Dodge: A Client-Side Framework for Video Fingerprinting Defenses

Steps Towards Deployable Video Fingerprinting Protection

Ethan Witwer David Hasselquist Tobias Pulls Niklas Carlsson

SVTA Players and Playback Working Group

March 30, 2026

Overview

1. The Threat: Video Fingerprinting
2. Why Existing Defenses are Insufficient
3. Introducing Dodge
4. Proof of Concept: Dodge-mimic
5. Why Have Dodge in `dash.js`?
6. Conclusion

The Threat: Video Fingerprinting

What is Video Fingerprinting?

Video fingerprinting is a local, passive attack that can identify *which video* a user is watching by analyzing their encrypted network traffic.

How it Works

1. Attacker sits on network path between client and video server
2. **Collects encrypted traffic** — fully hidden by HTTPS/TLS
3. **Applies an ML classifier** trained on MPDs or traffic data
4. **Identifies the video** being watched

Key Insight

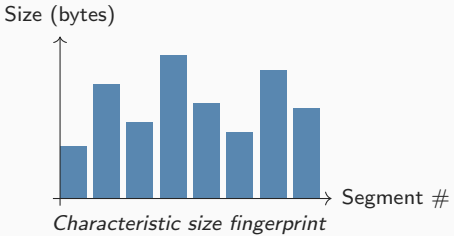
HTTPS hides content, but not **traffic patterns** (request/response sizes, timing, etc.). Every video has a characteristic pattern that survives encryption.

Why Encryption is Not Enough

DASH video uses **variable bitrate (VBR)** encoding: the size of each segment depends on the *visual complexity of its content*.

Video Fingerprints

Each video has a unique sequence of segment sizes that survives encryption.



State-of-the-Art Attacks

Deep learning classifiers and heuristic methods identify with high accuracy, even through VPNs and wireless encryption.

Scope of the Problem

Smaller streaming platforms are especially exposed: their entire video catalogs can be fully fingerprinted.

A Real and Overlooked Privacy Risk

Who is at Risk?

- Journalists or activists watching sensitive content
- Users in countries where certain content is prohibited or monitored
- Corporate users watching confidential training/briefing material
- Anyone who wants to keep their viewing habits private

The Gap

Despite the existence of several highly accurate attacks, work on **practical, deployable defenses** is almost nonexistent.

Our Goal

Make defense development, deployment, and usage easy with a **simple client-side framework** — and propose it as a feature of `dash.js` itself.

Why Existing Defenses are Insufficient

The Limitations of Current Approaches

Network-Layer Defenses

- Require changes to servers and/or network infrastructure
- No access to application-layer (video) semantics $\Rightarrow$ suboptimal
- Volatility: impractical under variable network conditions

Server-Side Approaches

- *Video reencoding*: enormous cost
- *Hard-coded defenses*: bind applications to one technique, quickly obsoleted as attacks improve
- All require laborious and expensive server or infrastructure changes

The Application-Layer Opportunity

DASH gives video *players* rich control over what bytes are requested and when, enough to implement strong, practical defenses **entirely on the client side**.

Introducing Dodge

Dodge: Overview and Design Goals

Dodge is a **client-side, application-layer framework** for video fingerprinting defenses, implemented as a fork of `dash.js`.

For Defense Designers

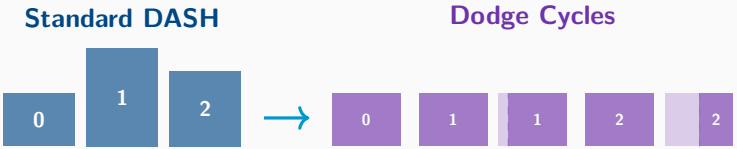
- **Precise, byte-level control** over every request and response
- A minimal **domain-specific language** (JSON) for expressing defenses declaratively
- Building blocks usable for many defense strategies

For Web Developers/Integrators

- Drop-in replacement for `dash.js`
- **Plug-and-play**: 1 URL parameter
- **No changes required** to servers, infrastructure, or CDNs
- Defenses can be hosted anywhere: video server, trusted external server, or localhost

Generalized Video Streaming: Cycles

Dodge replaces standard DASH segment downloads with **cycles**, a generalized unit consisting of a single HTTP request–response exchange:



One cycle: **segment index** + **byte range** + opt. **padding** flag + opt. **buffer** flush flag

Key Capability

Defense designers can control *exactly* which bytes are requested and in what order. QoE and overhead trade-offs can be made; QoE can remain essentially unaffected.

The Extended Manifest for Defense Specification

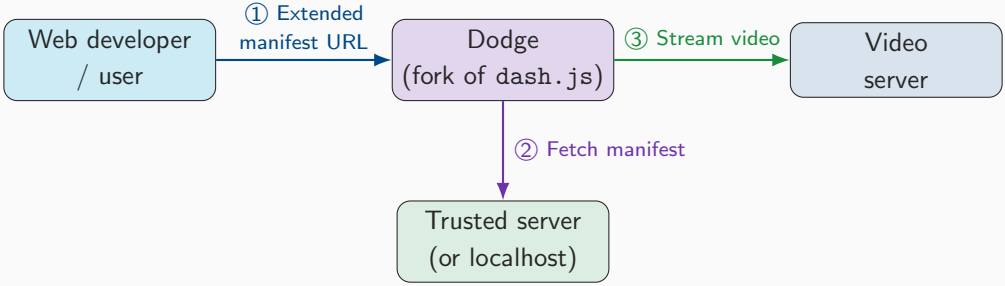
Dodge replaces the DASH MPD with an **extended manifest**, a JSON file that fully specifies a defended stream (all cycles to be downloaded).

```
{
  "start": {
    "mpd": "<MPD xmlns='urn:mpeg:dash:schema:mpd [...]",
    "base_uri": "https://cdn.example.com/v/"
  },
  "streams": [{
    "label": "video_1000k",
    "data": [
      {"index": 0, "buffer": true},
      {"index": 1, "range": "-50000"},
      {"index": 1, "range": "40000-",
                   "buffer": true},
      {"index": 2, "range": "-50000"},
      {"index": 2, "range": "20000-",
                   "buffer": true}
    ]
  }]
}
```

Fields:

- `index`: segment index
- `range`: HTTP byte range to request (for partial segments)
- `padding`: download and discard?
- `buffer`: flush data to buffer?

Plug-and-Play Deployment



Same Interface as `dash.js`

The only change for a web developer or user is **one URL parameter** pointing to the extended manifest. Everything else behaves as in standard `dash.js`.

Zero Server Requirements

Video servers, infrastructure, and CDNs are untouched. Defense happens entirely in the **extended manifest** and **player**. No coordination needed.

Behind the Scenes

Dodge's generalized model accounts for the full complexity of real DASH deployments.

- **Adaptive Bitrate (ABR)**: ABR rules are applied to cycles in a video-independent way, preventing information leakage due to quality switch dynamics
- **Audio and subtitle tracks**: all media types are defended together, not just the video stream
- **Stream duration**: trailing cycles can handle variable-length videos without introducing wait time
- **HTTP dynamics**: request sizes and header variation are accounted for in cycle size calculations
- **Request timing**: a random-walk scheduler decouples request timing from cycle type and content, removing a potential timing side channel
- **Extended manifest traffic**: manifests' on-wire sizes are also controlled and protected

Proof of Concept: Dodge-mimic

Dodge-mimic: A Mimicry Defense

To validate Dodge's capabilities, we implement Dodge-mimic, a **k-anonymity-based mimicry defense** expressed entirely in Dodge's extended manifest format.

Stage 1: Grouping

Videos are grouped into **anonymity sets** by minimizing overhead for all k videos in a set to have identical cycle sizes.

Stage 2: Manifest Generation

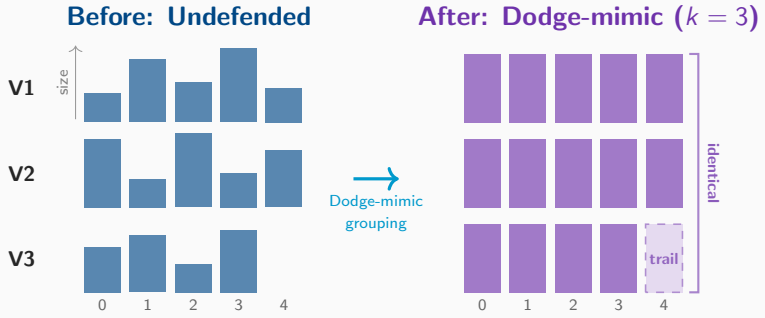
For each set, extended manifests are generated that pad requests/responses to a common *byte-level* profile.

Practical Notes

Manifests are generated once per catalog update and hosted statically. k is a single tunable parameter: larger k = stronger privacy, higher overhead.

k-anonymity has known limitations, and we present Dodge-mimic only as a **proof of concept** for Dodge's capabilities – many other defenses are possible.

Dodge-mimic: Making Traffic Patterns Identical



All videos in the group are made to have **identical** cycle sequences.

The trailing cycle pads the shorter video to match the group duration.

Experimental Setup

Video Dataset

- 1,400 YouTube videos (100 videos from each of 14 categories), 5–7 minutes, $\geq$720p
- 4 representations: 1/2/4 Mbps + audio
- Hosted on an nginx server

Data Collection

- Chrome via Selenium, web page with Dodge
- Network traces (pcap) + player's QoE metrics
- HTTP/1.1 and HTTP/2 evaluated separately

Attacks Evaluated

- **vRF**: CNN on sequences of byte counts (state of the art)
- **WHE**: heuristic that matches observed sizes against a database of segment fingerprints

Result 1: Attack Accuracy Near Random Guessing

Table: Attack accuracy against Dodge-mimic, with $k = 10$. Random guessing = **10%**.

Scenario	vRF (CNN)	WHE (heuristic)
Video, one quality	10–16%	$\leq$10%
Video + ABR	10–16%	$\leq$10%
Video + audio	10–14%	$\leq$10%

Tested Across

- Startup phase, steady-state streaming, trailing cycles
- Quality switches and audio track enabled/disabled
- HTTP/1.1 and HTTP/2

Residual Leakage

Due to **intra-response packet timing**, the pace at which the server delivers bytes within a single HTTP response. This is below the application layer and *cannot be controlled by any player-side defense*.

Verified Independently

tshark decryption confirms **identical** application-layer request/response sequences within each group. All differences are timing noise.

Result 2: Modest, Tunable Bandwidth Overhead

Table: Average overhead for 4 Mbps video

k	**16 threads**	**32 threads**
2	34.9%	21.0%
3	47.5%	32.5%
5	60.7%	48.7%
10	75.8%	73.1%

- At $k = 2$: $\sim$21% extra bandwidth
- At $k = 10$: $\sim$73%, equivalent to a few large images per minute
- Audio overhead: **negligible**
- k is **tunable** and controls the protection-overhead trade-off

Perspective

On a 50 Mbps connection streaming 4 Mbps video, $k = 10$ uses only $\sim$10% of available bandwidth.

Result 3: Minimal QoE Impact

QoE metrics measured:

- Buffer size over time
- Video bitrate and quality switches
- Stall / wait / seek events

Results

- **68%** of sessions: *zero stalls*
- Average initial stall: **1.6 s** (only in the 32% of sessions with stalls)
- Steady-state streaming: **completely uninterrupted**
- Bitrate reaches maximum quickly

Worst Case: Brief Startup Stall

In a minority of sessions, the startup phase is extended by a $\approx$1.6 s stall. This is comparable to normal network variability on many connections.

Steady-State Streaming Unaffected

Once the buffer target is reached, Dodge streams at maximum quality with no interruptions. The defended throughput pattern differs from undefended traffic, which is exactly the point.

Why Have Dodge in `dash.js`?

Why `dash.js` Is the Right Place

`dash.js` is the **reference open-source DASH player**, widely deployed by major content providers and platforms around the world.

Unmatched Reach

Merging Dodge into `dash.js` means that **opt-in privacy protection becomes immediately available** to every `dash.js`-based platform's users.

Minimal Code Changes

Our core additions to `dash.js` are relatively small can be easily isolated: extended manifest loader, cycle-based download loop, no new dependencies.

Dodge's Complementary Roles

1. **Feature flag/module**: Dodge as an optional module in `dash.js`, turned on via an extended manifest URL
2. **Reference implementation**: the fork as a documented reference for building defenses in other players
3. **Research testbed**: enabling community contributions of defense manifests and future standardization

What Adoption Would Look Like in Practice

For a privacy-conscious content provider:

1. Run a defense to generate extended manifests (offline for the entire catalog, or dynamically per-download)
2. Host extended manifests somewhere
3. Pass these manifest URLs to `dash.js`
4. **No CDN, server, or encoding changes**

For a user or browser extension:

1. Point `dash.js` to a self-hosted or community-maintained manifest
2. Works even without cooperation from the content provider
3. Seamless fallback to standard `dash.js` if needed

Mature Prototypes Already Available

- **Source**: publicly available on GitHub
- **Live demo**: a basic demo is available at the author's website
- **Full paper**: *Proceedings on Privacy Enhancing Technologies* (PoPETs) 2026

Open Questions — Feedback Is Much Appreciated!

While within reach, integration raises many practical questions that would benefit greatly from your expertise and experience in the WG, such as:

- Your opinions on the work in general, and the best ways forward
- Which content providers may be particularly interested in Dodge
- Whether standardization of some sort is a reasonable option
- How Dodge should interact with ABR algorithms, present and future
- Development considerations, such as making Dodge its own module

Our Position

We are excited to continue working on Dodge and support future work with development/maintenance, documentation efforts, and research support.

Conclusion

Summary

1. **Video fingerprinting** is a real and growing privacy threat against streaming users.
2. **Existing defenses** are not deployable by a `dash.js`-based platform today.
3. **Dodge** is a client-side, application-layer defense framework that:
   - Supports effective defenses with high QoE
   - Requires *no server or infrastructure changes*
   - Works as a *drop-in replacement* for `dash.js`
4. Merging Dodge into `dash.js` would bring **free, opt-in privacy protection** to millions of users, a significant step forward for video privacy on the Internet.

Resources

Paper (PoPETs 2026):
Available as PDF

Source code:
Available on GitHub

Live demo:
Available at author's website

Thank you.

Questions & discussion welcome.

Selected References

- Björklund & Duvignau. Endangered Privacy: Large-Scale Monitoring of Video Streaming Services. USENIX Security 2025.
- Carlson et al. Understanding and Improving Video Fingerprinting Attack Accuracy under Challenging Conditions. WPES 2024.
- Hasselquist et al. Raising the Bar: Improved Fingerprinting Attacks and Defenses for Video Streaming Traffic. PoPETS 2024.
- Schuster et al. Beauty and the Burst: Remote Identification of Encrypted Video Streams. USENIX Security 2017.
- Vaskevich et al. SMAUG: Streaming Media Augmentation Using CGANs as a Defence Against Video Fingerprinting. IEEE NCA 2023.
- Witwer et al. **Dodge: A Client-Side Framework for Application-Layer Video Fingerprinting Defenses.** PoPETs 2026.
- Zhang et al. Statistical Privacy for Streaming Traffic. NDSS 2022.