Australia
vijay@unsw.edu.au

Abstract

Internet Service Providers (ISPs) bear the brunt of being the first port of call for poor video streaming experience. ISPs can benefit from knowing the user's device type (*e.g.,* Android, iOS) and software agent (*e.g.,* native app, Chrome) to troubleshoot platform-specific issues, plan capacity and create custom bundles. Unfortunately, encryption and NAT have limited ISPs' visibility into user platforms across video streaming providers. We develop a methodology to identify user platforms for video streams from four popular providers, namely YouTube, Netflix, Disney, and Amazon, by analyzing network traffic in real-time. First, we study the anatomy of the connection establishment process to show how TCP/QUIC and TLS handshakes vary across user platforms. We then develop a classification pipeline that uses 62 attributes extracted from the handshake messages to determine the user device and software agent of video flows with over 96% accuracy. Our method is evaluated and deployed in a large campus network (mimicking a residential broadband network) serving users including dormitory residents. Analysis of 100+ million video streams over a four-month period reveals insights into the mix of user platforms across the video providers, variations in bandwidth consumption across operating systems and browsers, and differences in peak hours of usage.

CCS Concepts

• **Networks** → **Network measurement**; **Network monitoring**; • **Information systems** → *Multimedia streaming*; • **Computing methodologies** → *Machine learning approaches.*

Keywords

Network traffic analysis, user platform identification, video streaming, TLS fingerprinting

ACM Reference Format:
Yifan Wang, Minzhao Lyu, and Vijay Sivaraman. 2024. Characterizing User Platforms for Video Streaming in Broadband Networks. In *Proceedings of the 2024 ACM Internet Measurement Conference (IMC '24), November 4–6, 2024, Madrid, Spain.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3646547.3688435

IMC '24, November 4–6, 2024, Madrid, Spain

ACM ISBN 979-8-4007-0592-2/24/11
https://doi.org/10.1145/3646547.3688435

1 Introduction

Broadband Internet Service Providers (ISPs) are often the first to be blamed by users experiencing video freeze or grainy resolution, even when the issue is unrelated to the network. In many instances, the issues arise from the user platform – choppy video playback on Pixel devices in 2020 was attributed to Android 11 Beta issues [69]; the YouTube app threw up errors on iOS devices in late 2022 [35]; a software update to Roku devices in 2021 caused intermittent video freeze [50]; and Hulu deliberately lowered resolution on PC browsers in order to force users to download their proprietary app [13, 66]. ISPs, who bear the brunt of customer support calls, can hugely benefit by knowing the user platform, namely device type (iOS or Android smartphone or tablet, Windows or Mac PC, smart TV, Xbox or PlayStation console) and software agent (native app versus a specific browser such as Chrome, Firefox, Safari or Edge) on which a household user is having a poor video streaming experience. This will allow their customer care staff to rapidly filter out known platform-specific issues, prioritize handling of support tickets based on platform prevalence, and issue preemptive advisories to users, all of which can substantially reduce support costs.

Visibility into the user platform has other benefits as well for ISPs. The same video watched via a content provider's app may consume significantly higher bandwidth than when watched on a web browser, and these differences can amplify across operating systems [7, 26]. Given that video streaming dominates network traffic, ISP bandwidth provisioning and management models need to account for user device type and software agent heterogeneity, which vary widely from one content provider (*e.g.,* YouTube) to another (*e.g.,* Netflix). Further, user platform visibility also enables ISPs to perform better customer segmentation (*e.g.,* fans who stream live sports via set-top boxes), allowing them to create innovative custom bundles and to run effective up-sell/cross-sell marketing campaigns. Collectively, these initiatives can give an ISP significant competitive advantage, helping reduce costs as well as generate new revenues.

Deducing the user platform associated with a streaming session is unfortunately non-trivial for the ISP. Much of the traffic to/from the home today is generally on a single IPv4 address, which is shared among all household devices via network address translation (NAT). While IPv6 may eventually overcome this issue, deployment is immature in many ISPs globally, and is unlikely to displace IPv4 anytime soon. Further, deducing the software agent, such as a browser versus a native app, will require a different technique irrespective of whether the traffic is IPv6 or IPv4, since client-server interactions are predominantly carried within encrypted SSL/TLS sessions today.

Much attention has been given to classifying application streams, such as web browsing [47, 63], video streaming [19, 41], video conferencing [42], online gaming [36], cloud gaming [33] and metaverse [34], but less to identifying the client platform (OS and software agent) associated with each application stream. OS fingerprinting is done in [20, 57], but not coupled with the software agent – native apps are often optimized differently than browsers for streaming video [62]. Prior works have leveraged distinguishable patterns of TCP-based handshake fields across device firmware [14, 28] and application types [6, 52], with cipher suites shown to vary from one OS to another [22], and certain TLS handshake fields being unique to certain browsers [6]. Unlike prior studies that have a broad focus, our study focuses exclusively on video streams, which constitute over 60% of Internet traffic. We dive deep into the differences across OSes, browsers, content providers, and their native apps. Further, we consider the presence of QUIC, which is increasingly being used by video content providers – measurements in tier-1 ISPs indicate that QUIC accounts for nearly 30% of traffic in EMEA and 16% in North America [21]. Our methods are therefore more comprehensive in giving ISPs visibility into both device type and software agent for every video stream in their network.

Our **first** contribution (§3) comprehensively studies the communication process involved in the establishment of a video streaming session across 30 user platforms, *i.e.,* combinations of device types and software agents. The device types comprise mobiles, laptops, tablets, PCs and smart TVs running iOS, Android, Windows, macOS, Android TV, etc. For software agents, we consider native apps developed by streaming video providers and browsers such as Chrome, Firefox, Safari and Edge. By collecting and analyzing over 10,000 video flows in our lab from the four major providers, namely YouTube, Netflix, Amazon Prime Video and Disney+, we highlight the variations in their TCP/QUIC and TLS handshake parameters across the 30 user platforms.

For our **second** contribution (§4), we develop a machine learning-based pipeline to classify the user platform of a video streaming flow. We systematically extract 62 attributes from the initial connection establishment phase of the TCP or QUIC video flow; we then train machine learning models that use different algorithms, hyperparameters and attribute subsets grouped by their computational cost. We demonstrate that our ML models achieve an accuracy of above 96% in our lab datasets, outperforming prior methods across every video provider considered.

Our **third** contribution (§5) is the implementation and deployment of our system in a university campus serving tens of thousands of users and a characterization of the video streaming traffic observed across 4 months – over 100 million video flows spanning 400k hours of watch time. Our data reveals that mobile devices are more commonly used for YouTube, while PCs/laptops predominate for the other (subscription-based) video services. The latter also seem to demand higher bandwidth, with Amazon Prime Video being the most demanding, especially on Mac PCs that consume 50% more on average than even smart TVs. We believe that our system and the insights collected may be of help for network operators seeking to link video streaming services to capacity planning, user experience troubleshooting and user segmentation. In the spirit of reproducibility and to foster discussion in the research community, we make publicly available the code and training data at [70].

2 Related Work

OS/application classification: Classifying operating systems and applications has received considerable attention [9, 15, 46, 48]. Prior works that purely use common network and transport layer signatures to classify OS and application have been rendered inaccurate with the growth of complexity in user platforms [2, 8, 32, 73]. Therefore, recent research works leverage certain fields in TLS ClientHello (CHLO) messages to classify device OSes and application types [29, 72], as such information is directly related to client-side firmware and software configurations for their encryption preferences. For example, M. Husak *et al.* [22] used cipher suites from TLS CHLO messages to classify web browsers. M. Lastovicka *et al.* [28] used machine learning models to classify operating systems using 7 features (*e.g.,* server name and TLS version) extracted from TLS CHLO messages. B. Anderson *et al.* [5] jointly considered current and past TLS CHLO characteristics of a certain device for high-confidence detection of device OS versions. Other works combined TLS CHLO fields with flow statistics (*e.g.,* packet size distribution) to classify OS and/or application types [14, 25, 44, 58, 68]. Compared to prior research, our work focuses on the classification of user platforms (including both device types and software agents), instead of traffic (*i.e.,* application type) classification which in our case is based on TLS SNI matching – our ML models are activated once the video streaming application is detected. Specifically, we consider per-video streaming flow using TCP/QUIC and TLS handshake fields, without relying on aggregated statistics per device IP which can be rendered ineffective in residential networks with NAT in place. Instead of cherry-picking certain header fields, we evaluate 62 attributes that cover all available handshake fields of a video flow that can vary across user platforms, many of which (particularly for QUIC) are not considered in prior works. Later in §4.3.4, we empirically demonstrate the superior performance of our method over prior techniques.

Fingerprinting handshake fields: A group of prior research works developed tools to measure variations of (TLS) handshake fields that can exist in different device types (*e.g.,* client or server), applications (*e.g.,* browsers or social media apps), and vulnerable firmware and software configurations. For such purposes, JA3 [4] has been developed as a popular tool for fingerprinting TLS CHLO fields from a network device, which has been extended to include server-side fingerprinting in JA3S [3]. A. Razaghpanah *et al.* [52] developed a tool for extracting certain TLS CHLO fields from network traffic from Android devices, which are useful in identifying those with security vulnerabilities or misconfigurations. B. Anderson *et al.* [6] proposed a system that extracts TLS CHLO messages from standard operational environment devices in an enterprise network as an effective monitoring measure for the running processes on each host and their TLS configurations. M. Sosnowski *et al.* [61] evaluated five popular server-side TLS fingerprinting methods and developed an active server TLS scanner. Prior works only focused on certain handshake fields in TCP-based TLS flows. In this paper, we develop a system that automatically extracts and formalizes all available handshake fields from not only TCP-based TLS but also QUIC-based TLS for video flows.

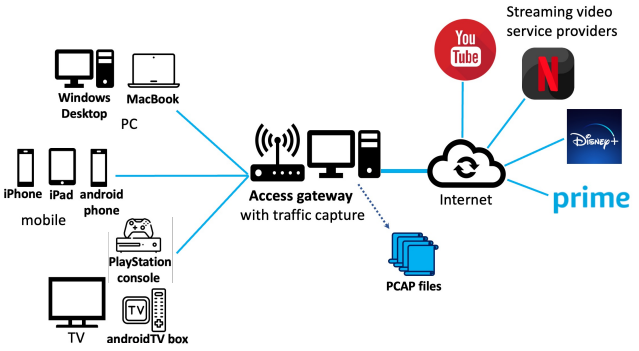


Figure 1: Experimental setup for video streaming traffic trace collection.

Traffic analysis of streaming video services: There is a large body of prior research works that analyze network traffic for streaming video services to provide visibility into the usage and user experience of video sessions for network operators. Machine learning algorithms that consume statistical attributes extracted from video flows such as packet inter-arrival times, downstream/upstream packet rates and throughput are extensively used by these works. Their prediction objectives include detecting video flows [12, 19], classifying content providers [10, 37], modeling video adaptation behaviors [43, 71], and inferring the status of streaming experience such as resolution, stall, and startup delay [1, 11, 18, 67]. An important aspect that has not been well captured by prior works is identifying user platforms of video flows, which can influence the observed streaming behavior and the user experience [30]. In this paper, we develop a lightweight method focused on responsively characterizing user platforms for each streaming video flow using only handshake messages prior to the delivery of actual video content.

3 Handshake Characteristics of Video Streaming Sessions

In this section, we describe our experimental setup for capturing traffic traces (§3.1), then systematically study the communication process involved in establishing video streaming sessions using a variety of device types and software agents (§3.2), and finally identify the handshake fields that take on different values across user platforms and video content providers (§3.3).

3.1 Experimental Setup and Dataset

In Fig. 1, we show our experimental framework to collect traffic trace files (*i.e.,* PCAPs) for video streaming sessions from the four major video content providers, namely YouTube, Netflix, Amazon Prime Video and Disney+. The setup consists of 3 mobile devices (iPhone, iPad, Android phone) running iOS and Android operating systems, 2 Windows desktop PCs, 2 macOS MacBooks, 2 smart TVs (including one with a built-in Android TV system and the other one connected to an Android TV set-top box), and a PlayStation gaming console. Having a variety of different devices expedites data collection with multiple users streaming videos concurrently.

Table 1: Number of video flows per content provider, *i.e.,* YouTube (YT), Netflix (NF), Amazon Prime Video (AP) or Disney+ (DN) captured for each combination of device type and software agent in our collected traffic traces. A user platform not supported by the content provider is marked as —.

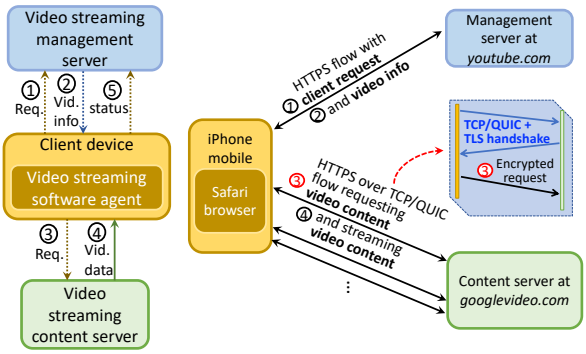
Device	OS Type	Software Agent	Number of video flows			
			YT	NF	DN	AP
PC	Windows	Chrome	411	202	199	215
		Edge	406	208	200	200
		Firefox	466	207	204	195
		Native app	—	204	211	186
	macOS	Safari	200	204	200	201
		Chrome	407	213	202	208
		Edge	402	204	202	210
		Firefox	467	212	202	199
		Native app	—	—	—	200
Mobile	Android	Chrome	107	—	—	—
		Samsung Internet	103	—	—	—
		Native app	100	102	106	111
	iOS	Safari	203	—	—	—
		Chrome	213	—	—	—
		Native app	203	215	306	372
TV	Android TV	Native app	200	116	107	113
	PlayStation	Native app	105	100	100	103

These devices connect via Wi-Fi to the access gateway for Internet access, from which we collect PCAP files using the Wireshark tool.

We capture traffic traces containing more than 100 video sessions for each of the 30 user platforms in our lab, comprising various device types and software agents. Our dataset contains 17 unique types of user platforms as we have redundancy for some OS types such as 2 Windows and 2 macOS. Each session is composed of one or multiple video flows, for a total of nearly 10,000 flows. We show the detailed composition of our dataset in Table 1. Some of the devices run the same OS, thus are considered as being the same from a user platform perspective. For example, both iPhone and iPad in our setup are using iOS, and so are captured under a single (Mobile, iOS) category in Table 1. The duration of each session was at least one minute, sufficient to capture all the handshake messages exchanged between the user device and the server of the respective provider. Note that we are not interested in capturing packets containing any video content payload. The devices were running the most recent versions of operating systems, browsers and apps. Some browsers, *e.g.,* Chrome, Firefox and Edge on Windows/Mac PCs, allow users to configure the transport layer protocol as either QUIC or TCP, which can impact the connection establishment messages exchanged. Our dataset encompasses all these different scenarios and has comprehensive coverage across all different configuration options. Our dataset is available on our university cloud storage platform and can be shared upon request, as detailed in [70].

3.2 Anatomy of Video Streaming Sessions

We now discuss the typical communication process involved in establishing a video streaming session between a client device and a streaming server.



(a) Communication process. (b) Practical example using YouTube.

Figure 2: Anatomy of network communication for a video streaming session.

3.2.1 Communication process. A streaming video session essentially consists of two sequential stages, *i.e.,* initialization and playback. The initialization stage, comprising connection setup handshake exchanges, alongside sharing of relevant metadata information, has received little attention from the community; later we will show its importance in characterizing user platforms. The playback stage on the other hand, comprising streaming of the video itself, has been extensively studied for detecting video traffic streams, quantifying QoE, and so on, as summarized in §2.

A flow diagram depicting the two stages is shown in Fig. 2(a). In the initialization stage, the client device interacts with a management server specifying the service request, user configurations and connection parameters such as the device type, software agent used and supported network protocols; step ① in Fig. 2(a). Then, in step ②, the management server responds with the streaming information (*e.g.,* URLs of the content servers and video/audio formats) along with control parameters (*e.g.,* media player configurations) to be adapted by the software agent. In step ③, the actual video playback process starts after the client requests video and audio data from the content server located using the URL information acquired in the previous step. This request also specifies video quality metrics such as resolution which can be adjusted at any time during the playback process, either manually or dynamically by the client-side player depending on the network conditions and playback quality. In step ④, the video and audio data are streamed to the client. The software agent on it may periodically send playback status information to a management server, as indicated in ⑤, to help the content provider keep track of service usage, session status, video quality and the like.

Steps ① to ④ exist in all video streaming traces we have collected across the different providers, whereas step ⑤ is only observed in certain video sessions such as on macOS devices watching YouTube on a Chrome browser.

3.2.2 Anatomy of network communication. From the perspective of network communication, the above steps are carried by HTTPS flows over either TCP or QUIC as the transport layer protocol. To illustrate the detailed communication process, Fig. 2(b) shows an example of a YouTube session operated from an iPhone using Safari browser. Video sessions on other device types, software agents and content providers share a similar anatomy, omitted for brevity. The steps circled in Fig. 2(b) correspond to those shown in Fig. 2(a).

The initial client request flow, *i.e.,* step ①, is always sent via a single HTTPS flow to a management server, typically *youtube.com*, *netflix.com*, *primevideo.com*, *disneyplus.com* for the four content providers, respectively. For YouTube, the flow is either carried by TCP or QUIC depending on the client configuration, whereas the other three providers, *i.e.,* Netflix, Amazon and Disney+ use only TCP. In step ②, the management server sends streaming information and player configuration to the client on the same HTTPS flow. However, if the client is configured to pre-load video metadata on a menu page, rather than requesting it dynamically from the server, then this information could be carried over multiple subsequent HTTPS flows.

During the video playback stages, steps ③ and ④, the client fetches the video from a content server (*e.g., googlevideo.com* for YouTube) using one or more HTTPS flows over either TCP or QUIC. We have observed three scenarios in this process. One, in which playback is delivered by a single HTTPS flow containing both video and audio data. Two, in which playback is delivered over multiple concurrent HTTPS flows carrying video and audio. Three, in which multiple HTTPS flows are activated in different time slots, each delivering chunks of video and audio data. For example, in some YouTube sessions we have collected, there are flows that send several chunks of video and audio data in the first few seconds (*e.g.,* 3 seconds) and then go idle while the remaining video and audio data is streamed by another flow. Since client information and playback data are all encrypted by TLS, network operators only have visibility into the TCP/QUIC and TLS handshake messages, as visually shown in the blue region in Fig. 2(b), and volumetric information of the payloads.

Next, we show that handshake messages carried by the first few (< 5) connection establishment packets contain fields that are strong indicators to characterize user platforms.

3.3 Handshake Characteristics Across User Platforms

The flows that stream video content are first initialized by a series of handshake messages. By analyzing the trace files, we find that information within these handshake messages is highly correlated with the OS, software agent and the provider of video content. In the following, we systematically discuss these properties.

3.3.1 Handshake fields and their categorization. A video flow delivered by HTTPS has two handshake processes: one for the transport layer protocol (*i.e.,* TCP or QUIC) and the other for TLS encryption, both of which occur prior to the encrypted video content being streamed.

Transport layer handshake. Both TCP and QUIC require a handshake process to establish connections. A TCP three-way handshake contains TCP header flags and options, such as window size, selective acknowledgment and max segment size, which are set by the user device and the software agent. We ignore most of the TCP flags in handshake messages such as SYN, ACK, RST and FIN as

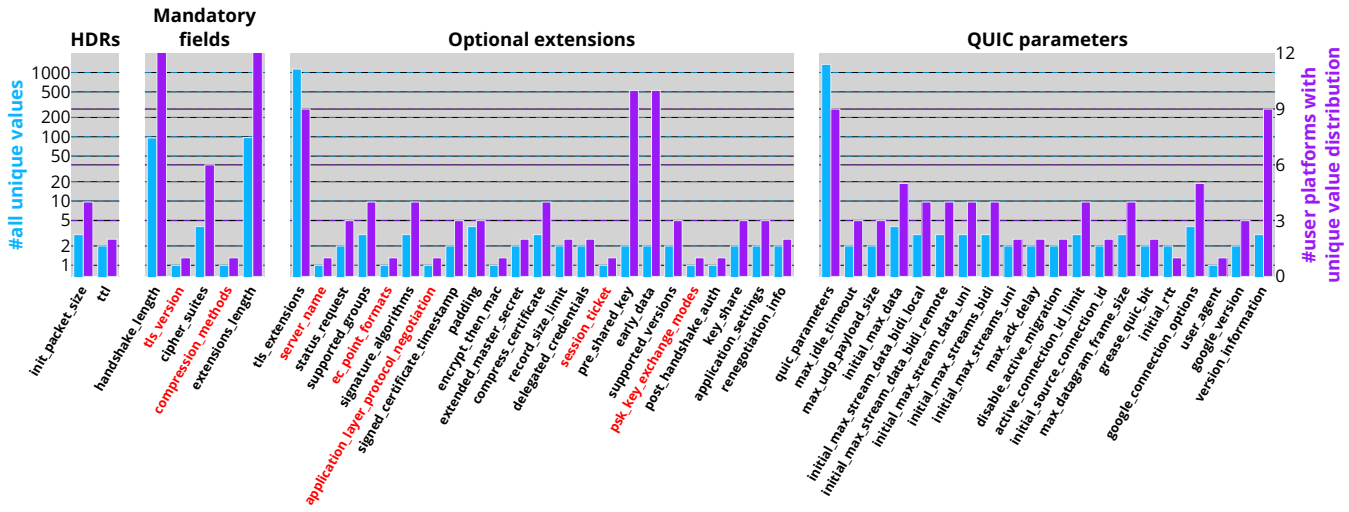


Figure 3: Number of unique values (left blue in log scale) and number of user platforms with different value distributions (right purple in linear scale) for each handshake field in YouTube flows over QUIC.

they do not differ across user platforms. Exceptions are TCP CWR and ECE, which are related to congestion control policies used by the device OS or software agent [16].

QUIC is designed to reduce the connection setup latency overhead and so its handshake via the first flow packet (*i.e.,* QUIC Initial packet) is integrated with the TLS handshake, as discussed below. In addition, we find that the time-to-live field and packet size in the IP header of the Initial packet are often correlated with the device type, as reported by [60].

TLS handshake. TLS handshake consists of a ClientHello followed by a ServerHello and subsequent encryption negotiations, and is executed right after the TCP three-way handshake or along with the first QUIC flow packet. The ClientHello contains customized information provided by the user device and is highly correlated with the device type and software agent that plays the video. We decompose the TLS handshake fields into three categories.

The first category is called **mandatory fields**. These fields always appear in the ClientHello of streaming video flows regardless of the underlying transport layer protocol (TCP or QUIC) and the specifications of user devices, OSes and software agents. These include handshake length, TLS version, cipher suites and compression methods.

The second category is **optional extensions**. These fields only appear in video flows as defined by the logic embedded in a device OS and software agent. In our dataset, a given user platform typically uses unique combinations of optional extensions, each set to a specific value. For example, Firefox browsers running on Windows and macOS PCs typically set the value of *record_size_limit* extension to 16385, whereas other user platforms do not use this extension.

In addition to the above categories, there are **parameters** in the ClientHello that are specifically available for video flows over **QUIC**. These parameters are contained in the collection *quic_transport_parameters* with extension code 57 [23, 65], which are set for specific QUIC preferences in connection establishment of certain user platforms. For example, Firefox browsers on Windows desktop PCs use the parameter *grease_quic_bit* to indicate its deprecation of certain flag bits in QUIC headers [64].

3.3.2 Handshake fields across user platforms. We now analyze the distribution of values contained in the fields of various TCP/QUIC and TLS handshake messages. The aim is to demonstrate the similarities and differences in the values contained within these fields across user platforms, which will form the basis of our machine learning model. We note that some fields are not numerical but categorical or lists, such as the mandatory fields *tls_version* and *cipher_suites* in TLS CHLO and *supported_groups*, *signature_algorithms* in TLS optional extensions. To simplify the analysis, we convert the values contained in such fields to integers by a 1:1 mapping between the values contained in the fields to a unique number. For instance, in our dataset, the field *compress_certificate* takes 2 values when carried over QUIC, *i.e., zlib* and *brotli*, which are uniquely mapped as 1 and 2 for the purposes of our analysis. Therefore, a video flow containing *zlib* as the value for *compress_certificate* is represented as 1 in our dataset. If a field does not appear in a flow, a value of 0 is assigned to it.

For each handshake field in YouTube QUIC video flows, we show the number of unique values as a blue bar in Fig. 3 (in log scale), while the number of user platforms having a unique value distribution compared to their counterparts for this particular field is shown as a purple bar (in linear scale). There are 7 fields that only have one unique value regardless of user platforms. These are highlighted as red labels in Fig. 3. Apparently, those fields are useless in differentiating user platforms for YouTube QUIC video flows. However, four of these fields, *i.e., ec_point_formats*, *ALPN*, *session_ticket* and *psk_key_exchange_modes* take different values across user platforms for TCP flows, thus, can serve as useful indicators for TCP scenarios. We provide two heatmaps as Appendix B Fig. 12(a) and Fig. 12(b) to show the median values of all handshake fields in

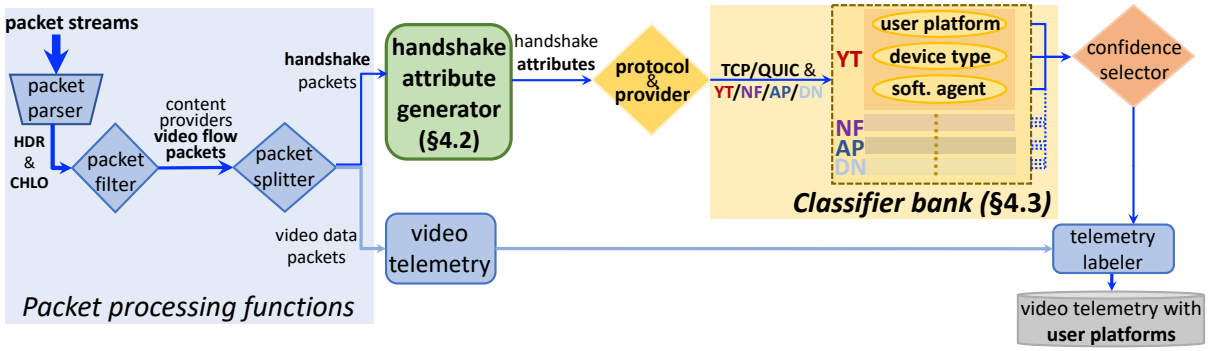


Figure 4: Packet processing pipeline for classification of video streaming user platforms.

both TCP and QUIC YouTube video flows respectively, rather than just their counts.

As detailed in Appendix B, similar conclusions on value distributions of handshake fields can be reached for other three video providers we have studied in this work, which are not explicitly discussed here for simplicity. In the next section, aiming for better classification performance, we systematically evaluate the importance of formalized attributes extracted from those handshake fields.

4 Classifying User Platforms for Video Streams in Real-Time

In this section, we first develop a classification pipeline (§4.1), then identify the fields that are relevant for classification purposes (§4.2), and finally evaluate the efficacy of three machine learning algorithms to achieve our desired objectives (§4.3).

4.1 Packet Processing Pipeline

We develop a generalized packet processing pipeline to classify user platforms for each streaming video flow and apply it to the four providers considered in this paper. The schematic of the pipeline is shown in Fig. 4.

The pipeline first takes raw packet streams as input, and parses them to identify video flows that belong to the four providers using port numbers and service names extracted from unencrypted packet headers (HDR) and ClientHello (CHLO) SNIs. These packets are further split into handshake packets for classification and payload packets for telemetry, *e.g.,* to obtain session duration, volume and throughput. This forms the preprocessing stage.

Next, each handshake packet is processed to extract attributes that can be readily fed into three machine learning classifiers that predict the composite user platform, device type and software agent for the respective streaming service provider. Therefore, in our later implementation that analyzes four video streaming providers, we have twelve classifiers (*i.e.,* three classifiers per provider) in total. The code for extracting handshake attributes from Client Hello packets is made publicly available at [70]. In the small minority of cases where the confidence (*i.e.,* probability of the predicted class) in predicting the composite user platform is $< 80\%$, we consider predicting the device type (*i.e.,* OS) and software agent (*i.e.,* native app or specific browser) individually, to have high confidence in classifying either of them accurately, so that at least partial information related to the user platform can be predicted confidently. The predicted user platform for each video flow can then be correlated using flow metadata and timestamps with real-time telemetry for further insight analysis. If none of the classifiers can produce an inference result with $> 80\%$ confidence, we reject the classification and determine the video flow to be from an unknown user platform. Further classifying the unknown user platform requires respective ground-truth training data to be collected, thus, is not within the scope of this paper.

4.2 Attributes Derived from Handshake Fields

We identify 62 fields of interest, of which 20 are numerical, 31 categorical, and 11 of type list. The detailed specifications of these attributes, applicable to both QUIC and TCP video flows, are provided in Table 2.

4.2.1 Creating attributes from fields contained in handshake messages. For the purpose of feeding the fields of interest into our machine learning models, we seek to first convert them into numerical attributes. We begin by considering those that are inherently numerical, such as *handshake_length* and *extensions_length*, for which no transformation is needed.

Fields that are categorical take values from a finite set of elements. For example, *compress_certificate* can take one of *zlib* and *brotli*. We assign a unique positive integer to them, *i.e.,* 1 and 2 respectively, and transform this field into a numerical attribute. We assign 0 to categorical fields that are not present in a flow.

Fields that are lists can in turn include many categorical fields. For example, *cipher_suites* contains multiple cipher suites from a finite list supported by a client. The order of the categorical items in the field indicates the client's preference. Therefore, to preserve the information provided by the choice of items and their order in a list-type field, we use a fixed-length vector (*i.e.,* by encoding the list as separate positional attributes) to indicate the placement of each item, with zero-padding for non-existing items. This simplified representation can be readily processed by the classifiers.

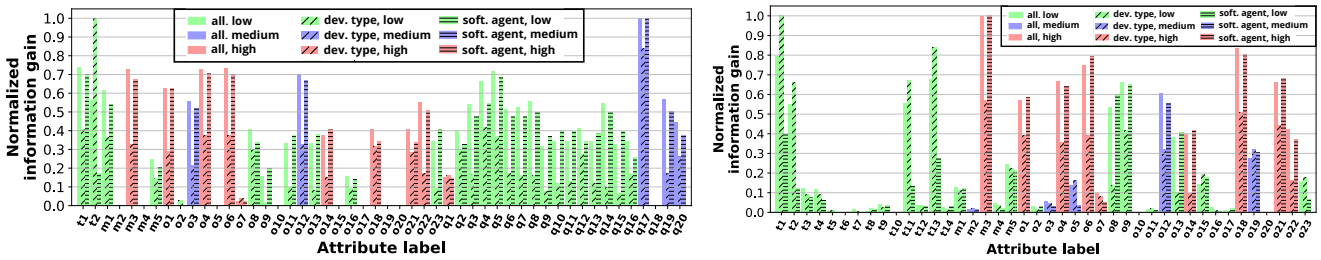
For 17 fields such as *grease_quic_bit*, they do not have any associated value. However, whether they are present or not differs across device types and software agents. Therefore, we assign 1 for their presence in a video flow and 0 otherwise. Also, for 7 fields such as *initial_source_connection_id* in QUIC, the values they contain are not useful since they are randomly chosen [24], but the length of

Table 2: Handshake fields of video flows and formalized attributes.

Handshake field	Transport protocol	Category	Attribute label	Attribute type	Attribute cost
init_packet_size	TCP, QUIC	transport layer	t_1	numerical	low
ttl	TCP, QUIC	transport layer	t_2	numerical	low
tcp_cwr	TCP	transport layer	t_3	presence	low
tcp_ece	TCP	transport layer	t_4	presence	low
tcp_urg	TCP	transport layer	t_5	presence	low
tcp_ack	TCP	transport layer	t_6	presence	low
tcp_psh	TCP	transport layer	t_7	presence	low
tcp_rst	TCP	transport layer	t_8	presence	low
tcp_syn	TCP	transport layer	t_9	presence	low
tcp_fin	TCP	transport layer	t_{10}	presence	low
tcp_window_size	TCP	transport layer	t_{11}	numerical	low
tcp_mss	TCP	transport layer	t_{12}	numerical	low
tcp_window_scale	TCP	transport layer	t_{13}	numerical	low
tcp_sack_permitted	TCP	transport layer	t_{14}	presence	low
handshake_length	TCP, QUIC	mandatory fields	m_1	numerical	low
tls_version	TCP, QUIC	mandatory fields	m_2	categorical	medium
cipher_suites	TCP, QUIC	mandatory fields	m_3	list	high
compression_methods	TCP, QUIC	mandatory fields	m_4	length	low
extensions_length	TCP, QUIC	mandatory fields	m_5	numerical	low
tls_extensions	TCP, QUIC	optional extensions	o_1	list	high
server_name	TCP, QUIC	optional extensions	o_2	length	low
status_request	TCP, QUIC	optional extensions	o_3	categorical	medium
supported_groups	TCP, QUIC	optional extensions	o_4	list	high
ec_point_formats	TCP, QUIC	optional extensions	o_5	categorical	medium
signature_algorithms	TCP, QUIC	optional extensions	o_6	list	high
application_layer_protocol_negotiation	TCP, QUIC	optional extensions	o_7	list	high
signed_certificate_timestamp	TCP, QUIC	optional extensions	o_8	length	low
padding	TCP, QUIC	optional extensions	o_9	length	low
encrypt_then_mac	TCP, QUIC	optional extensions	o_{10}	presence	low
extended_master_secret	TCP, QUIC	optional extensions	o_{11}	presence	low
compress_certificate	TCP, QUIC	optional extensions	o_{12}	categorical	medium
record_size_limit	TCP, QUIC	optional extensions	o_{13}	numerical	low
delegated_credentials	TCP, QUIC	optional extensions	o_{14}	list	high
session_ticket	TCP, QUIC	optional extensions	o_{15}	length	low
pre_shared_key	TCP, QUIC	optional extensions	o_{16}	presence	low
early_data	TCP, QUIC	optional extensions	o_{17}	length	low
supported_versions	TCP, QUIC	optional extensions	o_{18}	list	high
psk_key_exchange_modes	TCP, QUIC	optional extensions	o_{19}	categorical	medium
post_handshake_auth	TCP, QUIC	optional extensions	o_{20}	presence	low
key_share	TCP, QUIC	optional extensions	o_{21}	list	high
application_settings	TCP, QUIC	optional extensions	o_{22}	list	high
renegotiation_info	TCP, QUIC	optional extensions	o_{23}	presence	low
quic_parameters	QUIC	QUIC parameters	q_1	list	high
max_idle_timeout	QUIC	QUIC parameters	q_2	numerical	low
max_udp_payload_size	QUIC	QUIC parameters	q_3	numerical	low
initial_max_data	QUIC	QUIC parameters	q_4	numerical	low
initial_max_stream_data_bidi_local	QUIC	QUIC parameters	q_5	numerical	low
initial_max_stream_data_bidi_remote	QUIC	QUIC parameters	q_6	numerical	low
initial_max_stream_data_uni	QUIC	QUIC parameters	q_7	numerical	low
initial_max_streams_bidi	QUIC	QUIC parameters	q_8	numerical	low
initial_max_streams_uni	QUIC	QUIC parameters	q_9	numerical	low
max_ack_delay	QUIC	QUIC parameters	q_{10}	numerical	low
disable_active_migration	QUIC	QUIC parameters	q_{11}	presence	low
active_connection_id_limit	QUIC	QUIC parameters	q_{12}	numerical	low
initial_source_connection_id	QUIC	QUIC parameters	q_{13}	length	low
max_datagram_frame_size	QUIC	QUIC parameters	q_{14}	numerical	low
grease_quic_bit	QUIC	QUIC parameters	q_{15}	presence	low
initial_rtt	QUIC	QUIC parameters	q_{16}	presence	low
google_connection_options	QUIC	QUIC parameters	q_{17}	categorical	medium
user_agent	QUIC	QUIC parameters	q_{18}	categorical	medium
google_version	QUIC	QUIC parameters	q_{19}	categorical	medium
version_information	QUIC	QUIC parameters	q_{20}	categorical	medium

the values, in terms of the number of bytes they contain, could be helpful, and hence we treat them as length-based attributes.

We note that converting each field to its corresponding attribute (*i.e.,* the green box in Fig. 4) can come with different levels of processing steps that can require different levels of computational costs, especially when deployed for high-speed broadband networks in real time. In our real-time packet processing pipeline as shown in Fig. 4, numerical fields are directly taken from handshake packets as input attributes. Length-based and presence-based attributes are also directly taken from their corresponding header fields without



(a) Attribute importance for **YouTube** **QUIC** flows.

(b) Attribute importance for **YouTube** **TCP** flows.

Figure 5: Attribute importance in different classification objectives including user platforms, only device types, or only software agents for YouTube (a) QUIC and (b) TCP flows. The level of preprocessing required (annotated by low-, medium- or high-cost) and classification objectives are represented by different colors and patterns, respectively.

additional processing. These attributes can hence be considered low-cost. Attributes that are converted from categorical fields each requires an additional value mapping process after packet parsing, which is typically done through basic dictionary lookup operations. Albeit small in terms of time consumption (*e.g.,* several microseconds), there is an additional processing step involved, hence these attributes are considered medium-cost. A list-type field can contain multiple (categorical) items, each requiring an additional value mapping process to convert the entire list-type field into one attribute in the form of an array of numerical values. For example, the list-type field *cipher_suites* often has over ten *cipher_suite* items. To convert one such field into its corresponding attribute (*i.e.,* m_3) as an array of numerical values, the system needs to loop through every single item in the list, resulting in a total of over ten additional value mapping processes required. Therefore, such attributes are considered high-cost. As we will show next, using attributes with higher preprocessing costs in classification tasks does not necessarily increase the overall predictive capability.

4.2.2 Importance of attributes. As discussed in §3.3, the attributes derived from the handshake fields are not all equally important in predicting user platforms. We now systematically benchmark the importance of each attribute using the *information gain* metric [51]. It is indeed the mutual information between an attribute and the prediction target, which is calculated as the sum of the entropies of the attribute and the predicted class deduced by their joint entropy [27, 56]. Therefore, attributes of the highest importance have information gain close to 1 and an irrelevant attribute has information gain 0. In our analysis, the information gains are computed for both TCP and QUIC video flows for each of the four video providers, with prediction objectives being user platform, device type, and software agent. We now discuss the relative importance of the attributes using YouTube video flows over QUIC as a representative example.

Fig. 5(a) shows the importance (*i.e.,* normalized information gain) of attributes for YouTube over QUIC flows. Each bar is color-coded by its level of preprocessing required and pattern-coded by its prediction objective. Attributes are denoted by their ordered labels, and a full mapping between labels and attribute names is provided in Table 2.

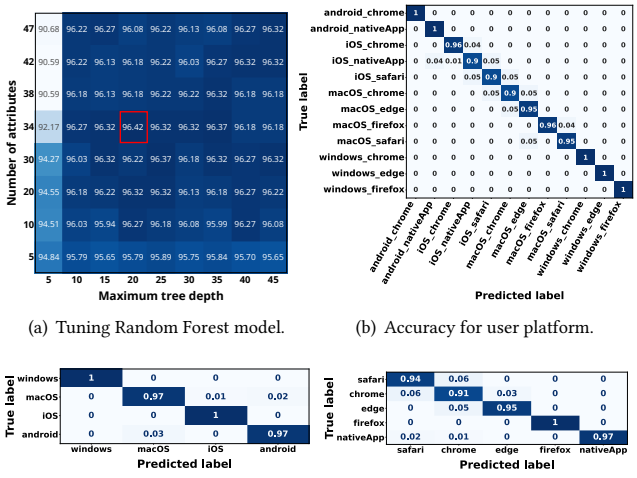
To elicit the relative importance of attributes in satisfying our prediction objectives, we empirically define two thresholds, 0.2 and 0.1, and rate the importance of attributes as high, medium or low if their information gain value is > 0.2, between 0.1 and 0.2 or < 0.1, respectively. We can see that 17 attributes including t_1, m_1, m_3, o_1, o_3, o_4, o_6, o_8, o_{12}, o_{18}, o_{21}, q_2, q_4, q_5, q_{12}, q_{17} and q_{20} have high importance for all three prediction objectives, *i.e.,* user platform, only device type, and only software agent. Eleven attributes, *i.e.,* m_2, m_4, o_2, o_5, o_7, o_{10}, o_{15}, o_{17}, o_{19}, o_{20} and q_{18}, have information gain < 0.1 for all three prediction objectives, implying their effectiveness is limited.

Other attributes have at least one high or medium score, and one medium or low score. For instance, t_2 has a normalized importance score of 1 (high) for device type but 0.18 (medium) for software agent. This is not surprising since certain handshake fields (*e.g.,* time-to-live) are highly dependent on device types while others (*e.g.,* version information) exhibit strong variations across software agents; see §3.3. In addition, as shown in Fig. 5(b), an attribute with low importance for QUIC video flows may have medium or high importance for TCP flows. An example is o_{15}, which has a value near 0 for QUIC but over 0.1 for TCP. These observations indicate that a select combination of attributes with medium to high importance can also serve as useful predictors.

Finally, we highlight that even though certain types of attributes require only zero or minimal preprocessing steps, such as the numerical fields in handshake messages, relying solely on them does not necessarily compromise prediction accuracy. For instance, out of 43 numerical and length-/presence-based (*i.e.,* low-cost) attributes, 3 (*i.e.,* t_1, t_2, o_8) have high importance for both QUIC and TCP video flows. On the other hand, 4 out of 9 categorical (*i.e.,* medium-cost) and 1 out of 10 list-type (*i.e.,* high-cost) attributes have low importance. In production environments, switches might not have the necessary computational resources to execute the entire packet processing and classification pipeline in real-time. Under these circumstances, one can (carefully) select only those attributes requiring minimal preprocessing to achieve a certain acceptable degree of prediction accuracy. This trade-off between the cost of preprocessing attributes and the accuracy of predicting user platforms is discussed next.

4.3 Classification Models

We now present our machine learning models to predict user platforms for the four video content providers. Note that at the time of



(a) Tuning Random Forest model.

(b) Accuracy for user platform.

(c) Accuracy for only device type.

(d) Accuracy for only software agent.

Figure 6: Hyperparameter tuning of random forest models for YouTube over QUIC (a). The model's classification accuracy to classify user platform is shown in (b), only device type is depicted in (c), and only software agent is presented in (d).

this writing, only YouTube supports QUIC while the others support only TCP.

4.3.1 Model training, tuning and selection. We consider three popular machine learning algorithms for our classification tasks, *i.e.,* random forest (decision tree-based), MLP (neural network), and KNN (clustering-based), and tune their hyperparameters accordingly. For random forest, these parameters include maximum tree depth, number of trees, and number of attributes. MLP models are tuned for the number of hidden layers, number of perceptrons per layer and activation functions. KNN classifiers are tuned for the number of neighbors, weight functions and leaf size.

We trained each of these algorithms on the dataset collected from our experimental setup shown in Fig. 1. To recall, it consists of attributes extracted from traces spanning about 10,000 video flows across 30 user platforms. The performance of each of the models was evaluated in terms of the overall accuracy using 10-fold cross-validation. Our evaluation shows that the random forest model outperforms both the MLP and KNN classifiers, not only for YouTube but for all other providers and user platforms as well. For example, in classifying the user platform for YouTube flows over QUIC, the random forest model achieves an overall accuracy of 96.4%, while the MLP and KNN models achieve an accuracy of 65.1% and 69.1%, respectively. This observation is consistent with prior research that shows decision tree-based models are better suited for network traffic classification problems [54]. We therefore selected the random forest model for real-time deployment and evaluation in our campus network.

Fig. 6(a) depicts how we select the best random forest model. Out of the 62 attributes overall, only 50 are applicable to QUIC. Further, the figure shows the overall classification accuracy when tuning the number of attributes (vertical axis) and the maximum tree depth (horizontal axis). The highest test accuracy of 96.4% was attained when the two hyperparameters were set to 34 and 20, respectively. We use this selection of hyperparameters as our best-performing random forest model.

Table 3: Model performance in open-set evaluation for three classification objectives, namely user platform, device type only and software agent only.

Provider	Objective	Accuracy
YouTube (TCP/QUIC)	User platform	98.7%/94.5%
	Device type	99.1%/98.4%
	Software agent	96.6%/95.4%
Netflix (TCP)	User platform	91.2%
	Device type	92.4%
	Software agent	90.6%
Disney (TCP)	User platform	90.9%
	Device type	91.6%
	Software agent	88.6%
Amazon (TCP)	User platform	88.2%
	Device type	89.4%
	Software agent	87.9%

For the above model, we show its classification accuracy per prediction class (*i.e.,* confusion matrix) in Fig. 6(b). It is clear that 5 out of 12 prediction classes are with 100% accuracy, including all browser types on Windows PC, as well as Chrome and native YouTube application on Android phone. Misclassified instances are observed within two groups, *i.e.,* iOS and macOS devices. For example, native YouTube app on iOS has a small chance (*i.e.,* $\leq 4\%$) of being misclassified as native app on Android. For the iOS native app instances, their device types (*i.e.,* iOS) are classified with 96% accuracy, while their software agents (*i.e.,* native app) might be misclassified as Chrome or Safari with a slim chance of less than 6%.

Delving deeper into the fields for (iOS, Safari) and (iOS, Chrome) platforms, see columns 10 and 11 in Fig. 12(a), we note that a vast majority of them take on similar values. Only a small number of attributes, *e.g., handshake_length*, *extensions_length* have different values. Moreover, Chrome has started to randomize TLS extension orders since version 110 [17]. These variations in attribute values explain why we see a small fraction of misclassifications in these user platforms.

We also observe that those misclassified instances are with low confidence, *i.e.,* less than 50%, while the correctly classified ones are with high confidence, *i.e.,* over 80%. The performance of our random forest models in classifying only the device type and the software agent for YouTube QUIC is depicted in Fig 6(c) and 6(d). We note that their accuracy in identifying the device type is high, $\geq 97\%$ for all device types, while it can predict all software agents with $\geq 91\%$ accuracy. The marginal decline in the accuracy of the latter is due to the reasons explained above. Nonetheless, the overall results demonstrate that our random forest classifiers offer high prediction accuracy with high confidence.

4.3.2 Open-set evaluation. Relying solely on the accuracy reported by 10-fold validation can result in over-fitting, which can mask the true accuracy of a classifier. To overcome this problem, we further evaluate the performance of our random forest model on a dataset

Table 4: Median classification confidence of correct and incorrect instances in the open-set evaluation for all four providers and three classification objectives.

Provider	Objective	Med. conf. (correct)	Med. conf. (incorrect)
YouTube (TCP/QUIC)	User platform	98.5%/91.4%	86.5%/54.4%
	Device type	89.6%/91.8%	46.7%/57.5%
	Software agent	98.2%/90.9%	89.3%/52.7%
Netflix (TCP)	User platform	88.7%	53.9%
	Device type	99.3%	60.0%
	Software agent	91.0%	59.1%
Disney (TCP)	User platform	91.5%	67.6%
	Device type	98.2%	83.5%
	Software agent	91.6%	67.6%
Amazon (TCP)	User platform	89.1%	60.6%
	Device type	99.4%	50.0%
	Software agent	91.3%	64.3%

collected by one of the authors from their home network. While the devices in the home are the same as those in our experimental setup, the OS versions as well as those of the software agents are different. These variations could impact the values of the different attributes. The aim of this exercise therefore is to validate the accuracy of the model, trained on the lab trace data, in predicting the user platforms seen in a different environment, *i.e.,* the home.

This dataset contains over 2000 video flows spread evenly across all user platforms. We note from Table 3 that the results are comparable to the ones reported earlier, confirming the high efficacy of the classifier. User platforms are classified with > 94% accuracy for YouTube, > 91% for Netflix, > 90% for Disney+ and > 88% for Amazon Prime Video. Additionally, as shown in Table 4, the majority of correctly classified instances are of very high confidence (*i.e.,* > 88%) whereas incorrect classifications tend to be of low confidence (*i.e.,* < 70%). Exceptions are also observed in minor cases for certain types of classes due to their similar networking and kernel suites to the misclassified classes. For example, video flows from Apple's mobile iOS devices sometimes behave very similarly to Apple's desktop macOS devices, thus, can be misclassified with high confidence.

4.3.3 Models with a subset of attributes. We have discussed in §4.2.2 that not all attributes are equally important from the perspective of information gain, especially list attributes which require high costs for preprocessing. An ISP carrying very high data rates, *e.g.,* multi-hundreds of Gbps, may require servers with significant computational capability to deploy the end-to-end classification pipeline in real-time. In the absence of such resources, one may choose to discard the use of high-cost low-importance attributes to reduce the processing load with negligible impact on classification accuracy, as described next.

We train random forest models with three subsets of attributes. Each subset excludes attributes that are deemed to be of low importance (*i.e.,* < 0.1 information gain). Additionally, the first subset excludes only high-cost low-importance attributes, the second subset excludes both high- and medium-cost attributes that are of low importance, and the third subset excludes all attributes that are of low importance.

Table 5: Accuracy of models for YouTube QUIC video flows with three subsets of attributes. Each subset excludes low-importance attributes associated with high cost; high and medium cost; high, medium and low cost.

Excluded low-imp. attributes	Objective	Accuracy
High cost	User platform	93.3%
	Device type	97.2%
	Software agent	94.6%
High + **medium** cost	User platform	93.0%
	Device type	97.2%
	Software agent	92.8%
High + **medium** + **low** cost	User platform	92.8%
	Device type	97.1%
	Software agent	92.9%

The overall accuracy of the random forest classifier for YouTube QUIC flows is reported in Table 5. Compared with the accuracy of the model that uses the full (*i.e.,* 50) attribute set, we observe a slight reduction (≈ 3%) across the three scenarios considered here. For example, the accuracy for classifying user platforms of QUIC YouTube flows is 96.4% with the full attribute set. This drops to 93.3%, 93.0% and 92.8%, with each subset, respectively. The performance is similar when predicting only the device type or software agent.

Thus, in scenarios with constrained computational resources, one can safely ignore attributes with low information gain requiring varying degrees of processing costs, to deploy the end-to-end user platform classification pipeline in real-time. In our campus-wide deployment, our server has sufficient computational resources to handle real-time traffic streams at 20 Gbps peak rate and process the observed maximum of over 1000 concurrent video flows from considered content providers. Thus, we use classifiers trained with the full attribute set to achieve the highest possible accuracy.

4.3.4 Benchmarking against state-of-the-art. Now, we benchmark our method with six state-of-the-art techniques spanning the last five years of literature in user platform identification using the ground-truth dataset discussed in §3.1. Three qualitative aspects, as specified in the second to fourth columns of Table 6, including inference objective, covered protocol and inference granularity are compared to demonstrate the superior visibility our method can provide. Our method outperforms all alternatives.

Two out of the six prior techniques [40, 55] require collecting statistics of all flows from a candidate host, and thus cannot be used to identify user platforms of individual video flows from clients behind NAT. The other four techniques [6, 14, 28, 53], either directly offer flow-level granularity, or have a subset of articulated attributes from individual flows, and thus can be adapted for flow-level identification as specified in the fifth column of Table 6. First of all, since all six prior techniques are designed only for TLS over TCP flows, a generic adaptation has to be made for all prior works to handle video flows over QUIC, including identifying and decrypting QUIC Initial packets and extracting handshake attributes from TLS CHLO messages over QUIC. In addition, to make the four techniques applicable for user platform identification, the required adaptations specific to each prior work are listed in the fifth column of Table 6 such as extracting fine-grained flow-level telemetry, constructing

Table 6: Benchmarking the classification accuracy of our user platform identification method against state-of-the-art after necessary methodological adaptations.

Work	Objective	Protocol	Granularity	Required specific adaptations	Accuracy after adaptation				
					YT (QUIC)	YT (TCP)	NF (TCP)	DN (TCP)	AP (TCP)
Ours	Dev. type + Soft. agent	TLS/QUIC	flow	–	94.5%	98.7%	91.2%	90.9%	88.2%
[6]	Dev. type + Soft. agent	TLS	flow	feature construct.; classi. process	90.1%	97.5%	84.0%	82.8%	80.3%
[14]	Dev. type	TLS	host	flow granularity; inference object.	94.0%	96.8%	86.0%	80.1%	84.1%
[28]	Dev. type	TLS	host	flow granularity; inference object.	68.1%	95.1%	82.7%	83.1%	79.0%
[55]	Dev. type + Soft. agent	non-TLS	host	not adaptable	–	–	–	–	–
[53]	Soft. agent	TLS	flow	inference objective	11.3%	51.0%	53.4%	56.5%	38.1%
[40]	Soft. agent	TLS	host	not adaptable	–	–	–	–	–

features from collected statistics, expanding inference objective, developing classification pipeline and models. For example, the method in [6] combines TLS handshake fields to generate string-based fingerprints of applications. Therefore, for benchmarking purposes, we adapt this method by constructing usable features from their fingerprint strings and developing a classification process. Also, the works in [14] and [28] classify device types with IP-level attributes. Thus, their attributes are adapted to be extracted for individual video flows and used to classify not only device types but also software agents.

The quantitative accuracy of our method in comparison to other prior techniques (after necessary adaptations) is reported in the last five columns of Table 6. We can clearly observe that our method outperforms other techniques in all five classification scenarios, *i.e.,* different providers and their supported protocols. Noting that the three techniques [6, 14, 28] that can achieve overall 80% accuracy require significant adaptations from constructing flow-level telemetry to articulating attributes and developing classification models, whereas the technique in [53] uses attributes extracted from flow metadata (*e.g.,* length) and only one TLS field "*TLS_message_type*" which becomes unavailable/encrypted in QUIC, and thus has only 11.3% accuracy for YouTube flows over QUIC and less than 60% for other scenarios.

5 Characterizing User Platforms in a Campus Network

To showcase the usability of our system for residential broadband network operators, in §5.1, we report on the deployment of our user platform classification pipeline on a commodity server and use it to analyze traffic between our university campus network (mimicking a residential broadband network) and the Internet in real time. Then in §5.2, we discuss insights obtained from our deployment over a four-month period, demonstrating how video streaming consumption patterns vary across user platforms and providers. Last, in §5.3, we conclude discussing insights collected as well as general considerations from our deployment that might be of interest to ISPs' network operation teams interested in video streaming traffic analytics.

5.1 Prototype and Campus Deployment

We implement our packet processing pipeline, shown in Fig. 4, as a virtual network function (VNF) system. DPDK [31] and NFF-Go [45] packet processing frameworks are used for the packet *Preprocessing* stage in Fig 4, which parses input packet streams and handles the handshake and data packets of video flows. The handshake attribute generator and video telemetry modules are developed using Golang while the classifier banks use Python scikit-learn library [49]. The video session telemetry, along with user platform labels, is stored in a PostgreSQL database for analysis.

The system runs on a commodity blade server configured with an 8-core Intel Xeon E5-2620 CPU and 64 GB DDR4 RAM. The server receives a copy of the traffic from our university network border router that is connected to the Internet. The traffic is delivered to two 10 Gbps network interfaces on our server for inbound and outbound traffic, respectively. We have obtained ethical clearance from our university ethics board as detailed in Appendix A.

As an additional sanity check of the classification methodology, we played over 1000 video sessions using all available user platforms in our university lab, which were captured by our system. The classification accuracy of those ground-truth video sessions is similar to our open-set evaluation. Almost all misclassified instances (which are less than 4% of streams) were with relatively low confidence, *i.e.,* < 50%.

5.2 Insights from the Deployment

Our university campus network serves staff, students, visitors, and over 10 residential dormitories. Over the four months from July 7th 2023, 00:00 to November 9th 2023, 23:59, our system collected telemetry statistics, *i.e.,* duration, volume and throughput, of every video flow from the four providers, and tagged each of them by their user platforms. Over 100 million streaming video flows were collected with a total of 400k hours of watch time. For reliability of our insights, we exclude about 20% of the sessions with low classification confidence that may be due to unknown types of user platforms not in our training dataset. We now analyze how different user platforms impact usage patterns from the point of view of watch time (§5.2.1), bandwidth demand (§5.2.2), and temporal usage patterns (§5.2.3).

5.2.1 Video watch time across user platforms. Taking our four-month deployment as an example, we start by examining how the level of engagement, measured in terms of total watch time, varies by device type across the four video content providers. This is shown in Fig. 7. Not surprisingly, across our entire campus demographic, YouTube, where content is mostly free, dominates engagement with an average daily total watch time of 2000 hours. It is followed by subscription-based providers such as Netflix, Disney+, and Amazon Prime Video. Furthermore, the majority of subscription-based videos are watched on PCs (Windows/Mac) rather than on mobile devices. In contrast, up to 40% of YouTube engagement occurs on mobile devices (iOS/Android). These insights enable ISPs

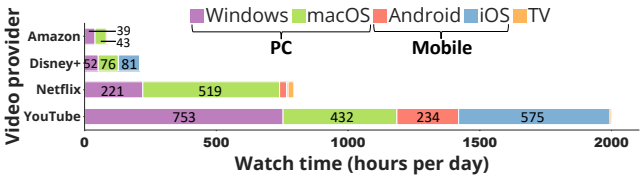
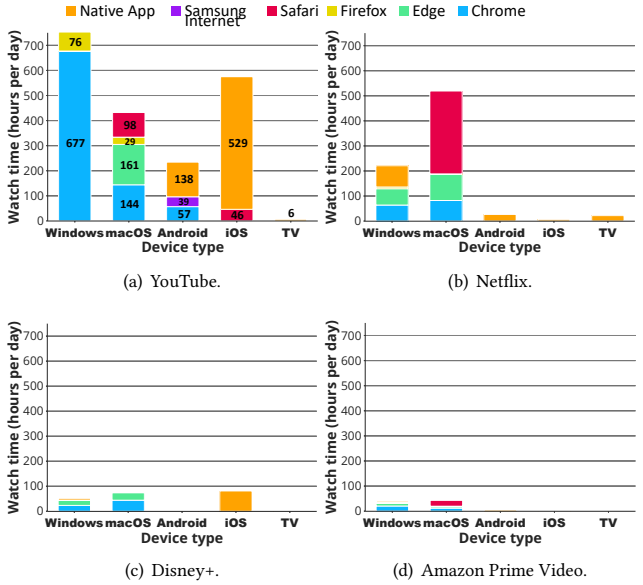


Figure 7: Video watch time for the four video content providers across device types.



(a) YouTube. (b) Netflix.

(c) Disney+. (d) Amazon Prime Video.

Figure 8: Video watch time for the four video content providers across software agents on each device type.

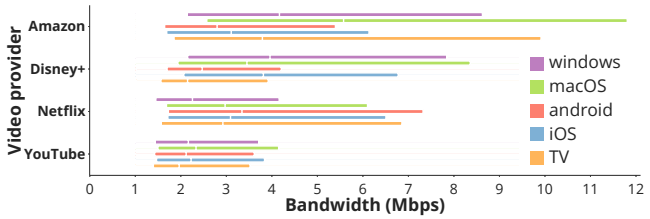
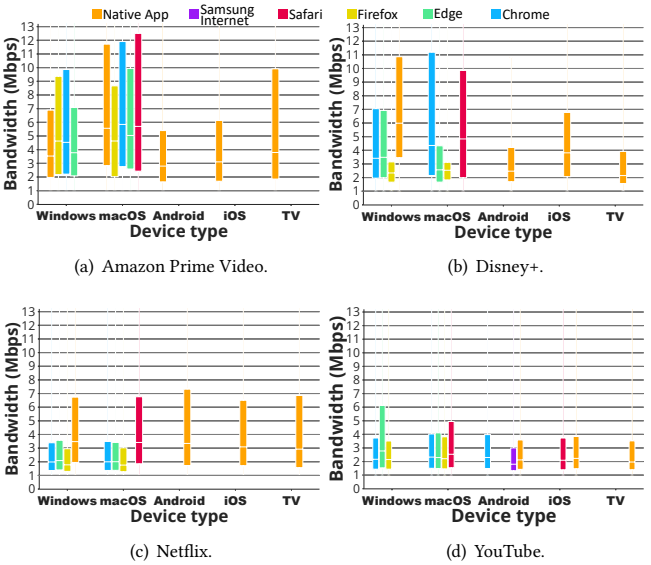


Figure 9: Bandwidth demand for the four video providers across device types.



(a) Amazon Prime Video. (b) Disney+.

(c) Netflix. (d) YouTube.

Figure 10: Bandwidth demand for the four video providers across software agents on each device type.

to prioritize the troubleshooting of issues, such as they can expect more support calls relating to PCs than mobiles for Netflix, and the converse for YouTube.

Fig. 8 shows the breakdown of software agents per video provider. Chrome browser on Windows PCs is the most popular software agent used to watch YouTube, clocking up 677 hours, as shown in Fig. 8(a). In addition, amongst mobiles, iOS is preferred with over 90% of watch time on its YouTube native app. The other devices use a relatively diverse set of software agents, as shown in the figure.

The watch time profiles for Netflix, Disney+ and Amazon are provided in Fig. 8(b), 8(c) and 8(d), respectively. While Safari on Mac PCs is popular for viewing Netflix and Amazon, the native Disney+ app on iOS dominates engagement of mobile users by over 90%.

5.2.2 Bandwidth demand. As video streaming are bandwidth demanding, in Fig. 9, we show the distribution of bandwidth consumption, as box plots indicating the median and the quartiles to either side of the median, for the four video streaming providers across different device types. It is apparent that the bandwidth demand imposed by subscription-based videos is higher than that of YouTube as the interquartile range for these providers is 3 to 9 Mbps higher. Notably, videos streamed from Amazon Prime Video to Mac PCs demand the highest median bandwidth (of 5.7 Mbps), which is 50% higher compared to smart TVs.

The impact of software agents on bandwidth for the different video providers is plotted in Figures 10(a)-(d). Android mobiles, iOS mobiles and TVs only support native Amazon Prime Video apps and as shown in Figures 10(a), consume less bandwidth (median < 3 Mbps) than their PC counterparts. All browsers on Windows/Mac PCs exhibit higher median bandwidth and interquartile range spread for Amazon compared to native mobile apps. In addition, Mac PCs generally require higher median throughput than Windows PCs. Interestingly, Netflix streamed to PCs on browsers (excluding Safari) consumes lower median bandwidth (< 2 Mbps), which might suggest lower resolution supported via browsers than the native app.

5.2.3 Temporal usage patterns. As traffic demand changes over time, in Fig. 11, we depict the median traffic volume during each hour of the day consumed by Amazon, Disney+, Netflix and YouTube videos on PCs and mobile devices. Overall, Amazon and Disney+

exhibit fairly similar daily usage patterns with a 4-hour peak period from about 7 pm to 11 pm. Also, mobile usage for Amazon is low compared to Disney+.

Comparing YouTube and Netflix, we see that the former has a long and sustained peak window from about 4 pm to midnight while the latter has a shorter peak, *i.e.,* between 8 pm to 10 pm. In terms of mobile usage, YouTube dominates with relatively steady hourly peak usage in the range of 17 to 20 GB from about 4 pm to midnight.

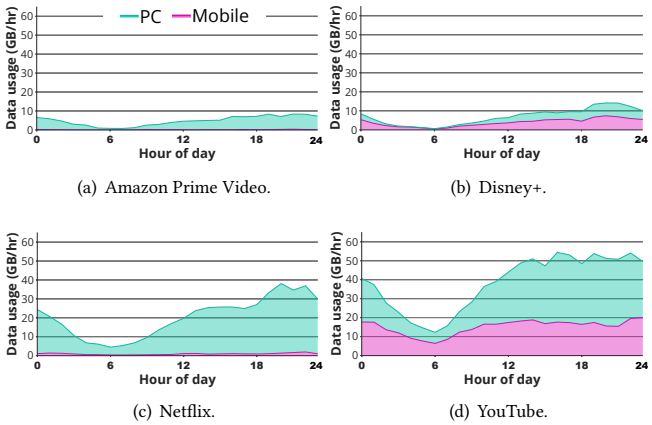
5.3 Discussion on Insights and Considerations

For network operation teams interested in video streaming, visibility into engagement patterns across user platforms can equip them with knowledge of their customer segments and preferences, as well as the ability to identify customer issues pertinent to popular content providers and/or certain firmware/software. Second, bandwidth capacity planning is a key ongoing activity for ISPs as they strive to meet the ever-increasing demands for service quality assurance from their customers. Given the significant load imposed by video streaming, fine-grained insights into the bandwidth demand of popular streaming service users, broken down by device types and software agents, may help ISPs improve the fidelity of their bandwidth forecasting models. In addition, temporal usage patterns of video streaming services can also provide valuable information to ISPs. For example, by knowing when peak usage occurs and for what kind of content, ISPs can proactively allocate adequate bandwidth to ensure high levels of service assurance. Conversely, during off-peak periods, bandwidth can be allocated differently to save costs. Alternatively, traffic management policies can be implemented that prioritize video streaming during peak hours and other types of traffic during off-peak hours.

We acknowledge that our study focused particularly on four popular content providers and thirty user platforms to demonstrate the efficacy of our method. While it is a necessary engineering extension to continuously collect comprehensive datasets and retrain models, the methodology itself is readily extensible to incorporate other streaming services, emerging user platforms, and changes in behaviors over time. Therefore, in practical deployment for a broadband network, two key **deployment considerations** arise.

First, for deployments in different broadband networks, the operators can have various video streaming services of interest. For example, a deployment in Asia-Pacific region may require the system to be able to measure streaming video sessions from regionally popular providers such as Bilibili and Hotstar. Therefore, the deployment team would have to collect ground-truth data of video streaming flows for each video provider to be included. Similarly, new popular user platforms for video streaming may emerge. For example, broadband ISPs are interested in understanding the streaming videos watched on Apple Vision Pro after it was released in February 2024, which requires the collection of ground-truth data to augment the classifiers.

Second, although our four-month deployment did not exhibit any observable drop in prediction confidence from our classifiers over time, it is acknowledged that the overall prediction accuracy and confidence will decline over a longer deployment period due to evolving traffic characteristics of video streaming services and



(a) Amazon Prime Video. (b) Disney+.

(c) Netflix. (d) YouTube.

Figure 11: Temporal data usage patterns for the four content providers across PCs and mobile devices.

(firmware/software) updates of user platforms, which is known as "concept drift". Therefore, in practice, the deployment team will have to periodically retrain the under-performing classifiers with updated ground-truth training data to adapt to these changes. While developing a continuous re-training process is not within the scope of this paper, we acknowledge that there are established techniques to detect and mitigate concept drifts [38, 39, 59].

6 Conclusion

Our work discussed in this paper provides network operators with fine-grained visibility into user platforms of streaming video flows over both TCP and QUIC. We first systematically understand the network communication anatomy of streaming video sessions and categorize handshake fields of video flows that can vary across OSes, browsers and provider-native applications. Following our observation on the variations of handshake fields across user platforms, we developed and evaluated a pipeline that processes network packet streams to classify user platforms of streaming video flows using well-trained machine learning models on formalized attributes from handshake fields of video flows, achieving over 96% accuracy. We then prototype our system on a commodity server and deploy it in a large university campus network that mimics a residential broadband network for a four-month period. The usage patterns from over 100 million video flows across various user platforms accessing four major content providers, namely YouTube, Netflix, Amazon Prime Video and Disney+ are discussed. Our method provides ISPs with valuable insights to better understand their customer segments, provision bandwidth, and troubleshoot video streaming issues pertinent to device firmware, OS, or software for customer experience and satisfaction.

Acknowledgments

We thank our shepherd Alessandro Finamore and the five anonymous reviewers for their comprehensive and insightful feedback. This work is supported by Australian Government's National Industry PhD Program award reference number 35063 and Cooperative Research Centres Projects (CRC-P) Grant CRCPXIV000099.

References

[1] Shahryar Afzal, Jiasi Chen, and K. K. Ramakrishnan. 2017. Characterization of 360-degree Videos. In *Proc. Workshop on Virtual Reality and Augmented Reality Network*. Los Angeles, CA, USA, 1–6.

[2] Hasan Faik Alan and Jasleen Kaur. 2016. Can Android Applications Be Identified Using Only TCP/IP Headers of Their Launch Time Traffic?. In *Proc. ACM WiSec*. Darmstadt, Germany, 61–66.

[3] John Althouse. 2019. TLS Fingerprinting with JA3 and JA3S. https://sforce.co/3vUe86r Accessed: 2023-09-29.

[4] John Althouse, Jeff Atkinson, and Josh Atkins. 2023. JA3 - A Method for Profiling SSL/TLS Clients. https://github.com/salesforce/ja3 Accessed: 2023-04-20.

[5] Blake Anderson and David McGrew. 2017. OS Fingerprinting: New Techniques and a Study of Information Gain and Obfuscation. In *Proc. IEEE Conference on Communications and Network Security*. Las Vegas, NV, USA, 1–9.

[6] Blake Anderson and David McGrew. 2019. TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior. In *Proc. ACM IMC*. Amsterdam, Netherlands, 379–392.

[7] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. 2013. Developing a Predictive Model of Quality of Experience for Internet Video. *SIGCOMM Comput. Commun. Rev.* 43, 4 (aug 2013), 339–350.

[8] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. 2006. Early application identification. In *Proc. ACM CoNEXT*. Lisboa, Portugal, Article 6, 12 pages.

[9] Arthur Callado, Carlos Kamienski, Geza Szabo, Balazs Peter Gero, Judith Kelner, Stenio Fernandes, and Djamel Sadok. 2009. A Survey on Internet Traffic Identification. *IEEE Communications Surveys & Tutorials* 11, 3 (2009), 37–52.

[10] Klenilmar Lopes Dias, Mateus Almeida Pongelupe, Walmir Matos Caminhas, and Luciano De Errico. 2019. An Innovative Approach for Real-time Network Traffic Classification. *Computer Networks* 158 (July 2019), 143–157.

[11] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-Ros, and Konstantina Papagiannaki. 2016. Measuring Video QoE from Encrypted Traffic. In *Proc. ACM IMC*. Santa Monica, CA, USA, 513–526.

[12] Yu-ning Dong, Jia-jie Zhao, and Jiong Jin. 2017. Novel Feature Selection and Classification of Internet Video Traffic based on a Hierarchical Scheme. *Computer Networks* 119 (June 2017), 102–111.

[13] Dristone. 2023. Poor Video Quality on My PC but Not Firestick. https://community.hulu.com/s/question/0D51L00006PQiwJSAT/poor-video-quality-on-my-pc-but-not-firestick Accessed: 2023-08-29.

[14] Xinlei Fan, Gaopeng Gou, Cuicui Kang, Junzheng Shi, and Gang Xiong. 2019. Identify OS from Encrypted Traffic with TCP/IP Stack Fingerprinting. In *Proc. IEEE International Performance Computing and Communications Conference*. London, United Kingdom, 1–7.

[15] Michael Finsterbusch, Chris Richter, Eduardo Rocha, Jean-Alexander Muller, and Klaus Hanssgen. 2014. A Survey of Payload-Based Traffic Classification Approaches. *IEEE Communications Surveys & Tutorials* 16, 2 (2014), 1135–1156.

[16] Sally Floyd, K. K. Ramakrishnan, and David L. Black. 2001. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168. doi:10.17487/RFC3168.

[17] Google Chrome. 2023. Chrome Platform Status. https://chromestatus.com/feature/5124606246518784 Accessed: 2023-09-29.

[18] Craig Gutterman, Katherine Guo, Sarthak Arora, Xiaoyang Wang, Les Wu, Ethan Katz-Bassett, and Gil Zussman. 2019. Requet: Real-time QoE Detection for Encrypted YouTube Traffic. In *Proc. ACM MMSys*. Amherst, MA, USA, 48–59.

[19] Hassan Habibi Gharakheili, Minzhao Lyu, Yu Wang, Himal Kumar, and Vijay Sivaraman. 2019. iTeleScope: Softwarized Network Middle-Box for Real-Time Video Telemetry and Classification. *IEEE Transactions on Network and Service Management* 16, 3 (Sept. 2019), 1071–1085.

[20] Desta Haileselassie Hagos, Anis Yazidi, Øivind Kure, and Paal E. Engelstad. 2021. A Machine-Learning-Based Tool for Passive OS Fingerprinting With TCP Variant as a Novel Feature. *IEEE Internet of Things Journal* 8, 5 (2021), 3534–3553.

[21] Alexander Havang. 2022. QUIC is QUICkly taking over! https://www.sandvine.com/blog/quic-is-quickly-taking-over Accessed: 2023-08-29.

[22] Martin Husak, Milan Cermak, Toma Jirsik, and Pavel Celeda. 2015. Network-Based HTTPS Client Identification Using SSL/TLS Fingerprinting. In *Proc. IEEE International Conference on Availability, Reliability and Security*. Toulouse, France, 389–396.

[23] Internet Assigned Numbers Authority. 2023. Transport Layer Security (TLS) Extensions. https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml Accessed: 2023-04-25.

[24] Jana Iyengar and Martin Thomson. 2021. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. doi:10.17487/RFC9000.

[25] Xi Jiang, Shinan Liu, Saloua Naama, Francesco Bronzino, Paul Schmitt, and Nick Feamster. 2023. AC-DC: Adaptive Ensemble Classification for Network Traffic Identification. arXiv:2302.11718 [cs.NI] https://arxiv.org/abs/2302.11718

[26] Arsalan Ali Jumani, Fizza Zafar, Zafar Ayyub Qazi, and Ihsan Ayyub Qazi. 2019. Device-Aware Adaptive Video Streaming. In *Proc. ACM SIGCOMM Conference Posters and Demos*. Beijing, China, 98–100.

[27] L. F. Kozachenko and N. N. Leonenko. 1987. Sample Estimate of the Entropy of a Random Vector. *Problemy Peredachi Informatsii* 23 (1987), 9–16. Issue 2.

[28] Martin Lastovicka, Stanislav Spacek, Petr Velan, and Pavel Celeda. 2020. Using TLS Fingerprints for OS Identification in Encrypted Traffic. In *Proc. IEEE/IFIP NOMS*. Budapest, Hungary, 1–6.

[29] Martin Laštovička, Martin Husák, Petr Velan, Tomáš Jirsík, and Pavel Čeleda. 2023. Passive Operating System Fingerprinting Revisited: Evaluation and Current Challenges. *Computer Networks* 229 (June 2023), 109782.

[30] Jing Li, Lukáš Krasula, Patrick Le Callet, Zhi Li, and Yoann Baveye. 2018. Quantifying the Influence of Devices on Quality of Experience for Video Streaming. In *Proc. IEEE Picture Coding Symposium*. San Francisco, CA, USA, 308–312.

[31] Linux Foundation. 2015. Data Plane Development Kit (DPDK). https://www.dpdk.org Accessed: 2023-08-11.

[32] Richard Lippmann, David Fried, Keith Piwowarski, and William Streilein. 2003. Passive Operating System Identification From TCP/IP Packet Headers. In *Proc. ICDM Workshop on Data Mining for Computer Security*. Melbourne, FL, USA, 40–49.

[33] Minzhao Lyu, Sharat Chandra Madanapalli, Arun Vishwanath, and Vijay Sivaraman. 2024. Network Anatomy and Real-Time Measurement of Nvidia GeForce NOW Cloud Gaming. In *Proc. Passive and Active Measurement*. Virtual Event.

[34] Minzhao Lyu, Rahul Dev Tripathi, and Vijay Sivaraman. 2023. MetaVRadar: Measuring Metaverse Virtual Reality Network Activity. *Proc. ACM Meas. Anal. Comput. Syst.* 7, 3 (Dec. 2023).

[35] Riya Madaan. 2023. YouTube 'Something went wrong' error in iOS being looked into. https://piunikaweb.com/2023/02/28/youtube-something-went-wrong-error-in-ios-being-looked-into/ Accessed: 2023-08-29.

[36] Sharat Chandra Madanapalli, Hassan Habibi Gharakheili, and Vijay Sivaraman. 2022. Know Thy Lag: In-Network Game Detection and Latency Measurement. In *Proc. Passive and Active Measurement*, Vol. 13210. Virtual Event, 395–410.

[37] Sharat Chandra Madanapalli, Alex Mathai, Hassan Habibi Gharakheili, and Vijay Sivaraman. 2021. ReCLive: Real-Time Classification and QoE Inference of Live Video Streaming Services. In *Proc. IEEE/ACM IWQoS*. Tokyo, Japan, 1–7.

[38] Navid Malekghaini, Elham Akbari, Mohammad A. Salahuddin, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. 2022. Data Drift in DL: Lessons Learned from Encrypted Traffic Classification. In *Proc. IFIP Networking*. Catania, Italy, 1–9.

[39] Navid Malekghaini, Elham Akbari, Mohammad A. Salahuddin, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. 2023. Deep Learning for Encrypted Traffic Classification in the Face of Data Drift: An Empirical Study. *Computer Networks* 225 (2023), 109648.

[40] Fatemeh Marzani, Fatemeh Ghassemi, Zeynab Sabahi-Kaviani, Thijs Van Ede, and Maarten Van Steen. 2023. Mobile App Fingerprinting through Automata Learning and Machine Learning. In *Proc. IFIP Networking*. Barcelona, Spain, 1–9.

[41] M. Hammad Mazhar and Zubair Shafiq. 2018. Real-time Video Quality of Experience Monitoring for HTTPS and QUIC. In *Proc. IEEE INFOCOM*. Honolulu, HI, USA, 1331–1339.

[42] Oliver Michel, Satadal Sengupta, Hyojoon Kim, Ravi Netravali, and Jennifer Rexford. 2022. Enabling Passive Measurement of Zoom Performance in Production Networks. In *Proc. ACM IMC*. Nice, France, 244–260.

[43] Abhijit Mondal, Satadal Sengupta, Bachu Rikith Reddy, M. J.V. Koundinya, Chander Govindarajan, Pradipta De, Niloy Ganguly, and Sandip Chakraborty. 2017. Candid with YouTube: Adaptive Streaming Behavior and Implications on Data Consumption. In *Proc. ACM NOSSDAV*. Taipei, Taiwan, 19–24.

[44] Jonathan Muehlstein, Yehonatan Zion, Maor Bahumi, Itay Kirshenboim, Ran Dubin, Amit Dvir, and Ofir Pele. 2017. Analyzing HTTPS Encrypted Traffic to Identify User's Operating System, Browser and Application. In *Proc. IEEE Annual Consumer Communications & Networking Conference*. Las Vegas, NV, USA, 1–6.

[45] NFF-Go. 2022. aregm/nff-go: NFF-Go -Network Function Framework for GO (former YANFF). https://github.com/aregm/nff-go Accessed: 2023-09-29.

[46] Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar. 2019. Towards the Deployment of Machine Learning Solutions in Network Traffic Classification: A Systematic Survey. *IEEE Communications Surveys & Tutorials* 21, 2 (2019), 1988–2014.

[47] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. 2016. Website Fingerprinting at Internet Scale. In *Proc. NDSS*. San Diego, CA, USA.

[48] Eva Papadogiannaki and Sotiris Ioannidis. 2021. A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures. *ACM Comput. Surv.* 54, 6, Article 123 (jul 2021), 35 pages.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12, 85 (2011), 2825–2830.

[50] Sarah Perez. 2021. Roku customers report streaming issues after 10.5 update. https://techcrunch.com/2021/11/22/roku-customers-report-streaming-issues-after-10-5-update/ Accessed: 2023-08-29.

[51] J. R. Quinlan. 1986. Induction of Decision Trees. *Machine Learning* 1, 1 (March 1986), 81–106.

[52] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. 2017. Studying TLS Usage in Android Apps. In *Proc. ACM CoNEXT*. Incheon, Republic of Korea, 350–362.

[53] Qiuning Ren, Chao Yang, and Jianfeng Ma. 2021. App Identification Based on Encrypted Multi-smartphone Sources Traffic Fingerprints. *Computer Networks* 201 (Dec. 2021), 108590.
[54] Paulo Angelo Alves Resende and André Costa Drummond. 2018. A Survey of Random Forest Based Methods for Intrusion Detection Systems. *ACM Comput. Surv.* 51, 3, Article 48 (may 2018), 36 pages.
[55] Omar Richardson and Johan Garcia. 2020. A Novel Flow-level Session Descriptor With Application to OS and Browser Identification. In *Proc. IEEE/IFIP NOMS*. Budapest, Hungary, 1–9.
[56] scikit-learn developers. [n. d.]. mutual_info_classif. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html Accessed: 2024-08-14.
[57] Zain Shamsi, Daren B. H. Cline, and Dmitri Loguinov. 2021. Faulds: A Non-Parametric Iterative Classifier for Internet-Wide OS Fingerprinting. *IEEE/ACM Transactions on Networking* 29, 5 (2021), 2339–2352.
[58] Danil Shamsimukhametov, Anton Kurapov, Mikhail Liubogoshchev, and Evgeny Khorov. 2022. Is Encrypted ClientHello a Challenge for Traffic Classification? *IEEE Access* 10 (2022), 77883–77897.
[59] O.I. Sheluhin and S.A Sekretarev. 2021. Concept Drift Detection in Streaming Classification of Mobile Application Traffic. *Automatic Control and Computer Sciences* 55 (2021), 253–262.
[60] sk. 2021. Identify Operating System Using TTL Value And Ping. https://ostechnix.com/identify-operating-system-ttl-ping/ Accessed: 2023-07-31.
[61] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, and Georg Carle. 2023. DissecTLS: A Scalable Active Scanner for TLS Server Configurations, Capabilities, and TLS Fingerprinting. In *Proc. Passive and Active Measurement*. Virtual Event, 110–126.
[62] Bruce Spang, Shravya Kunamalla, Renata Teixeira, Te-Yuan Huang, Grenville Armitage, Ramesh Johari, and Nick McKeown. 2023. Sammy: Smoothing Video Traffic to Be a Friendly Internet Neighbor. In *Proc. ACM SIGCOMM*. New York, NY, USA, 754–768.
[63] Qixiang Sun, D.R. Simon, Yi-Min Wang, W. Russell, V.N. Padmanabhan, and Lili Qiu. 2002. Statistical identification of encrypted Web browsing traffic. In *Proc. IEEE S&P*. Berkeley, CA, USA, 19–30.
[64] Martin Thomson. 2022. *Greasing the QUIC Bit*. RFC 9287. doi:10.17487/RFC9287.
[65] Martin Thomson and Sean Turner. 2021. *Using TLS to Secure QUIC*. RFC 9001. doi:10.17487/RFC9001.
[66] Daniel Villarreal. 2020. Hulu Accused of Throttling Bandwidth in New Lawsuit as Viewing Numbers Rise. https://bit.ly/hulu-throttling-bandwidth Accessed: 2023-09-01.
[67] Sarah Wassermann, Michael Seufert, Pedro Casas, Li Gang, and Kuang Li. 2020. ViCrypt to the Rescue: Real-Time, Machine-Learning-Driven Video-QoE Monitoring for Encrypted Streaming Traffic. *IEEE Transactions on Network and Service Management* 17, 4 (Dec. 2020), 2007–2023.
[68] Qizhi Zhang Weiping Zheng, Jianhao Zhong and Gansen Zhao. 2022. MTT: An Efficient Model for Encrypted Network Traffic Classification Using Multi-task Transformer. *Applied Intelligence* 52 (2022), 10741–10756.
[69] Damien Wilde. 2020. A fix for choppy Android 11 Beta video playback on Pixel devices is coming soon. https://9to5google.com/2020/06/18/android-11-beta-video-playback-issues/ Accessed: 2023-08-29.
[70] WYF99. 2024. chlo_extract. https://github.com/WYF99/chlo_extract
[71] Shichang Xu, Subhabrata Sen, and Z. Morley Mao. 2020. CSI: Inferring Mobile ABR Video Adaptation Behavior under HTTPS and QUIC. In *Proc. ACM EuroSys*. Heraklion, Greece, 1–16.
[72] Jiwon Yang, Jargalsaikhan Narantuya, and Hyuk Lim. 2019. Bayesian Neural Network Based Encrypted Traffic Classification using Initial Handshake Packets. In *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks*. Portland, OR, USA, 19–20.
[73] Michal Zalewski. 2014. p0f v3. https://lcamtuf.coredump.cx/p0f3/ Accessed: 2023-04-20.

A Ethics

We have obtained ethical clearance from our university ethics board (UNSW Human Research Ethics Advisory Panel approval number HC211007) that allows us to analyze campus traffic for streaming video flows without being privy to user identities such as ID numbers and names. In our campus deployment, insights into video streaming user platforms were reported in an aggregated manner, preserving anonymity rather than identifying specific users. In our analysis, no attempt was made to associate video streaming sessions with personal identities.

B Handshake Field Values of Video Flows

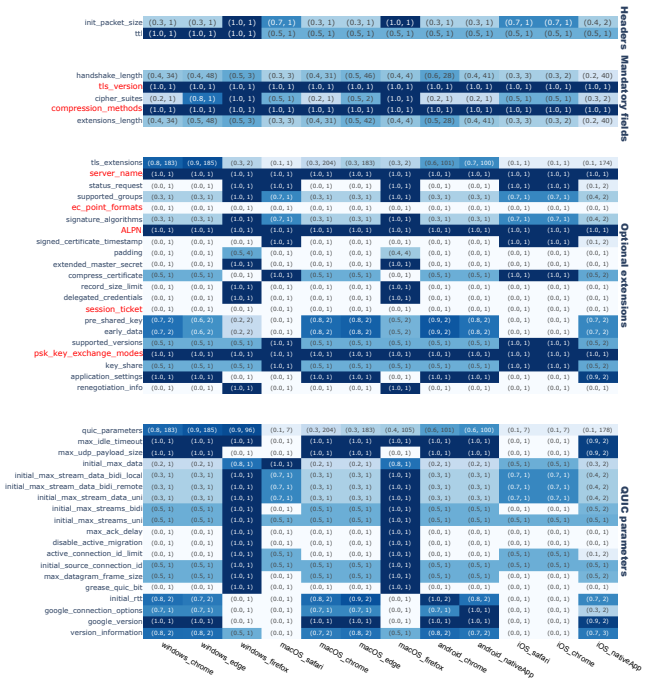
In this section, we show the distribution of handshake field values across all user platforms for all four considered streaming services (*i.e.,* YouTube, Netflix, Disney and Amazon).

In Fig. 12(a), we show a heatmap where each cell, corresponding to a user platform (indicated at the bottom of the figure), is a two tuple (x, y); x is the median value of the field, shown via labels on the left-hand side of the figure, and y is the number of distinct values that field takes as seen in our dataset. The heatmap is based on YouTube over QUIC flows across 12 user platforms. We consider this example for illustrative purposes.

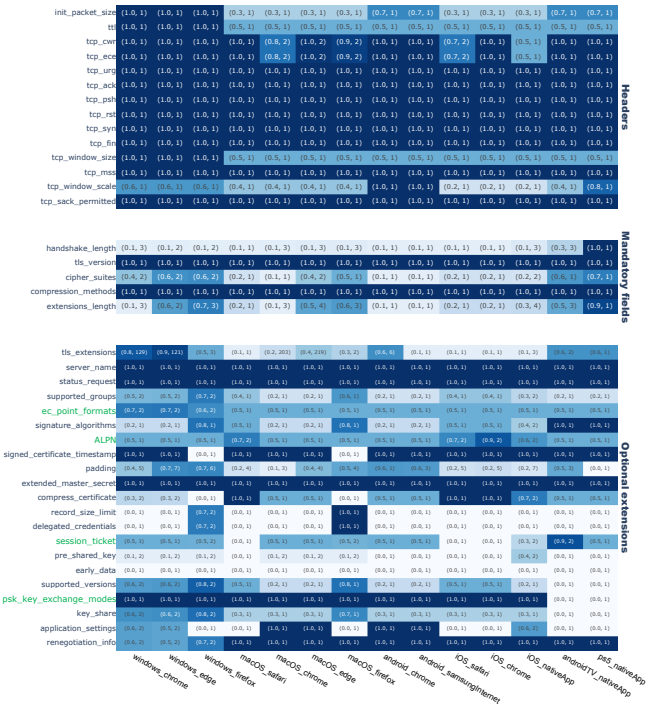
The median field values are normalized between 0 and 1. The x-axis shows 12 combinations of user platforms and the y-axis depicts the important fields obtained from different categories, as explained above.

As shown in Fig. 3, there are 7 fields whose median values are all the same (consistently either 0 or 1) across user platforms. These are highlighted as red labels in the figure and include *tls_version*, *compression_methods*, *server_name*, *ec_point_formats*, *ALPN*, *session_ticket* and *psk_key_exchange_modes*. This means they are not useful in differentiating user platforms for YouTube video flows over QUIC. However, as shown in green in Fig. 12(b), 4 of these fields, *i.e., ec_point_formats*, *ALPN*, *session_ticket* and *psk_key_exchange_modes* take different values across user platforms for YouTube video flows over TCP, meaning they can serve as useful indicators in identifying specific user platforms.

For the other three platforms (*i.e.,* Netflix, Disney and Amazon) which delivers video flows over TCP only, we show the overall number of unique field values and number of user platforms with different value distributions for each handshake field in Fig. 13, using a similar layout as Fig. 3. For most handshake fields, the value distributions either vary significantly across user platforms (e.g., *cipher_suites*) or remain consistently stable across user platforms (e.g., *compression_methods*), regardless of the video provider. There are also exceptions such as *tcp_syn* where six unique value distributions are observed across Disney+ user platforms, whereas the value distribution of this field remains consistent across user platforms for Netflix and Amazon. Such instances suggest that the indicative power of a certain handshake fields may vary for different video providers.

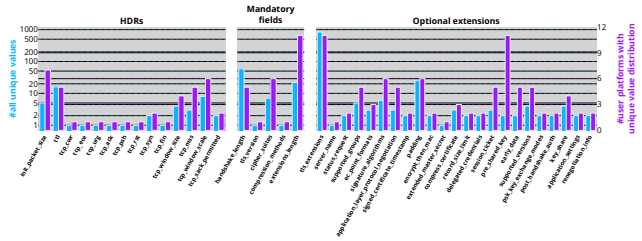


(a) Handshake field values of **YouTube** flows over QUIC across 12 user platforms.

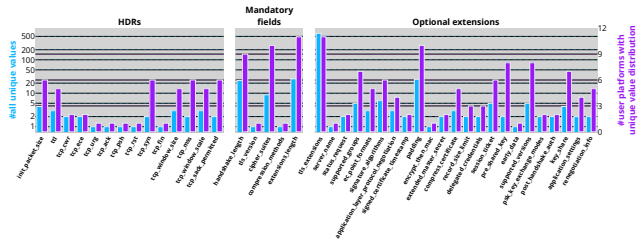


(b) Handshake field values of **YouTube** flows over TCP across 14 user platforms.

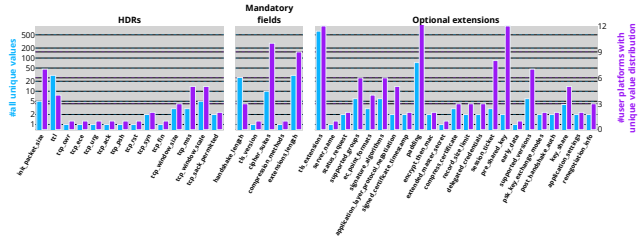
Figure 12: Median (normalized) and number of unique values shown as (x, y) taken by fields in handshake messages for YouTube flows.



(a) Handshake field value distribution of **Netflix** flows.



(b) Handshake field value distribution of **Disney+** flows.



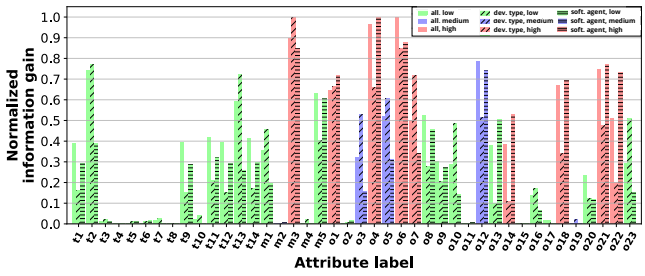
(c) Handshake field value distribution of **Amazon Prime Video** flows.

Figure 13: Number of unique values (left blue) and number of user platforms with different value distributions (right purple) for each handshake field in Netflix, Disney+ and Amazon Prime Video flows over TCP.

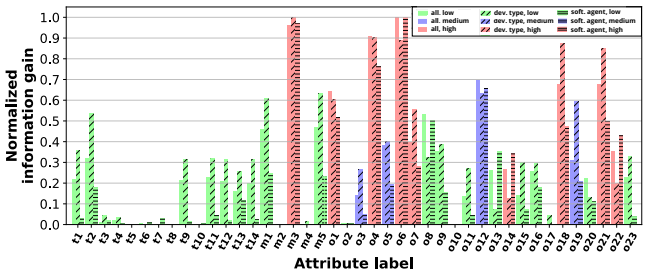
C Attribute Importance

In Fig. 14, we show the importance of attributes (defined in Table 2) in predicting user platforms for Netflix, Disney and Amazon Prime video flows over TCP. The attributes are color- and pattern-coded by their computational costs and prediction objectives.

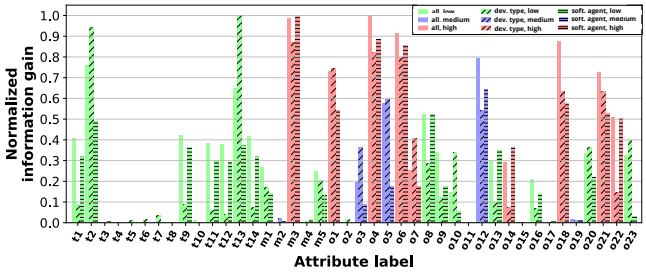
We highlight that the importance (*i.e.*, normalized information gain) of a certain attribute can differ across video providers. For example, o_{19} has very low importance (around 0) for all classification objectives related to Netflix and Amazon. However, it is highly useful for classifying user platforms for Disney, especially device types, where it has an importance score of nearly 0.6. Additionally, even for attributes that are indeed of high importance across all video providers, the specific classification objectives they are useful for can differ. For instance, t_9 is of high importance for classifying device types for Disney, whereas its importance for Netflix and Amazon is medium and low, respectively. Instead, it proves to be highly useful in classifying the software agent for those other two video providers.



(a) Attribute importance for classifying **Netflix** user platforms.



(b) Attribute importance for classifying **Disney+** user platforms.



(c) Attribute importance for classifying **Amazon Prime Video** user platforms.

Figure 14: Importance of different attributes in classifying user platforms of (a) Netflix, (b) Disney and (c) Amazon TCP video flows.