Endangered Privacy: Large-Scale Monitoring of Video Streaming Services

USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Endangered Privacy: Large-Scale Monitoring of Video Streaming Services

Martin Björklund and Romaric Duvignau,
Chalmers University of Technology and University of Gothenburg

https://www.usenix.org/conference/usenixsecurity25/presentation/bjorklund

This paper is included in the Proceedings of the
34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the
34th USENIX Security Symposium is sponsored by USENIX.

Endangered Privacy: Large-Scale Monitoring of Video Streaming Services

Martin Björklund, Romaric Duvignau
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
martibjo@chalmers.se, duvignau@chalmers.se

Abstract

Despite the widespread adoption of HTTPS for enhanced web privacy, encrypted network traffic may still leave traces that can lead to privacy breaches. One such case concerns MPEG-DASH, one of the most popular protocols for video streaming, where video identification attacks have exploited the protocol's side-channel vulnerabilities. As shown by several works in recent years, the distinctive traffic patterns generated by DASH's adaptive bitrate streaming reveal streamed content despite TLS-protection. However, these earlier studies have not demonstrated that the vulnerability remains exploitable in large-scale attack scenarios, even when making strong assumptions about network details. To that end, this work presents a protocol-agnostic system capable of identifying videos independent of network layer information, and demonstrates a practical attack over the largest dataset to date, comprising over 240,000 videos covering three entire streaming services. Using a combination of k-d tree search and time series methods, our system achieves an accuracy of over 99.5% in real-time video identification and remains effective even in scenarios involving victims behind VPNs or where Wi-Fi eavesdropping occurs. Since large-scale video identification can compromise user privacy and enable potential mass surveillance of video services, we complement our work with an analysis of the vulnerability root cause when using adaptive bitrate streaming and propose a mitigation strategy to stand against such vulnerabilities. Recognizing the lack of open-source tooling in this domain, we publish an extensive dataset of video fingerprints, network capture data, and tools to foster awareness and prompt timely solutions within the video streaming community to address these privacy concerns effectively.

1 Introduction

In the realm of user privacy preservation, HTTPS has become the standard, ensuring end-to-end encryption and enhancing privacy on the web. However, even encrypted network traffic leaves traces that can be exploited to compromise user privacy. In the last eight years, numerous video identification attacks (among others [5, 7, 9, 10, 22–24, 31]) have exploited a side-channel in the standardized MPEG-DASH protocol [3] (referred to hereafter simply as DASH), revealing streamed content despite all traffic being encrypted. DASH, an adaptive bitrate streaming protocol, segments videos into variously encoded bitrate chunks. This segmentation, while optimizing streaming, inherently generates a distinctive, *bursty* traffic pattern that can be analyzed to deduce the content being watched. Surprisingly, despite substantial privacy implications and the potential for mass surveillance of video services, the issue has not received enough attention. We believe the two key reasons behind this lack of attention are (1) *non-scalable* identification systems incapable of handling large datasets, and (2) *non-robust* identification systems that are sensitive to variations in transport layer details or noisy traffic. However, as this work highlights, these reasons do not justify overlooking the problem. Addressing a concerning reality, our work demonstrates that entire streaming services can be systematically monitored at a low cost to the attacker, underscoring the immediate need for mitigative actions.

Current state of the art. Earlier video identification studies have assumed different attack models. In most studies, the attack is deployed at the network layer, where the attacker is assumed to have access to transport layer information and the encrypted payload exchanged between the streamer and the video service [7, 9, 13, 24, 30]. This scenario mimics that of an ISP or a network administrator acting as the attacker. Other works have employed more covert models, operating over LTE networks [5] or 802.11 traffic [22]. Beyond the attacker model, known attacks fall into two broad categories: machine learning (ML)-based methods and fingerprinting methods. Refer to Table 1 for a summary of known attacks compared to the identification system introduced here.

ML-based studies [5, 9, 15, 24] have adhered to the traditional ML pipeline, i.e., streaming videos multiple times and extracting features from the resulting capture files to train a

Table 1: Comparison of our work against previous demonstrated privacy attacks over encrypted video streams.

Year [Ref.]	Targeted service(s)[1]	Monitored videos	Protocol-agnostic	Data collection[2]	Method[3]
2016 [22]	N	<1k	✓	◐	FP
2017 [24]	A,N,Y,V	<1k	✓	○	ML
2017 [9]	Y	<1k	✓	○	ML
2017 [23]	N	42k	✗	◐	FP
2019 [13]	Ah	<1k	✓	◐	FP
2020 [30]	F	<1k	✗	◐	FP
2020 [31]	Y	<1k	✓	◐	FP
2022 [5]	A,N,Y	<1k	✓	○	ML
2022 [10]	N,S	<1k	✓	○	ML
2022 [4, 15]	Y	<1k	✓	○	ML
2022 [29]	Ah	<1k	✗	◐	FP
2023 [7]	S	20k	✗	◐	FP
2023 [27]	Ah	<1k	✗	◐	FP
2023 [8]	Y	<1k	✗	◐	FP
2024 [32]	Y	<1k	✗	◐	FP
This work	**A,M,S**	**242k**	✓	●	**FP**

[1]Targeted service(s): Amazon Prime Video (A), Facebook (F), Max (M), Netflix (N), SVT Play (S), Youtube (Y), Vimeo (V); Ad Hoc service (Ah).
[2] ○ Slow, require watching videos (multiple times); ◐ Moderately efficient, partially automatized; ● Efficient & fully automatized data collection.
[3]Based on Machine Learning (ML) or Fingerprinting (FP).

classifier, whether using deep learning [15, 24] or a combination of other models [9, 10]. This approach's main advantage is to allow the system to learn the characteristics of targeted videos simply by repeatedly watching them, without requiring particular expertise in the targeted streaming protocol or video service. ML-based methods have shown success on modest datasets but face significant bottlenecks, particularly in data collection time (which may involve watching the same video hundreds of times under varying network conditions [9, 31]) and the cost of training models. Additionally, ML-based attacks require that the videos be streamed under the same timeframe and quality conditions as seen during training. As a result, all demonstrated attacks in this category, often implicitly, assume that the video is captured from its very start. These limitations pose major challenges in scaling ML-based methods to large-scale attack scenarios.

In contrast, *fingerprinting methods* [13, 22, 23] exploit the fact that DASH exposes information about segment sizes to any client initiating playback. This information serves as ground truth reference data, which can then be used to compare with live-captured traffic. Consequently, data collection is more efficient, as videos do not need to be streamed beforehand. However, strategies must be developed both for aggregating encrypted packet sizes into larger chunks analogous to segment sizes and for searching an existing fingerprint database. Recognizing that the term *fingerprint* is overloaded in this domain and can refer to anything from observed traffic patterns to statistical signatures, we clarify that in this work, a fingerprint specifically refers to the exact sequence of segment sizes corresponding to a particular video representation (i.e., quality). A more detailed explanation follows later, but at a high level, this definition is essential for distinguishing fingerprinting from ML-based methods. Building on this definition, prior fingerprinting studies have employed various techniques to use observed burst sizes as input for searching a fingerprint database, enabling video identification. These techniques include k-d trees [7, 22, 23], dynamic time warping [13], and Markov chains [31]. Among these, the k-d tree has proven to be the most scalable, supporting datasets with tens of thousands of identifiable videos. However, using a searchable data structure requires a distance measure and is weakened by noisy inputs that can disproportionately affect the distance calculation. Fingerprinting-based methods typically assume that the attacker captures traffic at the network layer, isolating communication between the video service and the streamer. To refine captured data, overhead from protocol headers and TLS-encryption may be estimated and removed from the trace [8, 23], aiming to closely approximate true segment sizes. However, the adoption of QUIC in HTTP/3 complicates this process, prompting newer works to target QUIC traffic specifically [27, 28]. Only a few works [5, 15, 22] attempt to bypass network-level access assumptions, such as dealing with VPN-protected victims [15], but these methods either fail to scale effectively or do not achieve high accuracy.

Our contributions. We present in this work a robust and versatile system for video identification over encrypted packet traces, with demonstrated attack efficacy in real video stream captures over the largest published dataset of video fingerprints. Specifically, our system employs an efficient and scalable data collection method, retrieving 242,000 videos and 3 million fingerprints from three well-established video streaming services. Furthermore, we combine the logarithmic search of the k-d tree with a time series method to create a system capable of identifying videos inside our collected dataset. To differentiate our approach from works that rely on transport layer information, we emphasize that our system is *protocol-agnostic*; it relies on the characteristics of adaptive bitrate streaming, independent of the network protocol used to deliver the stream. To demonstrate this property, we show the effectiveness of the attack in an authentic network environment as a network attacker (e.g. ISP or network administrator), and also more covertly as a passive 802.11 eavesdropper (without access to the network but close enough to sniff the encrypted 802.11 frames) and despite the victim using a VPN. Our system requires no synchronization between the target and attacker, meaning videos can be identified at any point in playback, provided that a sniffing time of approximately 60 to 90 seconds is available. Unlike previous works, which focus on static, one-time classification, our system is designed as a continuous identification tool, adapting dynamically as the target switches videos. Additionally, our approach is practical to deploy and mitigates base rate concerns by maintaining a

comprehensive dataset of all available videos for a service. Our data collection tool may also be used to keep the dataset up to date as streaming service libraries evolve.

The code and tools developed for this project are publicly available[1]. By publishing our large-scale and efficient identification system, we aim to raise awareness among all actors within video streaming services that privacy solutions for adaptive bitrate streaming must be urgently considered and implemented. Failing to address these vulnerabilities could leave the video streaming ecosystem exposed to large-scale surveillance and potential censorship. We also explore mitigation strategies to alleviate this attack and strongly urge streaming service developers to take them into account as early as possible.

Paper's organization. In Section 2, we introduce the necessary background for our work, presenting adaptive bitrate streaming (including the DASH and HLS protocols), the targeted streaming services, and the considered attack models. Section 3 presents our identification method in detail from data collection and building the fingerprint database to capturing traffic and identifying video streams. A thorough evaluation of our systems is conducted in Section 4. Finally, Section 5 explores mitigation strategies, and the last section presents our conclusions.

2 Background

2.1 Adaptive Bitrate Streaming

In the past, video streaming was facilitated by protocols such as RTP and RTSP. These early protocols struggled with issues such as firewall and NAT traversal, lack of efficient caching mechanisms, and inability to adjust to varying network conditions due to their fixed bitrate design. Recognizing these challenges, the industry has shifted towards *Adaptive Bitrate* (ABR) streaming protocols designed for performance in large distributed HTTP networks. ABR protocols work by segmenting the source media into smaller parts or chunks designated as *segments*, typically between two and fifteen seconds long, and encodes each segment at multiple bitrates. A client device will then request segments based on its available bandwidth and buffer consumption rate. At the start of a stream, the client downloads a *manifest file*, which contains metadata, including the available encodings and the location of their corresponding segments. Since manifests and segments are static files, ABR streaming works seamlessly with Content Delivery Networks (CDNs) where these files are cached at edge servers. Widely recognized and supported by Bitmovin's 7th Annual Video Developer Report [6], the industry leading ABR protocol implementations are *Dynamic Adaptive Streaming over HTTP* (DASH), also known as *MPEG-DASH*, and Apple's *HTTP Live Streaming* (HLS), with other options seeing minimal use.

[1]All artifacts, including datasets and capture files for reproducibility, are available on Zenodo: https://zenodo.org/records/14676527.

For convenience, the code is also accessible to view on GitHub: https://github.com/trustcom/endangered-privacy.

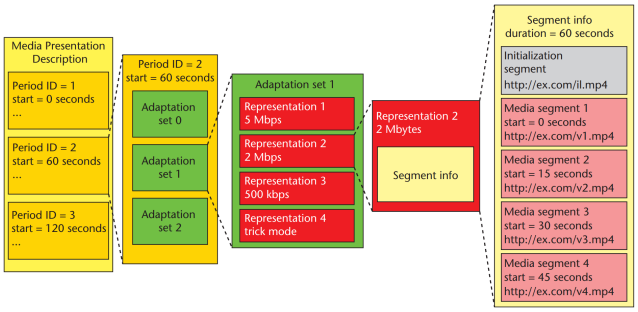

Figure 1: The hierarchical structure of a DASH manifest; source: Sodagar, 2011 [26].

DASH and HLS. DASH is the only ABR protocol that is an international standard [3]. It was developed by the Moving Pictures Expert Group (MPEG) with the goal of having an open standard alternative to HLS, which is proprietary to Apple. DASH is widely supported by content providers and devices, with iOS devices being a notable exception. DASH's manifest file (MPD extension, for *Media Presentation Description*) is an XML-based document organized in a hierarchical structure (cf. Figure 1), and summarized hereafter:

1. **Media Presentation** (root): primary component holding all the other elements.
2. **Periods** (one or multiple): individual chapters of a video or strategic breakpoints where advertisements can be inserted.
3. **Adaptation Sets** (for each period): comprise various representations of the media (e.g. several encodings at various bitrate levels, all using the same video codec); different video codecs are stored in separate adaptation sets.
4. **Representations**: distinct versions of media within adaptation sets containing multiple segments identified by an average bitrate; the representation is selected by the client based on its available bandwidth level.
5. **Segments**: references to the individual variable bitrate encoded segments that the client fetches and plays back.

ABR streaming protocols define how the media segments are delivered from the server to the client, facilitated by the manifest. The segments themselves are wrapped in a container file format, with additional metadata, encryption if using Digital Rights Management (DRM), and a codec that indicates how the video data is compressed and decompressed. DASH's MPD most often reference fragmented MP4 containers that are both encryption and codec agnostic. The individual media segments can be part of a contiguous file, in which case the client sends HTTP byte-range requests to fetch them. We lend our terminology from the DASH Industry Forum [11] and call

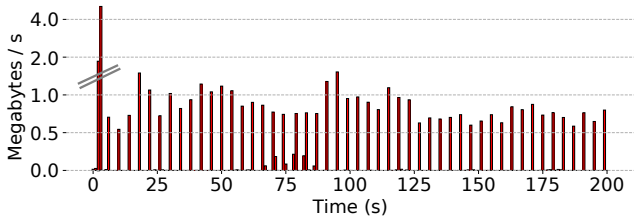

Figure 2: The *bursty* behavior of ABR streaming, starting off with the buffer-fill before transitioning to the steady state.

this mode *indexed addressing*. When indexed addressing is used, the MPD specifies the location of the file and the range of the *index segment* that contains the byte ranges of all other media segments. Segments can also be addressed with *explicit addressing* and fetched via individual URLs that are either directly listed in the MPD or constructed using a template based on sequence numbers or explicitly defined time spans.

HLS [19] predates DASH and has wide support even though it is not an open standard. HLS manifests are represented by M3U8 files known as (master) *playlists*. While structurally different than an MPD, its purpose is nevertheless the same and the concepts of representations and addressing modes also exist in HLS.

CMAF. To be able to stream, devices and various software must support the streaming protocol, the underlying media and its encryption scheme. Consequently, content providers typically package their content for both DASH and HLS with their respective common stacks, i.e. MP4 for DASH and MPEG-TS for HLS. This inefficiency of both packaging and storing twice as much data is what the *Common Application Media Format* (CMAF) [1] was introduced to solve. CMAF allows for a single fragmented MP4 container based on the ISO base media file format to be referenced by both DASH and HLS manifests, making use of MPEG Common Encryption [2] that is agnostic to the DRM employed. In this paper, to keep consistency, we adhere to the DASH terminology but it is worth noting that the concepts also apply to HLS.

2.2 ABR Streaming Leak

When a client initiates a video stream, it first requests the manifest that facilitates the stream. During start-up, the client has an empty buffer and typically requests segments from a low bandwidth representation to ensure earliest possible playback. Once the client has a full buffer it reaches a steady state, where the time between segment requests depends on the buffer consumption rate. The client continuously estimates its bandwidth capacity to decide from which representation it should request segments from. This on and off period causes a pattern in the network trace that has been described as *bursty* [24], visualized in Figure 2. Each burst can be associated with the client requesting and downloading one or more segments. Because segments are encoded with a variable bitrate, their sizes depend on the content, with more bits allocated to complex ones (e.g. vibrant action scenes) than to simpler ones (e.g. mostly static scenes). The leak relies on the distinctive network trace that aligns with the order and sizes of the segments belonging to a specific video representation.

Fingerprints. In previous works, fingerprints have had varied definitions. For instance, in machine learning works [5, 24], the observed network trace is labeled a fingerprint, and multiple streams of the same video is used as training data. As mentioned previously, we define a *fingerprint* as the complete sequence of exact segment sizes corresponding to a specific representation of a video. Consequently, a video available on a streaming service possesses a unique fingerprint for each of its available representations. Our method, like previous fingerprinting methods [7, 13, 22, 23, 30], makes use of the ground truth fingerprint data to identify videos and does not require streaming videos beforehand. To acquire a fingerprint, an adversary is required to first locate a manifest file or an equivalent JSON schema (e.g. YouTube and Vimeo specific format equivalent to the MPD). In scenarios where indexed addressing is employed, the adversary must access the index segment to get the byte ranges that constitute the fingerprint while in explicit addressing, the information of each segment is retrieved via its corresponding URL.

2.3 Studied Streaming Services

We evaluate our methods across the complete libraries of three well-established video-on-demand streaming services: Amazon Prime Video, Max (the product of the recent merger of Discovery, Inc. and WarnerMedia), and SVT Play, Sweden's national public television broadcaster's streaming service. Amazon and Max utilize indexed addressing for segments, whereas SVT employs explicit addressing. These distinctions require slightly different data collection strategies, to be explained in § 3.1. Besides available bandwidth, the representations a client can select during streaming are influenced by device capabilities and DRM restrictions. For example, on Amazon, browsers on Linux-based systems are limited to standard definition. A key advantage of our fingerprinting method is that it can collect all such representations, enabling device-independent identification.

Our focus on over-the-top (OTT) video-on-demand streaming services, rather than online video-sharing platforms, is intentional. OTT services typically have more manageable libraries, allowing us to collect data for every video, ensuring that network traces from these service's video CDN servers correspond to videos in our dataset. In contrast, previous studies targeting platforms such as YouTube and Facebook have demonstrated accuracy on subsets of videos between 100 and 1000, but need to consider the base rate when making claims about generalization (for context, YouTube is estimated to host more than 10 billion videos [16]).

2.4 Attack Models

Strong attacker. This model, common in previous works, represents the typical setting where an attacker (e.g. a monitoring ISP or network admin) can isolate IP communication between the target and the streaming service. The attacker can achieve this by maintaining a lookup table of known IP addresses or by intercepting the TLS handshake to obtain the hostname from the Server Name Indication (SNI). In our work, we do not make any distinctions between transport layer details. Although TCP information can be exploited by the attacker [23], and recent studies have focused on the increased complexities of QUIC-based traffic [27, 28], our approach relies almost exclusively on the characteristics of ABR streaming and look no further than the IP-layer. In this model, according to its available information, the attacker has knowledge of the streaming service being used.

Weak attacker. In this model, we consider two scenarios: VPN and Wi-Fi eavesdropping. For the VPN scenario, the attacker is on the network path, but the target uses a VPN (e.g. a commercial VPN proxy service), funneling all of the target's internet traffic through an encrypted tunnel. In the Wi-Fi scenario, a Wi-Fi eavesdropper, not connected to the network but within range, would observe an aggregated stream where multiple IP conversations are hidden behind encrypted link-layer communication. In this case, the attacker captures frames using a monitor mode device.

We assume the target is not *actively* doing anything other than streaming a video. This corresponds to the typical and realistic attack scenario we aim to reproduce, as most viewers are likely focused on the video itself, making additional online activities less probable during the streaming session. For instance, smart TVs often enforce this behavior by allowing only one application to run in the foreground. However, note that the target's device still operates in a typical setting, generating routine background traffic from system processes and passive network activity, which differs from intentional user-driven actions. This is to mimic a real-world scenario, hence, we do not disable protocols or standard applications that would otherwise generate traffic. The attacker has to deal with that traffic as additional noise, including data from the streaming services themselves that are not the actual stream, such as ad and telemetry data.

In the weak attacker model, the attacker cannot identify the streaming service directly from the communication addresses, as the VPN conceals the service's IP, and Wi-Fi eavesdropping reveals only the device's MAC address. In this work, we do not make any assumptions about the attacker's ability to identify a service through other means[2]. In any event, we believe it to be valuable to test our system's capabilities to handle the noisier data of VPN and Wi-Fi traffic. We extend our evaluation with an experiment to see how the system behaves when queried with data unassociated with Amazon, Max and SVT. Note that, in the worst case, the attacker must consider the set of all possible internet traffic as input, i.e., already billions of videos from other services even if restricted to video streaming only.

Common assumptions. For both models, we assume that the target maintains a stable enough connection during streaming to consistently select segments from the same representation for continuous periods of up to ca. 1 minute (more specifically $k+1$ bursts as detailed in § 3.4). As previously discussed, this steady state is typical for DASH or HLS streams under good network conditions, where the device strives to maintain the highest possible representation. However, in unstable network environments, such as those in some developing countries, frequent representation switching can occur, making identification more challenging. Such conditions are not considered and is a limitation of our work.

[2]For instance, one may run separate service detection methods prior to using our system, possibly based on service-specific buffer management strategies as discussed in § 5. Alternatively, the attacker may already suspect that the target is using a specific service and only then employ directed monitoring, e.g. setup a Wi-Fi device in close proximity to the target.

3 Protocol-agnostic Classification of Video Network Traces

Attack overview. The attack consists of two phases. In the first, video metadata and fingerprints belonging to the targeted services are collected and preprocessed fragments of all fingerprints are stored in a k-d tree data structure *per attacked service*. In the second or active phase, traffic from the target is captured and analyzed in real-time. Bursts present in the traffic are isolated and fed to an identification subsystem in charge of querying the fingerprint databases. The identification logic assigns a score to candidates and when the highest score is beyond a predetermined threshold the system outputs a match. The entire identification system functions on a continuous basis and identification is updated for instance when the target switches to another video. Our method is explained in detail hereafter.

3.1 Data Collection

We have developed a dedicated tool, *karl*, designed to fingerprint videos compatible with ABR streaming. It provides a common interface for fingerprinting videos across different services and also supports arbitrary DASH and HLS manifests (currently limited to commonly used recent versions) or fragmented MP4 files. While previous fingerprinting methods are relatively efficient compared to ML-based approaches, they typically rely on using a web driver[3] to crawl and start streams in order to intercept manifest information. For example, Reed and Kranch [23] collected a dataset of 42,000

[3]https://www.w3.org/TR/webdriver2

Table 2: Our dataset covering the full libraries of Amazon Prime Video, Max, and SVT Play.

Service	# Videos	# Representations
Amazon	116,070	1,285,713
Max	99,478	1,544,778
SVT	26,816	230,582
Total	242,364	3,061,073

Netflix videos and 330,000 fingerprints over two months using such a crawler. In contrast, karl is purely HTTP-based, eliminating the need to launch full browser instances. Although this approach requires understanding internal service-specific details to retrieve metadata and manifest locations, the process of extracting fingerprints remains the same.

Amazon and Max are subscription services, but aside from initiating DRM-protected playback, authorization is only required to retrieve video metadata. In fact, manifests and media files, which account for approximately 95% of requests, can be collected entirely unauthenticated from their respective CDN servers. Recall from § 2 that we require only the index segment (which is not DRM-protected) or the individual URLs if explicit addressing is used to derive fingerprints. Furthermore, when DASH and HLS manifests reference the same underlying CMAF-compliant containers, extracting fingerprints from either format is sufficient. In detail, when indexed addressing is used, karl fetches the byte range of the index segment and derives the fingerprint directly from the *sidx* atom of the fragmented MP4 file. For explicit addressing, karl constructs the URLs and sends HTTP HEAD requests to each media URL to obtain the segment size from the `Content-Length` header field.

As mentioned previously, our data collection method differs significantly from prior efforts, where data collection is a major bottleneck, often requiring hundreds or even thousands of hours [5] to collect data for a modest number of videos. In contrast, karl collects our entire dataset in less than a day from a single host, even with throttling applied to avoid IP-based blocking. On the other hand, ML models can learn network-based variations during streaming, which has been identified as a challenge for fingerprinting-based systems. However, as demonstrated in this work, our heuristic successfully distinguishes videos among 242,363 others, even in VPN and Wi-Fi attack scenarios.

Dataset. Our dataset (cf. Table 2) comprises the libraries of Amazon, Max, and SVT around the time of our evaluation. In this work, we deliberately collect only DASH manifests, which also allow for the identification of HLS streams when CMAF-compliant containers are used. However, HLS-based MPEG-TS streams are excluded. karl can fingerprint all such cases, but due to time, storage, and memory constraints, we limit our dataset to fingerprints derived from DASH manifests. We make our dataset publicly available as a set of Tab-Separated Values (TSV) files totaling 19.8 GB. Additionally, 264 videos were intentionally discarded due to their short duration (under one minute) as our system is not capable of identifying videos that fit entirely within a buffer's length. A real-world application of our methods could keep the dataset up to date as content is rotated in and out by running karl on a daily schedule.

3.2 Organizing the Data: *k*-d tree

Database(s). Similar to some earlier works [7, 22, 23], we employ a k-d tree for the initial lookup. To achieve this, we preprocess our collected dataset to create a database $\mathcal{D}_k$ (fully defined below) suitable as input to the tree, k being a fixed parameter of our system.

Let I be the set of all representations targeted by the attack. For each representation $i \in I$, let $S_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,n_i})$ be its fingerprint, with $s_{i,j}$ representing the size of its j-th segment and n_i its total number of segments. Let the set of all segment sizes be denoted by $S = \{s_{i,j} \mid i \in I, 1 \le j \le n_i\}$. We define the normalized fingerprint, with each element rescaled to be in the $[0,1]$ range, as $S'_i = (s'_{i,1}, s'_{i,2}, \ldots, s'_{i,n_i})$ where

$$s'_{i,j} = \frac{s_{i,j} - \min(S)}{\max(S) - \min(S)}.$$

Furthermore, let $\partial S'_i$ be the discrete derivative of the normalized fingerprint S'_i, i.e., for $i \in I$, $\partial S'_i = (\Delta s'_{i,2}, \Delta s'_{i,3}, \ldots, \Delta s'_{i,n_i})$, where $\Delta s'_{i,j} = s'_{i,j} - s'_{i,j-1}$ for $j = 2, 3, \ldots, n_i$. The reasoning behind computing the differences of each adjacent segment size is to make later comparison with imperfect data easier, explained further in § 3.4.

For a given dimension $k \ge 1$ and representation $i \in I$, we denote W_i^k the *window view* of width k over $\partial S'_i$, defined as:

$$W_i^k = \begin{bmatrix} \Delta s'_{i,2} & \Delta s'_{i,3} & \cdots & \Delta s'_{i,k+1} \\ \Delta s'_{i,3} & \Delta s'_{i,4} & \cdots & \Delta s'_{i,k+2} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta s'_{i,n_i-k+1} & \Delta s'_{i,n_i-k+2} & \cdots & \Delta s'_{i,n_i} \end{bmatrix}$$

The rows of the window view W_i^k can be conceptualized as each k-length sequence observed as a sliding window traverses $\partial S'_i$. Regardless of the attacker model, for each attacked streaming service ν, we build its *associated database* $\mathcal{D}_k^\nu = \bigcup_{i \in I_\nu} W_i^k$ by repeating the above process for every normalized fingerprint S'_i with $i \in I_\nu$, where I_ν is the set of targeted representations from service ν.

k-d tree. For a given database $\mathcal{D}_k^\nu$, we store every row of every window view $W_i^k \in \mathcal{D}_k^\nu$ in a k-d tree $\mathcal{T}_k^\nu$, where the ℓ-th row

$$W_i^k[\ell] = \Delta s'_{i,\ell+1}, \Delta s'_{i,\ell+2}, \cdots, \Delta s'_{i,\ell+k}$$

of W_i^k can be interpreted as a k-dimensional point in space.

We populate our data structure with triplets of the form $\langle W_i^k[\ell], i, \ell \rangle$, with $W_i^k[\ell]$ being used to calculate distances between points as used in the k-d tree's $O(\log n)$ lookup operation, whereas (i, ℓ) is used to retrieve the video information corresponding to the k-dimensional point $W_i^k[\ell]$. The `search` operation provided by our fingerprint database can be viewed as providing an implementation for a function

$$T_{\tau,r}^k : \mathbb{R}^k \to \mathcal{P}(\mathbb{N} \times \mathbb{N} \times \mathbb{R}),$$

that maps a k-dimensional point p to the τ closest points to p within radius r in $\mathcal{T}_k^{\mathrm{v}}$. Technically, the k-d tree $\mathcal{T}_k^{\mathrm{v}}$ is used to efficiently retrieve $\mathcal{T}_k^{\mathrm{v}}(p, \tau)$ the τ closest points to p in the tree, before filtering out points outside of radius r. Formally, the output set $T_{\tau,r}^k(p)$ is made of triplets of the form $\langle i, \ell, d \rangle$ where $i \in I$ is a video representation, $1 \leq \ell \leq n_i - k$ is a row number of W_i^k (i.e., $W_i^k[\ell]$ is a specific sequence of length $k+1$ within the video associated to the i-th representation; 1 added due to the normalization step) and $0 \leq d \leq r$ is a distance such that:

$$\langle i, \ell, d \rangle \in T_{\tau,r}^k(p) \Leftrightarrow W_i^k[\ell] \in \mathcal{T}_k^{\mathrm{v}}(p, \tau) \wedge d = \mathrm{dist}(p, W_i^k[\ell]) \leq r,$$

with $\mathrm{dist}(p,q) = \sqrt{\sum_{j=1}^k (p_j - q_j)^2}$ being the euclidean distance between the two k-d points $p = (p_1, \ldots, p_k)$ and $q = (q_1, \ldots, q_k)$. The introduced output format is relevant for the identification heuristic outlined further in § 3.4.

3.3 Capturing Traffic: Packets and Bursts

The objective of the capture is for the attacker to aggregate packet (or frame) sizes into chunks analogous to the actual segment sizes downloaded during video streaming. These chunks can subsequently be used to query the k-d trees to obtain a shortlist of potential candidates. Reed and Kranch [23] used TCP sequence and acknowledgment numbers to infer the sizes of HTTP application data on each connection. Other fingerprinting based works have used some version of a heuristic that group packets into time-based intervals. While not as exact as the method of [23], it suites ABR streaming traces well given their bursty behavior (cf. Figure 2).

In our work, we do not assume anything about the network protocol in use other than being able to distinguish a source, destination, a time and a size of a packet or frame. This allows the attacker to be efficient, needing to parse only the IP header of each packet (assuming the attacker can also keep a lookup table for service addresses). We created a tool, *BurstShark*, that wraps TShark to also aggregate sequences of packets or frames into bursts based on throughput, with some additional features such as real-time output, allowing us to carry out the attack and identify videos in a live setting by querying our system as it creates bursts. While BurstShark is effective for live analysis, ISP-level traffic monitoring would require more specialized tooling that is not based on *libpcap*. BurstShark performs some data cleaning, such as discarding bursts under a certain threshold unlikely to be video, and accounting for the header sizes on the outermost layer, but it is far from rigorous. For example, we do not know if packets unassociated with the video stream are interleaved in a burst. We do not make assumptions about transport details, e.g. multiple connections on different ports typical of HTTP/1.1 and TCP that offers insight to the attacker and those are instead aggregated grouped by IP address. Even as the strong attacker, the IP conversation that carries the video may also contain non-video traffic, e.g. audio segments or text. However, instead of attempting to clean the data on the network level, we accept that our query points contain such noise and rely on our identification heuristic to manage it.

Wi-Fi capturing. BurstShark also reads 802.11 traffic, provided a wireless interface in monitor mode. Here, we bin conversations into a tuple consisting of the source and target MAC address. Capturing 802.11 data frames is not always trivial; while control and management frames can be simpler to catch, data frames are usually transferred at full speed. The monitor device must match the capabilities of the transmitting device such as frequency, bandwidth and spatial streams. Based on the equipment we have available, we limit ourselves to 2.4 GHz 802.11n set to a fixed channel. Despite this, it is common for our device to miss sequences of frames. However, since we only need the total volume of transmitted data, we estimate missing bytes by using sequence numbers, identifying gaps, and computing each missing frame's size based on the average size of adjacent frames.

3.4 Identification Logic

In this section, we detail how bursts produced by our capture tool are converted into the system's predictions: which video is being streamed at each timestep. It introduces a number of system parameters, whose values and trade-offs are discussed in the next section on system configuration.

Querying the k-d tree (strong attacker). Let $X = (x_t)_{t \geq 1}$ be an unbounded series of bursts produced by our capturing program BurstShark at discrete time steps; timestep t is referred as "time t" whenever the context makes it unambiguous. As for the ground-truth datapoints (see the preprocessing steps outlined in § 3.2), let $X' = (\Delta x'_t)_{t \geq 2}$ be the discrete derivative of the normalized burst sequence X, i.e. $\Delta x'_t = x'_t - x'_{t-1}$ for $t \geq 2$, where $x'_t = \frac{x_t - \min(S)}{\max(S) - \min(S)}$.

The reason behind manipulating the *differences* instead of the absolute numbers is to eliminate potential constant overhead present in adjacent bursts. For example, while the video is variable bitrate encoded, audio is typically constant bitrate and has a similar size in each burst. Also, the added noise due to background traffic (particularly relevant for VPN and Wi-Fi scenarios) will mostly on average cancel out if we assume the volume of background traffic stays roughly constant over time (at least when aggregated over segments

duration). We define the *query (k-d) point* X_t^k at time step $t \geq k+1$ as follows: $X_t^k = [\Delta x'_{t-k}, \Delta x'_{t-k+1}, \ldots, \Delta x'_t]$ is the derivative of the normalized $k+1$ last recorded bursts at time $t \geq k+1$. Note X_{t+1}^k is efficiently calculated from X_t^k using a sliding window of length $k+1$ over the bursts sequence.

At last, we query our fingerprint database $\mathcal{T}_k^{\nu}$ for the corresponding service ν (that was already built for a specific value of k before the capturing tool started) to retrieve the set of triplets $T_{\tau,r}^k(X_t^k)$ following the fixed parameters τ and r used to configure the system; see § 3.2 for the definition of $T_{\tau,r}^k$. We call $C_t^k = T_{\tau,r}^k(X_t^k)$ the set of *candidate triplets* at time step t.

Querying the k-d trees (weak attacker). Since the weak attacker potentially does not know which video streaming service is being used (or even if the captured traffic corresponds to streaming traffic), we query in this case all the available fingerprint databases and consider as candidates the (at most) τ triplets with smallest distance from the queried point X_t^k among all retrieved triplets.

Grouping candidate triplets into alignment groups. In previous k-d tree works [7, 23], a single query was used to declare an identification, with each query treated independently without regard to queries at different points in time. Due to isolation of the conversation at the transport layer and data cleaning, a query point could then be almost identical to its counterpart stored in the tree. In this work, we cannot rely solely on a single query point due to increased noise, which generates too many false positives (cf. comparison in Appendix C). Instead, we introduce the concept of *candidates* (distinct from candidate triplets), an illustrative example being included at the end of the paragraph. In detail, we group candidate triplets $\langle i, \ell, d \rangle \in C_t^k$ by their representation i and *alignment* $j = t - \ell$ into the *alignment group* $(i, j) \in I \times \mathbb{Z}$.

We say that two candidate triplets $\langle i_a, \ell_a, d_a \rangle \in C_{t_a}^k$ seen at time step t_a, and $\langle i_b, \ell_b, d_b \rangle \in C_{t_b}^k$ seen at time step $t_b \geq t_a$ are *aligned* if

$$i_a = i_b \quad \wedge \quad \ell_b - \ell_a = t_b - t_a.$$

According to the *time series lag* parameter L, an alignment group $A(i,j)_t$ "at timestep t" is made of all candidate triplets seen during the last L time steps that belong to the alignment group (i, j) – all such triples are then aligned with each other (note that being aligned is indeed transitive), i.e.,

$$A(i,j)_t = \bigcup_{t-L \leq t' \leq t} \{ \langle i, \ell, d \rangle \in C_{t'}^k \mid t' - \ell = j \}.$$

We call *candidate* at time t any non-empty alignment group $A(i,j)_t$ at time t, with

$$\mathcal{C}_t = \{ A(i,j)_t \mid A(i,j)_t \neq \emptyset \}_{i \in I, j \in \mathbb{Z}}$$

being the set of candidates at time t.

An alignment group (i, j) is said to be *present* at time t if a triplet corresponding to that alignment group has been observed for time step t, i.e., $\langle i, t-\ell, d \rangle \in A(i, t-\ell)_t$ for some $d \in \mathbb{R}$. In other words, the set of candidates $\mathcal{C}_t$ at time t correspond to all alignment groups that have been present at least once in the most recent L timesteps.

Example: To give an example of candidates, assume the target is watching some video (representation) at timestamp 1:15:30 and one of the triplets obtained by querying our k-d tree identifies this part of the video. Now, assume there are no matches, i.e. candidate triplets, for the video in the next minute because of noise. Then, one minute later a match for the same representation at timestamp 1:16:30 is retrieved, correlating with where the target would be in the video, assuming uninterrupted playback. Our underlying hypothesis is that this does not appear out of mere luck. Hence, whenever candidate triplets are aligned, it is a strong indication that the considered representation likely belongs to the video being played. Note we used real timestamps here for the sake of the example as opposed to the discrete time steps of our burst-based system.

Scoring. Recall, we have so far built at time t a set of candidates $\mathcal{C}_t$ from which we need to decide if one of the candidates $c \in \mathcal{C}_t$ produces a *match*, i.e., if our identification system is confident enough in the candidate c to output it for time step t (meaning its score exceeds a predefined threshold θ). To decide for a match, we rank all candidates of $\mathcal{C}_t$ using a heuristic scoring function. Intuitively, the score of a candidate indicates the likelihood of the candidate to correspond to the actual video being played with the following aspects increasing the score of a candidate: repetitive presence of the candidate in previous timesteps and closeness of previous query points to the candidate when it is present.

Let us now explain our scoring function $\texttt{score}(c,t)$ for any candidate $(i, j) \in \mathcal{C}_t$ in detail. For each candidate $c = (i, j) \in \mathcal{C}_t$ that is present at time step $t \geq k+1$, i.e., such that

$$\langle i, \ell, \mathrm{dist}(X_t^k, W_i^k[\ell]) \rangle \in A(i,j)_t$$

with $j = t - \ell$, we assign first a *base score* D_t^c with $0 < D_t^c \leq 1$ and determined by the exponential distance decay from the query point X_t^k at time t, i.e., calculated as

$$D_t^c = e^{-\lambda_{dist} \cdot \mathrm{dist}(X_t^k, W_i^k[\ell])},$$

where the constant λ_{dist} represents the *distance decay factor*. If a candidate $(i, j) \in \mathcal{C}_t$ is not present at time t then we set its base score to zero, i.e. for any alignment group $c = (i, j)$ we have

$$\neg(\exists d \in \mathbb{R}, \langle i, j, d \rangle \in C_t^k) \implies D_t^c = 0.$$

The base scores vector $\mathbf{D_t^c}$ for candidate $c \in \mathcal{C}_t$ at time step t is defined as

$$\mathbf{D_t^c} = [D^c_{\max\{k+1, t-L+1\}}, \ldots, D^c_{t-1}, D^c_t]$$

and contains the L most recent base scores for c; note if $t < L$, the vector $\mathbf{D_t^c}$ is of smaller size.

Figure 3: Illustration of weights for some w, k and L emphasizing recent scores as less important.

Furthermore, we introduce a vector $\mathbf{W} = [w_1, w_2, \ldots, w_L]$ made of L weights where $w_1, \ldots, w_{L-k-1}$ are all set to a fixed constant $w \geq 1$ and the remaining $k+1$ elements $w_{L-k}, \ldots, w_L$ are set to 1. The main input for the score for any candidate c present at time t is the weighted sum $S_t^c = \mathbf{W} \cdot \mathbf{D_t^c}$ of the base scores by the fixed weights. As our system is non-conclusive it needs to adapt its opinion over time, therefore we also log scale S_t^c and apply the *decay constant* λ_{score} to a candidate's previous score when it is not present at time t.

The score of candidate $c \in \mathcal{C}_t$ at timestep t is calculated as:

$$\texttt{score}(c,t) = \begin{cases} \ln\left(1 + \sum_{\substack{n \geq 1, \\ n \geq k+1+L-t}}^{L} D^c_{t-L+n} \cdot w_n\right) & \text{if c is present at time t,} \\ \texttt{score}(c,t-1) \cdot \lambda_{score} & \text{otherwise.} \end{cases}$$

Summary and justifications: In brief, the score of alignment groups that are absent from the last L timesteps is 0. Among candidates, when not present, the score of a candidate decreases exponentially by a factor λ_{score} at every time step. If present, the score is the product of the base scores of the time steps when it appears in the last L steps, weighted so that the most recent $k+1$ scores have a lesser importance. The prioritization of older base scores over recent ones is illustrated in Figure 3. We motivate this decision by assuming a *false* candidate c has been produced by the query point $X_t^k = [\Delta x'_{t-k}, \Delta x'_{t-k+1}, \ldots, \Delta x'_t]$. As the sequence evolves to $X_{t+1}^k = [\Delta x'_{t-k+1}, \Delta x'_{t-k+2}, \ldots, \Delta x'_{t+1}]$, all elements in X_t^k exist in X_{t+1}^k except for the discarded element $\Delta x'_{t-k}$ and the added element $\Delta x'_{t+1}$, highlighting this latter one as the focal point for determining the likelihood of c reappearing as a result of X_{t+1}^k. With each new time step, the influence of the initially overlapping elements on the likelihood of c diminishes. This diminishing likelihood can be expressed as $\mathbb{P}(c \mid X_{t+1}^k) > \mathbb{P}(c \mid X_{t+2}^k) > \ldots > \mathbb{P}(c \mid X_{t+k}^k)$. Consequently, we place less emphasis on recent base scores, and our empirical data have shown this to be effective in preventing high scores for false candidates.

Prediction. In each time step t we select the highest score candidate above a predefined threshold θ which becomes the prediction. If no candidate has a score above the threshold, then the system does not make a prediction, instead returning -1 to signal that it is not confident enough. Additionally, let us observe that because a candidate also includes the alignment $j = t - \ell$, we can derive the playback position of the target by using ℓ, the segment duration and accounting for the buffer length used by the service.

Table 3: Configurable system parameters and default values.

Variable (§ def.)	**Value**	**Description**
k (§ 3.2)	8	Dimension of the k-d tree
r (§ 3.2)	0.025	Radius of the k-d tree search
τ (§ 3.2)	50	Max elements retrieved from $\mathcal{T}_k^v$
L (§ 3.4)	75	Time series lag
w (§ 3.4)	3.0	Weight for $t-L, \ldots, t-(k+1)$ scores
λ_{dist} (§ 3.4)	100.0	Decay constant for base scores
λ_{score} (§ 3.4)	0.9	Decay constant for candidate scores
θ (§ 3.4)	2.2	Confidence threshold

3.5 System Configuration

System parameters. The value that we set for our system parameters are presented in Table 3. $k = 8$ is a trade-off between the *curse of dimensionality*, identification time and accuracy. We set $r = 0.025$ and $\tau = 50$ to limit the lookup in both sparse and dense regions of the k-d trees. To prioritize older scores, the weight parameter is set to $w = 3.0$ to lessen the impact of false candidates appearing in clusters (as explained in § 3.4) and handle highly similar footage. The lag is set to $L = 75$ as a trade-off memory usage and system accuracy. We set the decays to $\lambda_{dist} = 100.0$ and $\lambda_{score} = 0.9$ to have base scores close to 0 when points are close to r. With these parameters and when the system is the most confident, scores converge to 5.0 and drops below 2.2 upon changing videos (as visualized in Figure 5 where $\theta = 2.2$). The effect of different thresholds is demonstrated in § 4. The same parameter set is applied across all of our attack scenarios, although different system configuration could potentially be more effective given the variable quality of data. Prior to expanding our dataset and evaluation to include Amazon and Max videos, tuning was performed on 200 SVT video streams (0.1% of our dataset) in a VPN-scenario. While these parameters are not necessarily optimal, they have proven sufficiently effective for our heuristic approach. The reasons behind the selection of all system parameters are described in more depth in Appendix B.

Implementation. Our implementation uses NumPy, pandas and the k-d tree implementation of scikit-learn. The 8-dimensional trees consist of approximately 1.89 billion points derived from 3 million fingerprints, requiring around 90 minutes to build and approximately 152 GB of memory. In practice, an attacker may update the dataset and rebuild the trees once per day. A lookup, i.e. call to the `search` function of one k-d tree, takes about 18 ms on average on our midrange hardware. Being in the order of milliseconds, for our evaluation it is deemed negligible against the playback (wall) time.

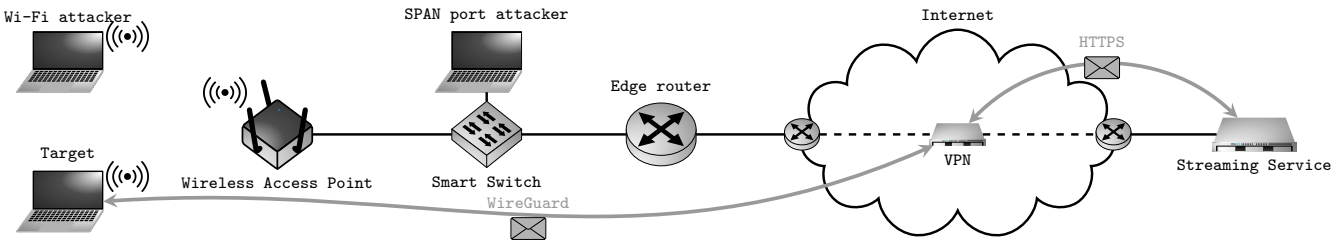

Figure 4: Illustration of the attack and testing environment; note that the testing setup has been designed to facilitate our evaluation and most elements are not required for the attack to be performed.

Table 4: Videos streamed per device (evenly distributed among the services): (a) HTTPS streams for the strong attacker model and (b) VPN streams for the weak attacker.

	(a) HTTPS		(b) VPN	
Device	**# Videos**	**Time (h)**	**# Videos**	**Time (h)**
Linux	339	56.5	0	0
Mac	255	42.5	429	71.5
Windows	306	51	411	68.5
Total	900	150	840	140

4 Evaluation

We present in this section a thorough evaluation of the identification method presented in § 3.1-3.4 and parameterized by the default values as presented in Table 3 and explained in § 3.5. We first explain in detail the real network environment that is used for our evaluation, before presenting our results.

4.1 Experimental Setup

Network environment. Our aim is to mimic a typical real life streaming scenario, therefore we setup a regular home network, see Figure 4, with an internet-facing router. LAN devices, i.e. the *targets*, connect to the network wirelessly through a separate access point. Between the router and the access point we setup a layer 3 switch with monitoring capabilities. We connect the *SPAN port attacker* to the switch, such that it receives a copy of all network packets on the Access Point (AP) port. While the attacker is part of the LAN in our experimental setup, it could be deployed anywhere on the network path between the target and the video service. For the evaluation of the strong attacker model, there are three target laptops streaming with HTTPS only, a Lenovo laptop running Ubuntu 24.04, a MacBook running macOS 11, and an Acer laptop running Windows 11. We refer to them as the Linux, Mac, and Windows devices respectively. For the weak model, we do not stream on the Linux device. Instead, we configure it as a VPN (WireGuard) relay positioned outside the LAN, allowing the Mac and Windows devices to use it as an exit node. As a result, from the SPAN port attacker's perspective, all network traffic is now going through an encrypted tunnel. Furthermore, we set up the *Wi-Fi attacker* in close proximity to the target devices, a device equipped with a network card in monitor mode to capture the Wi-Fi traffic. For clarity, both the VPN and Wi-Fi evaluations are conducted using the same VPN-streamed sessions. The target devices are fully updated and use default OS-level settings. As mentioned, and relevant for the VPN and Wi-Fi attacks, we purposefully try to emulate a typical user, so the Mac is signed in with an Apple ID, and the Windows device is signed in with a Microsoft account. Consequently, each device generates some idle network traffic, such as syncing, telemetry, and checking for updates. A shared web driver program for the Chrome browser is used to navigate each service and start playback automatically.

Test set. We select randomly videos from Amazon, Max and SVT that are at least 15 minutes long for each device to stream. Because our system is designed to be continuous and adaptive to changes, each device streams three videos (one from each service) at a time, also in a random order. Each video is streamed for 10 minutes, seeking to a random position with enough time remaining as the starting point. This duration is chosen to provide sufficient playback for querying while also balancing the evaluation cost in terms of time spent watching videos and storage needed for saving PCAPs. The attacker runs the capture tool for the full 30 minutes. The system is capable of real-time identification but the evaluation is instead done by saving the burst files and the original PCAPs for reproducibility. Note, however, that there is no required post-processing step, and evaluation results for a certain configuration apply to the real-time use case as well. Table 4 shows the number of streams and total duration per device in both evaluation settings. Our webdriver script often failed to initiate playback, or crashed due to stale web elements or CAPTCHA prompts. Sessions that contained such failures were discarded, as they do not accurately emulate real user behavior.

In the weak attacker model, we also conduct an experiment to evaluate how the system reacts to data unassociated with Amazon, Max and SVT videos. For this, we continuously stream random videos on YouTube using YTRoulette[4], obtaining 149 hours of VPN traces. Additionally, we use capture

[4] https://ytroulette.com

traces from HBO Max, from our own earlier experiments prior to the launch of its successor Max, comprising 68.8 hours of VPN and Wi-Fi captures. On top of our own collected data, we make use of the public VNAT dataset [14] that includes streaming traces from YouTube, Vimeo and Netflix as well as a range of other applications in a VPN and non-VPN setting, accounting for 490 hours of network traffic (cf. Table 6 in Appendix D).

4.2 Results

The system's performance at any given time step t is determined by whether it outputs a correct prediction for representation i or returns -1 if confidence falls below the predetermined threshold θ. This non-conclusive approach necessitates an evaluation over time, particularly as the target content changes. In order to assess the system in a way that is interpretable, we account for the playback time in seconds $0 \leq t \leq 600$, and define the following three mutually exclusive outcomes for a single playback at time t:

- **Identified**(t) (abbrv. **Id.**): At least one correct prediction within time t with no incorrect predictions[5] inside the full 600 seconds watching session.
- **Misidentified**(t) (abbrv. **Mis.**): At least one incorrect prediction inside the watching session; constant for all t.
- **Unknown**(t) (abbrv. **Unk.**): Neither identified within t nor misidentified.

Here, a correct prediction means that the predicted representation belongs to the video that is being streamed by the target, while an incorrect prediction belongs to some other video. When the target changes video, the score of the previous video automatically decays. For this reason we allow a 90 second grace period, but only if the system's prediction is associated with the previous video played by the target. Additionally, since we classify a playback as misidentified if there is a single incorrect prediction inside the full 600 seconds, we can safely say that a video identified at some time $t = t_0$ will also be identified for all $t_0 < t \leq 600$. Furthermore, for Amazon, Max and SVT videos (i.e. data known to the system) we define the *precision* and the *recall* as follows:

$$\textbf{Prec.}(t) = \frac{\sum_t \textbf{Id.}(t)}{\sum_t \textbf{Id.}(t) + \sum_t \textbf{Mis.}(t)}$$

$$\textbf{Rec.}(t) = \frac{\sum_t \textbf{Id.}(t)}{\sum_t \textbf{Id.}(t) + \sum_t \textbf{Mis.}(t) + \sum_t \textbf{Unk.}(t)}$$

[5] Except during a 90 second grace period when the target switches to another video during which only the previously watched representation is allowed to be "misidentified".

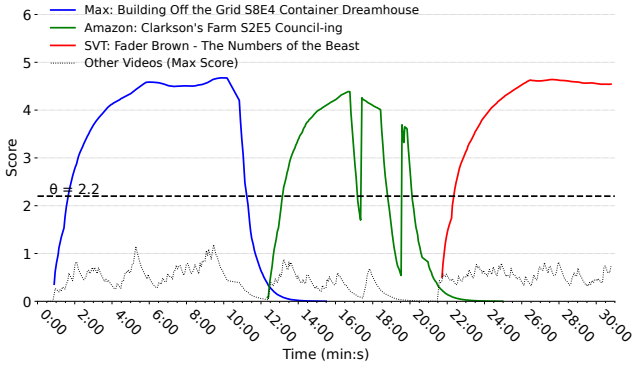

Figure 5: System scoring over time as target streams three videos with a VPN, demonstrating how it adapts over time.

Figure 6 shows the accuracy over time for each evaluation setting. Unsurprisingly, the system more rapidly identifies the HTTPS-based traffic compared to the noisier VPN and Wi-Fi traffic, with approximately 90% of streams being identified within 1:30 and the very last identification occurring at the 5:30 mark. Table 5 shows the cumulative results after 10 minutes of playback time, using a threshold of $\theta = 2.2$. The unidentified SVT videos in the HTTPS and VPN settings were HLS-based MPEG-TS streams, whose fingerprints were absent from our dataset as noted earlier. Nevertheless, any attacker with a more capable setup than ours could easily gather all such fingerprints as well. On the other hand, one unknown Amazon video was a cartoon, characterized by smaller segment sizes typical of static scenes and was streamed by the Linux device, resulting in reduced quality as well. Here, our set threshold for discarding bursts in our capture tool was set too high at 50 KB, causing legitimate segments to be ignored. Evaluating very low-quality streams may warrant further investigation. The other unknown Amazon video had an outdated fingerprint, possibly due to re-encoding. While our evaluation took place over a few days after collection, an attacker could run the data collection tool daily to avoid such cases. In the Wi-Fi evaluation, many Amazon videos remained unidentified, a stark contrast to the VPN scenario where the very same streams were largely identified. Analysis of the raw capture data indicates prolonged periods of frame loss in these cases, surpassing what our previously described recovery technique could handle. This issue likely stems from a limitation in our monitoring device, specifically its inability to process the large volumes of data received during Amazon's initial buffer fill, which may overwhelm the network card's capacity. During evaluation we observed a small number of misidentifications, but these turned out to be mirrors with separate video IDs (e.g. the same video with a different title or for another language) and such cases were ignored.

Unknown traffic experiment. We provide empirical evidence for the expected false positive rate of our identification

Table 5: Results with θ = 2.2 at t = 600 s.

(a) Strong attacker: HTTPS

Service	Id.	Unk.	Mis.	Prec.	Rec.
Amazon	299	1	0	1.000	0.996
Max	300	0	0	1.000	1.000
SVT	299	1	0	1.000	0.996
Total	898	2	0	1.000	0.998

(b) Weak attacker: VPN

Service	Id.	Unk.	Mis.	Prec.	Rec.
Amazon	279	1	0	1.000	0.996
Max	280	0	0	1.000	1.000
SVT	276	4	0	1.000	0.985
Total	835	5	0	1.000	0.994

(c) Weak attacker: VPN + Wi-Fi

Service	Id.	Unk.	Mis.	Prec.	Rec.
Amazon	165	115	0	1.000	0.589
Max	279	1	0	1.000	0.996
SVT	275	5	0	1.000	0.982
Total	835	121	0	1.000	0.856

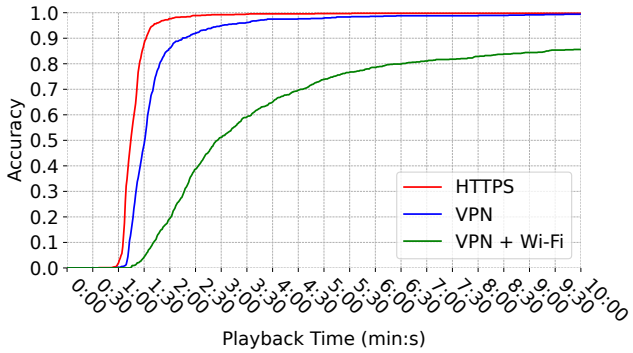

Figure 6: Cumulative accuracy for the evaluated attack scenarios over the playback time.

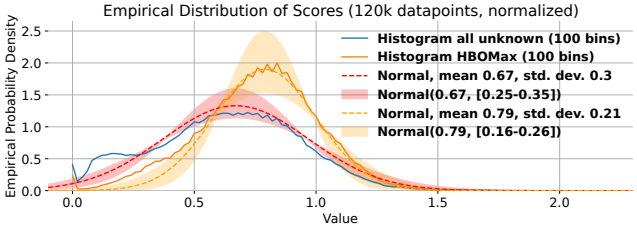

Figure 7: Empirical maximum scores over unknown traffic.

system, assuming a threshold θ set at 2.2. Consider the nearly 500 hours of unknown network traffic selected for our evaluation. Recall that this traffic is generated by various networking applications (including SSH, SCP, etc.) and not exclusively by adaptive bitrate streaming traffic. As a result, our burst identifier tool often fails to form query points, leading to an average of only one datapoint generated every 15 seconds (details are provided in Table 6 in Appendix D). The highest score ever produced by our system is 1.99, resulting in zero identifications and therefore no false positives.

Figure 7 displays the empirical distribution of the maximum score obtained by querying our system, i.e., the highest score among the 50 closest matches in our fingerprint database over all our unknown data. It also highlights HBO Max, the "unknown" application with the highest average maximum score. Following our experiments, the empirical maximum score distribution is well modeled by a normal distribution.

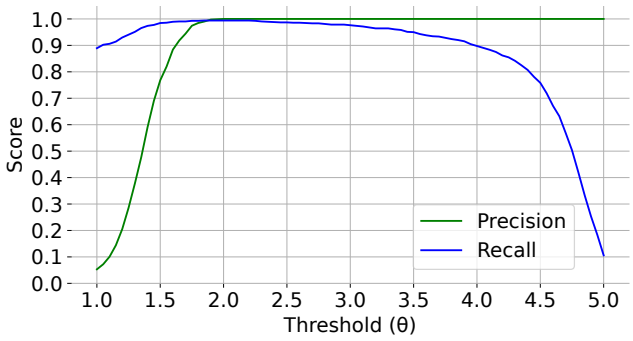

Figure 8: Effect of θ for 140 hours of known VPN traffic and 490 hours of unknown traffic.

Such a distribution quickly approaches zero, thus under this assumption, the probability that an unknown score is above θ =2.2 is $6.17 \cdot 10^{-6}$ with the mean μ and standard deviation σ parameters of the normal distribution pessimistically set to $\mu = 0.67$ and $\sigma = 0.35$ so to be well above the tail of the empirical distribution (cf. the zoomed-in plot Figure 10 on the tails of the distributions in Appendix D). Following this observation, we estimate that the maximum score from unknown traffic will exceed the set threshold only once every hundred thousand query points. Combining this with the average empirical time between query points, this translates to a false positive being (pessimistically) expected approximately once per month when the system is continuously fed unknown network traffic from a single source. This estimate is obtained under the simplified assumption that the observed data is representative of the overall distribution of such traffic. Furthermore, Figure 8, with precision modified to include false positives from the unknown set, demonstrates how the threshold can be adjusted to fine-tune the sensitivity of the system.

Summary. Our evaluation highlights the concerning aspect of the demonstrated privacy attack. Indeed, among a massive database of 242,000 videos and 3 million fingerprints, 99.5% of streams are identified within 10 minutes with successful identification occurring on average within 2 minutes of starting our capturing program. Given the ease with which our system can be deployed, these results are particularly alarm-

ing. In addition, the robustness of our system is maintained in noisier conditions: it can effectively identify streams when targets use VPNs to mask their traffic or if the attacker has no network access but simply monitor the wireless traffic within reach. In these cases, our preliminary results indicate some resistance to false positives, but further research is required to validate its effectiveness in more diverse conditions.

5 Mitigation

5.1 Traffic Manipulation

Recent studies [12, 21, 25] have explored defensive measures against encrypted traffic analysis attacks, primarily focusing on website fingerprinting. The VPN service Mullvad recently introduced an optional feature called Defense against AI-guided Traffic Analysis (DAITA) in their beta client [17]. DAITA is built using the Maybenot framework, introduced by Pulls *et al.* [21]. It works by ensuring constant packet sizes, interspersing random background traffic, and distorting data patterns to make it difficult for observers to identify meaningful activity or specific website visits. We conducted a brief experiment streaming five videos with and without DAITA activated. Our results, whose capture files are available as part of our published record, indicate that DAITA effectively mitigates the attack outlined in this paper but doubles bandwidth usage at best, potentially impacting quality of experience (QoE). These findings align with the authors' evaluation of the initial DAITA servers [20]. Nevertheless, we advocate for the development of such privacy-preserving features. DAITA, although in its early stages, shows promise and is likely to improve with further research. In its current state, DAITA appears to be a viable option for mitigating the attack when privacy is paramount, but further research is required to thoroughly evaluate its efficacy.

5.2 Addressing the Leak Source

Traffic manipulation is a band-aid solution to the leak, which stems from the design of ABR streaming clients. We could pad segments on the server side using a padding scheme [18], but due to the already large size of video segments, excessive padding could negate many of the advantages of using variable bitrate over constant bitrate in the first place. These advantages include reducing bandwidth costs and maintaining QoE, which are top priorities for content providers [6]. Since ABR protocols are inherently client-driven, we believe that the largest part of the prevention strategy should be implemented at the client level, perhaps in conjunction with variations of previously mentioned approaches. The remainder of this section discusses the cause of the leak and proposes a number of conditions to mitigate it.

Buffer management. In this paper, we assume that one burst, excluding audio and other noise, corresponds to one video segment during the streaming steady state, meaning the buffer is full and network conditions are stable. This reflects a buffer strategy of a request being triggered when the buffer falls below a predefined threshold, typically a segment length, and seems to be the most common strategy from our observations. Consequently, this means that all network traces for a representation playback during the steady state, have a single permutation, or mapping, to its fingerprint and lets us build a single k-d tree to find candidates for all such videos. A different strategy employed by some services is the use of shorter segments where the client requests multiple ones in quick succession. This can be beneficial for latency assuming a whole segment needs to be decoded to initiate playback, but has the drawback of increased load on servers because of the higher request frequency. Here, the requested segments in each "on period" depend on when the steady state is reached. This raises the question if the attacker can deal with a variable number of segments interleaved in a single burst.

HTTP. To address this, we assume a network attacker that can observe the transport layer, albeit TLS-protected, conversation between the target and the streaming service, i.e. the model of many previous works [7, 9, 13, 24, 30]. The information that can be dissected from the multiple segments in the same on period depend on the HTTP version. It is known that HTTP/1.1 suffers from head-of-line blocking, and to overcome this, simultaneous requests for resources use multiple TCP connections. Consequently, if the client uses HTTP/1.1, the attacker can derive the individual segment sizes from the different connections. HTTP/2 offers multiplexing at the HTTP level, meaning resources can share a single TCP connection. HTTP/3 further improves on this, addressing some shortcomings of TCP itself and employs the UDP-based QUIC protocol instead. The use of HTTP/2 and HTTP/3 thus poses challenges to the attacker when multiple segments are requested, in that deriving their individual sizes from a single burst will be difficult. In its current state, the attack outlined in this paper would not work on a streaming service that employs a buffer management strategy of requesting a variable number of segments in each on period. However, as we discuss next, this is not a general countermeasure to the vulnerability and we do not recommend settling for such a solution.

Timing considerations. In this work, as in previous fingerprinting works, the variable bitrate encoded segment sizes have been the focus. Setting aside concerns about inefficiency, consider an encoding strategy where each representation is divided into segments of equal size. The observed bursts for all videos will thus be similar in size, effectively stopping known fingerprinting methods. However, this approach would only shift the focus of the attack to the time domain, since the segment durations would now vary significantly. Here, the attacker could use the time between bursts, and the fingerprint would comprise the segment durations. We note that this is possible because the buffer will, during the steady state, try

to remain constant. This is generally true and not restricted to equal sized segments. We believe that the buffer striving to stay at a constant size during the steady state is the *root cause* of the vulnerability and that there is, with this assumption, potential for a method that is also agnostic to the buffer strategy. For example, such a solution would incorporate both the segment sizes and the segment durations for a generalized fingerprint. This remains to be evaluated by future work, but as a proactive measure, we suggest the use of a *randomized* buffer. In this scenario, the buffer could be configured to maintain a capacity between a predefined lower bound, e.g. the previous max size of the buffer, and an upper bound. The buffer would then be allowed to deplete to a random level within this range before the next request phase is triggered which requests between 1 and N segments, ensuring that the number of segments requested does not cause the buffer to exceed its upper bound. This layer of randomness would challenge any fingerprinting system derived from constant features.

Summary. To mitigate the attack, we propose three conditions that should all be satisfied by a ABR client implementation: **(1)** a buffer strategy where multiple segments may be requested simultaneously, **(2)** use of HTTP/2 or higher, and **(3)** the use of a randomized buffer. The effectiveness of this approach, potentially in combination with other methods, and its impact on QoE remain areas for future research.

6 Conclusion

We have presented a concerning narrative about the video streaming landscape, demonstrating that the ABR leak is more extensive than previously documented. Our system shows remarkable accuracy and precision even with a massive set of monitored videos and would in its current state be practical to deploy in smaller networks, leveraging our efficient data collection method to keep the datasets fresh. Also, its portability to programmable network devices could be explored to evaluate its scalability at larger scales, such as ISP-level deployments.

The privacy implications of this work are very substantial, extending beyond mere privacy invasion to potential surveillance, censorship or discriminatory treatment. We hope our work raises public awareness about the vulnerabilities in video streaming, encouraging informed consumer choices. We also release our tools and datasets to support ongoing research in this domain. At last, we would like to call on streaming developers to take our proposed mitigation strategy into account and develop solutions promptly to address the leak.

7 Ethics Considerations

This work highlights a significant privacy issue with potential implications for covert surveillance, where individuals could be monitored without their knowledge. A key concern is whether publicizing this vulnerability does more harm than good. However, exposing weaknesses in widely used systems is a crucial step toward improving security. Identifying and addressing vulnerabilities is a fundamental principle in network security, allowing the community to develop stronger defenses. Importantly, this is not a newly discovered or zero-day vulnerability. Rather, it builds on a known issue, now shown to be more severe than previously understood. The ABR leak has been documented since 2016 [22], and our findings suggest that, given its feasibility over large datasets, well-resourced actors could potentially already be exploiting it.

Beyond security research, this work also serves an educational purpose by raising awareness about the privacy risks in video streaming. Even if the issue is not immediately resolved, users and organizations can make more informed decisions when considering the privacy implications of streaming content online. Additionally, we discuss mitigation strategies, contributing to the broader effort to address these vulnerabilities. Striking a balance between responsible disclosure and the potential risks of misuse is always important. However, by adhering to ethical disclosure practices and prioritizing public awareness, this work ultimately contributes to strengthening digital privacy and network security.

8 Open Science

All our tools are made publicly available to facilitate reproducibility and encourage a timely response from streaming services. Additionally, we are releasing our entire fingerprint dataset—the largest of its kind ever published, exceeding previous datasets by at least an order of magnitude. We hope this will not only foster research into mitigation strategies for the demonstrated attack, with reproducible results, but also stimulate further studies on network traffic analysis, particularly in video streaming. All artifacts, available on Zenodo as referenced earlier, include:

- TSV datasets containing video metadata and fingerprints from Amazon, Max, and SVT;
- PCAP capture files from the main evaluation, the unknown traffic experiment, and DAITA testing;
- karl, our data collection tool for retrieving fingerprints;
- BurstShark, our TShark wrapper tool for network traffic capture;
- Models implementing the video identification logic;
- An evaluation script to reproduce the main results;
- Miscellaneous scripts, e.g., the automated WebDriver streaming script.

References

[1] ISO/IEC 23000-19:2024. Multimedia application format (mpeg-a) — part 19: Common media application format (cmaf) for segmented media. Standard, International Organization for Standardization, February 2024.

[2] ISO/IEC 23001-7:2023. Mpeg systems technologies — part 7: Common encryption in iso base media file format files. Standard, International Organization for Standardization, August 2023.

[3] ISO/IEC 23009-1:2022. Dynamic adaptive streaming over http (dash) — part 1: Media presentation description and segment formats. Standard, International Organization for Standardization, August 2022.

[4] Waleed Afandi, Syed Muhammad Ammar Hassan Bukhari, Muhammad US Khan, Tahir Maqsood, and Samee U Khan. Fingerprinting technique for youtube videos identification in network traffic. *IEEE Access*, 10:76731–76741, 2022.

[5] Sangwook Bae, Mincheol Son, Dongkwan Kim, CheolJun Park, Jiho Lee, Sooel Son, and Yongdae Kim. Watching the watchers: Practical video identification attack in LTE networks. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1307–1324, Boston, MA, August 2022. USENIX Association.

[6] Bitmovin. The 7th annual video developer report. Technical report, Bitmovin, January 2024.

[7] Martin Björklund, Marcus Julin, Philip Antonsson, Andreas Stenwreth, Malte Åkvist, Tobias Hjalmarsson, and Romaric Duvignau. I see what you're watching on your streaming service: Fast identification of dash encrypted network traces. In *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, pages 1116–1122. IEEE, 2023.

[8] Meijie Du, Minchao Xu, Kedong Liu, Weitao Tang, Lijuan Zheng, and Qingyun Liu. Long-short terms frequency: A method for encrypted video streaming identification. In *2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 1581–1588. IEEE, 2023.

[9] Ran Dubin, Amit Dvir, Ofir Pele, and Ofer Hadar. I know what you saw last minute—encrypted http adaptive video streaming title classification. *IEEE Transactions on Information Forensics and Security*, 12(12):3039–3049, 2017.

[10] Romaric Duvignau. Metainformation extraction from encrypted streaming video packet traces. In *Proc. of the International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 1–6. IEEE, 2022.

[11] DASH Industry Forum. Dash-if implementation guidelines: restricted timing model. Technical report, DASH Industry Forum, June 2020.

[12] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. Wfdefproxy: Real world implementation and evaluation of website fingerprinting defenses. *IEEE Transactions on Information Forensics and Security*, 19:1357–1371, 2024.

[13] Jiaxi Gu, Jiliang Wang, Zhiwen Yu, and Kele Shen. Traffic-based side-channel attack in video streaming. *IEEE/ACM Transactions on Networking*, 27(3):972–985, 2019.

[14] Steven Jorgensen, John Holodnak, Jensen Dempsey, Karla de Souza, Anandithа Raghunath, Vernon Rivet, Noah DeMoes, Andrés Alejos, and Allan Wollaber. Extensible machine learning for encrypted network traffic application labeling via uncertainty quantification. *IEEE Transactions on Artificial Intelligence*, 5(1):420–433, 2024.

[15] Muhammad US Khan, Syed MAH Bukhari, Tahir Maqsood, Muhammad AB Fayyaz, Darren Dancey, and Raheel Nawaz. Scnn-attack: A side-channel attack to identify youtube videos in a vpn and non-vpn network traffic. *Electronics*, 11(3):350, 2022.

[16] Ryan McGrady, Kevin Zheng, Rebecca Curran, Jason Baumgartner, and Ethan Zuckerman. Dialing for videos: A random sample of youtube. *Journal of Quantitative Description: Digital Media*, 3, Dec. 2023.

[17] Mullvad. Introducing defense against ai-guided traffic analysis (daita), May 2024. https://mullvad.net/en/blog/introducing-defense-against-ai-guided-traffic-analysis-daita.

[18] Kirill Nikitin, Ludovic Barman, Wouter Lueks, Matthew Underwood, Jean-Pierre Hubaux, and Bryan Ford. Reducing metadata leakage from encrypted files and communication with purbs. *Proceedings on Privacy Enhancing Technologies*, 2019(4):6–33, 2019.

[19] Roger Pantos and William May. Http live streaming. RFC 8216, RFC Editor, August 2017.

[20] Tobias Pulls. Evaluating using the first eight daita servers, June 2024. https://pulls.name/blog/2024-06-05-eval-first-daita-servers.

[21] Tobias Pulls and Ethan Witwer. Maybenot: A framework for traffic analysis defenses. In *Proceedings of the 22nd Workshop on Privacy in the Electronic Society*, WPES '23, page 75–89. ACM, 2023.

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

[23] Andrew Reed and Michael Kranch. Identifying https-protected netflix videos in real-time. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (Codaspy)*, pages 361–368, 2017.

[24] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1357–1374, 2017.

[25] Jean-Pierre Smith, Luca Dolfi, Prateek Mittal, and Adrian Perrig. QCSD: A QUIC Client-Side Website-Fingerprinting defence framework. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 771–789, Boston, MA, August 2022. USENIX Association.

[26] Iraj Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE multimedia*, 18(4):62–67, 2011.

[27] Weitao Tang, Meijie Du, Zhao Li, Shu Li, Zhou Zhou, and Qingyun Liu. Shrink: Identification of encrypted video traffic based on quic. In *2023 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 140–149. IEEE, 2023.

[28] Weitao Tang, Jianqiang Li, Meijie Du, Die Hu, and Qingyun Liu. Zenith: Real-time identification of DASH encrypted video traffic with distortion. In *ACM Multimedia 2024*, page to appear, 2024.

[29] Mingkai Wang, Zengkun Xie, Xiangdong Tang, and Fei Chen. Noise-resistant video channel identification. *Security and Communication Networks*, 2022.

[30] Hua Wu, Zhenhua Yu, Guang Cheng, and Shuyi Guo. Identification of encrypted video streaming based on differential fingerprints. In *IEEE Conference on Computer Commun. Workshops (INFOCOM WKSHPS)*, pages 74–79, 2020.

[31] Luming Yang, Shaojing Fu, Yuchuan Luo, and Jiangyong Shi. Markov probability fingerprints: A method for identifying encrypted video traffic. In *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, pages 283–290. IEEE, 2020.

[32] Xiyuan Zhang, Gang Xiong, Zhen Li, Chen Yang, Xinjie Lin, Gaopeng Gou, and Binxing Fang. Traffic spills the beans: A robust video identification attack against youtube. *Computers & Security*, 137:103623, 2024.

A Test Environment

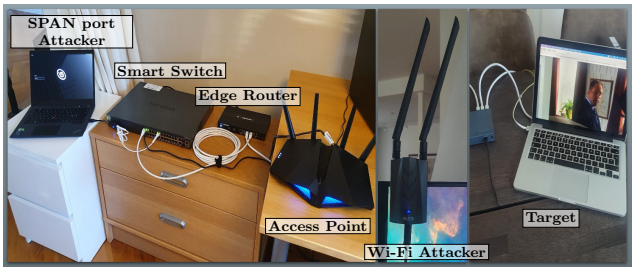

Figure 9: Deployed experimental setting.

Hardware involved in our experimental setup are displayed in Figure 9.

B System Configuration and Parameter Tuning

Earlier k-d tree works [7, 22, 23] have used a value for k between 3 and 6, noting a trade-off between identification time and accuracy. It is known that k-d trees suffer in performance as k grows, a concept known as the *curse of dimensionality*. In this work, we did not have query points of sufficient quality to obtain accurate queries with a $k < 6$. We found that an 8-dimensional tree was more accurate than 6 and 7, and that identification time differences were negligible due to our method requiring more than a single point to make a prediction. The accuracy of the search was met with diminishing returns at $k > 8$, with cost in both query time and memory use. We note that our heuristic works by assuming points belonging to the representation we are searching for will, over time, appear more often and with a closer distance than other points. The previous k-d tree works only needed a single query point to surpass a set threshold based on the Pearson correlation with one of the returned points in order to accurately declare a match. In our method, the overlap between true and false points greatly reduced the effectiveness of the previous strategy, as demonstrated by our comparison presented in Appendix C.

We use a radius value r of 0.025 which can be thought of as a generous upper-bound, with most lookups being limited to the τ closest points, which we keep at 50. r and τ complement each other in that r limits the lookup in sparse regions of the tree, while τ limits more dense regions. For example, as noted in earlier k-d tree works [7, 23] some videos have periods of darkness, resulting in a sequence of equal sized segments. Our implementation is not as sensitive to such sequences due to requiring multiple points and prioritizing older scores, for which we use $w = 3.0$. This lessens the impact of false candidates appearing in clusters, as explained in § 3.4, and consequently improves accuracy (about ~3% in our parameter

Table 6: Number of query points and average elapsed time between them per application, from processing unknown traffic.

Application	Total capture time (h)	Number of generated query points	Average time between points (s)
HBO Max	68.8	52561	4.7
Netflix	1.3	314	14.9
Skype	123	0	–
SFTP,Rsync,SCP	14.8	514	104
SSH,RDP	127.4	126	3640
Vimeo	1.0	479	7.5
YouTube	151.5	58358	9.3
Zoiper	2.7	0	–
Total	490.5	112382	15.7

Table 7: Evaluation results using the method of Björklund *et al.* [7]. Testing multiple suggested settings.

w	k	Pearson Thr.	Id.	Unk.	Mis.	Prec.	Rec.
5	5	1 - 1e-5	8	86	746	0.011	0.009
12	3	1 - 1e-2	2	783	55	0.036	0.002
12	4	1 - 1e-2	7	682	151	0.046	0.008
12	6	1 - 1e-2	49	307	484	0.092	0.058

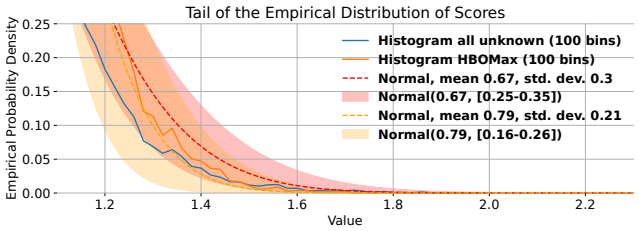

Figure 10: Empirical maximum scores over unknown traffic, tail of the empirical distributions.

tuning evaluation), identification time and the system's ability to retain its predictions, by allowing a lower threshold to be set. We use a lag of $L = 75$, ensuring that the system retains only the most recent 75 time steps without indefinitely consuming memory, and further increasing it had no significant impact on accuracy. For the base scores, we use $\lambda_{dist} = 100.0$, resulting in base scores close to 0 for points closer to r. Note the difference with tightening the r-value, since in that case the candidate would not be considered as present, while λ_{dist} allows us to both give low scores to false candidates, while simultaneously accounting for previous candidates appearing again but with a bit more noise. We set $\lambda_{score} = 0.9$, setting a lower value technically allows the system to adapt faster to videos changing, but consider that a sequence of $k+1$, i.e 9 bursts are required at minimum to initially predict a new video. With the suggested parameters, even when the system is the most confident, the score will converge around 5.0, allowing it to drop to approximately 2.2 during the transition period. This can be visualized in Figure 5 presenting score over time, where the threshold θ is also set to 2.2. Uncertainty due to noise is characterized by the score dropping, before the system sees the candidate present again.

C Comparison with Previous Method

Among the previous works summarized in Table 1, many are hard to properly reproduce in our setting (e.g. require transport details). Machine-learning based approaches require multiple streams of the same videos, something we cannot do given the size of our monitored videos set. The majority of works also lack open-source implementations and public datasets. However, we can do a reasonable comparison with previous works based on the k-d tree. For this, we implemented the conclusive heuristic used by Björklund *et al.* [7], as they do not rely on TCP sequence and acknowledgement numbers used by Reed and Kranch [23]. Their method was demonstrated to be 99.5% accurate for a dataset of 20,000 videos, having full network-level access, and similarly to Reed and Kranch, use a dimensionality reduction technique for windows, condensing w-dimensional sequences to k-dimensional query points for the k-d tree to retrieve a set of candidates that are subsequently compared with the query point using the Pearson correlation coefficient and declaring a match if it surpasses a predefined threshold. We evaluate the method using our VPN streams for Amazon, Max and SVT. Table 7 demonstrates the difference in the quality of the input data in our attack model, reaching 0.058 recall in the best case, using their proposed settings.

D Empirical Scores for Unknown Traffic

Table 6 summarizes the selected *unknown traffic* dataset covering 490 hours of captured traces. For each application, total duration of the captures is given as well as number of generated query points by our packet capturing tool and the average time between query points. Figure 10 is a zoomed-in version of Figure 7 focusing on the scores above 1.1. It highlights that, following the distribution of max scores obtained over 500 hours of unknown network capture of various applications, the probability that a false positive is generated by our system (with the score threshold set to $\theta = 2.2$) is very low.