Dodge: A Client-Side Framework for Application-Layer Video Fingerprinting Defenses

Ethan Witwer
Linköping University
ethan.witwer@liu.se

David Hasselquist
Linköping University
Sectra Communications
david.hasselquist@liu.se

Tobias Pulls
Karlstad University
tobias.pulls@kau.se

Niklas Carlsson
Linköping University
niklas.carlsson@liu.se

Abstract

As reliance on online video continues to increase throughout all facets of society, it is critical to address the security and privacy threat of video fingerprinting, in which a local, passive adversary monitors a victim's encrypted connection to a video server to infer which videos they are watching. These attacks attain high accuracy in realistic scenarios, while defenses that offer an acceptable trade-off between protection, overhead, and user experience are lacking. In this paper, we motivate *application-layer defenses* against video fingerprinting and present Dodge, a *client-side framework* for application-layer video fingerprinting defenses, implemented as a fork of the `dash.js` video player. Dodge provides the infrastructure and building blocks for defenses as well as a plug-and-play interface, making defense development and use straightforward while still providing maximal control over video flows. As a proof of concept, we use Dodge to implement a mimicry defense and show through live deployments that Dodge and the defense operate seamlessly, with modest overhead and very low user experience impact, while reducing attacker success close to theoretical bounds. Dodge can easily be deployed at scale and in a number of scenarios, with no changes to servers or network components; our analyses also lead to a host of insights that we hope will aid in deployment efforts.

Keywords

traffic analysis, video fingerprinting, framework, defenses

1 Introduction

With video already constituting a majority of the traffic sent over the Internet every day, both video-on-demand and live streaming continue to increase in popularity and use [50, 70]. At the same time, despite most video traffic being encrypted, it has been shown that *video fingerprinting* attacks can be used to identify which videos a user is watching with alarmingly high accuracy. These attacks are straightforward and can be carried out by a weak local, passive adversary that need only observe a user's network traffic. Moreover, a plethora of such attacks have been proposed [6, 10, 13, 24, 25, 31, 40, 65, 66, 71, 90, 91, 94]. In contrast, relatively few studies [32, 83, 93] have focused on potential defenses, leaving an urgent and substantial gap between demonstrated attack capabilities and practical, deployable defenses.

Many suggestions to defend against video fingerprinting are incidental and would require significant implementation efforts while leading to unacceptable performance degradation. For example, constraining or moving away from the variable bitrate encoding typically used to store videos would degrade performance markedly while requiring undue effort on the part of content providers, as well as increased storage and bandwidth costs [65, 71].

Recently, a few defenses that specifically target video fingerprinting have been proposed [32, 61, 83, 93]. Unfortunately, though, these systems come with their own pitfalls, including deployment challenges [83, 93], high overheads [32, 93], unclear effects on user experience [83, 93], and the inability to protect all videos [32]. A practical solution should be usable and effective for privacy-conscious users while remaining manageable for content providers with limited resources. Recognizing this need and the promise of *application-layer defenses*, we develop Dodge, a *client-side framework* with the building blocks for a wide range of video fingerprinting defenses, eliminating the need for defense designers to consider many implementation and integration details, bind applications to a specific defense, or acquire support from servers. We implement Dodge as a fork of the widely deployed `dash.js` video player [20, 76].

Dodge represents a substantial advancement of the state of the art of video fingerprinting defenses and a major step towards practical deployment. Its generalized model of DASH streaming gives defense designers ***precise control over video flows***. Furthermore, thanks to Dodge's simple domain-specific language for expressing defenses, it is both ***quick and easy to implement defenses***. Once a defense has been written, it can be passed to the Dodge player via a single input parameter, in a ***plug-and-play*** fashion, by a web developer integrating the player or the user (depending on the application). With Dodge, we provide a full implementation that can be deployed in place of `dash.js` by simply replacing a JavaScript file, and a reference for safely implementing defenses in other players.

As a proof of concept, we implement `Dodge-mimic`, a mimicry defense expressed in Dodge's domain-specific language. It is based on ***precise, byte-level control over every part of video traffic*** and provides theoretical guarantees on protection. Thanks to these

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–16
© YYYY Copyright held by the owner/author(s).
https://doi.org/XXXXXXX.XXXXXXX

properties, our evaluations show that Dodge provides the granular level of control needed to enact challenging defenses successfully and that Dodge supports effective and efficient defenses: it reduces classifier accuracy close to the theoretical bounds of `Dodge-mimic`, with modest overhead and ***very small impacts on user experience***.

Besides being a practical tool and testbed for defenses, Dodge also leads to a number of important traffic analysis insights. We demonstrate how to construct a purely client-side framework that provides maximal control over HTTP/DASH traffic and that client-side application-layer defenses can achieve strong protection with moderate overhead, offering concrete evidence and insights regarding the viability of deploying defenses at the application layer. With `Dodge-mimic`, we demonstrate the promise of *traffic regularization*, while illustrating the importance of ensuring that defenses are fit for their deployment scenarios; and we discover compelling reasons for defense designers to consider defenses based on varying degrees of randomness, which can be effective in live deployments [61].

At a high level, our primary contributions are:

- Dodge, a framework + full implementation with the building blocks for video fingerprinting defenses, a simple domain-specific language, and a bare-bones interface (Section 3).
- `Dodge-mimic`, a mimicry defense + full implementation that demonstrates Dodge's capabilities and usage, and provides theoretical guarantees regarding protection (Section 4).
- A comprehensive evaluation with live deployments, showing that Dodge provides granular control over video flows and supports effective and efficient defenses (Section 5).
- Practical guidance on selecting applicable defense strategies, fitting defenses to their deployment scenarios, and avoiding external side channels in deployment (Section 6).

We begin by presenting key concepts used in Dodge and the observations that motivate Dodge's design (Section 2). Then, we describe the Dodge framework and all of the components that allow it to support effective and efficient defenses (Section 3). We proceed by describing our proof-of-concept mimicry defense, `Dodge-mimic`, expressed in Dodge's domain-specific language (Section 4). To show that Dodge supports effective and efficient defenses, we evaluate Dodge/`Dodge-mimic` through live deployments (Section 5). Later, we provide practical guidance and discuss lessons learned and real-world considerations (Section 6). We wrap up by presenting related work on traffic analysis (Section 7) and our conclusions (Section 8).

2 Background and Motivation

This work addresses the threat of **video fingerprinting**, a type of traffic analysis of encrypted video flows. In video fingerprinting, an adversary is positioned on the network path between a client and video server and monitors the encrypted connection, using the observed traffic to infer which video the user is watching. We follow prior studies in assuming a **local, passive adversary**: local since they directly observe the connection between the client and video server, passive because they only collect traffic and do not modify it in any way. In other words, the attacker is relatively weak and could be another user on the local network, an Internet service provider, or any other party along the network path between the client and server. This threat model is depicted in Figure 1.

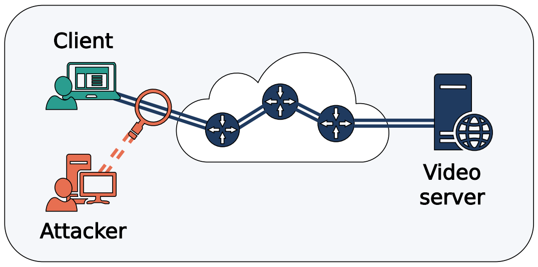


Figure 1: Video fingerprinting threat model.

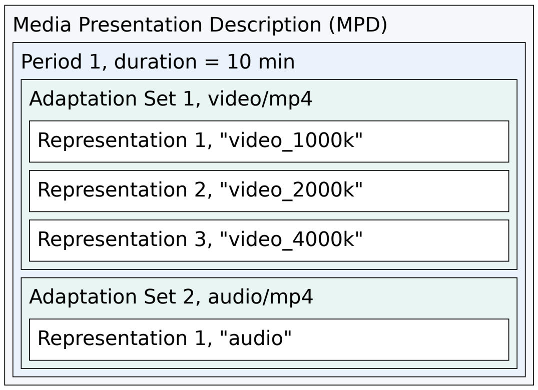

Media Presentation Description (MPD)
Period 1, duration = 10 min
Adaptation Set 1, video/mp4
Representation 1, "video_1000k"
Representation 2, "video_2000k"
Representation 3, "video_4000k"
Adaptation Set 2, audio/mp4
Representation 1, "audio"

Figure 2: Manifest structure in DASH streaming.

2.1 Video Traffic

On-demand video traffic may be best understood as follows:

(1) Videos are divided into **segments** of a fixed duration.
(2) Video players maintain a **playback buffer** with all of the segments received so far.
(3) Whenever the buffer is below a **target size**, segments are downloaded one-by-one.

From these key points, it is possible to deduce the two primary "phases" of video traffic: (1) the **startup phase** at the beginning of playback, in which back-to-back segment downloads occur to quickly fill the initially empty buffer; (2) **steady-state streaming**, in which new segments are downloaded periodically as the buffer is depleted (each time a segment duration has elapsed).

The most widespread streaming standard is **Dynamic Adaptive Streaming over HTTP (DASH)** [79]. As the name suggests, segments are made available on a web server for video players to download. This is done in a hierarchical fashion. A content provider wishing to host a video encodes and segments the video several times, resulting in a number of segmented **representations**, each with a different quality. This process is repeated for each version of the video (e.g., in different languages) and other media types (audio and subtitle text), producing **adaptation sets**, each consisting of a unique set of representations. Additionally, videos may be divided into **periods**, temporal sections that contain adaptation sets. This can, for example, be done for longer videos or to mark breakpoints for advertisements. This hierarchy is exemplified in Figure 2.

Each video's hierarchical structure is stored in a **Media Presentation Description (MPD)**, or **manifest**, an XML file that contains metadata on available periods, adaptation sets, representations, and segments. To play the video, a video player fetches the manifest, selects one or multiple adaptation sets/representations, and requests segments to reach a buffer target. Each media type has its own **stream** with independent segment downloading and buffer logic, though typically over the same connection. The player can switch representations dynamically to adapt to varying network conditions, a technique called **Adaptive Bitrate Streaming (ABR)**. Finally, a small **initialization segment** may be downloaded before initial playback and when switching qualities on a stream [1, 2].

Video traffic is vulnerable to traffic analysis because videos are usually encoded with **Variable Bitrate Encoding (VBR)**, in which an *average bitrate* is set for each representation, and the *true bitrate* (and, consequently, size) of every segment depends on its content. As such, each representation has a sequence of segments that is strongly correlated with its content: an adversary can monitor the network path between a client and server and use the observed segment sizes to determine which video the user is watching.

2.2 Attacks

Many video fingerprinting attacks [10, 24, 31, 65, 66, 90–92, 94] have been successful using basic heuristics and machine learning with sequences of segment sizes. Two notable examples are Walls Have Ears [31], which computes segment size *differentials* for monitored videos (via their manifests) and matches them against captured network traces; and a recent attack by Björklund and Duvignau [10] that employs a k-d tree and sliding window over segment size differentials to identify videos in real time with very high accuracy, even when the victim uses a VPN or traffic is snooped from the victim's wireless network, with link-layer encryption in place.

Other attacks [6, 13, 25, 40, 71] use deep learning: Beauty and the Burst [71] operates on a $3 \times N$ matrix, with rows representing the number of incoming packets (from the client's perspective), outgoing packets, and the sum of both in N 0.25-second windows; or other time series. Subsequent works [6, 40] have built on Beauty and the Burst to be effective even against snooped wireless traffic. vDF and vRF [13], which are adaptations of the website fingerprinting attacks Deep Fingerprinting (DF) [77] and Robust Fingerprinting (RF) [73], respectively, also use packet-level features and have impressive performance under a host of challenging conditions.

Despite these results, uncertainty as to the magnitude of the video fingerprinting threat remains: Walsh et al. [85, 86] show that, with deep learning attacks, false positives are a practical issue when considering large popular video platforms and Tor traffic. Though this implies that current techniques are limited in such settings, smaller streaming services remain extremely vulnerable; in fact, their entire catalogs can be successfully fingerprinted [10].

2.3 Defenses

A few defenses against video fingerprinting have been proposed, all of which directly modify the sequence of packets leveraged by attacks. Zhang et al. [93] consider adversarial machine learning and differential privacy, finding that differential privacy is effective but requires prior knowledge of network traces and high **overhead** (additional bandwidth and latency cost), with a rigorous user experience analysis and large-scale evaluation of its protection lacking. Vaskevich et al. [83] use GANs to generate network traces and mold flows to match them, resulting in good protection but with the need to train a GAN on the server, potentially share the GAN with the client via some mechanism, and unclear user experience impacts. Hasselquist et al. [32] adapt two website fingerprinting defenses to video traffic and develop a new defense, Scrambler, which achieves good protection in most cases but depends on stable network conditions and fails to protect videos with large segments.

Note that video streaming, like other real-time services, entails strict user experience demands. **Quality of Experience (QoE)** encompasses the user's subjective experience of video playback, which has been shown to depend on several objective factors, including player metrics such as stalls and quality switches [14, 69**?**]. Hasselquist et al. [32] evaluate the QoE impact of the three defenses they implement, showing in a systematic way for the first time that defenses can have a major effect on QoE – this cannot be ignored by practical defense proposals. Hasselquist et al.'s evaluation lays the groundwork for our consideration of QoE in this work.

2.4 HTTP Concepts

As mentioned, DASH traffic includes a manifest and segments requested and sent via HTTP. While lower-layer protocols have, generally speaking, fixed-size headers (whose sizes are not correlated to content), HTTP is a text-based protocol with an arbitrary number of key-value pairs as headers. This has a number of implications for designing application-layer defenses for video.

Content providers can choose to host each segment separately (in separate files, with unique paths) or contiguously (as one big file), in which case a video player uses the `Range` header to request a specific byte range from the server. The server's response will then contain a partial resource rather than a full resource.

In HTTP/1.1, headers are transmitted as plaintext key-value pairs (above TCP/TLS). In an HTTP *response*, from server to client, some headers of interest for defense designers include [46]:

- `Content-Length`, always transmitted. The size of the partial or full resource being sent from server to client.
- `Content-Range`, transmitted in reply to range requests. The byte range being sent and the resource's total size.
- `ETag` (entity tag), optional. A unique identifier assigned to each resource by the server for caching purposes.

These headers are of interest because the number of characters they contain (and, thus, the total size of the response) depends on the request parameters, resource characteristics, and server.

While HTTP/2 uses the same header format as HTTP/1.1, it also uses a tailored header compression algorithm, HPACK [55], to compress request and response headers. Compression involves first consulting a static dictionary with common headers, then a dynamic dictionary that is filled and updated as data is sent on a connection, and Huffman encoding as a "last resort". HTTP/3 also has a dedicated compression algorithm, QPACK [39], based on similar principles and designed to work efficiently with QUIC.

2.5 Dodge's Motivation

Existing video fingerprinting defenses are **network-layer defenses**, which operate below the application layer, directly modifying the

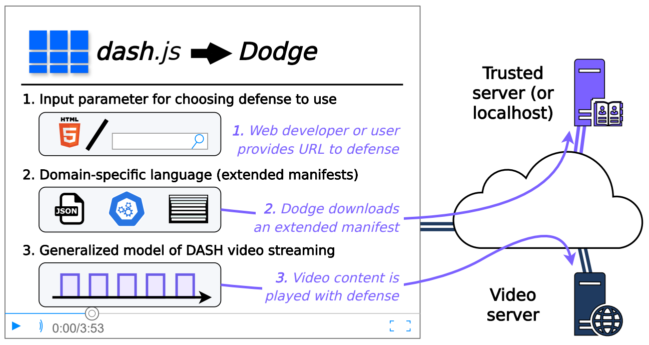

dash.js → Dodge
1. Input parameter for choosing defense to use
1. Web developer or user provides URL to defense
2. Domain-specific language (extended manifests)
2. Dodge downloads an extended manifest
3. Generalized model of DASH video streaming
3. Video content is played with defense
Trusted server (or localhost)
Video server
0:00/3:53

Figure 3: Overview of Dodge's functionality (c.f. Figure 1).

flow of packets rather than higher-level semantics. This has benefits including granular control over traffic and generalizability to different video players, but it does not allow direct access to or modification of the fundamental elements of video traffic. **Application-layer defenses**, on the other hand, stand to achieve better protection as well as lower overhead and QoE impacts by taking advantage of the semantics of video traffic (Section 2.1).

Another key observation is that **hard-coded defenses** limit applications to one defense technique, which can be obsoleted as more effective and efficient defenses are built or defeated as attacks improve, leading to disuse; this has occurred in other traffic analysis domains [57, 89]. Some authors have thus proposed **defense frameworks** that provide the *building blocks* for many different types of defenses: these are essentially modules or libraries that handle the details of integrating defenses with applications. Instead of focusing on implementation details, defense designers can use a simpler domain-specific language to specify defenses, feeding them into a defense framework in a plug-and-play fashion [28, 57, 62, 78]. This means that no software changes are required if defenses are defeated or different defenses are needed for a particular purpose (research, development, different threat models, etc.). This approach has been adopted by Tor [45, 57] and Mullvad VPN [48, 61].

Finally, many defenses require **server-side modifications**, as traffic must be shaped in both directions (from client to server and vice versa) to protect against attacks. This may include integrating a defense framework [32], deploying GANs and traffic shapers [83], or even reencoding videos [25, 71, 94]. This represents laborious and costly efforts for content providers, which may not be motivated by available resources or incentives. Such issues can be avoided by **client-side defenses**, which only operate on the client but modify traffic in both directions through, e.g., protocol features.

3 Dodge Framework

We combine the benefits of application-layer defenses, the robustness of defense frameworks, and the practicality of client-side defenses in Dodge, a client-side framework for application-layer defenses against video fingerprinting. Our implementation of Dodge has its foundation in the `dash.js` video player [20], a JavaScript implementation of the DASH standard based on best practices in DASH streaming and state-of-the-art ABR algorithms. At a high level, Dodge introduces:

- A ***generalized model*** of video streaming that provides the building blocks for many different types of defenses.
- A minimal ***domain-specific language*** for both easily and intuitively expressing defenses within this new model.
- A single ***input parameter***, available to web developers or users (depending on the application using the Dodge player), who want to employ video fingerprinting defenses.

`dash.js` is widely used on the modern Internet, including by many prominent content providers [4, 5, 19, 76]. This affords Dodge ***excellent potential for real-world deployment***: it could, for example, simply be pushed as a major update of `dash.js` and put into use immediately. Dodge has no additional dependencies beyond those already present in `dash.js` and imposes no special requirements on network infrastructure or video servers. In practice, Dodge handles the logic of enacting defenses and all low-level details, while defense designers express defenses in its domain-specific language and web developers/users employ the *same interface* from vanilla `dash.js` with a different manifest URL (see Figure 3).

In the remainder of this section, we provide a high-level overview of Dodge. We begin with an overview of our most significant changes to `dash.js` (Sections 3.1 and 3.2), later addressing a number of smaller changes that are just as crucial as well as considerations for defense designers (Sections 3.3 and 3.4). Along the way, we describe how Dodge fulfills its primary design goals: maximizing deployment potential by supporting many different defense types and deployment scenarios; making defense implementation simple and straightforward; and retaining a cyclic traffic pattern (defenses that attempt to conceal the mere fact that a video is being watched—that is, the protocol/service in use—are out of scope).

3.1 Generalized Video Streaming

At the core of Dodge, we introduce a generalization of the typical pattern of video traffic, as depicted in Figure 4. Typically, video streams are made up of segment downloads that occur at regular intervals during steady-state playback (left); in Dodge, these are replaced by **cycles** (right), which are defined by a segment `index`, optional `range` (as in the HTTP Range header), and optional `padding` and `buffer` flags. A stream in Dodge thus consists of cycles that are downloaded in a controlled way.

The primary difference between undefended DASH streams and Dodge streams is that partial segments can be downloaded (`range`) instead of full segments, affording defense designers precise control over each of the resource requests that comprise a video stream. Additionally, full or partial segments can be downloaded and discarded (`padding`). To combine these capabilities in a seamless and flexible way, Dodge passes data to the video player's buffer only when a cycle has the `buffer` flag set: once such a cycle is encountered, all pending segments are buffered. This allows defense designers to directly and explicitly manage the size of the buffer and affect request timing, two key capabilities when crafting defenses.

Moreover, this design permits Dodge streams to maintain the general characteristics of DASH, allowing network operators to determine that the traffic carries a video stream. Additionally, Dodge's design affords defense designers full control over request sequences and buffer behavior with minimal changes to `dash.js` and thereby facilitates low-overhead implementation of defenses.

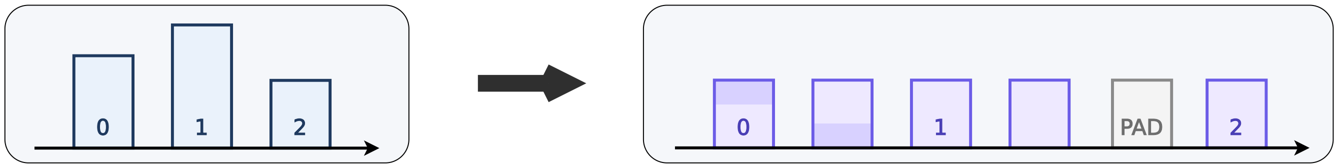


Figure 4: Example of segment downloads (left) replaced by cycles (right). Cycles can be full segment downloads (segment 2), partial segment downloads (1 and 0, where overlap between two cycles is indicated with shading), or padding.

```
{
  "start": {
    "mpd": "<MPD xmlns='urn:mpeg:dash:schema:mpd [...]",
    "base_uri": "https://example.com/segments/"
  },
  "streams": [
    {
      "label": "video_1000k",
      "init": [ /* init segment cycles */
        {"range": "-855"},
        {"range": "44-898"}
      ],
      "data": [ /* media segment cycles */
        {"index": 0, "range": "-43999"},
        {"index": 0, "range": "16000-", "buffer": true},
        {"index": 1, "range": "-43999"},
        {"index": 1, "range": "44000-", "buffer": true},
        {"index": 5, "range": "-43999", "padding": true},
        {"index": 2, "buffer": true}
      ]
    }
  ]
}
```

Figure 5: Sample extended manifest corresponding to the scenario in Figure 4. Here, segment index 5 is used for padding.

3.2 Domain-Specific Language

The cycles that comprise a video stream are specified in an **extended manifest**, a JSON file constrained by a schema that is used in place of the DASH manifest. The extended manifest contains the original `mpd` for the video; a `base_uri` to be used for segment downloads, in most cases pointing to the location of the MPD; and `streams`, an array of objects which specify how media content will be downloaded via sequences of cycles. An example corresponding to the video stream in Figure 4 is shown in Figure 5.

The heart of the extended manifest is the `streams` array, which contains **defended stream info** for every *representation*. It is of particular interest that this implies all media types: video streams typically do not come alone; rather, audio and subtitle text are downloaded simultaneously over the same connection. In the interest of promoting comprehensive defenses that provide useful protection in practice, Dodge will not download audio or fragmented subtitle text if defended stream info is not specified for them.

Each defended stream info object has a `label`, which must be a *representation ID* from the original manifest. In the `cycles` array, each entry is an object that represents a cycle, with the four components described above. Segments are indexed from zero, and partial downloads of the same segment are allowed to overlap. The `range` flag can be omitted to download a full segment directly, and the `padding` and `buffer` flags may be omitted rather than set to false. During playback of the relevant representation, the cycles specified here are downloaded sequentially; the data is placed in an intermediate buffer, which is emptied into the video player's buffer as soon as a cycle with the `buffer` flag has been downloaded. Similarly, in the `init` array, cycles specify how the initialization segment for the representation should be downloaded. These cycles, in contrast to cycles for media content, do not need a segment index.

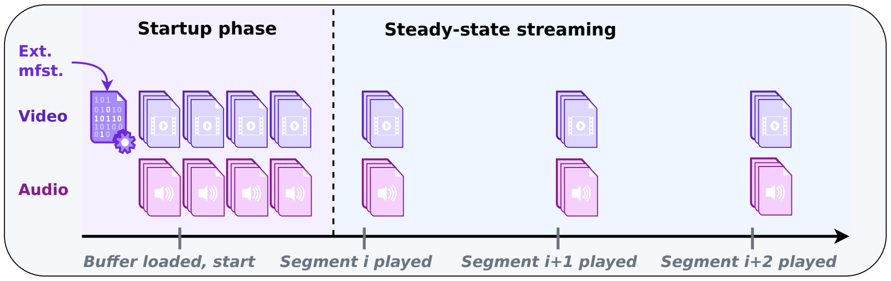
This design has implications for the integration of the Dodge player in a web page. We have opted not to modify the `dash.js` interface in any way; instead, the URL of the extended manifest is used in place of the original manifest. Since the extended manifest explicitly specifies a base URI for segment downloads, it may be hosted anywhere: on the video server, an external trusted server, or locally (accessed via a loopback address – no support is required from servers). This results in significant flexibility and deployment potential in a variety of real-world scenarios (see Section 6).

3.3 Eliminating Information Leakage

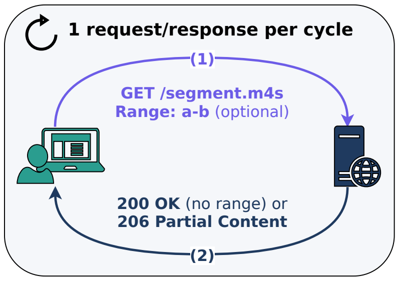
A number of subtle factors must be controlled to build an effective defense framework and ensure information correlated to video contents cannot be leaked in unexpected ways. These are accounted for during the development of Dodge to the extent possible, but some must be addressed by defense designers.

Defense designers must consider (1) extended manifest traffic, (2) which streams are included in an extended manifest and their durations, and (3) the effects of HTTP headers on server-side traffic. Here, we go through many additional factors that Dodge addresses. To concretize, we explain the role each factor plays in the design of our proof-of-concept defense and experiments. Figure 6 provides an informative overview of the traffic in a Dodge session that aids in visualizing the importance of many of these factors.

3.3.1 Extended manifest traffic. If extended manifests are hosted locally and requested via a loopback address, no specific measures are needed to protect them. However, if extended manifests are hosted alongside video content or on another remote server, they must be defended, as their on-wire sizes are visible to an attacker – this can be accomplished by padding them (by adding fields to some JSON object) to a selected size or a random size per download, which is the most practical option if HTTP content compression is in place. `Dodge-mimic` (Section 4) takes the former approach, padding all extended manifests in an anonymity set to the same size, as compression is disabled on our extended manifest server.



(a) Dodge session timeline



(b) Per-cycle HTTP exchange

Figure 6: Example timeline representing the traffic makeup of a typical Dodge session (and in our experiments). Each pair of ticks is separated by an MPD segment duration: resource downloads occur once a segment has been played. A stack of media file icons represents the cycles that must be downloaded before a segment is buffered (all cycles up until a `buffer` flag).

3.3.2 Adaptive bitrate streaming. It is important that ABR rules are applied to cycles, not the underlying segments, and in a video-independent way; otherwise, active attacks or fluctuations in network conditions could make it possible to identify videos via quality switching behavior. This is especially critical when traffic is strictly regulated: in `Dodge-mimic`, videos in an anonymity set have matching sequences of cycles at every quality, but quality switch dynamics could be used for identification if they were video dependent. In the initial release of Dodge, we only enable the primary video-on-demand rules in `dash.js`, throughput and BOLA [80], disabling optional supplementary rules. We apply these rules *when segments are added to the buffer* (after cycles with `buffer` flag).

3.3.3 Media types beyond video. As noted in the previous subsection, each representation containing audio or subtitle text must have associated defended stream info to be requested by Dodge. Thus, they are protected in the same way as video content. `Dodge-mimic` illustrates how audio can be defended, as a vast number of videos (including all of the videos in our dataset) have accompanying audio tracks; fragmented subtitle text may be handled identically.

3.3.4 Stream duration. Streams may be identified based on their durations, even if the traffic leading up to the end of playback is otherwise identical. To account for this, **trailing cycles** (with the `padding` flag) can be included in the extended manifest, downloaded after all playable content. A **mock buffer** is used to time these requests; the `buffer` flag, when set in trailing cycles, increments it by one segment (if any real data is still pending, it is buffered, and the mock buffer is incremented by the segment duration specified in the manifest minus the amount of data actually buffered; this accounts for the variable duration of the last segment). A relatively large number of trailing cycles can be downloaded before playback ends, but excessively many may result in a wait time after playback. `Dodge-mimic` ensures videos in the same anonymity set have the same duration but avoids wait times by placing videos with similar durations in the same anonymity set. Other strategies could also be used, such as randomization of start times or durations.

3.3.5 Client-side traffic. This consists of HTTP `GET` requests for resources, whose timing is based on buffer sizes and thus controllable. However, the *sizes* of requests could leak information to an attacker if not regulated. In the initial release of Dodge, a configurable HTTP header is used to pad all resource requests to the same size. As a real-world example of the practicality of this approach, the popular content delivery network Akamai allows custom `hdntl` and `hdnts` headers in cross-origin requests [82]. If no header is available, the header setting in Dodge may be left empty to use a query string for padding instead. Note also that an `OPTIONS` request is sent before cross-origin `GET` requests. To pad these, Dodge adds a query parameter `padding` with a variable amount of random data to the resource paths for segments upon each request.

3.3.6 Server-side traffic. HTTP headers in the server's response can vary, as described in Section 2.4. Thus, to truly control response sizes, defense designers must also take expected headers into account. `Dodge-mimic` has cycles with ranges adjusted to account for differences in HTTP headers by default, resulting in precise control over the size of the server's response. However, the need for this level of granularity can be avoided through small random perturbations – we demonstrate that this works with HTTP/2, where header compression is a factor. Even defenses that are not based on randomization, such as `Dodge-mimic`, may simply add or subtract some random amount of bytes from desired ranges per extended manifest download, among other possible approaches.

3.3.7 Scheduling logic. In `dash.js`, segments are downloaded *one-by-one* when the buffer level is below a target value. From an implementation perspective, this involves an event that is fired when a segment download is completed, which results in the segment being added to the video player's buffer, some other relevant operations and checks being performed, and *subsequently* a new request being made. In Dodge, the tasks performed on download completion vary in complexity depending on cycle characteristics.

There are several different cases when a cycle download is finished: the cycle may have been padding and the data should be discarded, it may have been a partial segment which should be stored, or a `buffer` directive may have specified that all pending segments should be buffered. This results in a classic timing side channel: differences in the code execution time of these tasks affect when the *next* request is scheduled, which can result in large video-dependent cascading effects on request timing.

One way to account for this would be to have strict clock-based scheduling logic, but we consider this solution inadequate due to its complexity in the face of volatile bandwidth conditions and

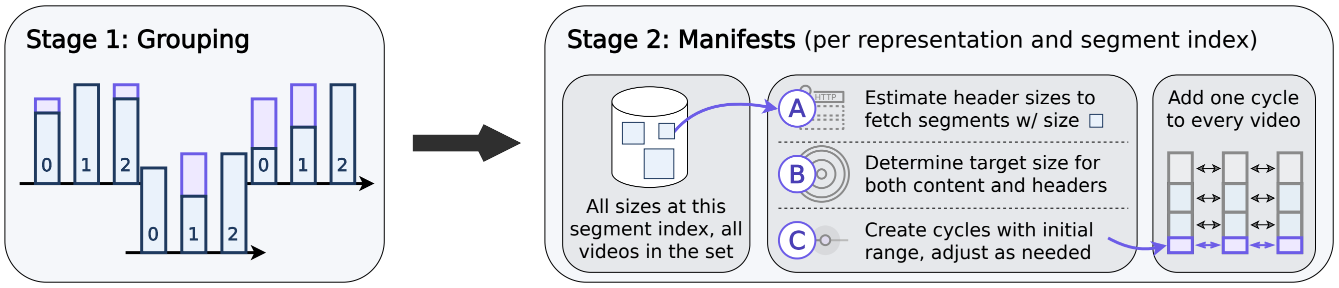

Stage 1: Grouping
0
1
2
0
1
2
0
1
2
Stage 2: Manifests (per representation and segment index)
All sizes at this segment index, all videos in the set
A
Estimate header sizes to fetch segments w/ size □
B
Determine target size for both content and headers
C
Create cycles with initial range, adjust as needed
Add one cycle to every video

Figure 7: The full workflow of anonymity set and extended manifest generation in `Dodge-mimic`.

many quality switches/stalls. Instead, we make these differences in request timing a *random walk* by waiting for a random amount of time between each request, which decouples cycle type, segment size, and other such factors from request timing. We have found this to be a crucial contributing factor to Dodge's success.

It is also important to note that segment durations can deviate slightly, by fractions of a second, from the segment size specified in the manifest (and, as mentioned in Section 3.3.4, the last segment's duration can vary even more). To account for this, the mock buffer is used not only for trailing but through the entire playback session, with small updates to account for variable segment durations.

3.4 On User Behavior

We argue that variations in network traffic due to user actions are acceptable as long as they cannot be correlated to the video being watched. A significant challenge is present in the fact that user behavior, such as pausing and seeking, *is* correlated to video content. Consider YouTube's heatmaps [81]: when hovering the mouse cursor above the timeline for a video, "most replayed" moments are labeled, and these tend to be subjectively noteworthy parts of the video. Such human factors cannot be eliminated by defenses.

Assume briefly that user behavior *is not* correlated with video content in any way. In such a case, pausing results in a temporary lapse in the sequence of cycles, indicating to an attacker that the user has likely paused the video. Similarly, seeking causes multiple cycle downloads in a row, similar to when playback starts, and is most certainly detectable. However, these facts provide no *identifying* information, and the sequence of cycles is not impacted in a video-dependent way. Considering this, we posit that Dodge's design inherently accounts for user behavior to the greatest extent possible while still giving defense designers maximal control over the different components of video traffic and maintaining DASH's traffic characteristics. See Section 6 for further discussion.

4 A Mimicry Defense for Dodge

To demonstrate that Dodge provides granular control over video traffic patterns and supports practical defenses, we design a defense, `Dodge-mimic`, that creates explicit anonymity sets, based on the optimal k-anonymity algorithm by Bayardo and Agrawal [8]. Note that k-anonymity is vulnerable to several types of attacks [16]; nonetheless, k-anonymity allows us to derive theoretical bounds for an attacker that *only uses classification models* and show that defenses deployed in Dodge can reach them. `Dodge-mimic` has two stages (Figure 7): grouping and manifest generation.

4.1 Stage 1: Grouping

`Dodge-mimic` is based on an algorithm that groups and continuously regroups videos into anonymity sets, progressively generating better sets, to minimize a cost function computed per set. By default, the cost function is the overhead in bytes required to pad all corresponding segments (with the same index) to the size of the largest segment in the set at that index, the target size.

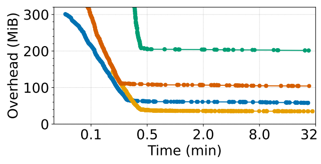
Consider the example anonymity set with three videos in Figure 7. The segment sizes at index zero are $\{500, 1000, 1200\}$ kB, so the cost function reflects the overhead needed to pad the first segment to 1200 kB for all videos. These calculations are done using a configurable video quality (the highest, by default), and streams for other representations are padded as needed, in the same way, to fit the generated anonymity sets. Note that any video quality could, in principle, be used to generate the same optimal anonymity sets, as segment size roughly doubles when bitrate doubles.

4.2 Stage 2: Extended Manifests

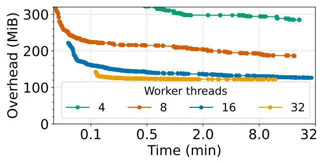
Once optimal anonymity sets have been found or the algorithm is stopped by the defender, extended manifests are created. For each anonymity set, the program loops through every representation and segment index, and a sequence of cycles is generated for all segments at the same index simultaneously. Cycles are constructed iteratively, with a size determined by removing the smallest value from a bucket containing all segment sizes in the anonymity set at the current segment index. This size is capped if it is greater than the size of the largest segment in any video in the anonymity set or the remaining data needed at this segment index.

Recall from Section 3.3 that both header and content sizes need to be considered when crafting resource downloads of identical size, as the values of HTTP headers may differ in the number of characters. Thus, once a size has been selected from the bucket and capped, the header sizes needed to download a partial segment of this size are computed for segments that still need content, and these are used to determine a target size for both content and headers. Cycles are then generated with ranges adjusted to match the target size.

After the manifest generation process, the sizes of corresponding cycles (cycle c in a given representation for different videos in an anonymity set), including both headers and content, are identical. This granular level of control is applicable to HTTP/1.1 (which does not employ header compression) and would result in different behavior if HTTP/2 or HTTP/3-QUIC were in use; we address this with random perturbations of cycle sizes in Section 5.4.



(a) k = 2



(b) k = 10

Figure 8: Mean per-video overhead of top quality video (`video_4000k`) with the best sets generated over time, shown for two anonymity set sizes and varying worker count.

5 Experiments: Live Deployment

In this section, we systematically demonstrate that Dodge is true to its intended behavior in practice and provides granular enough control over video traffic to implement defenses that achieve their theoretical guarantees. We do this by deploying `Dodge-mimic` in Dodge, which not only verifies Dodge but also results in a very reasonable trade-off between protection, overhead, and QoE.

We begin by generating anonymity sets of different sizes and with varying parameters (Section 5.1), highlighting performance aspects of `Dodge-mimic` and its protection/bandwidth trade-offs. We then use these sets to collect several datasets through ***systematic live deployment*** of Dodge. We collect a large number of traffic samples from videos in a *single anonymity set*, to show decisively that Dodge correctly applies the defense and that videos in the same anonymity set are virtually indistinguishable (Section 5.2). We then collect traffic and QoE metrics for *multiple anonymity sets*, further demonstrating Dodge's efficacy, and that `Dodge-mimic`'s protection scales with very small QoE impacts (Section 5.3). We do this with HTTP/1.1 and header adjustment; later, we show that Dodge and `Dodge-mimic` are just as effective with HTTP/2 (Section 5.4).

Our data collection setup mirrors prior work on video fingerprinting [13, 32]: we directly connect two physically adjacent machines with a 10 Gbps full-duplex link, to act as a client and server. Whenever a sample is to be collected, the *client*:

(1) Runs Chrome via Selenium [72] browser automation tool
(2) Loads a small, locally hosted web page that integrates Dodge
(3) Inputs the extended manifest URL (a loopback address that leads to a minimal Flask [53] server) via a text box
(4) Presses a button that results in the extended manifest being requested and content being fetched from the video server
(5) Collects network traffic (extended manifest and video/audio) and QoE metrics/player logs until playback is done

Caching is disabled in Chrome, and HTTP content compression is disabled on our Flask extended manifest server. We run up to 10 data collection sessions in parallel. The *server* machine has 10 virtual network namespaces, one for each simultaneous data collection session, and makes videos available via nginx [51]. Each namespace is limited to 100/100 Mbps throughput. We also add a 5 ms delay to each packet exiting a network namespace.

We choose videos via the YouTube Data API [30]. First, we collect over 100,000 video metadata objects from all *assignable* categories (excluding shorts, film-only categories, etc.). Then, for each of the 14 categories, we randomly select 100 videos between 5 and 7 minutes in length with a resolution of at least 720p, resulting in a total of 1,400 videos. We obtain the contents of these videos and encode them to create 4 representations with 2-second segments: 1000 kbps video (`video_1000k`), 2000 kbps video (`video_2000k`), 4000 kbps video (`video_4000k`), and `audio`. Finally, we generate static MPDs and host the videos on our nginx server.

Table 1: Overhead (%) of top quality video (`video_4000k`) with varying anonymity set size and thread count. These are the lowest overheads after 30 minutes, where ± indicates differences when protecting the other qualities in our datasets.

		Worker threads (W)			
		4	8	16	32
Set size (k)	2	120.8 (± 10.8)	62.5 (± 5.0)	34.9 (± 3.0)	21.0 (± 0.8)
	3	133.2 (± 8.0)	82.6 (± 6.0)	47.5 (± 3.5)	32.5 (± 1.3)
	4	143.7 (± 8.3)	91.9 (± 5.2)	54.6 (± 3.3)	41.2 (± 1.6)
	5	154.9 (± 5.5)	97.9 (± 5.8)	60.7 (± 3.8)	48.7 (± 2.0)
	10	170.9 (± 6.5)	111.6 (± 4.1)	75.8 (± 2.2)	73.1 (± 2.8)
	20	161.3 (± 4.1)	111.7 (± 0.9)	104.1 (± 2.5)	107.4 (± 1.5)

5.1 Anonymity Set Selection

Here, we use the `Dodge-mimic` defense code to generate anonymity sets. Recall that this is a live algorithm that progressively improves anonymity sets and terminates when the sets are optimal according to the cost function, which by default is based on the overall bandwidth overhead of the generated sets.

The primary tunable parameter in `Dodge-mimic` is k, the number of videos per anonymity set. This is (with the default cost function based on bandwidth overhead) the major determiner of the trade-off between protection and overhead. Since the videos in a set will be made to have identical traffic patterns, an attacker can determine which *anonymity set* a video belongs to by observing its traffic, but not the specific video: accuracy is $(100/k)\%$. As traffic patterns are made identical by downloading the amount of data in the largest segment at each index, it is more likely that efficient sets will be generated if k is small, while better protection can be attained with a higher k. Note that other cost functions could be used, resulting in a looser correlation between k and overhead/QoE.

We test values in the set $k \in \{2, 3, 4, 5, 10, 20\}$ with different numbers of worker threads $W \in \{4, 8, 16, 32\}$. To do this, we sort all 1 400 videos by their durations and run set generation separately in W threads, with at least $\lceil 1\,400/W \rceil$ videos per thread. This results in an additional trade-off between computation and overhead: multithreading eliminates many suboptimal sets from the search tree (much overhead is needed to pad a short video to the duration of a long one) but also potentially optimal sets, especially near thread boundaries (e.g., videos $\lceil 1\,400/W \rceil$ and $\lceil 1\,400/W \rceil + 1$).

We use a computer with an AMD Ryzen 9 7950X 16-Core Processor and 128 GiB RAM for evaluation. Set generation over 30 minutes' time is illustrated in Figure 8 with two representative values of k. Here, the x-axis is time since the algorithm starts, and the y-axis is the average per-video overhead (total overhead in bytes divided by number of videos in the dataset) of the best anonymity sets found, at the top video quality, with dots showing when new anonymity sets are found. Audio overhead is negligible.

A primary observation is that *overhead reduces very quickly* in all cases, meaning that little time is required to generate low-overhead anonymity sets. In Figure 8(b), average overhead reduces

Table 2: Accuracy (%, $\mu \pm \sigma$) of *vRF* when training and testing on videos in the same anonymity set.

Comb. size and target ↓	(a) **Video, one quality** — *A*	*B*	*C*	(b) **Video, ABR enabled** — *A*	*B*	*C*	(c) **Video and audio** — *A*	*B*	*C*
2, start	51.0 ± 3.4	53.1 ± 6.6	51.3 ± 3.7	51.2 ± 2.1	50.0 ± 1.5	50.8 ± 2.5	50.2 ± 1.3	50.3 ± 1.5	50.7 ± 2.8
2, middle	51.0 ± 2.8	54.2 ± 8.8	53.7 ± 10.1	52.5 ± 3.7	52.6 ± 5.2	51.9 ± 4.9	51.7 ± 4.4	51.5 ± 4.6	51.7 ± 5.2
2, finish	51.6 ± 6.4	51.7 ± 7.4	53.6 ± 9.2	53.1 ± 5.9	50.0 ± 2.0	52.9 ± 7.5	50.8 ± 1.9	51.4 ± 4.4	53.7 ± 8.2
10, start	11.8 ± 2.6	12.9 ± 2.6	15.6 ± 2.9	11.7 ± 2.7	10.5 ± 3.8	12.0 ± 2.3	11.2 ± 3.0	9.4 ± 2.4	12.4 ± 2.7
10, middle	13.2 ± 3.7	13.1 ± 3.5	11.3 ± 3.1	15.6 ± 3.8	15.8 ± 2.7	12.4 ± 3.1	13.3 ± 3.9	13.1 ± 1.5	13.1 ± 3.1
10, finish	10.6 ± 2.3	12.7 ± 3.8	13.0 ± 2.8	13.4 ± 3.6	10.8 ± 3.0	13.5 ± 2.7	12.5 ± 4.3	11.9 ± 3.5	14.2 ± 3.8

drastically within the first few seconds and levels off after roughly 2 minutes. This goes even faster with a smaller k: Figure 8(a) shows that average overhead levels off after less than 30 seconds. It is also worth noting that *increasing the number of worker threads decreases overhead notably*, albeit with diminishing returns around 32 threads.

Recall that videos are between 5 and 7 minutes in length, meaning that the best sets with $k = 2$ incur less than 10 MiB overhead per minute, while the best sets with $k = 10$ incur about 20 MiB. This is equivalent to downloading *a few large segments per minute.* To provide further perspective, Table 1 shows average overhead as a percentage of undefended video content for several values of k. Here, similar trends as in Figure 8 appear: lower values of k result in less overhead, and more threads decrease overhead.

5.2 Video Identification: One Set

In the following experiments, we play all of the videos in a single anonymity set 100 times, from start to finish, and collect network traces. We use the anonymity sets generated in Section 5.1 with $k = 10$ and $W = 32$ (but note that the choice of k and W is irrelevant, since we are comparing within a single anonymity set).

We randomly pick three anonymity sets: A, B and C – each with 10 videos, resulting in 1 000 video-on-demand playback sessions per dataset. For each anonymity set, we collect three datasets: only video, played at a single quality; only video, with ABR enabled; and video and audio, with ABR enabled (a typical streaming scenario). This results in 9 datasets, with over 1 000 hours of playback data.

The goal of these experiments is to determine whether videos in the same anonymity set are indistinguishable. We do this by running state-of-the-art video fingerprinting attacks against combinations of the videos in each dataset: of size 2 (are all *pairs* of videos indistinguishable?) and 10 (are *all videos* indistinguishable?). We also describe and illustrate verification steps that we go through to ensure that videos have indistinguishable traffic patterns.

5.2.1 Attack 1: vRF. We begin with the deep learning attack *vRF* [13], which is the state-of-the-art in its class for video fingerprinting. We apply vRF to the first minute (*start*) and last minute (*finish*) of traces, as in prior work [13, 32], as well as the third minute (*middle*, since videos are 5-7 minutes long). In prior work, the first and last minutes capture the startup phase and steady-state streaming, respectively. Here, the first, third, and last minutes cover the extended manifest, startup phase, steady state, and trailing, including the transition from normal playback to trailing. The result is a comprehensive evaluation of all of the phases of Dodge traffic.

We use an 80-20 train-test split and 10-fold cross-validation. Thus, when we discuss *mean accuracy*, we refer to the mean of cross-validated accuracy values. With combination size 2, there are 45 pairs of videos whose accuracies are included in the mean and standard deviation; with size 10, the mean is a single cross-validated accuracy and standard deviation is based on folds.

The mean accuracies with each combination size for the 9 datasets are depicted in Table 2. In all cases, *vRF* has similar performance to random guessing, regardless of the combination size or scenario. This is true for start, middle, and finish, indicating that Dodge traffic with `Dodge-mimic` is nearly indistinguishable during both phases of DASH streaming (the startup phase and steady-state playback) and that extended manifest downloads, quality switches, and trailing cycles do not introduce information leakage.

Note, however, that a small amount of information is still leaked, indicated by mean accuracies that are slightly higher than random guessing and relatively low variance. This is due to *intra-response timing*—that is, timing characteristics of individual packets within resource transmissions from server to client—which cannot be controlled on the client side. See Section 6 for further discussion.

5.2.2 Attack 2: WHE. We also employ the heuristic-based attack Walls Have Ears [31], both in its default form (with a database of segment sizes, *WHE-seg*) and modified for a more fair evaluation against Dodge traffic, with a database of *cycle* sizes (*WHE-cyk*). We use the implementation by Hasselquist et al. [32], extended to work with cycles. We apply *WHE-seg* to the last 90 seconds (45 segments) of each trace, as they did – Walls Have Ears is not effective against the startup phase because the heuristic it uses to extract segment sizes from network traces assumes steady-state streaming.

As with vRF, both variants of the attack are completely ineffective. This is to be expected with *WHE-seg* because the cycle sizes in `Dodge-mimic` have no correlation with the underlying segment sizes (accuracy never exceeds 8.5% regardless of the set tested). *WHE-cyk* does not provide any benefit because cycle sizes do not differ between videos in an anonymity set (accuracy 9.7%).

5.2.3 Verification. We also perform deterministic evaluation of our datasets to show that traffic patterns within an anonymity set are as indistinguishable as possible (given that Dodge is a client-side, application-layer framework). During data collection, we use `tshark` [88] to decrypt `pcap` traces, exposing HTTP requests and responses. We confirm that videos in the same anonymity set have an identical sequence of request-response sizes at the application layer, though TLS chunking varies from trace to trace.

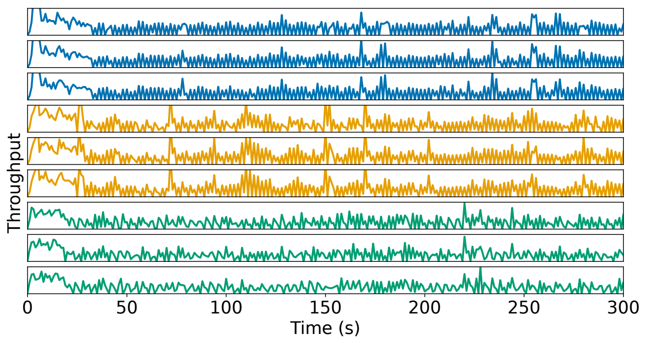


Figure 9: Throughput of 9 distinct videos taken from 3 anonymity sets, with ABR and audio.

All differences are due to (1) Dodge's scheduling logic, which we verify to be a random walk with no correlation to video contents (Section 3.3.7), and (2) how the server sends individual resources. To illustrate this, the throughput of videos from anonymity sets A, B, and C with ABR and audio enabled is shown in Figure 9. Note that all differences are due to randomness in timing between cycles (throughput spikes) and how the server shapes responses.

Takeaway: Dodge correctly applies the defense, `Dodge-mimic`, specified in our extended manifests. We verify this with state-of-the-art attacks, which do not surpass random guessing, and verification scripts/visualizations of network traffic.

5.3 Video Identification: Multiple Sets

In the next round of experiments, we play videos from different anonymity sets 10 times, from beginning to end, and collect network traces and QoE metrics. We use the anonymity sets from Section 5.1 with $k = 10$ and $W = 32$, as in Section 5.2.

Here, the goal is to show that the protection provided by Dodge/-`Dodge-mimic` scales to multiple anonymity sets and that QoE impacts are negligible. We run the attacks from the previous section against datasets with a progressively increasing number of anonymity sets, starting with 2. Note that we employ the same verification scripts and obtain only positive results, but do not focus on these results. Instead, we turn our attention to QoE, evaluating several metrics proposed by Hasselquist et al. [32].

5.3.1 Attack 1: vRF. To test vRF's classification ability when it must distinguish between different anonymity sets, we compute the mean and standard deviation of vRF's accuracy against 5 different combinations of anonymity sets with a number of combination sizes; these results are shown in Figure 10 for video at one quality and video with audio. Video with ABR enabled is omitted for clarity, as the results are nearly identical to video at one quality.

In general, accuracy is slightly lower than expected with a small number of anonymity sets (between 2 and 4), before rising and leveling off near random guessing. vRF's performance also becomes more stable with more anonymity sets, as indicated by markedly decreasing standard deviation. vRF is thus effective at determining which anonymity set a video belongs to, but not the exact video. However, vRF appears to capture the same small leakage as in Section 5.2 with only video and at least 5 anonymity sets, while

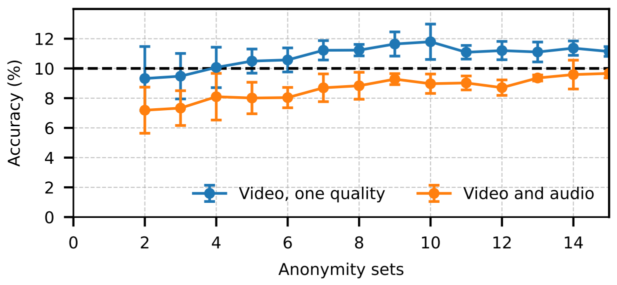


Figure 10: Accuracy (%, $\mu \pm \sigma$) of *vRF* when training and testing on videos in different anonymity sets, with $k = 10$.

accuracy remains lower than expected in all cases with video and audio, which is the most common real streaming scenario.

5.3.2 Attack 2: WHE. Since *WHE-seg* has no mechanism to learn which anonymity set a video belongs to, we expect its accuracy to be close to random guessing. This is true in practice: its accuracy peaks at 4.3% with 4 anonymity sets, subsequently decreasing as the number of sets increases. *WHE-cyk* is more effective, often correctly identifying the anonymity set, but not specific videos.

5.3.3 QoE. We consider a number of video player metrics that can substantially affect the subjective experience of viewers [42]:

- **Buffer size**, the amount of video data currently stored.
- **Playback rate**, the video content speed. This is included to facilitate comparisons with prior work on live streaming.
- **Bitrate**, the video quality: 1000k, 2000k, or 4000k.
- Number of **quality switch** occurrences.
- Occurrences of **wait**, small pauses during buffering.
- Occurrences of **stall**, potentially longer pauses when the buffer runs out and there is no video data left to play.
- Occurrences of **seek**, jumps to a different time.

Throughput, buffer size, playback rate, and bitrate are shown for an undefended video (played with vanilla `dash.js`) in Figure 11 to provide a baseline. Buffer size increases continually until the buffer target is reached, playback rate remains at 100% throughout the duration of the video, and bitrate starts at 1000k before quickly switching to 4000k for the remainder of the video. There are no wait, stall, or seek events – playback is uninterrupted.

The QoE of the same video played with `Dodge-mimic` (video with ABR enabled and audio) is shown in Figure 12. Here, stalls are also shown, with a red background in the buffer size plot. As expected, the defended throughput pattern does not resemble undefended throughput. It also has a longer duration, as does the buffer size plot, because the extended manifest includes trailing cycles. Bitrate rises from 1000k to 4000k quickly and remains there, and the playback rate is always at 100%, as in the undefended session. However, it takes longer to reach the buffer target with `Dodge-mimic`, and there are some short stalls at the beginning of playback.

This pattern represents the worst case: about 68% of playback sessions have *no stalls*, meaning that their QoE is effectively perfect. In Figure 12, the buffer is steadily filled and then remains near the buffer target throughout the playback session, after 1-2 seconds of stall time when switching to the highest quality. The initial stall time, which is 1.6 seconds on average, considering only the 32% of

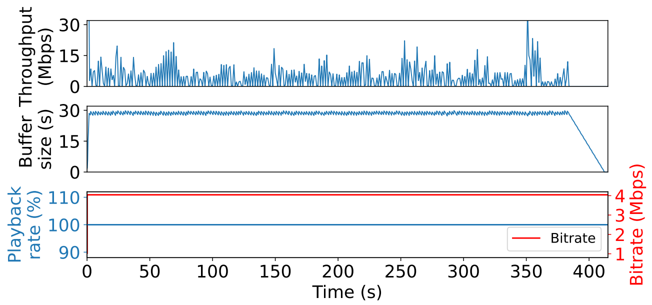


Figure 11: Example QoE metrics with no defense (`dash.js`).

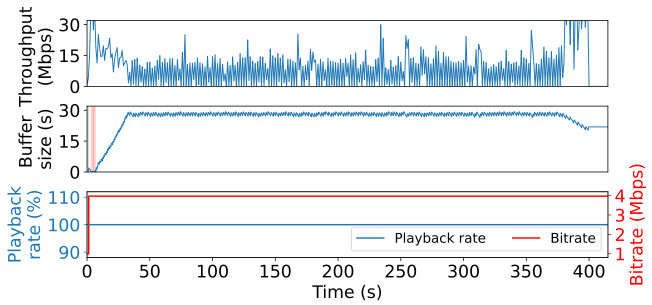


Figure 12: Worst-case QoE metrics with Dodge/`Dodge-mimic`.

sessions that have stalls, can be thought of as an extension of the startup delay; otherwise, QoE is unaffected by `Dodge-mimic`.

Beyond `Dodge-mimic`'s impressive QoE given the protection it provides, we note that all downloads are completed about 20 seconds before playback end, even though our example video has many large cycles near the end of playback, including trailing cycles. This indicates that even extended manifests with a large number of trailing cycles should not result in a wait time after playback.

Takeaway: Dodge is effective in applying a specified defense even with a larger number of videos and supports defenses with very high QoE. `Dodge-mimic`'s protection scales to many sets, and its QoE impacts are very small on all metrics.

5.4 HTTP/2, HTTP/3, and QUIC

In this subsection, we demonstrate that Dodge/`Dodge-mimic` works with HTTP/2 and above. We do this by perturbing extended manifests each time they are downloaded, adding or subtracting a random amount of bytes from each range. A defense that regularizes traffic to set a specific pattern is still used (`Dodge-mimic`), but small adjustments are made to this pattern to account for unpredictable HTTP header differences and avoid the need for perfect adjustment. We also modify Dodge's settings to pad client-side requests to a random value rather than a fixed one.

We collect the same datasets as in Section 5.2, for three different anonymity sets. However, we now use a unique perturbed extended manifest for each session rather than the originals from `Dodge-mimic`, random request padding, and HTTP/2. Since HTTP/3 has a similar header compression algorithm, these results are generalizable to HTTP/3-QUIC, whose adoption is continually increasing [84]. Thus, these experiments mirror a realistic scenario representative of how video streaming will occur in the future.

Table 3: Accuracy (%, $\mu \pm \sigma$) of *vRF* when training and testing on videos in the same anonymity set with video and audio, over HTTP/2. Results are comparable in other scenarios.

Comb. size and target ↓	Anonymity set *A*	*B*	*C*
2, start	50.2 ± 1.9	50.4 ± 1.2	51.4 ± 2.6
2, middle	51.8 ± 3.5	52.2 ± 4.8	53.3 ± 6.6
2, finish	49.7 ± 1.8	51.5 ± 4.6	52.9 ± 6.7
10, start	10.5 ± 2.1	10.8 ± 2.0	13.9 ± 4.3
10, middle	11.8 ± 3.3	13.5 ± 3.9	12.4 ± 3.4
10, finish	11.9 ± 2.8	13.5 ± 2.8	14.2 ± 4.3

5.4.1 Attack 1: vRF. Dodge provides the same level of protection when streaming over HTTP/2: as can be seen in Table 3 (where only video with audio is shown since other results are comparable), accuracy is near random guessing in all cases. This indicates that random perturbations can account for HTTP headers, which also implies that there is no need for manual adjustment in practice.

5.4.2 Attack 2: WHE. As in Section 5.2, *WHE-seg* and *WHE-cyk* both achieve approximately the same accuracy as random guessing.

Takeaway: Dodge supports effective defenses when header compression is used, making it a viable option for defending video traffic in realistic scenarios in the future.

6 Discussion

6.1 Dodge Framework

Our implementation of Dodge can, in principle, be used to stream *any* DASH video, and this can be done in many different deployment scenarios. For example, an *individual user*, group, or organization could write a web page embedding Dodge, create extended manifests for desired videos, and host them (perhaps locally). Extended manifests could also be provided as a *third-party service*, allowing for high scalability with the downside that users must trust the service and/or verify extended manifests. This is a plausible scenario reminiscent of AdBlock lists [3]: lists could be created for specific content providers, device types/streaming scenarios, threat models, etc., and hosted in community-hosted or otherwise publicly accessible repositories. Finally, a *content provider* could support Dodge, a best-case scenario: regularization defenses become more practical because the provider can refrain from publishing video attributes that can be used in side-channel attacks, privacy-aware TLS implementations can be deployed, etc. This may be particularly desirable for providers of niche or sensitive content, or hidden services [23] – in which case the added latency of Tor could act as an additional mitigation against exploitation of timing features.

In the realm of defense frameworks, Dodge can be considered rather tightly integrated with the application (video player). For example, the Maybenot framework [62] for network-layer defenses is provided as a standalone Rust crate; applications input *events* from a connection and receive *actions* that should be applied to packets on the connection. In contrast, most of our implementation efforts were related to the *integration* of `dash.js` with Dodge's new model, and the most generalizable components include the parser and verifier of the domain-specific language. Other frameworks are also tightly integrated: QCSD [78] uses the protocol features of QUIC to support network-layer defenses and is implemented in Neqo (Mozilla's QUIC-HTTP/3 library) [47], involving fundamental modifications to Neqo's networking code. For those wishing to implement the Dodge framework in another video player, our work can best be considered a ***reference for safe implementation***.

Even larger service providers with proprietary variants of the DASH standard, like YouTube [94], could make use of Dodge's model as-is. Continuing with the example of YouTube, one major peculiarity in their implementation of DASH is that *varying numbers of segments are downloaded at once.* This could be generalized, resulting in varying numbers of *cycles* being downloaded at any given time. One possibility is to download all cycles up to the next `buffer` directive at the same time rather than one-by-one. Another approach is to slightly modify Dodge's model in a rather natural way: incorporate explicit request timing rather than relying on the player's buffer logic. A key takeaway is that ***small changes to DASH require only small changes to Dodge, if any at all***.

In principle, Dodge supports arbitrary sequences of cycles whose sizes are non-zero and do not exceed the size of the largest segment. This includes random cycle count/sizes, traffic regularization with specific patterns, explicit anonymity sets, and sequences of cycles generated with external algorithms (e.g., deep learning). Note that these categories correspond to the primary types of website fingerprinting defenses [44] – see Section 7.1.2 for more details. As such, ***Dodge can enact a myriad of different defenses***.

6.2 Defense Possibilities

The goal of testing Dodge with a mimicry defense was two-fold: (1) to demonstrate that Dodge provides granular control over video traffic, (2) to show that a vast array of challenges (from Section 3.3) can be overcome without resorting to server-side solutions. Thus, we have shown how Dodge supports defenses that are more "challenging" to implement and enact. We envision that future work may explore different types of defenses based on varying degrees of *traffic regularization* and different ways of grouping videos.

However, some of the details that need to be considered when designing such defenses are inherently addressed by *randomized defenses*. In some frameworks, it is possible to have a single defense whose behavior varies between connections [44, 62]; with Dodge's deterministic JSON format, a new extended manifest per session can be used to change a video's traffic pattern in random ways, as we demonstrated in Section 5.4 to account for headers in HTTP/2. This is analogous to using a randomly generated defense for each connection, a very effective strategy that is deployed in practice by Mullvad VPN for WireGuard tunnels [48, 61]. Pulls et al. [61] provide preliminary evidence that this strategy is also effective against video fingerprinting, though with less explainable results than can be attained with Dodge and an unknown impact on QoE.

Note also that defenses should account for the specific ***deployment scenario and threat models*** they are used in. For example, different HTTP headers may need to be protected than those we address in this work, and header and content compression may vary in nature and be applied to different file types. Thus, a defense implemented for one scenario may not be directly applicable to another (but compatibility across scenarios seems more likely with randomized defenses). This is also true because users stream from different vantage points – mobile, home networks, etc. – and under various bandwidth conditions. While including multiple qualities in an extended manifest inherently accounts for this variation to some extent (for example, `Dodge-mimic` could still attain undefended playback with 1/4 or less of the bandwidth cap we set at 1000k quality), it may be advantageous to have different sets of extended manifests with varying trade-offs for different user groups.

Dodge may also be deployed under variations of the threat model in Section 2, requiring special considerations. For example, Dodge is designed to hide which video is being watched, not the fact that a defense is in use. It is likely possible to conceal even the very use of Dodge by carefully crafting extended manifests, which may entail QoE sacrifices, and modifying the framework or its settings.

6.3 Framework Limitations

Dodge provides as much control as possible over video flows, but it is impossible to control ***how the server shapes responses*** from the application layer, at least with HTTP. In our experiments, Dodge reduced vRF's accuracy close to `Dodge-mimic`'s theoretical bounds, but a small amount of leakage is unavoidable. This leakage is present in inter-packet timing ***within individual resource downloads***, and changing the order of cycles in extended manifests, server I/O settings, HTTP version, etc., does not mitigate this.

In practice, this is likely a smaller problem than in our controlled setting with directly connected machines. Differences in segments' locations on disk and server-side caching lead to negligible timing differences compared to round-trip times on the Internet, and factors such as server load and CDN behavior are noisy and difficult to exploit. Such timing differences may actually *improve* defenses, and even if such information leakage sources are present, targeted training data would be needed to capture them, making identification more challenging (a global model would not suffice). It is known that different training/testing vantage points introduce difficulty [85], and we gave the attacker a major advantage by using a small closed world. The only scenario in which substantial information leakage due to timing differences is likely is if, for example, subsets of a content provider's videos are hosted on different servers.

We draw the conclusion nonetheless, as other framework designers have [78], that *privacy-aware protocols* can significantly boost potential for wide-scale adoption of (application-layer) defenses on the Internet. For example, QUIC provides a `PADDING` frame which can be used to increase packet lengths [37], and the TLS 1.3 specification also includes a padding mechanism and discussion of traffic analysis [67]. Further consideration of traffic analysis during protocol design and development is needed to ensure that building blocks for traffic analysis defenses are in place. This is especially important given the sheer number of deployed application-layer protocols and corresponding client and server implementations that need defending, one of many reasons why Dodge's purely client-side nature is favorable and improves its deployment potential.

6.4 Deployment Challenges

While Dodge can shape video flows, it cannot address *external* side channels or sources of information leakage. One example is defenses that group videos into anonymity sets, like `Dodge-mimic`: if a subset of videos in a dataset is available at a higher quality than the others, playback at this quality breaks the anonymity set. This also applies to statistical attributes that content providers publish (views, likes, etc.), which can be used to determine the base rate of videos or as *oracles*, for augmented observation [60].

Moreover, anonymity sets themselves may serve as useful labels; for example, this is the case when all of the videos in an anonymity set are from the same category or are the same type of content. `Dodge-mimic` is configured to group videos with similar durations into the same anonymity set, which could result in short videos – such as YouTube shorts or the sort of content that often appears in social media feeds – ending up in separate anonymity sets, and this fact alone could be useful to a traffic analysis adversary.

Finally, we emphasize that it is likely desirable to modify MPDs rather than directly including them in extended manifests when embedding external videos in a web page. One reason for doing this could be to remove certain adaptation sets that may reveal information if certain types of defenses are used, as in the case of k-anonymity. Another potential issue is advertisements: periods can be used to mark breakpoints for ads, and MPDs can specify where and how ads should appear. These should be removed, as contacting servers that host ad content (especially in the case of targeted ads) poses serious privacy risks, including and beyond traffic analysis.

7 Related Work

7.1 Website Fingerprinting

To provide further traffic analysis context for Dodge and complement our discussion of video fingerprinting in Section 2, we discuss website fingerprinting as an illustrative and noteworthy example. In website fingerprinting (WF), a local, passive adversary monitors the traffic between a client and encrypted tunnel, such as a VPN or Tor [23], to infer which websites are being visited.

7.1.1 Attacks. Early WF classifiers were based on basic machine learning and required manual feature engineering [33, 35, 54], while modern deep learning-based attacks operate directly on simple transformations of raw network traces [9, 22, 38, 64, 68, 73, 77]. Prominent examples are Deep Fingerprinting (DF) [77], which merely uses a sequence of packet directions as input to achieve high accuracy against Tor traffic; and Tik-Tok [64], an enhancement to DF that incorporates packet timestamps. More recently, Robust Fingerprinting (RF) [73] has demonstrated the even greater potential of coarse-grained time series, and Laserbeak [43] has surpassed all of these attacks thanks to multi-channel feature representation and a novel model architecture including Transformers, which have also proven useful in multi-tab browsing scenarios [22, 38].

7.1.2 Defenses. In response to improving attacks, many defense proposals have been put forth. Most are network-layer defenses, which directly shape the flow of packets by introducing *padding* packets or temporarily *blocking* (delaying) outgoing packets. They can be categorized as being based on randomization [27, 59, 61], regularization [11, 26, 36], mimicry (also regularization) [52, 74, 87], adversarial machine learning [29, 49, 63], or traffic splitting [21, 34]. The primary challenge in the space is the inherent trade-off between protection and overhead [17, 18, 44], which affects user experience and willingness to adopt privacy-enhancing technologies.

7.2 Application-Layer Defenses

Earlier WF studies addressed *application-layer defenses*, arguing that it is unnecessary to defend at lower layers since they carry no identifying details about websites. Proposed defenses involve batching requests, sending partial requests, and injecting padding requests (client side) [41, 56] and modifying the number and sizes of resources per page load (server side) [15]. Though these strategies are reminiscent of Dodge's, they have been overcome [12] or have major deployment challenges [75], and current WF work does not focus on application-layer defenses. Nonetheless, it is accepted that domain knowledge can improve defenses, as we have shown – one major factor that makes video easier to defend at the application layer is the use of a single connection, whereas web traffic includes many third-party resources hosted on different domains [75].

7.3 Defense Frameworks

Since "optimal" defenses are a moving target, several frameworks have been proposed that provide building blocks for defenses, decoupling integration with applications from defense design. These include the circuit padding framework in Tor [57], in which padding-only defenses are expressed as non-deterministic finite state machines; the standalone Maybenot framework [62], a Rust library based on probabilistic state machines that can be integrated into a variety of applications; and WFDefProxy [28], which allows for defenses implemented in Go to be deployed over a Tor Pluggable Transport [58]. QCSD [78] supports network-layer defenses in QUIC through protocol features. These are all network-layer frameworks with roots in website fingerprinting research; Dodge is the first application-layer framework and the first for video traffic.

8 Conclusions

We presented Dodge, a client-side framework for application-layer video fingerprinting defenses, and evaluated it using `Dodge-mimic` to demonstrate the control it provides over video traffic. Our hope is that Dodge's straightforward design, ease of use for all parties involved, and excellent deployment potential will inspire steps towards practical deployment of defenses. Dodge also represents a reference for the safe implementation of defenses in other video players and contributes new insights related to the potential of application-layer defenses. We urge protocol designers to consider traffic analysis where possible and encourage more systematic investigation of which defenses should be deployed and how as traffic analysis protection becomes more widespread on the Internet. Dodge and the insights it gives can help kick-start such efforts.

Artifacts, Ethics, and AI Use

The Dodge framework, `Dodge-mimic` defense, and associated research artifacts are available on GitHub under the BSD 3-Clause License: https://github.com/trafnex/dodge-framework

We only collected our own traffic from a local network; no real users' traffic was collected, defended, or used in attack evaluations. Grammarly was used for writing assistance.

Acknowledgments

This work was funded by Karlstad Internet Privacy Lab, the Graduate School in Computer Science (CUGS) at Linköping University, the Swedish Foundation for Strategic Research (SSF), the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and the Knowledge Foundation of Sweden.

References

[1] 2022. *DASH-IF Interoperability Points v4.3.* Technical Report. DASH Industry Forum. https://dashif.org/docs/DASH-IF-IOP-v4.3.pdf

[2] 2022. Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats.

[3] AdBlock. 2026. Introduction to Filter Lists. https://helpcenter.getadblock.com/hc/en-us/articles/9738523403027-Introduction-to-Filter-Lists.

[4] Akamai Technologies, Inc. 2025. Akamai Stream Validation and Player Test Page v0.1.1. players.akamai.com/players/dashjs. https://players.akamai.com/players/dashjs

[5] Amazon Web Services, Inc. 2025. *AWS Elemental MediaPackage v2: User Guide.* Amazon Web Services, Inc. https://docs.aws.amazon.com/pdfs/mediapackage/latest/userguide/mediapackage-guide.pdf

[6] Sangwook Bae, Mincheol Son, Dongkwan Kim, CheolJun Park, Jiho Lee, Sooel Son, and Yongdae Kim. 2022. Watching the Watchers: Practical Video Identification Attack in LTE Networks. In *Proc. USENIX Security.*

[7]]bampis2017study Christos George Bampis, Zhi Li, Anush Krishna Moorthy, Ioannis Katsavounidis, Anne Aaron, and Alan Conrad Bovik. [n. d.]. Study of temporal effects on subjective video quality of experience. *IEEE Trans. on Image Processing* 26, 11 ([n. d.]).

[8] Roberto J Bayardo and Rakesh Agrawal. 2005. Data privacy through optimal k-anonymization. In *21st International conference on data engineering (ICDE'05).* IEEE, 217–228.

[9] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2018. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. In *Proc. Privacy Enhancing Technologies (PETS).*

[10] Martin Björklund and Romaric Duvignau. 2025. Endangered Privacy:{Large-Scale} Monitoring of Video Streaming Services. In *34th USENIX Security Symposium (USENIX Security 25).* 6581–6597.

[11] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proc. ACM Computer and Communications Security (CCS).*

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

[13] August Carlson, David Hasselquist, Ethan Witwer, Niklas Johansson, and Niklas Carlsson. 2024. Understanding and Improving Video Fingerprinting Attack Accuracy under Challenging Conditions. (2024).

[14] Tianyu Chen, Yiheng Lin, Nicolas Christianson, Zahaib Akhtar, Sharath Dharmaji, Mohammad Hajiesmaili, Adam Wierman, and Ramesh K Sitaraman. 2024. SODA: An adaptive bitrate controller for consistent high-quality video streaming. In *Proc. ACM SIGCOMM.* 613–644.

[15] Giovanni Cherubin, Jamie Hayes, and Marc Juárez. 2017. Website fingerprinting defenses at the application layer. *Proceedings on Privacy Enhancing Technologies* 2017, 2 (2017), 168–185.

[16] Aloni Cohen. 2022. Attacks on Deidentification's Defenses. In *31st USENIX Security Symposium (USENIX Security 22).* 1469–1486.

[17] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2018. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency-choose two. In *Proc. IEEE Security and Privacy (S&P).*

[18] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2020. Comprehensive anonymity trilemma: User coordination is not enough. In *Proc. Privacy Enhancing Technologies (PETS).*

[19] DASH Industry Forum. 2025. Controlbar. dash.js Documentation. https://dashif.org/dash.js/pages/usage/controlbar.html

[20] DASH-Industry-Forum. 2026. dash.js. https://dashjs.org/.

[21] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, et al. 2020. TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting. In *Proc. ACM CCS.*

[22] Xinhao Deng, Qilei Yin, Zhuotao Liu, Xiyuan Zhao, Qi Li, Mingwei Xu, Ke Xu, and Jianping Wu. 2023. Robust multi-tab website fingerprinting attacks in the wild. In *Proc. IEEE S&P.*

[23] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *Proc. USENIX Security.* 303–320.

[24] Meijie Du, Minchao Xu, Kedong Liu, Weitao Tang, Lijuan Zheng, and Qingyun Liu. 2023. Long-Short Terms Frequency: A Method for Encrypted Video Streaming Identification. In *Proc. Computer Supported Cooperative Work in Design (CSCWD).*

[25] Ran Dubin, Amit Dvir, Ofir Pele, and Ofer Hadar. 2017. I Know What You Saw Last Minute—Encrypted HTTP Adaptive Video Streaming Title Classification. *IEEE Trans. on Information Forensics and Security (TIFS)* (2017).

[26] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proc. IEEE Symposium on Security and Privacy (S&P).*

[27] Jiajun Gong and Tao Wang. 2020. Zero-delay Lightweight Defenses against Website Fingerprinting. In *Proc. USENIX Security.*

[28] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. 2021. WFDefProxy: Modularly implementing and empirically evaluating website fingerprinting defenses. *arXiv:2111.12629* (2021).

[29] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. 2022. Surakav: Generating Realistic Traces for a Strong Website Fingerprinting Defense. In *Proc. IEEE Symposium on Security and Privacy (S&P).*

[30] Google. 2026. YouTube Data API. https://developers.google.com/youtube/v3.

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

[32] David Hasselquist, Ethan Witwer, August Carlson, Niklas Johansson, and Niklas Carlsson. 2024. Raising the Bar: Improved Fingerprinting Attacks and Defenses for Video Streaming Traffic. In *Proc. Privacy Enhancing Technologies (PETS).*

[33] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *Proc. USENIX Security.*

[34] Sébastien Henri, Gines Garcia-Aviles, Pablo Serrano, Albert Banchs, and Patrick Thiran. 2020. Protecting against Website Fingerprinting with Multihoming. In *Proc. Privacy Enhancing Technologies (PETS).*

[35] Andrew Hintz. 2002. Fingerprinting websites using traffic analysis. In *Proc. Workshop on Privacy Enhancing Technologies.*

[36] James K Holland and Nicholas Hopper. 2022. RegulaTor: A Straightforward Website Fingerprinting Defense. In *Proc. Privacy Enhancing Technologies (PETS).*

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

[38] Zhaoxin Jin, Tianbo Lu, Shuang Luo, and Jiaze Shang. 2023. Transformer-based Model for Multi-tab Website Fingerprinting Attack. In *Proc. ACM CCS.*

[39] Charles 'Buck' Krasic, Mike Bishop, and Alan Frindell. 2022. QPACK: Field Compression for HTTP/3. RFC 9204. https://doi.org/10.17487/RFC9204

[40] Ying Li, Yi Huang, Richard Xu, Suranga Seneviratne, Kanchana Thilakarathna, Adriel Cheng, Darren Webb, and Guillaume Jourjon. 2018. Deep Content: Unveiling Video Streaming Content from Encrypted WiFi Traffic. In *Proc. IEEE Network Computing and Applications (NCA).*

[41] Xiapu Luo, Peng Zhou, Edmond WW Chan, Wenke Lee, Rocky KC Chang, and Roberto Perdisci. 2011. HTTPOS: Sealing Information Leaks with Browser-Side Obfuscation of Encrypted Flows. In *Proc. NDSS.*

[42] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proc. ACM SIGCOMM.*

[43] Nate Mathews, James K Holland, Nicholas Hopper, and Matthew Wright. 2024. LASERBEAK: Evolving Website Fingerprinting Attacks with Attention and Multi-Channel Feature Representation. *IEEE Trans. on Information Forensics and Security* (2024).

[44] Nate Mathews, James K Holland, Se Eun Oh, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. 2023. SoK: A critical evaluation of efficient website fingerprinting defenses. In *Proc. IEEE S&P.*

[45] Nick Mathewson. 2021. Implement circuit padding machines (#63). https://gitlab.torproject.org/tpo/core/arti/-/issues/63.

[46] MDN contributors. 2025. *HTTP headers.* https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers

[47] Mozilla. 2025. *Neqo: Mozilla's QUIC and HTTP/3 implementation in Rust.*

[48] Mullvad VPN. 2025. DAITA: Defense Against AI-guided Traffic Analysis, https://web.archive.org/web/20250504100318/https://mullvad.net/en/vpn/daita.

[49] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2021. Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. In *Proc. USENIX Security.*

[50] AppLogic Networks. 2025. 2025 Global Internet Phenomena Report. https://www.applogicnetworks.com/phenomena.

[51] NGINX. 2025. https://www.nginx.com/.

[52] Rishab Nithyanand, Xiang Cai, and Rob Johnson. 2014. Glove: A bespoke website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society.* 131–134.

[53] Pallets. 2010. https://flask.palletsprojects.com/en/stable/.

[54] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, et al. 2016. Website Fingerprinting at Internet Scale. In *Proc. NDSS.*

[55] Roberto Peon and Herve Ruellan. 2015. HPACK: Header Compression for HTTP/2. RFC 7541. https://doi.org/10.17487/RFC7541

[56] Mike Perry. 2011. Experimental Defense for Website Traffic Fingerprinting. https://blog.torproject.org/experimental-defense-website-traffic-fingerprinting/.

[57] Mike Perry and George Kadianakis. 2020. Circuit Padding Developer Documentation. https://github.com/torproject/tor/blob/main/doc/HACKING/CircuitPaddingDevelopment.md.

[58] Tor Project. 2026. Pluggable Transport Specification (Version 1). https://gitweb.torproject.org/torspec.git/tree/pt-spec.txt.
[59] Tobias Pulls. 2020. Towards Effective and Efficient Padding Machines for Tor. *arXiv:2011.13471* (2020).
[60] Tobias Pulls and Rasmus Dahlberg. 2020. Website Fingerprinting with Website Oracles. *PoPETs* (2020).
[61] Tobias Pulls, Topi Korhonen, Ethan Witwer, and Niklas Carlsson. 2026. Ephemeral Network-Layer Fingerprinting Defenses. In *Proc. Privacy Enhancing Technologies Symposium (PETS)*.
[62] Tobias Pulls and Ethan Witwer. 2023. Maybenot: A Framework for Traffic Analysis Defenses. In *Proc. ACM Workshop on Privacy in the Electronic Society (WPES)*.
[63] Mohammad Saidur Rahman, Mohsen Imani, Nate Mathews, and Matthew Wright. 2021. Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces. *IEEE TIFS* (2021).
[64] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2020. Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks. In *Proc. Privacy Enhancing Technologies (PETS)*.
[65] Andrew Reed and Benjamin Klimkowski. 2016. Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11n connections. In *Proc. IEEE Consumer Communications & Networking Conference (CCNC)*.
[66] Andrew Reed and Michael Kranch. 2017. Identifying HTTPS-Protected Netflix Videos in Real-Time. In *Proc. ACM CODASPY*.
[67] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446
[68] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *Proc. Network and Distributed System Security (NDSS)*.
[69] David C Robinson, Yves Jutras, and Viorel Craciun. 2012. Subjective video quality assessment of HTTP adaptive streaming technologies. *Bell Labs Technical Journal* 16, 4 (2012), 5–23.
[70] Sandvine. 2024. 2024 Global Internet Phenomena Report. https://www.sandvine.com/global-internet-phenomena-report-2024.
[71] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2017. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *Proc. USENIX Security*.
[72] Selenium. 2025. https://www.selenium.dev/.
[73] Meng Shen, Kexin Ji, Zhenbo Gao, Qi Li, Liehuang Zhu, and Ke Xu. 2023. Subverting Website Fingerprinting Defenses with Robust Traffic Representation. In *Proc. USENIX Security*.
[74] Meng Shen, Kexin Ji, Jinhe Wu, Qi Li, Xiangdong Kong, Ke Xu, and Liehuang Zhu. 2024. Real-Time Website Fingerprinting Defense via Traffic Cluster Anonymization. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 263–263.
[75] Sandra Siby, Ludovic Barman, Christopher Wood, Marwan Fayed, Nick Sullivan, and Carmela Troncoso. 2023. Evaluating practical QUIC website fingerprinting defenses for the masses. *Proceedings on Privacy Enhancing Technologies* (2023).
[76] Daniel Silhavy. 2022. DASH-IF Reference Client – dash.js. Open-Source Media Application Reference Tools (OSMART) workshop 2022. https://dvb.org/wp-content/uploads/2022/05/OSMART-workshop-dashjs-presentation.pdf
[77] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *Proc. ACM Computer and Communications Security (CCS)*.
[78] Jean-Pierre Smith, Luca Dolfi, Prateek Mittal, and Adrian Perrig. 2022. {QCSD}: A {QUIC}{Client-Side}{Website-Fingerprinting} Defence Framework. In *31st USENIX Security Symposium (USENIX Security 22)*. 771–789.
[79] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia* (2011).
[80] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2020. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. *IEEE/ACM Transactions on Networking* 28, 4 (2020), 1698–1711. https://doi.org/10.1109/TNET.2020.2996964
[81] TeamYouTube. 2022. YouTube Video Player Updates: Most Replayed, Video Chapters, Single Loop & More. https://support.google.com/youtube/thread/164099151/youtube-video-player-updates-most-replayed-video-chapters-single-loop-more.
[82] Akamai Technologies. 2026. https://techdocs.akamai.com/adaptive-media-delivery/docs/default-cors-policy-rule.
[83] Alexander Vaskevich, Thilini Dahanayaka, Guillaume Jourjon, and Suranga Seneviratne. 2021. Smaug: Streaming media augmentation using CGANs as a defence against video fingerprinting. In *Proc. IEEE NCA*.
[84] W3Techs. 2025. Usage statistics of HTTP/3 for websites. https://w3techs.com/technologies/details/ce-http3.
[85] Timothy Walsh, Armon Barton, and Mathias Kölsch. 2025. Improved Open-World Fingerprinting Increases Threat to Streaming Video Privacy but Realistic Scenarios Remain Difficult. *Proceedings on Privacy Enhancing Technologies* (2025).
[86] Tim Walsh, Trevor Thomas, and Armon Barton. 2024. Exploring the Capabilities and Limitations of Video Stream Fingerprinting. In *2024 IEEE Security and Privacy Workshops (SPW)*. IEEE, 28–39.
[87] Tao Wang and Ian Goldberg. 2017. Walkie-Talkie: An efficient defense against passive website fingerprinting attacks. In *26th USENIX Security Symposium (USENIX Security 17)*. 1375–1390.
[88] Wireshark Foundation. 2025. *Wireshark and TShark*.
[89] Ethan Witwer, James K Holland, and Nicholas Hopper. 2022. Padding-only defenses add delay in Tor. In *Proc. ACM WPES*.
[90] Hua Wu, Zhenhua Yu, Guang Cheng, and Shuyi Guo. 2020. Identification of encrypted video streaming based on differential fingerprints. In *Proc. IEEE Computer Communications Workshops (INFOCOM WKSHPS)*.
[91] Luming Yang, Shaojing Fu, Yuchuan Luo, and Jiangyong Shi. 2020. Markov probability fingerprints: A method for identifying encrypted video traffic. In *Proc. Mobility, Sensing and Networking (MSN)*.
[92] Luming Yang, Yingming Zeng, Shaojing Fu, and Yuchuan Luo. 2020. Unsupervised analysis of encrypted video traffic based on Levenshtein distance. In *Proc. Security and Privacy in Social Networks and Big Data*.
[93] Xiaokuan Zhang, Jihun Hamm, Michael K Reiter, and Yinqian Zhang. 2019. Statistical privacy for streaming traffic. In *Proc. NDSS*.
[94] Xiyuan Zhang, Gang Xiong, Zhen Li, Chen Yang, Xinjie Lin, Gaopeng Gou, and Binxing Fang. 2024. Traffic spills the beans: A robust video identification attack against YouTube. *Computers & Security* (2024).

A Defense Comparisons

Here, we compare `Dodge-mimic` with three defenses that have also been evaluated with `dash.js`. These are Hasselquist et al.'s [32] Adapted FRONT, Adapted RegulaTor, and Scrambler. The former two defenses are adaptations of the website fingerprinting defenses FRONT [27] and RegulaTor [36], respectively, while the latter defense is specifically designed for video fingerprinting.

Note that these are *network-layer defenses*, and they were evaluated with vanilla dash.js and the Maybenot framework [62]. Nevertheless, they allow us to make more concrete statements about how Dodge compares to existing defenses. Rather than recreating Hasselquist et al.'s experimental setup, we run live deployments with `Dodge-mimic` applied to the videos in their LongEnough dataset, compare with the QoE results from their live deployments, and simulate their defenses on LongEnough with version 2 of the Maybenot simulator [62] to compute overhead and run attacks. We exclude other proposed video fingerprinting defenses [61, 83, 93], as comparing to the authors' results would result in an apples-to-oranges comparison, and much implementation effort would be required to thoroughly evaluate them on our dataset or LongEnough.

The LongEnough dataset contains 100 videos that are each 10 minutes long, played at 10 different offsets from the start of the video (increments of 60 seconds), 10 times. The intent of offsets is to capture the impact that cascading effects of defenses have after different amounts of time [32]. Deep learning-based attacks are applied to the last 60 seconds of each trace, and WHE uses the last 90 seconds of traces, as in our `Dodge-mimic` experiments.

Setup. In the following experiments, we collect 10 samples of each video in LongEnough with `Dodge-mimic` at offset 0. We compute overhead and QoE metrics from the first sample of each video, corresponding to Hasselquist et al.'s defense deployments, and apply vRF and WHE to the entire collected dataset. Then, we use the Maybenot simulator [62] to simulate Adapted FRONT, Adapted RegulaTor, and Scrambler 10 times at offset 0, compute overhead on the simulated datasets, and apply vRF and WHE to them.

We compare `Dodge-mimic` with $k = 10$, $W = 2$ to the defense configurations that have the closest *overall overhead*. This means ($N = 3500$, $W = 5$) for Adapted FRONT, ($R_0 = 1000$, $D = 0.95$) for Adapted RegulaTor, and ($\delta = 200$, $N = 700$) for Scrambler. Refer to Hasselquist et al.'s work [32] for an explanation of what these parameters mean and results with other parameter choices.

Table 4: Comparison of defenses: bandwidth overhead, attack accuracy, and QoE.

Defense	Bandwidth overhead			Attack accuracy (%)		Quality of Experience (QoE)									
						Occurrences				Activity per quality					
	Send	Receive	Overall	vRF	WHE	Wait	Stall	Seek	Q. switch	Wait 1k	Wait 2k	Wait 4k	Play 1k	Play 2k	Play 4k
Adapted FRONT	×1 467	49.2%	92.8%	86.9	3.6	1	0	0	2	0.3%	0.0%	0.0%	0.2%	0.0%	99.4%
Adapted RegulaTor	×1 777	135.9%	193.8%	55.0	0.2	7	8	0	363	0.4%	0.7%	0.1%	8.8%	47.1%	42.8%
Scrambler	×2 146	102.9%	174.6%	80.0	1.9	5	2	0	14	0.7%	0.1%	0.4%	4.7%	4.3%	89.9%
`Dodge-mimic`	×1.5	87.5%	88.7%	9.5	0.6	1	2	0	1	0.0%	0.0%	0.0%	0.0%	0.0%	99.8%

Results. Our findings are similar to the original evaluation results, except that we see higher overhead with the tested Adapted RegulaTor configuration, likely due to updates to the Maybenot simulator; and higher accuracy against Adapted RegulaTor and Scrambler due to the use of vRF, which outperforms Beauty and the Burst [71].

At a high level, Adapted FRONT incurs moderate overall overhead, has a small impact on QoE, and provides little protection against deep learning attacks. On the other hand, Adapted RegulaTor has high overhead, a major negative effect on QoE, and provides a significant level of protection against attacks. These two extremes clearly illustrate the trade-off between QoE and protection in the video fingerprinting domain: bandwidth overhead is less perceptible than in, e.g., website fingerprinting scenarios thanks to the video player's buffer, but delays can significantly degrade QoE.

Scrambler is more practical than both of these defenses, with high overhead, far better QoE than Adapted RegulaTor, and modest protection, which can be improved signficantly by using configurations that have slightly higher overhead [32]. Finally, `Dodge-mimic` has moderate overhead, QoE near that of undefended traffic, and provides a high degree of protection, similar to higher-overhead configurations of Scrambler. It is worth noting that `Dodge-mimic`'s send overhead is several orders of magnitude lower than the other three defenses; this is because Hasselquist et al. explicitly chose to use unnecessarily heavy defenses on the client side [32]. Both Scrambler and `Dodge-mimic` show an interesting QoE trade-off: QoE can remain unaffected for the majority of playback if some disturbances are acceptable in the startup phase, where bandwidth overhead and slight delays/bottlenecks have a stronger effect.

We conclude by remarking that considering only accuracy values is not an entirely meaningful way to compare defenses' protection: while Scrambler (with higher overhead than we test) and `Dodge-mimic` appear to provide similar protection, Scrambler is a general-purpose defense and simply leaks information when videos have very large segments; `Dodge-mimic` leaks which anonymity set a video belongs to as an intentional design decision. Thus, while an adversary that targets Scrambler can have high confidence when videos have huge segments, an adversary that targets `Dodge-mimic` can only be certain of the anonymity set that a video belongs to – these are fundamentally different types of information. We therefore reiterate the importance of considering use cases and deployment scenarios when designing/selecting defenses.