Raising the Bar: Improved Fingerprinting Attacks and Defenses for Video Streaming Traffic

David Hasselquist
Linköping University, Sweden
Sectra Communications, Sweden

Ethan Witwer
Linköping University, Sweden

August Carlson
Linköping University, Sweden

Niklas Johansson
Linköping University, Sweden
Sectra Communications, Sweden

Niklas Carlsson
Linköping University, Sweden

ABSTRACT

Despite the clear dominance of video streaming traffic on the Internet and the significant ramifications of disclosure of which videos users are streaming, video fingerprinting has received relatively little attention compared to other traffic analysis domains. Existing attacks are tailored to undefended traffic and mostly rely on a few manually crafted features. Meanwhile, potential defenses are ad hoc, often impractical, and typically only mentioned briefly. Drawing from progress made on website fingerprinting, we aim to improve current standards for attacks and defenses for video streaming traffic while highlighting a critical and underexplored issue on today's Internet. We show that directional and timing-based attacks that leverage CNNs are competitive with state-of-the-art video fingerprinting attacks, in many cases with far less training data. We also provide the first extensive study of potential defenses, which considers performance against attacks, overheads, and user QoE; and we present a novel defense design that boasts both broader applicability and greater efficacy than existing proposals.

1 INTRODUCTION

In a world increasingly reliant on online video and where disclosure of which videos users are streaming can have significant ramifications, the potential misuse of users' viewing habits extends far beyond individual privacy. Despite this, *video fingerprinting*, where an attacker eavesdropping on the encrypted connection between a client and video server tries to determine what content is being streamed, has received limited attention and almost no defenses have been proposed or evaluated for this attack scenario.

The status is quite different with *website fingerprinting*, where many traffic analysis research efforts have been invested in the last decade. These works typically consider a local, passive adversary that monitors a victim's connection to an encrypted tunnel, such as a VPN or Tor [19], with the goal of identifying the websites they are visiting. In this area, progress includes powerful attacks that leverage various deep learning architectures with automatic feature extraction [6, 56, 59, 62, 63]; many different defense approaches [9, 17, 22, 24, 26, 31, 34, 45, 46, 48, 52, 55]; public datasets that allow fair head-to-head comparisons [45, 52, 59, 63]; as well as several other theoretical and practical contributions [10, 28, 35, 41, 67, 68, 72].

Status of video fingerprinting research: Meanwhile, other domains, such as video fingerprinting, have seen relatively little advancement, with attacks typically employing heuristics or machine learning techniques on a few *manually crafted* features and potential defense schemes often being infeasible or only mentioned in passing. For example, despite several video fingerprinting attacks having been proposed and shown effective under some set of assumptions [3, 20, 21, 27, 42, 57, 60, 73, 74, 77], little work has been put into defenses. Existing suggestions [12, 60, 66, 76] often highlight a single feature, such as segment size or intervals between segments, overlooking other critical features exploitable in attacks; and implementing them would demand extensive modifications from system designers, highlighting the need for practical defense techniques that can be deployed seamlessly on existing infrastructure without burdensome efforts from service providers.

Other limitations of current state of the art: Two other large limitations with current video fingerprinting works are that most of them do not share the source code or datasets used for evaluation. Finally, to be practically applicable, we note that defenses must consider the tradeoff between the level of privacy protection they offer and the impact that they may have on the user's quality of experience (QoE). Despite utility being a central concept in privacy-enhancing technologies, the QoE has garnered almost no attention in the development of traffic analysis defenses thus far.

Design and evaluation of practical defenses: In this paper, we take several steps to address the above gaps and limitations. First, we turn our attention towards *network-layer* defenses. These operate independently below the application layer, shaping the flow of packets – a common strategy in the website fingerprinting setting. This has the benefit of removing the burden from service providers and providing more generalized protection for different implementations and setups. Second, we set out to *adapt* existing website fingerprinting defenses to video streaming traffic and *develop* a new tailored defense that is demonstrably successful against video fingerprinting. Third, to better evaluate the effectiveness of the defenses we test, we adapt attack and security estimation techniques from the website fingerprinting domain to work for video, and we show that the attacks often contend with the state-of-the-art while having far lower observation times. Fourth, we develop a large-scale evaluation framework that captures both the strengths of the different defenses and their impact on the user's QoE. Fifth, source code and datasets are shared with the paper. Finally, our

Proceedings on Privacy Enhancing Technologies 2024(4), 1–18

https://doi.org/XXXXXXX.XXXXXXX

evaluation provides the first comprehensive analysis of, and practical insights into, the effectiveness of different attacks and defenses in the video streaming context. Our results indicate that website fingerprinting defenses can be modified to provide some protection against video fingerprinting, but a targeted strategy can achieve much better results with lower overhead and QoE impact.

Overall, our contributions can be summarized as follows:

- We apply directional and timing-based website fingerprinting attacks to video traffic and show that they rival the state-of-the-art with far less training data.
- We implement video fingerprinting attacks from prior work and thoroughly evaluate them against our defenses.
- We develop a live evaluation framework and use it to collect *LongEnough*, a large-scale dataset containing both network traffic and QoE metric data in varying network conditions.
- We adapt two website fingerprinting defenses, FRONT and RegulaTor, to work with video traffic and assess their effectiveness with both simulations and live deployments.
- We develop and test a new targeted defense, Scrambler, which achieves competitive results against all evaluated attacks with moderate overhead and low QoE impact.
- We evaluate our defenses under variable bandwidth conditions, providing an indication of how poor network quality impacts the relative strengths of different types of defenses.
- We advocate for higher standards for video fingerprinting and pave the way for future defense work by releasing the source code and datasets used in the paper.

Outline: Section 2 provides background on traffic analysis and streaming standards. Section 3 outlines our data collection process, followed by Section 4, which presents our attacks. We then benchmark the constant-rate defense in Section 5; detail our adaptations of the website fingerprinting defenses FRONT and RegulaTor in Sections 6 and 7, respectively; and introduce our proposed defense, Scrambler, in Section 8. We conclude the paper in Section 9.

2 BACKGROUND AND RELATED WORK

2.1 Traffic Analysis

Traffic analysis refers generally to inferences made via observable patterns in (encrypted) communications, with prior work focusing primarily on using leaked information to harm user privacy. The most prominent example is *website fingerprinting*.

Website Fingerprinting: In website fingerprinting (WF), an adversary monitors traffic from some vantage point between a client and an anonymous communications service. This encompasses a broad range of potential actors, including malicious users on the same local network as the victim and the victim's Internet service provider, among many others. The attacker visits web pages of interest and collects the resulting packet traces – these observations comprise the *monitored set* and are used to train a classifier. He can then gather traces from a victim's connection for classification.

Attacks are typically evaluated in either the *closed world*, in which it is assumed a user will only visit pages in the monitored set; or the more realistic *open world*, requiring the adversary to determine if the visited page is in the monitored set before classifying further. These settings also apply to video fingerprinting. Since our main goal is to evaluate defenses, we opt for the closed world to provide an advantage to the attacker and approach an upper bound on attack accuracy. This is typical in WF, though other settings, such as the one-page setting [68], can provide improved worst-case analysis.

WF attacks: Early WF attacks used heuristics and manually crafted features, sometimes in combination with machine learning [30, 32, 33, 49]. However, over the past few years, researchers have shown that deep learning models such as CNNs [6, 56, 59, 62, 63] and Transformers [18, 37] offer greatly improved performance, especially against defended traffic, while eliminating the need for manual feature engineering. Techniques such as augmentation [4] have also been shown to improve attack performance. Even so, researchers have only recently shifted their focus to evaluating the severity of the WF threat in real-world conditions [11, 36], which pose challenges beyond those presented by artificial datasets.

Since deep learning-based WF classifiers make use of low-level features common to all types of traffic, such as packet sequences, directions, and timing, they can be adapted to domains beyond WF. To evaluate the effectiveness of WF classifiers against video, we adapt DF [63] and Tik-Tok [56], which we describe in more detail in Section 4. Notably, these attacks rely on packet sequences, allowing us to evaluate the utility of packet sequences for video identification. They are matched only by RF [62] (which has strong similarities with a video fingerprinting attack we test in terms of both input format and model structure) and Transformer-based attacks that are tailored to multi-tab browsing scenarios [18, 37].

WF defenses: Many defenses against WF have been proposed and scrutinized. Though some employ other techniques [17, 31, 43], most are *network-layer* defenses. These directly shape the flow of traffic by either (1) delaying packets or (2) introducing padding packets that are indistinguishable from those carrying real application data. Several approaches exist to do this, such as random padding [24, 38, 52], regularization/constant-rate traffic [9, 22, 34], imitation/supersequences [48, 69, 70], and adversarial machine learning [26, 46, 55]. Each one supposes a unique tradeoff between effectiveness, overhead, and other requirements [45].

Even though deep learning classifiers for WF need few modifications to be effective against video traffic, different defenses are required due to the continuous and periodic nature of video streaming, as opposed to discrete webpage downloads which consist of a single instance of back-to-back resource requests and responses. We evaluate two WF defenses that employ random padding and regularization, respectively, with changes to make them suitable for protecting long-lived video streams. FRONT [24] is a padding-only defense that has the goal of masking useful features that are present near the beginning of a webpage download trace. RegulaTor [34], on the other hand, regularizes traces by specifying a more rigid traffic pattern. We adapt FRONT and RegulaTor to video traffic and also design a new defense, which adopts a hybrid approach.

Frameworks: Research in traffic analysis has traditionally been an arms race, with attacks and defenses succeeding each other regularly – this has inspired the development of *frameworks* that provide a platform for defenses. Tor contains a "circuit padding framework" [50], and Gong et al. [25] propose WFDefProxy, which allows defenses to be deployed with the aid of Tor bridges [51]. Pulls and Witwer present the standalone Maybenot framework [53], which uses a probabilistic finite state machine model inspired by Tor's circuit padding framework to support network-layer defenses.

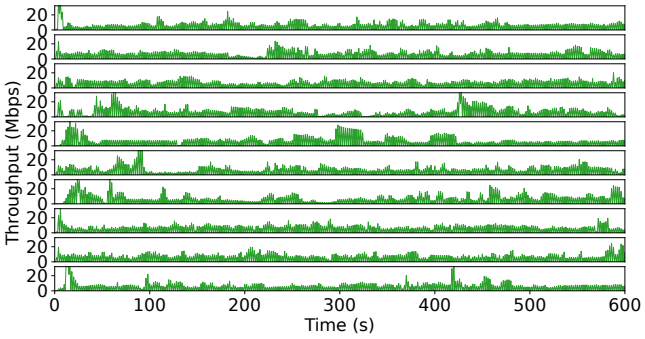


Figure 1: Throughput of 10 video streams using MPEG-DASH

It manages *machines* which operate on *events* related to a channel, such as packets sent or received, and produce *actions* as output – e.g., block/delay outgoing traffic or inject padding. In this work, we use Maybenot to implement all defenses and also provide the corresponding machines. We perform simulations via the Maybenot simulator [53] and integrate Maybenot with our deployments.

Other types of traffic analysis: Traffic analysis has also been shown to pose a significant privacy threat in other settings. For example, a local, passive adversary can determine smartphone app usage with high accuracy [13, 40, 65]; identify live streamed content, VoIP services, and voice assistant traffic including fine-grained usage details [1, 2, 8, 29, 39, 71]; and determine administrators, group members, and usage from instant messaging traffic [5, 23].

2.2 HTTP-Based Streaming

MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [64] is a widely adopted standard for streaming recorded and live video, with prominent providers such as Netflix [58] and YouTube [77] employing variations of the protocol. With MPEG-DASH, a content provider encodes a video file with several different bitrate settings and then splits each version into a number of *segments*, which typically have a fixed duration of a few seconds.

To stream the video, a client must first download a Media Presentation Description (MPD) containing information about the segments stored by the server. Using this information, the client can then request the segments individually using HTTP(S). To achieve smooth playout and improve robustness in unfavorable network environments, the client can vary the quality of the requested segments based on factors such as buffer status and perceived network conditions, a technique known as *adaptive bitrate streaming*.

Variable bitrate: To further save bandwidth, video streams are typically encoded using variable bitrate (VBR) encoding. With VBR, the bitrate of a video stream is adjusted based on the complexity of each scene in the video. Scenes with minimal motion or static images require lower bitrates, as they can be compressed more effectively without sacrificing quality, whereas action-packed scenes and those with intricate details demand higher bitrates.

Consequently, when streaming videos over a network, the throughput needed varies depending on the streamed content. To illustrate this concept, Figure 1 shows the throughput of 10 different streamed videos. Even though the traffic is encrypted, it is clear that an attacker can distinguish video streams by their network patterns.

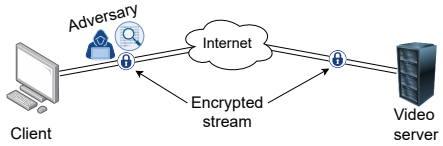


Figure 2: Video fingerprinting threat model

In our work, we use the MPEG-DASH standard with variable bitrate encoding when streaming videos. When encoding a video, we do so using an *average* bitrate throughout the video.

Quality of Experience: One significant aspect of our work is its focus on quality of experience (QoE). As the name suggests, this refers to the user's perception of the quality of the video playback. Though some works have endeavored to provide a unified quantitative definition of QoE, taking into account multiple factors (e.g., [44]), these are more useful in the context of optimization, whereas player-reported metrics are more informative when evaluating the utility of a system, as they are indicative of specific causes of poor streaming quality. The metrics include buffer size (amount of data temporarily stored to ensure uninterrupted playback during streaming), live latency (delay between recording and playback for live streams), playback rate (video content speed), and bitrate (video quality). We also extract occurrences of video player wait (small temporary pauses while buffering data), stall (playback halting due to a complete lack of data), seek (playback jumping to a different point), and quality switch (dynamic quality adjustments of DASH).

2.3 Video Fingerprinting

Threat model: The goal of video fingerprinting is to discern which video a user is watching by monitoring their encrypted connection to a video server. Figure 2 shows the threat model for this setting. Similar to WF, we assume a local, passive adversary able to observe all packets on the network and that the packet payload is secure. We note that the threat remains relevant regardless of whether a direct connection is employed: while an adversary could most likely use the destination IP address to identify the service provider, this does not reveal the specific videos being watched.

Attacks: In contrast to web traffic, it is not strictly necessary to have sample traces for a monitored set of videos. Though some attacks, particularly deep learning-based ones, do use sample traces, a few attacks are based on the notion that the possible sequences of segment sizes for any prerecorded video can be computed ahead of time and matched against the observed segment downloads.

Reed et al. [57, 58] propose to build a k-d tree with keys based on segment size statistics to identify candidate videos, with further matching performed using the correlation coefficient. Gu et al. [27] present a comparable attack that uses burst size *differentials* instead of raw values. They compute these for each reference video and extract them from captured video streams with a heuristic based on packet times; identification is done using a modification of Dynamic Time Warping. In this work, we compare our attacks with those by Reed and Klimkowski [57] and Gu et al. [27] since they are representative of different heuristic approaches to video fingerprinting and provide useful insights. We note that a plethora of similar attacks that use segment sizes have also been proposed [20, 73–75, 77].

Finally, some attacks use collected traces to build a training set. Dubin et al. [21] test nearest-neighbor classifiers and a support

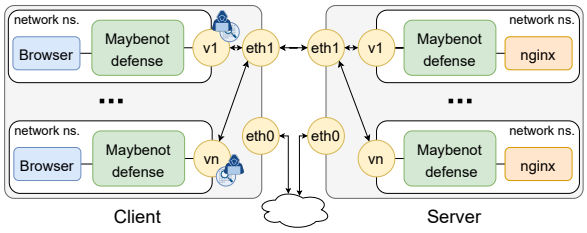
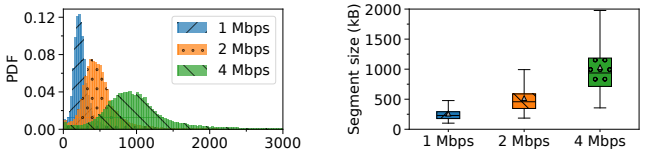


Figure 3: Overview of our data collection framework



(a) Probability density function (PDF) (b) Box plot

Figure 4: Segment size distribution for each quality level

vector machine, reporting high accuracy with features representing similarity between sets of segment sizes. Schuster et al. [60] use a CNN with time series of bytes/packets per second or average packet sizes as input. Li et al. [42] follow up on this work, employing similar features with several deep learning architectures against sniffed wireless traffic. Bae et al. [3] similarly propose a CNN with a time series of traffic volumes to identify videos on LTE networks. We implement Schuster et al.'s [60] attack as a point of comparison.

Defenses: Existing mentions of possible defenses mostly relate to the modification of segment sizes. Two common suggestions are to store videos with a constant bitrate (CBR) encoding or to constrain the VBR encoding [57, 60], resulting in a high QoE impact. It has also been proposed that segments should be requested at varying intervals [57, 60], but this would likely be ineffective against modern attacks. One promising proposition is to pad segments [21, 60, 77] – randomly or to some maximum size – but even this could likely be defeated on its own in the face of timing-based attacks [56].

Zhang et al. [76] evaluate adversarial machine learning to defend video traffic, showing that adversarial samples are not a robust solution. They also consider two ε-differential privacy approaches: d^*-privacy is effective, but at very high overheads; as such, the Fourier perturbation algorithm is their most promising proposal. However, it has an unclear impact on QoE, does not address packet timing, and requires machinery beyond Maybenot. Vaskevich et al. [66] propose using conditional GANs to generate traffic traces and shaping streams to match them, but this too has an unknown effect on QoE and presents significant deployment challenges.

3 DATA COLLECTION

3.1 Framework

Due to the lack of (1) public video fingerprinting traffic datasets and (2) fingerprinting datasets containing QoE metrics, we collect new datasets containing both of these and make them publicly available.[1] Figure 3 shows an overview of our data collection framework, which uses two physical machines to represent the client and server.

On the client side, we launch n data collection instances in parallel, each isolated in its own network namespace behind a virtual interface (v1,v2,...,vn). We use the dash.js [16] reference client implementation of MPEG-DASH, allowing us to retrieve QoE metrics, a key consideration when evaluating defense proposals. Each data collection instance contains a browser operated using Selenium [61] that receives a live low-latency stream via dash.js. In addition, each instance may contain a defense implemented with the Maybenot framework [53], adding or delaying packets; in the case of no defense, the browser sends packets directly to the virtual interface.

[1] https://github.com/trafnex/raising-the-bar

On the server side, we use corresponding network namespaces for each data collection instance. Here, each namespace contains an instance of an nginx web server [47] that serves video content and, as needed, a server-side defense implemented with Maybenot.

Our two machines each have two physical network interfaces, one of which (eth0) is connected to a management network to control the data collection; the other (eth1) directly connects the two machines. Packets from each namespace on one machine are routed to the corresponding namespace on the other machine over the interface that directly connects them (eth1). Furthermore, we add a 5 ms delay on each packet exiting a network namespace. This results in a round-trip time (RTT) of about 11 ms, with 10 ms of these added virtually. We also cap each virtual interface at 100 Mbps.

Due to our machine limitations, we run five data collection instances in parallel. Both machines run Ubuntu 22.04 on an 8-core Intel CPU (Xeon E3-1230 v6 @ 3.50 GHz) with 16 GB RAM and each network interface supporting 1 Gbps full duplex.

3.2 Datasets

Evaluation of WF attacks and defenses is typically done with a closed world of around 100 websites [9, 24, 34, 38, 56, 62, 63]. Using a similar setting, we collect the new *LongEnough dataset* containing both network data and QoE metrics. To consider a wide variety of videos, we select the first 100 available movies on YouTube from the playlist "The Best Free Movies on Youtube" [7].

We obtain the first 10 minutes of each movie and encode it using MPEG-DASH [64] into two-second segments with three average bitrates {1, 2, 4} Mbps. Figure 4 shows the segment size distribution for each quality. We observe an overall mean of 604 kB across all three quality levels and that segment size is proportional to bitrate: doubling the bitrate roughly doubles the segment size. Average segment sizes are 260 kB with 1 Mbps bitrate, 518 kB with 2 Mbps, and 1033 kB with 4 Mbps, though some may be smaller (e.g., still scenes) or larger (action-packed scenes). We refer to Appendix A for further analysis of video characteristics in our dataset.

After encoding, the segments are made available on our nginx server. Using a dynamic MPD file, we serve the live stream as described previously. Similar to current public WF datasets [45, 52, 59], we store packet timing, directions, and sizes as observed over the network for each trace. In addition, and in contrast to previous works, we collect QoE metrics for the stream as reported by dash.js.

For undefended traffic, we stream the 100 videos using 10 different starting points for the stream and collect 10 samples per video and starting point. This results in a total of $100 \times 10 \times 10 = 10\,000$ samples. We stream up to 10 minutes for every sample but offset the starting points by one minute, giving a total of $10 \times \sum_{i=1}^{10} i = 550$ minutes per video. Combined, our undefended dataset results in $100 \times 550 = 55\,000$ minutes $\approx$ 917 hours of streaming.

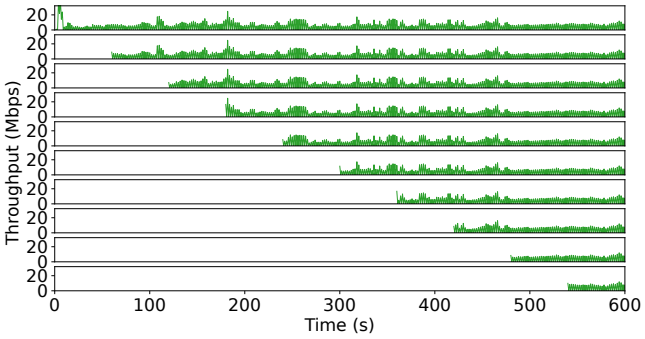


Figure 5: Throughput of same video with different offsets

To illustrate our concept of offset, Figure 5 shows the throughput of the same video, collected independently 10 times using different offsets. This represents a scenario in which users start the same live stream at varying points; we highlight that the observed network traffic is still similar across the independent data collection sessions.

Other user behavior, such as pausing the stream and seeking to different points, may require modifications to attacks to avoid performance decreases. Since our goal is to approach a worst-case evaluation of the defenses we test, we leave such considerations to future work and instead focus on streams that a user begins watching at a certain point and continues watching until completion.

While we mostly simulate defenses to quickly estimate performance, we also collect defended traces from our physical machines to obtain QoE metrics, resulting in the *LongEnough-defended* dataset. For each of the 28 defense configurations described later (1 for constant rate, 9 for Adapted FRONT, 9 for Adapted RegulaTor, and 9 for Scrambler), we stream each video once, without any offset. This results in $28 \times 100 \times 10 = 28\,000$ minutes ≈ 467 hours of streaming.

In addition, to better capture the impact of real-world network effects, we collect the *LongEnough-variable* dataset by streaming videos under variable bandwidth conditions. This dataset contains an additional 917 hours of data – we perform further evaluations of the defenses we present throughout the paper using it and present the results and analysis in Appendix B.

Combining both our undefended and defended datasets, and including the dataset with variable bandwidth conditions, *LongEnough* contains 2 300 hours of streamed network and QoE data.

4 ATTACKS AND UNDEFENDED TRAFFIC

We aim to provide a benchmark for the defenses we present throughout the paper by adapting WF attacks to video, evaluating them against undefended traffic, and testing prior video fingerprinting attacks that are based on a variety of identification strategies.

4.1 DF and Tik-Tok

We adapt DF [63] and Tik-Tok [56] – which use the same CNN trained on *directional* and *directional time* sequences, respectively – to video traffic. In the directional case, −1 is an incoming packet (to the client) and +1 an outgoing packet, for the first N packets in a trace; zero padding is used for shorter traces. Directional time is derived by multiplying each directional value by a timestamp. Also, we employ DeepSE-WF [67] to measure Bayes error rate and information leakage of the latent feature spaces of the two attacks.

Table 1: Attack accuracy and security estimation for different input sizes N and corresponding observation times

N	Time (s) $\mu \pm \sigma$	Attack accuracy DF	Attack accuracy Tik-Tok	Security est. BER	Security est. MI
5 000	15 ± 3	0.831	0.916	0.041	6.509
10 000	22 ± 9	0.928	0.951	0.024	6.603
15 000	29 ± 13	0.944	0.960	0.019	6.627
20 000	35 ± 16	0.962	0.966	0.016	6.646
25 000	40 ± 17	0.963	0.970	0.014	6.650
30 000	43 ± 17	0.966	0.970	0.013	6.660

With DF and Tik-Tok, each video is considered a unique class, and its 100 traces are individual data points. We truncate each trace to consider only the last 60 seconds of data transmission, and the first N of these packets are used as input to the classifiers. Though in practice an attacker may begin monitoring a stream at different points in time, using a constant offset represents an advantage for the attacker, which is desirable when evaluating defenses as it moves us closer to an *upper bound* on their protection. We use a 0.8/0.2 train-test split and 5-fold cross-validation for both models.

Hyperparameters: We find the default DF hyperparameters to be appropriate for video traffic, with the exception of input size: the canonical choice is 5k in WF, but given the distinct nature of video traffic, higher values can be far more effective. We show this by running the attacks and DeepSE-WF with $N = \{5\text{k}, 10\text{k}, ..., 30\text{k}\}$. Table 1 shows the attack accuracy as well as Bayes error rate (BER) and mutual information (MI) using different input sizes N. Longer packet sequences are strongly correlated with better attack performance and security estimation: DF's accuracy is only 83.1% with $N = 5\text{k}$, but it rises by nearly 10% when N is increased to 10k, with further improvements at higher values. Though Tik-Tok has better baseline performance, its accuracy also increases with N, from 91.6% with $N = 5\text{k}$ to 97.0% with $N = 30\text{k}$. Since the sharpest increase is seen between $N = 5\text{k}$ and 10k with both attacks, we adopt $N = 10\text{k}$ for analyzing defended traffic in subsequent sections.

Observation time: Given that *observation time* has traditionally been used to determine the amount of data provided to a video fingerprinting attack (since this determines the number of segments captured), we also analyze the relationship between packet sequence lengths and elapsed time. Table 1 shows mean observation time and standard deviation over different input sizes N. With the default DF input size of 5k, the mean observation time is about 15 seconds. However, the required capture time and standard deviation increase with input size: the mean time to reach 10k packets is about 22 seconds, and for 30k packets, 43 seconds of observation is needed on average. Despite this, more than 25% of videos still reach 30k packets in less than 30 seconds. In all cases, the expected observation time remains lower than in most previous attacks.

Summary: With only 15 seconds of observation on average, Tik-Tok can reach 91.6% accuracy on undefended traffic. Moreover, accuracy increases with observation time: this is especially true for DF, with a near 10% increase (from 83.1% to 92.8%) with a mean of just 7 seconds of further observation. Note that these values apply to the case of two-second segments, and different segment durations may yield different results. However, the primary takeaways are universal: high accuracy can be achieved in relatively little time, and an attacker who can observe for longer is at a significant advantage.

4.2 Comparison with Other Attacks

We also evaluate some previously presented video fingerprinting attacks [27, 57, 60]; as we could not obtain the code from the authors, we implement the attacks ourselves and share the code publicly.

Leaky Streams: We implement the Leaky Streams attack by Reed and Klimkowski [57], with some slight modifications, as a representative attack that uses features based directly on *segment sizes*. When building a video's fingerprint, we include only the last 90 seconds (45 segments), both for performance and since any matches found earlier would be false positives anyway. During testing, we take the last 60 seconds of each trace as input, with a slight offset when needed to align with two-second periods. This allows for better comparison among attacks and eliminates the need to account for different possible alignments of segment boundaries.

We also find that the thresholds used by Reed and Klimkowski for the k-d tree range search give low accuracy with our dataset. To account for this, we tune the attack by fixing the correlation needed for a match at 0.9 and testing different thresholds on one randomly selected trace per video (consistent across runs). By searching the space [0.01, 0.1] for the first threshold and [0.0001, 0.9] for the second, we see the best settings are $\{0.07, 0.08, 0.09, 0.1\}$ and 0.03. All combinations yield 91.0% accuracy; we select 0.07 for the first threshold since it is the most restrictive and should thus minimize false positives. We further evaluate this choice with 100 traces per video: similarly, accuracy is 90.8%. Note that fast mode is never invoked: two windows cannot overlap by more than five segments for this to occur, so it is not possible with our setup.

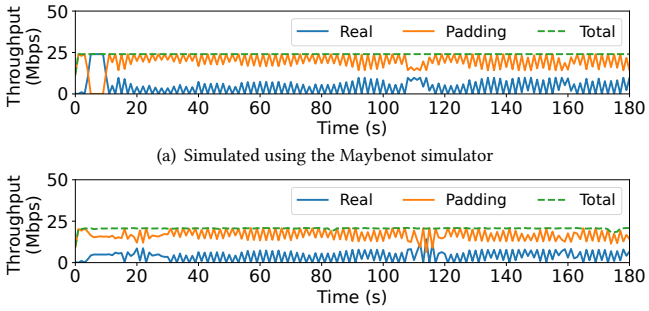
Walls Have Ears: We also implement the Walls Have Ears attack by Gu et al. [27] due to its use of segment size *differentials*. We use only the final 90 seconds (45 segments) of a video in its fingerprint and the last 60 seconds of each trace for evaluation. To better reflect the alignment of segment boundaries, we modify the estimation of segment sizes (Algorithm 1 in [27]) to count backwards from the end of a trace. We also constrain the P-DTW algorithm to only allow matches between subsequences of the same length; this is reasonable for undefended traffic since segment transmissions in steady-state streaming typically do not exceed two seconds. Finally, we omit the false rate threshold used by Gu et al. and only use the predicted label. With 100 traces per video, we attain 95.9% accuracy.

Beauty and the Burst: Lastly, we implement the CNN-based Beauty and the Burst attack by Schuster et al. [60] using Keras and TensorFlow v2.15.0. This attack differs from DF and Tik-Tok in that it uses coarse-grained time series as input instead of packet sequences. From preliminary testing, we select 500 epochs and a learning rate of 0.001. Using the same 0.7/0.3 train-test split as Schuster et al., we obtain 99.9% test accuracy with their PPS (packets per second) features over the last minute of each trace.

5 CONSTANT-RATE DEFENSE

Ignoring overhead and impact on QoE, the constant-rate defense, in which packets are simply sent at a constant pace, is provably optimal and can be considered the maximal defense [9, 22]. Thus, we evaluate it to provide a baseline for other defenses.

In theory, the defense leaks no timing information – the inter-packet delay is a fixed value – and since the videos in our dataset all have the same duration, the number of packets per trace will also be the same. Note that we additionally pad all packets to the MTU of 1 500 bytes, which prevents packet sizes from leaking information.



(a) Simulated using the Maybenot simulator

(b) Collected over physical machines using a Maybenot implementation

Figure 6: Throughput using constant-rate defense

Given that the rate selected does not affect the defense's theoretical properties, we elect to send at 2.4 Mbps on the client and 24 Mbps on the server. Though more precise tuning may be desirable in a real-world deployment scenario to decrease overhead, these settings are sustainable on our 100 Mbps links and allow all but a few colossal segments to be sent within their two-second windows.

Evaluation with CNNs: With an input size of 10k packets, DF has 1.0% accuracy and Tik-Tok achieves only 0.9% (close to random). Here, BER is 0.867 and MI only 0.119 bits. Similarly, Beauty and the Burst achieves only 0.67% test accuracy with the PPS features. This aligns well with our expectations: no information is leaked by the constant-rate defense, and any amount of observation should yield no benefit to the attacker. This makes the constant-rate defense an optimal choice for users seeking the best possible protection, as long as its QoE impacts and high overhead (462.45% receive and 518.47% overall bandwidth overhead) are considered acceptable.

Note that the BPS (bytes per second) features for Beauty and the Burst are equivalent after normalization since we pad to MTU; for the same reason, the PLEN (average packet size) features do not add any value. Thus, we focus on PPS for the remainder of the paper.

Evaluation with heuristic attacks: The Leaky Streams attack achieves 1.0% overall accuracy. This is also the case with Walls Have Ears, even after relaxing the condition that a query sequence can only be matched against reference sequences of the same length in P-DTW. Instead, we enforce that reference sequences should be *greater than or equal to* the length of the query sequence: adding padding between segments can collapse multiple segments into one period during Algorithm 1 in [27], but the opposite is not true.

Comparison to simulated traces: Figure 6 shows the defense's throughput for (a) the simulated case in Maybenot and (b) when run over physical machines. Here, the simulated case represents the theoretical defense, sending packets at an exact rate with equal size. However, we see some small differences during deployment. First, we note that throughput is about 4 Mbps lower due to Maybenot not accounting for code execution time. Second, we note some small variations in the sending rate. It is important to clarify that these arise from machine-specific performance fluctuations and are not correlated with packet contents. Therefore, we claim that our constant-rate implementation does not leak any *useful* information.

We also observe a significant deviation in the ratio of real packets to padding packets, primarily at the beginning of a trace. When

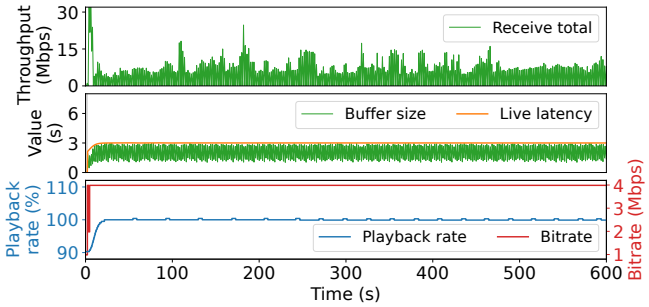


Figure 7: Throughput and video player QoE metrics for an undefended trace

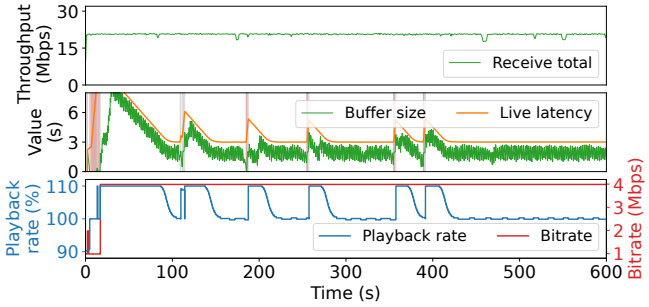


Figure 8: Throughput and video player QoE metrics using constant-rate defense

there is real traffic to send, the simulator sends it at a maximum pace and sends no padding packets; in contrast, on physical machines, we see padding being sent despite there being real packets available. This is due to congestion control, which tries to better control the entry of packets into the network and avoid congestion collapse.

Quality of Experience: To better understand how the user is affected, we also study the QoE metrics reported by the video player. For comparison, Figure 7 shows throughput and QoE metrics (buffer size, live latency, playback rate, and bitrate) for an undefended trace. We see that (1) throughput again varies over time; (2) buffer size fluctuates just below the targeted live latency of 3 seconds; (3) the playback rate is set at 100% after some initial adjustments; and (4) the only quality switch occurs at the beginning due to many segments being sent over an initially low congestion window.

On the other hand, Figure 8 shows throughput and QoE metrics when using the constant-rate defense. At the top, we again observe content-independent throughput drops. In the middle, we see periods with a temporary pause in playback while the player buffers more data (gray background) intermingled with periods in which playback has stalled (red), indicating a more severe issue: playback has completely stopped due to a lack of available data. After these periods, live latency increases and buffer size shortly follows.

With the increase of live latency, we also observe the playback rate increase to its maximum (110%), pushing the live latency down towards its target of 3 seconds. As live latency decreases, so does playback rate, eventually reaching 100%. With the exception of three quality switches in the beginning because of initial buffering, quality remains at the maximum of 4 Mbps, even with the occurrences of playback temporarily stopping throughout the stream.

Defense summary: While the constant-rate defense provides good protection, it comes at the cost of high bandwidth overhead,

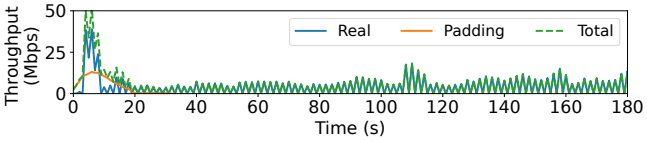


Figure 9: Throughput using FRONT defense with padding budget $N = 25\,000$ and padding window $W = 14$

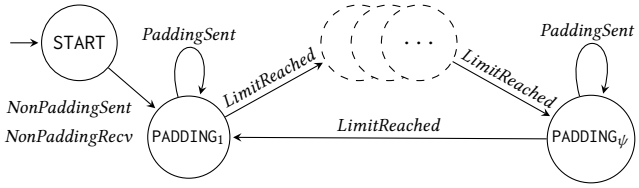


Figure 10: Adapted FRONT machine

and the user QoE is negatively impacted by short time periods in which the video player is paused, waiting for an additional buffer.

6 ADAPTED FRONT

FRONT is a padding-based defense against WF attacks [24]. Its operation consists of sampling time values from a Rayleigh distribution and sending padding at those times relative to download start. This results in a high volume of padding in the first few seconds, with the goal of masking useful features near the "front" of a trace.

Before a download, the client and server both sample a *padding count* from a discrete uniform distribution: $n_c = U([1, N_c])$, $n_s = U([1, N_s])$. N_c and N_s are parameters of the defense, representing the client and server's respective padding budgets. Also, both sides sample a *padding window* from a continuous uniform distribution: $w_c, w_s = U([W_{min}, W_{max}])$, where W_{min} and W_{max} are parameters that determine the range of possible windows for both parties.

After this, the client and server sample n_c and n_s values from Rayleigh distributions with $\sigma = w_c$ and $\sigma = w_s$, respectively. By subsequently sending a padding packet at each of these time offsets relative to download start until the download has finished, FRONT introduces randomness to the traffic flow to confuse classifiers.

Adapting FRONT: Due to the nature of website downloads and WF classifiers typically being trained on the initial part of a trace, one-shot defenses such as FRONT can provide effective protection in that context. However, video streaming presents different challenges: the adversary may have access to much more training data and, consequently, options regarding which part of a trace to use as input to a classifier. This renders such defenses useless over time. As an example, Figure 9 shows the throughput of a video when applying FRONT. The defense produces no padding or perturbation whatsoever after around 25 seconds, even with a high value of N.

To account for this, we test a straightforward modification of FRONT called *Adapted FRONT*, which simply repeats the defense throughout a stream; Figure 10 shows its Maybenot machine. We leverage an existing implementation, Maybenot FRONT [53], that uses a pipeline of `PADDING` states to approximate the Rayleigh distribution. Its parameters are N, corresponding to the padding budget; W, which is the maximum padding window; and ψ, the number of `PADDING` states. We adapt Maybenot FRONT to video by replacing the last transition $\texttt{PADDING}_\psi \rightarrow \texttt{StateEnd}$ with $\texttt{PADDING}_\psi \rightarrow \texttt{PADDING}_1$ so that the machine continues to pad indefinitely.

Table 2: Evaluation with selected parameters for Adapted FRONT defense

Parameters		Bandwidth overhead			Attack accuracy			Security estimation		Quality of Experience									
										Occurrences				Activity per quality					
N	W	Send (Mbps)	Receive (%)	Overall (%)	DF	Tik-Tok	Beauty	BER	MI	Wait	Stall	Seek	Q. sw.	Wait 1k	Wait 2k	Wait 4k	Play 1k	Play 2k	Play 4k
4 000	12	0.98	22.92	46.76	0.818	0.833	0.993	0.077	6.312	1	0	0	2	0.3%	0.0%	0.0%	0.2%	0.0%	99.4%
4 500	14	1.03	24.11	47.10	0.818	0.831	0.994	0.080	6.309	1	0	0	2	0.3%	0.0%	0.0%	0.2%	0.0%	99.4%
2 500	7	1.00	23.36	48.25	0.808	0.843	0.996	0.082	6.315	1	0	0	2	0.3%	0.0%	0.0%	0.2%	0.0%	99.4%
3 500	5	2.08	48.49	97.09	0.735	0.784	0.984	0.120	6.048	1	0	0	2	0.3%	0.0%	0.0%	0.2%	0.0%	99.4%
5 000	7	2.09	48.75	99.19	0.733	0.780	0.986	0.123	6.042	1	0	0	2	0.3%	0.0%	0.0%	0.2%	0.1%	99.4%
6 000	9	2.08	48.65	92.93	0.728	0.761	0.990	0.124	5.979	1	0	0	2	0.3%	0.0%	0.0%	0.2%	0.0%	99.4%
5 500	2	8.29	193.53	385.50	0.420	0.494	0.966	0.349	4.659	1	0	0	2	0.3%	0.0%	0.0%	0.2%	0.0%	99.4%
6 000	2	8.95	208.88	414.58	0.413	0.477	0.961	0.370	4.568	1	0	0	2	0.3%	0.0%	0.0%	0.2%	0.0%	99.4%
6 500	2	9.75	227.69	453.65	0.375	0.431	0.966	0.390	4.411	1	0	0	2	0.3%	0.0%	0.0%	0.2%	0.1%	99.4%

6.1 Tuning

To explore Adapted FRONT's potential to defend video traffic, we tune the defense by exhaustively searching through the parameter space $N = \{500, 1\,000, ..., 6\,500\}$ and $W = \{2, 3, ..., 14\}$. We select the ranges for N and W based on prior work [24, 53] and preliminary testing; ψ is fixed at 30 in all cases, as this yields an acceptable approximation of the Rayleigh distribution [53]. For each of the resulting 169 settings, we simulate the defense 100 times and record mean bandwidth overhead, attack accuracy, and BER/MI.

Evaluation: Given that the configurations we simulate result in different tradeoffs between overhead and protection, we divide them into three categories of roughly equal size for further analysis. Due to our focus on QoE, we do this by way of the overall bandwidth overhead, since it is the only metric from our simulations that may predict user experience impact. We then select the best three candidates from each group: *low overhead* (less than 50%), *moderate overhead* ([50, 100)%), and *high overhead* (100% or more). We use BER for this, as it has the best correlation with attack accuracies.

Table 2 shows the results of the tuning. Adapted FRONT appears to be moderately successful against DF and Tik-Tok, but it only excels with a prohibitive amount of padding. Over 385% overall overhead is needed to increase BER to 0.35, and none of the best three moderate configurations reduce accuracy to below 72.8%. In fact, Tik-Tok even achieves 43.1% on the best configuration, whose overhead nears 500%; BER remains below 0.40 and MI above 4 bits.

Note that we use $N = 10$k to balance the effects of using too few packets (underrepresenting an attacker's maximal capabilities) and too many packets (less apparent differences in accuracy). Considering the analysis in Section 4, this does not correspond to more than a few seconds in an *undefended* trace, and with much padding, even less observation is needed to capture the same number of packets. An attacker could defeat even the best configuration by increasing N: with 30k, DF's accuracy rises to 76.2% and Tik-Tok's to 83.2%.

Indeed, with Beauty and the Burst, we achieve a minimum of 96.1% accuracy across all configurations. Though this is due in large part to longer observation time, it is clear that both packet sequences and time series of packet volumes can be rather effective against padding-based approaches. We conclude that Adapted FRONT is ineffective even with exorbitant amounts of overhead.

6.2 Further Evaluation and Discussion

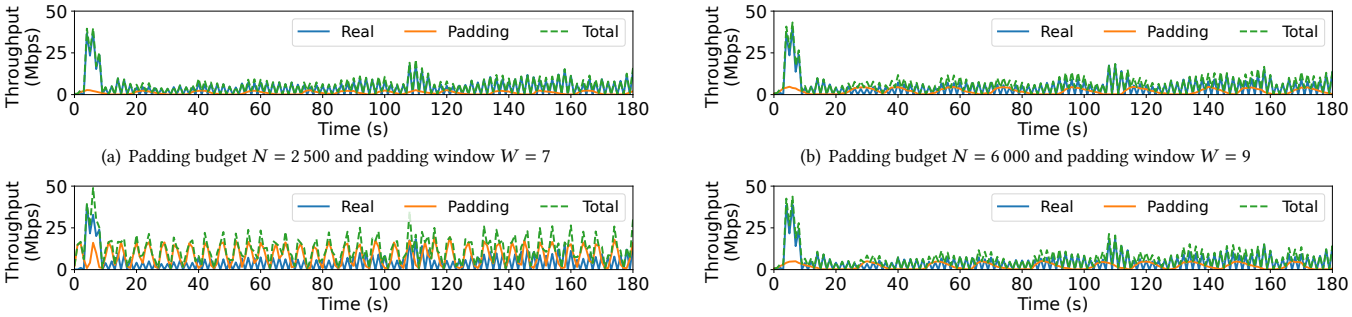
Evaluation with heuristic attacks: For all nine configurations in Table 2, we repeat the tuning of Leaky Streams, but with every threshold setting, we find that it only achieves 1.0% accuracy. Walls Have Ears is more effective, reaching 53.1% accuracy with the lightest configuration and 59.1% with ($N = 4\,500$, $W = 14$). This is because small amounts of padding do not significantly alter segment size differentials. Also, Walls Have Ears seems to perform better against configurations with higher W, even when N is increased slightly: we attribute this to trace-to-trace randomness and the fact that padding is more spread out over time with larger windows, reducing the overall distortion of the sequence of segment sizes. However, the attack degrades with more padding, with accuracy reduced to 4.8% on the weakest moderate configuration.

Throughput: To better understand the behavior of Adapted FRONT, Figures 11(a)–11(c) show the receive throughput for the best configuration in each overhead category. The throughput peaks corresponding to segment transmissions are still clearly visible with low overhead (Figure 11(a)). In contrast, with high overhead (Figure 11(c)), the peaks are more hidden but still rather distinguishable.

Comparison to simulated traces: When comparing the throughput of simulated traces vs. those collected over physical machines (Figure 11(d) to 11(b)), we observe minimal differences that can be accounted for by expected trace-to-trace randomness. That is, FRONT follows a simple padding scheme that only depends on the two sampled parameters n and w and not the video being defended.

Quality of Experience: Table 2 also shows the QoE metric values of each configuration averaged over all videos. Overall, we see that QoE is unaffected by the defense: playback is still at the highest quality for over 99% of the streaming time. This is because FRONT neither delays packets nor injects enough padding to exhaust our 100 Mbps link. Regardless of configuration, there is only one occurrence of *wait*, in the buffering period when initially loading the stream. Furthermore, we see on average two quality switches at this time; this is due to bulk segment downloads, as shown in Figure 7.

To illustrate this, Figure 12 shows the QoE for the best moderate configuration (Figure 11(b)). As with undefended traffic, buffer size fluctuates just below live latency and never reaches zero. Also, after some initial adjustments, playback rate stays at 100% and bitrate



(a) Padding budget $N = 2\,500$ and padding window $W = 7$

(b) Padding budget $N = 6\,000$ and padding window $W = 9$

(c) Padding budget $N = 6\,500$ and padding window $W = 2$

(d) Simulated: padding budget $N = 6\,000$ and padding window $W = 9$

Figure 11: Throughput using Adapted FRONT collected over physical machines (a-c) and via simulation (d)

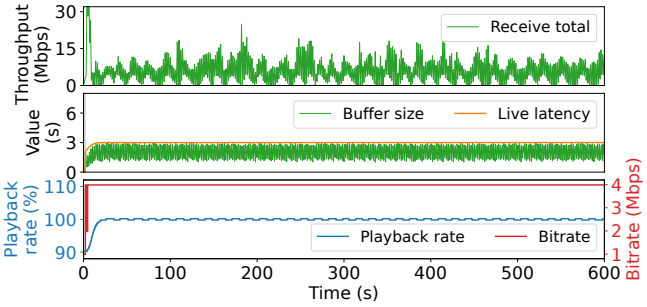


Figure 12: Throughput and video player QoE metrics using Adapted FRONT with $N = 6\,000$ and $W = 9$

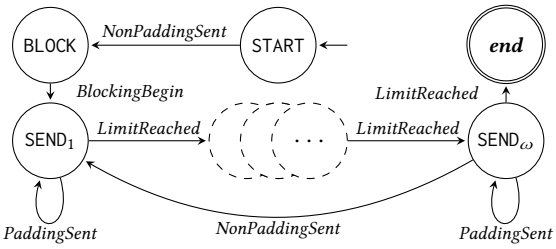


Figure 13: Adapted RegulaTor server-side machine

at 4 Mbps. For the best low- and high-overhead parameters (Figures 11(a) and 11(c)), we refer to Figures 23 and 24 in Appendix C.

Defense summary: The Adapted FRONT configurations we test do not negatively impact user QoE. As the defense is padding only, the same quality can be achieved as with no defense if there is sufficient bandwidth. However, in all cases, the defense leaks sensitive information and fails to adequately reduce attack accuracy.

7 ADAPTED REGULATOR

RegulaTor is a regularization defense tailored to WF against Tor [34]. As Tor traffic contains *surges*–spikes in packet volume surrounded by periods of lower activity–the defense begins by sending data at a high rate, which decreases until the number of queued packets exceeds a threshold. Then, the rate is reset, and the process repeats.

On the server side, RegulaTor accomplishes this by maintaining a sending rate of $R(t) = R_0 \cdot D^t$ packets per second, where R_0 is the *initial rate*, D is the *decay parameter*, and t is the time elapsed since the beginning of the current surge. The first surge begins when a download starts, and a new surge begins whenever the number of queued packets exceeds $T \cdot R(t)$, where T is the *threshold parameter*. On the client side, one packet is sent for every U packets received.

To reduce overhead, there are two exceptions to this basic model. After a *padding budget* of N packets has been exceeded, the server will no longer send padding to reach the target rate. Instead, the rate is merely *limited* by delaying real data. The client, in turn, sends any packet that has been queued for over C seconds immediately.

Adapting RegulaTor: We test the Maybenot RegulaTor implementation [53] with slight modifications to improve its practicality and applicability to video traffic. Figure 13 shows our modified version, *Adapted RegulaTor*. Given Maybenot's current limitations, RegulaTor's optimizations cannot be implemented, but a heuristic is included in Maybenot RegulaTor to approximate surge restarting. To reduce overhead, we modify the heuristic so that, instead of restarting probabilistically when non-padding packets are sent, the server machine transitions with 100% probability if the current rate is less than 200 packets per second (2.4 Mbps). We also omit BOOT states to begin sending at the target rate immediately.

7.1 Tuning

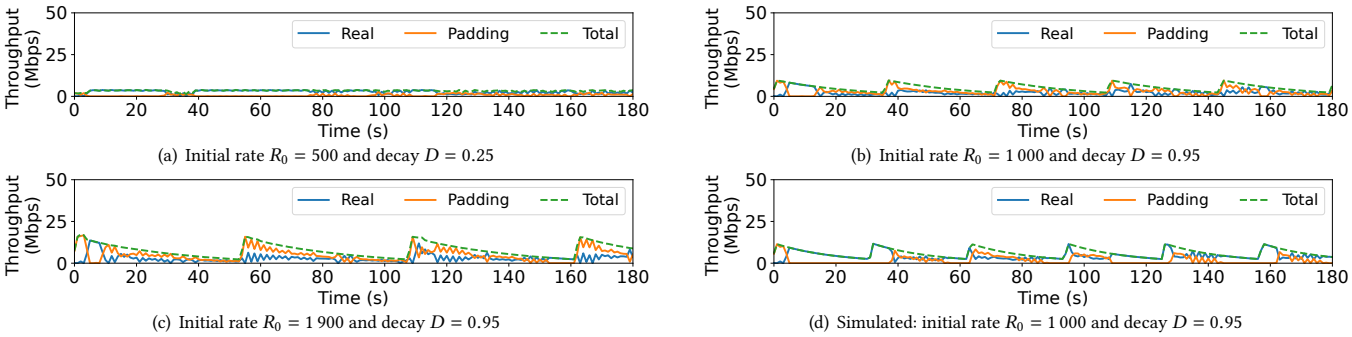
With our modifications of Maybenot RegulaTor, the remaining parameters are R_0, D, U, and ω, which is the number of SEND states used to approximate the decay function. We elect to set U to a constant value of 4, in line with settings tested in prior works [34, 53]. Similarly, we use a fixed value of $\omega = 20$, which has been shown to result in a suitable approximation of $R_0 \cdot D^t$ [53]. For tuning, we search the parameter space $R_0 = \{500, 600, ..., 2\,000\}$ and $D = \{0.15, 0.25, ..., 0.95\}$, leading to 144 total test combinations.

Table 3 shows our tuning results for Adapted RegulaTor. As with FRONT, we divide the tested configurations into three groups of roughly equal size: (1) less than 62.5% overhead, (2) [62.5, 125)% overhead, and (3) at least 125% overhead. We also consider latency metrics, including the *time to byte* at various progress points, *byte at time* at 250 ms intervals, and occurrences of spillage. However, we opt not to use these values in determining categorizations as they are correlated with bandwidth overhead. This is because of RegulaTor's fixed sending rate: if the rate is high, more padding will be sent when no real packets are queued, but progress will be made more quickly when data is queued. Thus, a *lower* QoE impact may be expected from higher-overhead configurations.

Evaluation: Adapted RegulaTor reduces the attacks to random guessing in nearly all cases, even with minimal overhead of about

Table 3: Evaluation with selected parameters for Adapted RegulaTor defense

Parameters		Bandwidth overhead			Attack accuracy			Security estimation		Quality of Experience									
										Occurrences				Activity per quality					
R_0	D	Send (Mbps)	Receive (%)	Overall (%)	DF	Tik-Tok	Beauty	BER	MI	Wait	Stall	Seek	Q. sw.	Wait 1k	Wait 2k	Wait 4k	Play 1k	Play 2k	Play 4k
500	0.75	0.03	0.70	25.83	0.008	0.010	0.007	0.870	0.117	5	7	0	324	0.4%	0.3%	0.2%	9.6%	75.4%	14.0%
500	0.45	0.02	0.61	25.71	0.008	0.010	0.006	0.878	0.112	2	4	0	357	0.4%	0.0%	0.0%	10.7%	74.1%	14.8%
500	0.25	0.02	0.45	25.51	0.009	0.010	0.207	0.878	0.116	2	4	0	352	0.4%	0.0%	0.0%	12.0%	73.8%	13.8%
1 400	0.95	3.19	74.82	118.46	0.008	0.010	0.081	0.784	0.123	10	11	0	294	0.4%	1.4%	0.4%	5.9%	36.5%	55.4%
1 300	0.95	2.89	67.55	109.38	0.010	0.010	0.255	0.790	0.112	9	10	0	319	0.4%	1.1%	0.2%	6.5%	39.7%	52.0%
1 000	0.95	1.75	40.78	75.91	0.007	0.010	0.006	0.796	0.121	7	8	0	363	0.4%	0.7%	0.1%	8.8%	47.1%	42.8%
1 500	0.85	3.52	82.26	127.75	0.008	0.010	0.097	0.776	0.118	11	19	0	255	0.4%	0.6%	0.8%	4.9%	25.1%	68.1%
1 600	0.95	3.89	90.99	138.66	0.007	0.010	0.063	0.777	0.121	11	11	0	265	0.5%	1.5%	0.4%	6.0%	34.6%	57.1%
1 900	0.95	4.93	115.15	168.85	0.007	0.010	0.053	0.787	0.120	11	12	0	238	0.5%	1.5%	0.4%	6.9%	30.5%	60.2%



(a) Initial rate $R_0 = 500$ and decay $D = 0.25$

(b) Initial rate $R_0 = 1\,000$ and decay $D = 0.95$

(c) Initial rate $R_0 = 1\,900$ and decay $D = 0.95$

(d) Simulated: initial rate $R_0 = 1\,000$ and decay $D = 0.95$

Figure 14: Throughput using Adapted Regulator collected over physical machines (a-c) and via simulation (d)

25%. This can be explained by its rigid sending rate: as long as there are queued packets when the rate drops below 200 packets per second, the server-side machine restarts. The only information leaked is, thus, a bound on packets in the queue whenever a certain SEND state is reached. Surprisingly, however, it appears that Beauty and the Burst can sometimes capitalize on this information: though three configurations of Adapted RegulaTor reduced its accuracy to random guessing, we find that the attack achieves 25.5% accuracy on the configuration ($R_0 = 1\,300$, $D = 0.95$), with varying success against others. This occurs when D is low (a bound on queued packets is leaked more often) and when the initial rate R_0 is high, in which case a reset to the initial rate may be more informative (videos with large segments or substantial variability; see Appendix A).

7.2 Further Evaluation and Discussion

Evaluation with heuristic attacks: As with FRONT, we tune the Leaky Streams attack against all configurations in Table 3, and the attack only reaches 1.0% accuracy with all tested thresholds. Similarly, Walls Have Ears achieves 1.0% accuracy, plus some negligible variance - the heuristic to determine segment boundaries is rendered useless by Adapted RegulaTor's strict traffic pattern.

Throughput: Figures 14(a)-14(c) show the receive throughput for the best configuration in each of the overhead categories. In contrast to Adapted FRONT, Adapted RegulaTor completely hides individual segment transmissions due to its strict sending rate.

Comparison to simulated traces: When comparing the simulated traces to those collected over physical machines (Figure 14(d) to 14(b)), we again only observe slight variations in terms of total throughput. Similar to the constant-rate defense, we see that the simulator fails to account for the effects of congestion control.

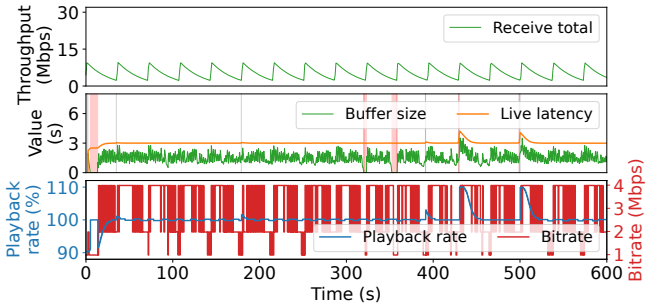
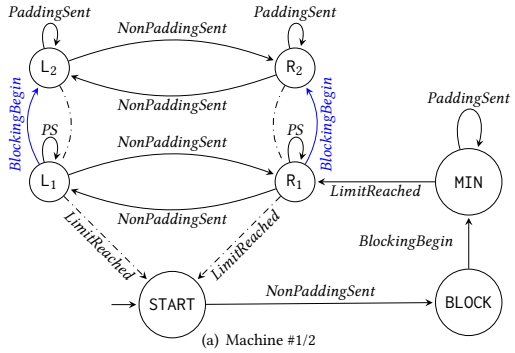


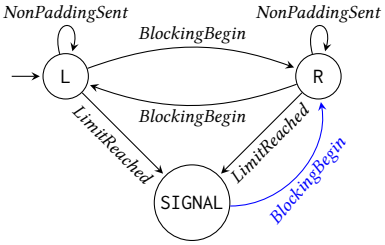
Figure 15: Throughput and video player QoE metrics using Adapted RegulaTor with $R_0 = 1\,000$ and $D = 0.95$

Quality of Experience: Unfortunately, despite Adapted RegulaTor's promise, it degrades QoE even more than the constant-rate defense. Table 3 shows the QoE of each configuration. While the number of occurrences of *wait* and *stall* is relatively high, the most significant QoE impact is caused by quality switches: the player's estimation of throughput causes a quality switch several hundred times during a 10-minute stream. We also see that the play percentage is no longer always dominated by the maximum quality.

Figure 15 shows the throughput and QoE for the best moderate-overhead parameters. Here, we see the reverse "sawtooth" behavior of the throughput and a few instances in which the buffer stops



(a) Machine #1/2



(b) Machine #2/2

Figure 16: Scrambler server-side machines

playing, but most clear are the excessive number of quality switches being requested. For the best parameters in the low- and high-overhead categories, we refer to Figures 25 and 26 in Appendix C.

Defense summary: Adapted RegulaTor is highly successful at reducing the accuracy of all attacks. However, it comes at the cost of significant QoE impact: unlike Adapted FRONT, it conforms video traffic to a target shape rather than adding random distortion, affecting segment transmissions and the player's ABR algorithms that determine when to switch qualities. This highlights the importance of evaluating not only a defense's performance but also its effects on QoE, an aspect that many prior works fail to consider.

8 SCRAMBLER: A TARGETED DEFENSE

We aim to create a defense that is well suited to the periodic nature of video traffic. Specifically, we leverage the typical pattern in DASH streaming, where each period (in our case, every two seconds) involves a small number of uplink packets requesting a segment, followed by the server's response burst. More critically, we observe that inter-packet timing does not vary much within segment transmissions, suggesting that timing information can be perfectly concealed by maintaining a constant rate close to the observed average. In any case, there should be no QoE impact as long as each segment is transmitted in full without spilling over to another window, as the client will still have buffered segments.

With no sensitive timing information leaked, the only remaining property to conceal is segment size. However, volatile network environments and buffering periods must also be taken into account. These considerations inform the design of Scrambler, our proposed defense for video streaming traffic. During steady-state streaming, its operation consists of (1) sending a configurable minimum number of packets at a constant rate during each burst and (2) introducing a random amount of trailing padding once all remaining application data has been sent. As a consequence of Scrambler's design, padding overhead should be decreased during buffering, and the minimum packet constraint is relaxed when spillage occurs. On the client side, we run a simple 3 Mbps constant-rate defense.

Operation: Figure 16 shows Scrambler's two machines, run concurrently on the server side. Machine #1 (Figure 16(a)) encodes the primary logic of the defense, while Machine #2 (Figure 16(b)) counts the number of real packets sent during a segment transmission. If a certain threshold is exceeded, a signal is sent to Machine #1 via a *BlockingBegin* event. Transitions that occur simultaneously in both machines due to signaling are indicated with blue arrows.

Machine #1 infers the start of a new burst when a non-padding packet is sent. This results in a transition from `START` to `BLOCK`, which enables infinite blocking of outgoing packets with the *bypass* and *replace* flags set. *bypass* allows padding actions to ignore any blocking in effect, and *replace* specifies that any queued application data can be sent instead of padding; the combination of the two allows a constant rate to be maintained. Once blocking starts, another transition occurs to `MIN`, which sends a packet with a timeout value of δ, a defense parameter specifying the inter-packet delay. Repeated self-transitions will occur until N packets (the minimum per burst) have been sent, at which time a *LimitReached* event will be triggered within the framework, causing a transition to $\mathtt{R}_1$.

$\mathtt{L}_1$ and $\mathtt{R}_1$'s actions are also to send a packet with timeout δ, but any non-padding data sent while in these states causes a transition from one to the other; thus, a new limit is sampled, and the transition back to `START` is delayed. On the other hand, if no further data is queued, a number of trailing packets are sent – which cause self-transitions via the resultant *PaddingSent* (*PS*) events – and the machine will then restart. The limit for both $\mathtt{L}_1$ and $\mathtt{R}_1$ is sampled uniformly from the range $[P_{min}, P_{max}]$, which are parameters that represent the minimum and maximum trailing padding count.

Two additional states, $\mathtt{L}_2$ and $\mathtt{R}_2$, are also included in Machine #1. These are designed to further reduce the QoE impact of the defense and improve its efficacy against attacks. When the number of non-padding packets sent during a burst exceeds $1.25 \cdot N$, Machine #2 causes a *BlockingBegin* event. If Machine #1 is in $\mathtt{L}_1$ or $\mathtt{R}_1$ at this time, it transitions to $\mathtt{L}_2$ or $\mathtt{R}_2$, which are identical to $\mathtt{L}_1$ and $\mathtt{R}_1$ except for their limits: these are sampled from the range $[P_{min}/4, P_{max}/4]$. Thus, less padding is sent following abnormally large segments, reducing their apparent sizes towards typical ones.

In Machine #2, `L` and `R` count non-padding packets sent during a single segment transmission (a cycle of Machine #1). They specify zero-duration blocking without *replace*, so they are effectively no-op states[2]. When a sufficiently small segment has been sent, a transition occurs from `L` to `R` or vice versa, resetting the count; but if the limit $1.25 \cdot N$ is reached, a transition to `SIGNAL` occurs. This state sets infinite blocking with *replace*, causing a *BlockingBegin* event. Afterward, counting resumes with a transition to `R`.

8.1 Tuning

We select the best parameters for Scrambler by exhaustively searching the space $\delta = \{120, 160, 200\}$ μs, $N = \{500, 700, ..., 1\,500\}$, and $P_{min} = P_{max} = \{400, 700, 1\,000\}$, excluding combinations in which

[2]Note that, in Maybenot v1, this requires a slightly modified integration: zero-duration blocking actions without *replace* must not cause a *BlockingBegin* event.

Table 4: Evaluation with selected parameters for Scrambler defense

Parameters				Bandwidth overhead			Attack accuracy			Security estimation		Quality of Experience									
												Occurrences				Activity per quality					
δ	N	P_{min}	P_{max}	Send (Mbps)	Receive (%)	Overall (%)	DF	Tik-Tok	Beauty	BER	MI	Wait	Stall	Seek	Q. sw.	Wait 1k	Wait 2k	Wait 4k	Play 1k	Play 2k	Play 4k
160	500	400	1 000	3.01	120.36	190.58	0.163	0.176	0.817	0.663	2.093	4	2	0	14	0.7%	0.1%	0.4%	2.8%	4.3%	91.7%
200	700	400	1 000	3.01	120.27	190.48	0.123	0.137	0.700	0.689	1.752	5	2	0	14	0.7%	0.1%	0.4%	4.7%	4.3%	89.8%
160	700	400	1 000	3.01	124.06	194.27	0.128	0.142	0.769	0.693	1.823	4	2	0	14	0.7%	0.1%	0.3%	5.7%	6.1%	87.1%
160	1 100	400	1 000	3.01	163.93	234.13	0.061	0.065	0.005	0.762	1.155	4	2	0	14	0.7%	0.1%	0.4%	5.7%	3.3%	89.8%
120	1 100	400	1 000	3.01	165.57	235.77	0.064	0.072	0.386	0.770	1.120	4	2	0	16	0.7%	0.1%	0.3%	2.9%	2.5%	93.6%
200	1 100	400	1 000	3.01	159.86	230.06	0.061	0.056	0.313	0.773	1.127	4	2	0	15	0.7%	0.1%	0.3%	2.8%	5.1%	90.9%
200	1 500	400	1 000	3.01	211.98	282.17	0.042	0.029	0.007	0.808	0.864	4	2	0	14	0.7%	0.1%	0.3%	3.8%	5.2%	89.9%
160	1 500	400	1 000	3.01	215.43	285.62	0.042	0.040	0.007	0.808	0.895	4	2	0	14	0.7%	0.1%	0.3%	3.8%	7.0%	88.1%
120	1 500	400	1 000	3.01	213.82	284.00	0.039	0.041	0.182	0.814	0.898	4	2	0	14	0.7%	0.1%	0.3%	3.8%	5.2%	89.8%

$P_{min} > P_{max}$. This results in a total of 108 configurations. The ranges of δ and N are based on the distributions of inter-packet times within segment transmissions and the 4 Mbps burst sizes in our dataset (Figure 4), and we test values for P_{min} and P_{max} that seem reasonable with 100 Mbps and two-second segments.

Evaluation: Table 4 shows the best configurations in three sets of similar size: (1) less than 200% overall overhead; (2) [200, 250)% overhead; and (3) at least 250% overhead. We find that Scrambler is highly effective above 120% receive and 190% overall overhead: the weakest configuration reduces the maximum accuracy of DF and Tik-Tok to 17.6% with 10k packets (BER 0.66, MI about two bits). The best configuration we test (δ = 120, N = 1 500) achieves an even further reduction to 4.1% accuracy (BER 0.81, less than one bit of MI) with 214% receive and 284% overall overhead, a significant overhead reduction compared to the constant-rate defense (462% receive, 518% overall) while still achieving strong protection.

Beauty and the Burst is more effective, with up to 81.7% accuracy on the three low-overhead configurations. However, accuracy drops markedly with a slight 35-40% increase in overall overhead: (δ = 160, N = 1 100) reduces accuracy to 0.5%, and the other moderate configurations still provide strong protection with a maximum of 38.6% accuracy. As expected, the heavy configurations are the most effective, with 0.7%, 0.7%, and 18.2% accuracy, respectively.

From this, it appears that random guessing can be achieved by only changing δ. However, we repeat the simulations and find that Beauty and the Burst's cross-validated accuracy is sometimes 1% with *every* moderate and high-overhead configuration, and accuracy can be as low as 30.5% with low overhead: Scrambler's randomization is effective in the worst case, but it is often exceptional.

Note that all of the best configurations in Table 4 have P_{min} = 400 and P_{max} = 1000: this is the most permissive possible range of trailing padding within the values we test and creates the most variance. Thus, Scrambler's success can be attributed in part to the randomness it adds to segment sizes, which serves to reduce the correspondence between undefended and defended segments. Larger values of N also lead to increased efficacy, since segments can be more effectively concealed when their size is less than N.

Finally, increasing the input size of DF and Tik-Tok does not substantially improve an attacker's performance against Scrambler: with the strongest configuration (δ = 120, N = 1 500), DF and Tik-Tok only achieve 6.2% and 6.0% accuracy, respectively, with 30k packets - a marginal increase from their 10k-packet accuracy. With the lighter (δ = 160, N = 500) setting, accuracy increases to 45.7% for DF and 47.0% for Tik-Tok. Though this change is more notable than with the best parameters, it suggests that well more than 43 seconds of observation (Table 1) would be needed, on average, to have high confidence in the classifiers' predictions.

8.2 Further Evaluation and Discussion

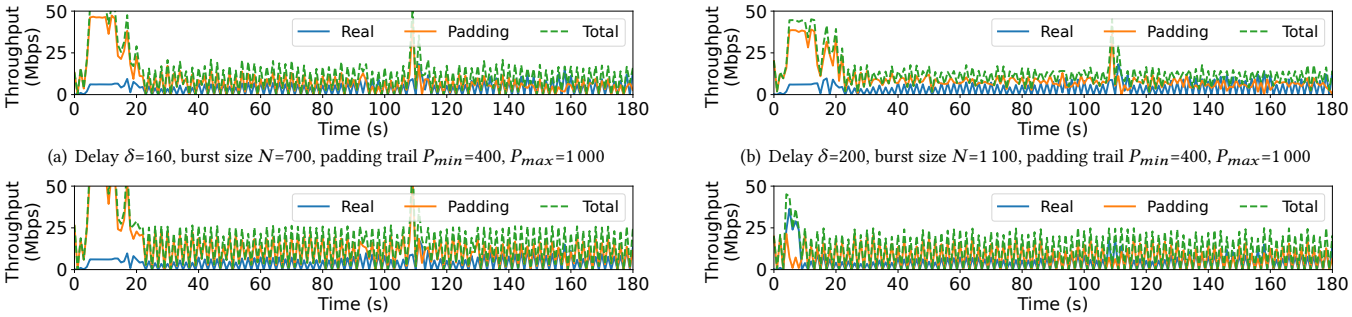
Evaluation with heuristic attacks: We again tune Leaky Streams and find that it cannot reach more than 1.0% accuracy against Scrambler. Similarly, Walls Have Ears achieves its highest accuracy of 3.9% against the weakest configuration (δ = 160, N = 500), and this decreases to random guessing with moderate overhead.

Throughput: Figures 17(a)–17(c) show the receive throughput for the best Scrambler configuration in each overhead category. We see that segment peaks are well hidden, and the defense operates periodically with two-second cycles, except when spillage occurs.

Comparison to simulated traces: Comparing simulated traces to those collected over physical machines (Figure 17(d) to 17(b)), we see more segment spillage near the beginning of a trace, again due to the simulator being able to model sending real packets at an unlimited speed unbounded by congestion control.

Quality of Experience: As expected, Scrambler's targeted design affords it a rather high QoE, as shown in Table 4. Here, regardless of configuration, the video is played at the highest quality the vast majority of the time. In fact, for 75% of the videos in *Long-Enough*, the highest quality is played over 96% of the time. While we do experience some periods in which the video player waits for more data, and some quality switches, these mostly occur at the beginning of the trace and only last a few seconds. Figure 18 shows the throughput and QoE for the best moderate-overhead configuration: it is clearly visible that after some initial buffering at the beginning, the video plays without any issues, and the defense does not impact QoE. For the best configurations in the low- and high-overhead categories, we refer to Figures 27 and 28 in Appendix C.

Defense analysis: Although we have a high degree of success with Scrambler, a general approach based on random sampling will *always* provide inadequate protection (with reasonable overhead) for certain videos with sufficiently unique fingerprints. For example, we observe that a few videos in our dataset contain unusually large segments within the last 60 seconds; thus, classification accuracy



(a) Delay δ=160, burst size N=700, padding trail P_{min}=400, P_{max}=1 000

(b) Delay δ=200, burst size N=1 100, padding trail P_{min}=400, P_{max}=1 000

(c) Delay δ=120, burst size N=1 500, padding trail P_{min}=400, P_{max}=1 000

(d) Simulated: delay δ=200, burst size N=1 100, padding trail P_{min}=400, P_{max}=1 000

Figure 17: Throughput using Scrambler collected over physical machines (a-c) and via simulation (d)

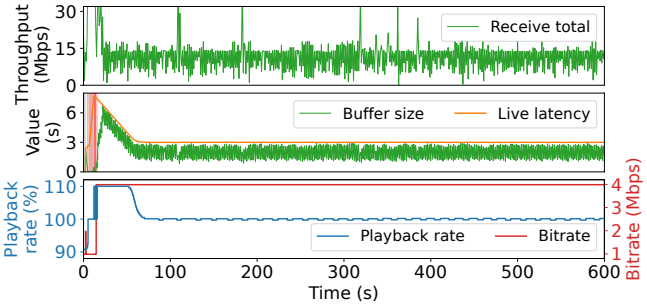


Figure 18: Throughput and video player QoE metrics using Scrambler with δ = 200, N = 1 100, P_{min} = 400, P_{max} = 1 000

for these particular videos is greater than the average. In particular, with (δ = 120, N = 1 500), most videos have nearly 0% accuracy, but we observe values as high as 85% with DF and 73% with Tik-Tok.

This alludes to an inherent tradeoff between overhead and protection that has been discussed by many authors [3, 14, 15]: while we could arbitrarily raise N to achieve perfect protection with Scrambler, this would imply prohibitive overhead and QoE reduction. Similarly, by reducing N, P_{min}, and P_{max}, we could minimize overhead, but trace patterns would not be well concealed. In summary, with Scrambler, and in fact with *any defense* that does not take advantage of prior knowledge of the characteristics of individual videos or sets of videos, we can only hope for an acceptable balance between average-case protection, overhead, and QoE.

Despite this, we expect similar results to those we report for Scrambler with any classifier, not just the attacks we test: it simply reduces the correlation between the original and defended sequences of segment sizes while concealing other features. Also, its general applicability affords it greater possibilities for deployment than, e.g., SMAUG [66], which requires significant extra infrastructure on the server side; and its minimal and predictable QoE impact makes it more practical than defenses with significant variance (such as [76]). Finally, we note that we use a 3 Mbps constant rate on the client side to further reduce the accuracy of packet sequence-based attacks, but Scrambler's protection is mostly provided by the server-side machines; thus, overall overhead could be further decreased in practice while maintaining strong protection.

Defense summary: Our proposed Scrambler defense, targeted for video streaming using MPEG-DASH, is successful in reducing attack accuracy while maintaining high QoE. It represents a tradeoff between the high overhead and limited efficacy of Adapted FRONT and the prohibitive QoE impacts of Adapted RegulaTor. Though the QoE is not as high as undefended traces (which we consider perfect), the defense provides accuracy close to that of the constant-rate defense, better QoE, and much lower bandwidth overhead.

9 CONCLUSIONS

We present the first rigorous study of video fingerprinting defenses, considering attack performance, overheads, and QoE. Our comprehensive analyses provide a host of important insights, which we hope will accelerate future video fingerprinting work and lead to the development of practical defenses that provide adequate protection while accounting for network impacts and user experience.

Attacks: We show that heuristic strategies and attacks based on manually crafted features – of which there are many – degrade significantly when confronted with defended traffic, regardless of the defense technique. In contrast, attacks that leverage deep learning provide the best performance against both undefended and defended traffic. As it is our intention to nudge video fingerprinting research in the direction of potential defenses, we suggest that future work should focus on such attacks, as they allow for better estimation of the maximal protection defenses can provide. However, we note that heuristic attacks are more explainable and may provide useful insights into effective defense strategies, particularly those that do not alter the periodic nature of the original traffic.

Defenses: Our evaluations hint that naive padding-only approaches cannot provide acceptable protection with reasonable overheads, yet strong regularization may have prohibitive impacts on QoE. Our proposed defense, Scrambler, achieves a more acceptable tradeoff; however, we identify weaknesses inherent to general defense strategies that may be considered in future work. We also note that factors such as code execution time and congestion control can cause real-world deployments to diverge from theoretical defense behavior in unexpected ways; consideration of these issues also represents a fruitful direction for future study. Overall, our work underscores the need for more effective defenses that take into account not only protection but also usability and practicality under wide-scale deployment. As video streaming continues to monopolize residential network traffic, it is crucial to address the very real threat of identification of users' viewing habits, which can be accomplished by a number of relatively weak adversaries.

ACKNOWLEDGMENTS

We would like to thank Matthias Beckerle and Tobias Pulls for helpful discussions that contributed substantially to the work presented in this paper. This work was partially supported by the Swedish Foundation for Strategic Research (SSF) and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

[1] Dilawer Ahmed, Aafaq Sabir, and Anupam Das. 2023. Spying through Your Voice Assistants: Realistic Voice Command Fingerprinting. In *Proc. USENIX Security*.
[2] J.S. Atkinson, M. Rio, J.E. Mitchell, and G. Matich. 2014. Your WiFi Is Leaking: Ignoring Encryption, Using Histograms to Remotely Detect Skype Traffic. In *Proc. IEEE Military Communications Conference (MILCOM)*.
[3] 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*.
[4] Alireza Bahramali, Ardavan Bozorgi, and Amir Houmansadr. 2023. Realistic Website Fingerprinting By Augmenting Network Traces. In *Proc. ACM CCS*.
[5] Alireza Bahramali, Ramin Soltani, Amir Houmansadr, Dennis Goeckel, and Don Towsley. 2020. Practical traffic analysis attacks on secure messaging applications. In *Proc. Network and Distributed System Security (NDSS)*.
[6] 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)*.
[7] BigtimeFreeMovies. 2023. The Best Free Movies on Youtube. https://www.youtube.com/playlist?list=PLm9l7EEbJuhzLTz8aVdDfYIn2TQ_X_-0S.
[8] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, et al. 2007. Revealing Skype traffic: when randomness plays with you. *SIGCOMM CCR* (2007).
[9] 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)*.
[10] Giovanni Cherubin. 2017. Bayes, not Naïve: Security Bounds on Website Fingerprinting Defenses. In *Proc. Privacy Enhancing Technologies (PETS)*.
[11] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. 2022. Online website fingerprinting: Evaluating website fingerprinting attacks on tor in the real world. In *Proc. USENIX Security*.
[12] Thilini Dahanayaka, Guillaume Jourjon, and Suranga Seneviratne. 2022. Dissecting traffic fingerprinting CNNs with filter activations. *Computer Networks* (2022).
[13] Shuaifu Dai, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. 2013. NetworkProfiler: Towards automatic fingerprinting of Android apps. In *Proc. IEEE INFOCOM*.
[14] 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)*.
[15] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2020. Comprehensive anonymity trilemma: User coordination is not enough. In *Proc. Privacy Enhancing Technologies (PETS)*.
[16] DASH-Industry-Forum. 2024. dash.js. https://dashjs.org/.
[17] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, et al. 2020. TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting. In *Proc. ACM CCS*.
[18] 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*.
[19] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *Proc. USENIX Security*. 303–320.
[20] 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)*.
[21] 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).
[22] 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)*.
[23] Yanjie Fu, Hui Xiong, Xinjiang Lu, Jin Yang, and Can Chen. 2016. Service usage classification with encrypted internet traffic in mobile messaging apps. *IEEE Trans. on Mobile Computing* (2016).
[24] Jiajun Gong and Tao Wang. 2020. Zero-delay Lightweight Defenses against Website Fingerprinting. In *Proc. USENIX Security*.
[25] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. 2021. WFDefProxy: Modularly implementing and empirically evaluating website fingerprinting defenses. *arXiv:2111.12629* (2021).
[26] 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)*.
[27] 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*.
[28] David Hasselquist, Martin Lindblom, and Niklas Carlsson. 2022. Lightweight Fingerprint Attack and Encrypted Traffic Analysis on News Articles. In *Proc. IFIP Networking*.
[29] David Hasselquist, Christian Vestlund, Niklas Johansson, and Niklas Carlsson. 2022. Twitch Chat Fingerprinting. In *Proc. IFIP TMA*.
[30] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *Proc. USENIX Security*.
[31] 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)*.
[32] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proc. ACM Workshop on Cloud Computing Security*.
[33] Andrew Hintz. 2002. Fingerprinting websites using traffic analysis. In *Proc. Workshop on Privacy Enhancing Technologies*.
[34] James K Holland and Nicholas Hopper. 2022. RegulaTor: A Straightforward Website Fingerprinting Defense. In *Proc. Privacy Enhancing Technologies (PETS)*.
[35] Rob Jansen and Nicholas J Hopper. 2011. Shadow: Running Tor in a box for accurate and efficient experimentation. (2011).
[36] Rob Jansen, Ryan Wails, and Aaron Johnson. 2024. A Measurement of Genuine Tor Traces for Realistic Website Fingerprinting. *arXiv:2404.07892* (2024).
[37] Zhaoxin Jin, Tianbo Lu, Shuang Luo, and Jiaze Shang. 2023. Transformer-based Model for Multi-tab Website Fingerprinting Attack. In *Proc. ACM CCS*.
[38] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. 2016. Toward an efficient website fingerprinting defense. In *Proc. ESORICS*.
[39] Vengatanathan Krishnamoorthi, Niklas Carlsson, Emir Halepovic, and Eric Petajan. 2017. BUFFEST: Predicting Buffer Conditions and Real-time Requirements of HTTP(S) Adaptive Streaming Clients. In *Proc. ACM MMSys*.
[40] Jianfeng Li, Shuohan Wu, Hao Zhou, Xiapu Luo, Ting Wang, et al. 2022. Packet-Level Open-World App Fingerprinting on Wireless Traffic. In *Proc. NDSS*.
[41] Shuai Li, Huajun Guo, and Nicholas Hopper. 2018. Measuring Information Leakage in Website Fingerprinting Attacks and Defenses. In *Proc. ACM CCS*.
[42] 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)*.
[43] 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*.
[44] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proc. ACM SIGCOMM*.
[45] 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*.
[46] 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*.
[47] NGINX. 2024. https://www.nginx.com/.
[48] Rishab Nithyanand, Xiang Cai, and Rob Johnson. 2014. Glove: A Bespoke Website Fingerprinting Defense. In *Proc. ACM WPES*.
[49] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, et al. 2016. Website Fingerprinting at Internet Scale. In *Proc. NDSS*.
[50] Mike Perry and George Kadianakis. 2020. Circuit Padding Developer Documentation. https://github.com/torproject/tor/blob/main/doc/HACKING/CircuitPaddingDevelopment.md.
[51] Tor Project. 2024. Pluggable Transport Specification (Version 1). https://gitweb.torproject.org/torspec.git/tree/pt-spec.txt.
[52] Tobias Pulls. 2020. Towards Effective and Efficient Padding Machines for Tor. *arXiv:2011.13471* (2020).
[53] Tobias Pulls and Ethan Witwer. 2023. Maybenot: A Framework for Traffic Analysis Defenses. In *Proc. ACM Workshop on Privacy in the Electronic Society (WPES)*.
[54] Darijo Raca, Jason J Quinlan, Ahmed H Zahran, and Cormac J Sreenan. 2018. Beyond throughput: A 4G LTE dataset with channel and context metrics. In *Proc. ACM Multimedia Systems Conference (MMSys)*.
[55] 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).
[56] 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)*.
[57] 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)*.

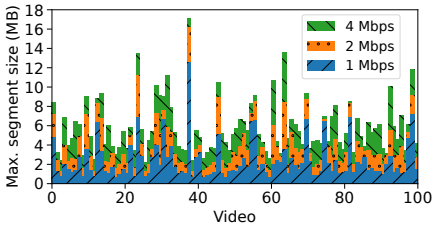


Figure 19: Maximum segment size per video for each quality level

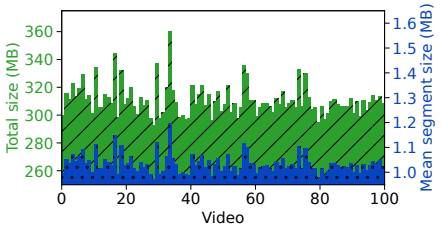


Figure 20: Total size vs. mean segment size per video at 4 Mbps bitrate

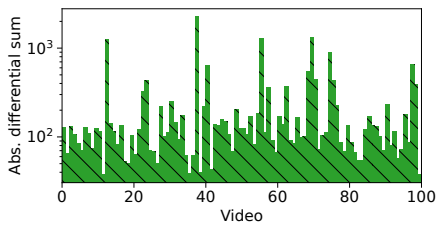


Figure 21: Absolute differential sum per video at 4 Mbps bitrate

[58] Andrew Reed and Michael Kranch. 2017. Identifying HTTPS-Protected Netflix Videos in Real-Time. In *Proc. ACM CODASPY*.

[59] 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)*.

[60] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2017. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *Proc. USENIX Security*.

[61] Selenium. 2024. https://www.selenium.dev/.

[62] 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*.

[63] 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)*.

[64] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia* (2011).

[65] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2018. Robust Smartphone App Identification via Encrypted Network Traffic Analysis. *IEEE Trans. on Information Forensics and Security (TIFS)* (2018).

[66] 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*.

[67] Alexander Veicht, Cedric Renggli, and Diogo Barradas. 2023. DeepSE-WF: Unified Security Estimation for Website Fingerprinting Defenses. In *Proc. PETS*.

[68] Tao Wang. 2021. The One-Page Setting: A Higher Standard for Evaluating Website Fingerprinting Defenses. In *Proc. ACM CCS*.

[69] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *Proc. USENIX Security*.

[70] Tao Wang and Ian Goldberg. 2017. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *Proc. USENIX Security*.

[71] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. 2005. Tracking anonymous peer-to-peer VoIP calls on the internet. In *Proc. ACM CCS*.

[72] Ethan Witwer, James K Holland, and Nicholas Hopper. 2022. Padding-only defenses add delay in Tor. In *Proc. ACM WPES*.

[73] 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)*.

[74] 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)*.

[75] 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*.

[76] Xiaokuan Zhang, Jihun Hamm, Michael K Reiter, and Yinqian Zhang. 2019. Statistical privacy for streaming traffic. In *Proc. NDSS*.

[77] 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 FURTHER ANALYSIS OF *LONGENOUGH*

We provide a top-down description of some of the most notable properties of *LongEnough* in order to explain why certain videos are identified more easily than others, starting from global statistics and then moving to individual videos.

As shown in Figure 4, the distribution of segment sizes for each quality level is roughly normal, and doubling the bitrate approximately doubles segment sizes. Table 5 provides a more detailed characterization of these distributions. The main feature that is more apparent here than in Figure 4 is that the maximum segment size at each quality level is extremely large, resulting in a subtly right-skewed distribution, as reflected by the mean segment sizes.

Table 5: Segment size statistics for each quality level

Average bitrate	Mean segment size (kB)	Segment size percentiles (kB)				
		0	25	50	75	100
1 000	260	3	178	229	292	12 508
2 000	518	4	346	458	588	16 151
4 000	1 033	6	713	937	1 185	17 071
All	604	3	255	451	809	17 071

Maximum segment size: We find unusually large segments to be particularly identifying: they are the primary reason that a few videos are identified by DF and Tik-Tok despite the use of Scrambler, as described in the defense analysis in Section 8.2. Figure 19 shows that every video in fact has a *unique* maximum segment size at all three bitrates—and much variation exists—making this a particularly useful feature against the videos in *LongEnough*.

Total size and mean segments: Large segment sizes are not the only feature that causes some videos to have rather distinct fingerprints. Another factor is total size: each video in *LongEnough* also has a unique size at all quality levels. Moreover, mean segment size and total size are perfectly correlated for videos of the same duration; though trivial, this relationship is exemplified in Figure 20 for the 4 Mbps bitrate. For videos without substantial temporal segment size variation, then, an adversary may gain an advantage by monitoring a stream for a limited period of time and recording the average throughput or mean of the observed segment sizes, which will be similar to the global mean over the entire video.

These observations help explain why FRONT fails: even with a large amount of padding, videos are likely still distinguishable to a certain extent by data volume, since the expected value of padding is independent of the video being defended and depends instead on the defense parameters. Similarly, the CNN classifiers we test apparently learn to distinguish the predictable Rayleigh distribution shape from the periodic segments that overlap it, but data volume–especially over more time–may also prove useful.

Bitrate variability: Another notable difference between videos is their bitrate variability: as mentioned in Section 2.2, we encode videos with VBR such that a target *average* bitrate is achieved. As such, some videos may have more pronounced differences in size between adjacent segments or many segments of extreme size compared to others encoded at the same bitrate. To quantify this, we

Table 6: Evaluation with best configurations under variable bandwidth conditions

Defense	Config.	Parameters	Bandwidth overhead			Attack accuracy			Security estimation	
			Send (Mbps)	Receive (%)	Overall (%)	DF	Tik-Tok	Beauty	BER	MI
Undefended	N/A	N/A	0.00	0.00	0.00	0.911	0.932	0.988	0.032	6.555
Adapted FRONT	low	($N = 2\,500$, $W = 7$)	1.12	26.56	53.46	0.687	0.755	0.979	0.151	5.890
	moderate	($N = 6\,000$, $W = 9$)	1.93	50.66	97.20	0.529	0.619	0.976	0.272	5.021
	high	($N = 6\,500$, $W = 2$)	9.81	237.87	474.19	0.073	0.107	0.915	0.761	1.572
Adapted RegulaTor	low	($R_0 = 500$, $D = 0.25$)	1.51	45.35	81.62	0.033	0.044	0.206	0.761	0.333
	moderate	($R_0 = 1\,000$, $D = 0.95$)	2.30	122.16	230.19	0.105	0.120	0.006	0.639	1.922
	high	($R_0 = 1\,900$, $D = 0.95$)	0.97	3.29	96.90	0.081	0.126	0.056	0.637	1.975
Scrambler	low	($\delta = 160$, $N = 700$)	3.01	529.54	602.88	0.030	0.038	0.454	0.897	0.416
	moderate	($\delta = 200$, $N = 1\,100$)	3.01	390.36	462.75	0.029	0.037	0.473	0.900	0.388
	high	($\delta = 120$, $N = 1\,500$)	3.01	326.75	398.16	0.026	0.030	0.417	0.893	0.349

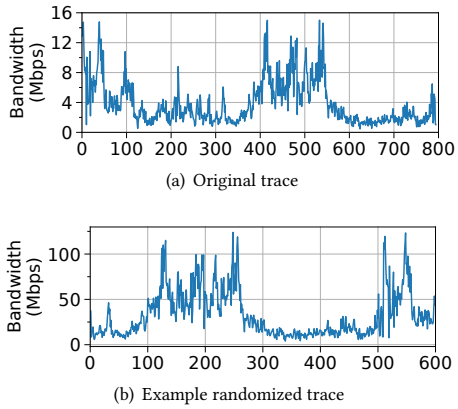
measure segment size differentials throughout the entire duration of every video in *LongEnough* (as is done for the last part of a video in Walls Have Ears) and calculate their distribution. The sum of the absolute differentials for each video is shown in Figure 21.

Though many videos have relatively low bitrate variation, there are a few that have substantial variation, indicating that adjacent segments are often of very different sizes and/or that many very large segments exist. Apart from resulting in a fingerprint that is more distinctive in general (and thus more difficult to defend), this partially explains the seemingly counterintuitive attack results for RegulaTor presented in Section 7.1: with substantial bitrate variation, a defended video may have a pronounced pattern of surge restarts due to varying queue sizes, leading to easier identification.

Summary: The *LongEnough* dataset exhibits a high degree of variability in terms of segment sizes, total video size, and the variability of segment sizes within videos, despite all videos being of the same duration. This reflects the varying types of content included by design in the dataset and results in a rich set of useful features for classifiers, facilitating our objective of evaluating defenses in conditions that are most favorable for an attacker.

B VARIABLE BANDWIDTH CONDITIONS

Data collection: To capture the impact of real-world network effects, we also perform experiments under variable bandwidth conditions. Using a real-world LTE sampled trace [54], we collect the *LongEnough-variable* dataset where we vary the bandwidth capacity on client-side links. Figure 22(a) shows the original sampled bandwidth trace over time as collected by Raca et al. [54], with the average rate being 3.85 Mbps. Based on the original trace, we first multiply the trace by a factor of 8 to better align the peaks with the original 100 Mbps bandwidth capability, as we want to capture the effect of the variability rather than focus on a low bandwidth value. We then create randomized traces for each data collection sample. Figure 22(b) illustrates an example of a unique randomized bandwidth trace. To obtain these, we randomize the starting point, loop the trace, and randomly multiply each bandwidth value by ±10%. This is done to ensure that the bandwidth trace becomes unique but still includes periods of both low and high bandwidth.



(a) Original trace

(b) Example randomized trace

Figure 22: Variable bandwidth traces

Evaluation: To provide an indication of how variable bandwidth conditions interact with the use of defenses, we perform attacks and calculate overhead for the best tuned configuration from each overhead category (low, moderate, and high) for Adapted FRONT, Adapted RegulaTor, and Scrambler. Table 6 shows these results, along with the undefended case for reference.

We find that QoE is impacted and highly dependent on the varying bandwidth conditions. As a reference, for the undefended case, we observe the average number of wait, stall, seek, and quality switch occurrences to be 8, 5, 0, and 17, respectively. The percentage of time spent waiting for data at 1k, 2k, and 4k bitrates is 0.4%, 0.0%, 1.3%, while the play percentages are 0.3%, 1.6%, 96.3%.

For attacks, we find that the accuracy of all CNN attacks against undefended traffic is similar to the performance achieved with constant bandwidth. The slight decrease (1–2%) can be accounted for by the randomization of bandwidth traces, which causes segments to be requested at different qualities more often during a stream.

With Adapted FRONT, which simply adds padding to the original trace, there is a notable reduction in accuracy compared to the constant bandwidth case, which is more pronounced with stronger configurations: DF and Tik-Tok's accuracy is about 10% lower with

low overhead, 15-20% lower with moderate overhead, and 30% lower with high overhead. Adapted FRONT's simple padding scheme is more effective at obfuscating packet sequences in the presence of many quality switches, at least with relatively low N; however, timing features apparently remain useful, as Tik-Tok still performs better than DF against all configurations. In any case, Adapted FRONT still fails to defeat Beauty and the Burst, with only a minor (1–5%) reduction in accuracy compared to constant bandwidth conditions. Overhead is largely unchanged, as the random padding added during simulation has no effect on the underlying traffic.

In contrast, with Adapted RegulaTor, attack accuracy and overheads deviate significantly from the constant bandwidth evaluations: this is to be expected when shaping traces with distinct characteristics. The best accuracy is achieved by Beauty and the Burst, with ($R_0 = 500$, $D = 0.25$), as surge restarts are more frequent when the rate is low. However, its accuracy decreases markedly with the other two configurations, since very few restarts happen when streaming most videos due to an abundance of lower-quality segments that are small compared to R_0. Note that ($R_0 = 1\,000$, $D = 0.95$) is more effective than the high-overhead configuration: this is likely because restarts are more informative when R_0 is high, as they signal the presence of large segments or queue buildup.

Surprisingly, DF and Tik-Tok perform better against the moderate and high-overhead configurations of Adapted RegulaTor. This is explained by more (smaller) segments being included in the $10k$-packet input than under constant bandwidth conditions and surge restarts being more informative with higher R_0. Adapted RegulaTor's overhead is higher than with constant bandwidth except with the high-overhead configuration: less frequent restarts allow a lower rate to be maintained for much of a stream.

In the case of Scrambler, attack accuracy is similar (ranging from 41.7% to 47.3%) across configurations. Since we tune all parameters with 4 Mbps segments in mind, it appears that Beauty and the Burst primarily takes advantage of the difference in size between segments of different qualities, though DF and Tik-Tok are not able to do so effectively due to their smaller input size: their accuracy is around 2–4% against every configuration. Also as a result of tuning, we observe much higher overhead in variable bandwidth conditions, since lower-quality segments are more frequently downloaded. These results primarily illustrate the importance of properly tuning defenses for the conditions in which they will be deployed.

Evaluation with heuristic attacks: We repeat the tuning of Leaky Streams against all tested configurations, as in previous sections. We find that Leaky Streams achieves 2% accuracy against RegulaTor with ($R_0 = 1000$, $D = 0.95$): the regulated traffic pattern closely matches actual segment sizes for one video during the last minute of streaming, so it is consistently identified. However, Leaky Streams has only 1% accuracy with the other two Adapted RegulaTor configurations and against Adapted FRONT and Scrambler.

Against Adapted FRONT, Walls Have Ears achieves 16.68% accuracy with the low-overhead configuration, 10.13% with moderate overhead, and 1.02% with high overhead: though the attack is not entirely ineffective, its accuracy is lower under variable bandwidth conditions due to (1) a modified sequence of differentials due to interleaved segments of different qualities and (2) perturbation of this sequence due to padding. Adapted RegulaTor is more effective at concealing segment sizes, with 1.02%, 0.84%, and 1.03% accuracy; as is Scrambler, with 2.08%, 2.44%, and 1.83% accuracy.

However, unlike in constant bandwidth conditions, the videos identified vary, and identification does not appear to depend on any specific characteristics. We conclude that, though variable bandwidth conditions tend to reduce attack accuracy, requesting segments at varying qualities can in some cases provide additional useful information to an attacker monitoring a stream.

Summary: Our preliminary evaluations show that both attack accuracy and overhead are affected in many unexpected ways under variable bandwidth conditions. Future defense proposals should carefully consider the impact of network conditions on defense operation; in the case of the defenses we test, our results demonstrate the importance of tuning. We also note that improved tooling, such as more realistic network models in the Maybenot simulator, could aid future work by reducing the number of live deployments required when testing new defenses and configurations.

C ADDITIONAL THROUGHPUT AND QOE

Figures 23 and 24 show the throughput and QoE for the low- and high-overhead categories, respectively, using the Adapted FRONT defense. As discussed in Section 6, we observe no QoE impact.

Figures 25 and 26 show the corresponding results for the Adapted RegulaTor defense. Again, as discussed in Section 7, we see a high QoE impact, especially in the form of many quality switches.

Finally, Figures 27 and 28 show the corresponding results for the Scrambler defense. As discussed in Section 8, QoE is only slightly impacted at the beginning during buffering and segment spills.

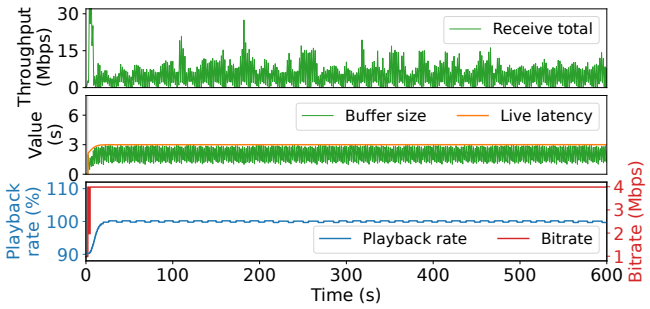


Figure 23: Throughput and video player QoE metrics using Adapted FRONT with $N = 2\,500$ and $W = 7$

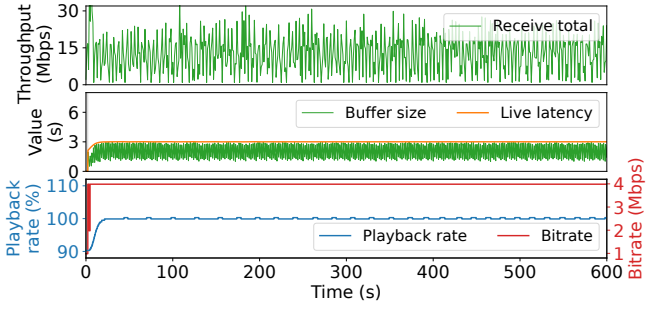


Figure 24: Throughput and video player QoE metrics using Adapted FRONT with $N = 6\,500$ and $W = 2$

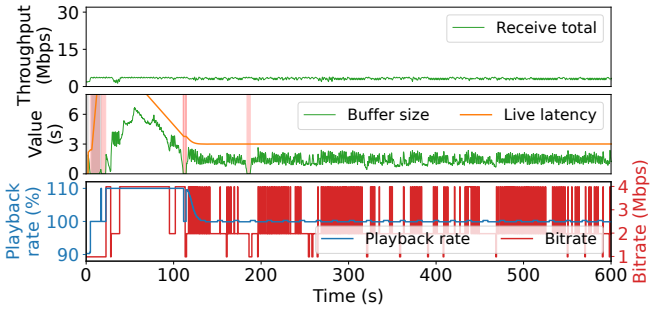


Figure 25: Throughput and video player QoE metrics using Adapted RegulaTor with $R_0 = 500$ and $D = 0.25$

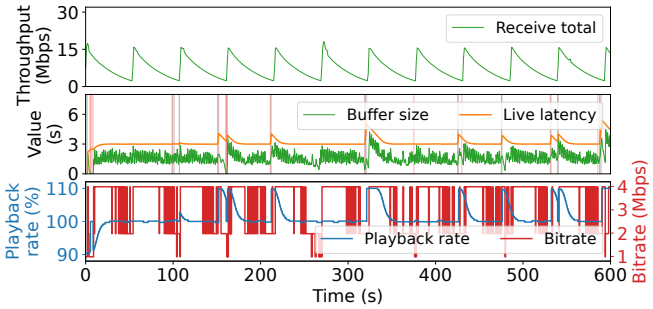


Figure 26: Throughput and video player QoE metrics using Adapted RegulaTor with $R_0 = 1\,900$ and $D = 0.95$

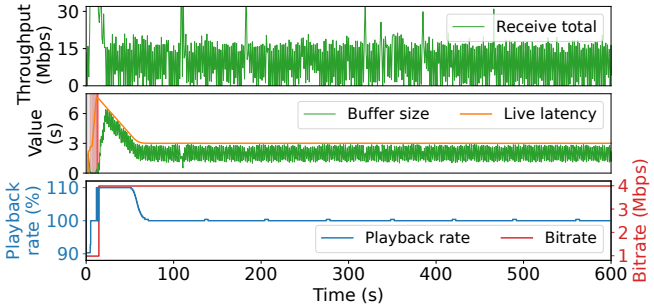


Figure 27: Throughput and video player QoE metrics using Scrambler with $\delta = 160$, $N = 700$, $P_{min} = 400$, $P_{max} = 1\,000$

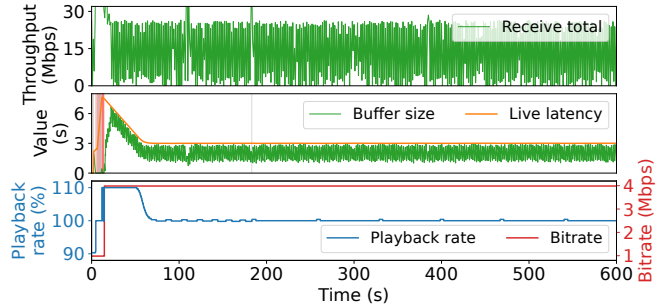


Figure 28: Throughput and video player QoE metrics using Scrambler with $\delta = 120$, $N = 1\,500$, $P_{min} = 400$, $P_{max} = 1\,000$