Twitch Chat Fingerprinting

David Hasselquist*†, Christian Vestlund†, Niklas Johansson†, and Niklas Carlsson*
*Linköping University, Sweden
†Sectra Communications, Sweden

***Abstract*—The streaming content that we choose to watch can reveal much about our thoughts, opinions, and interests. An adversary capable of determining what users watch therefore presents a significant privacy threat. In this paper, we present and evaluate the first fingerprinting attack on Twitch that allows viewers of individual live streams to be identified despite the traffic being encrypted. The attack targets the traffic patterns associated with chat messages associated with each stream. Our results show that high accuracy can be achieved by eavesdropping only for a short time (e.g., 90 seconds) and that the accuracy can be increased even further by interacting with the stream. We also take a closer look at how the accuracy and activity level differ between different Twitch channels and provide insights into the accuracy that attackers using different strategies for selecting target channels may have. Finally, we study countermeasures to protect against such attacks and demonstrate that the naive use of VPN is not enough. We instead present countermeasures altering packet timing and sizes. Our large-scale evaluation of different countermeasures provides important insights that help both the streaming providers and users better protect their privacy.**

***Index Terms*—Fingerprinting attack, Twitch, YouTube Live, Side-channel attack, Encrypted traffic analysis, Live streaming**

I. Introduction

Live video streaming is already responsible for a major part of modern internet activity and is becoming increasingly popular. With these services, *viewers* can tune in and watch any stream created by one of many live *streamers*. Two of the most popular live streaming services are Twitch and YouTube Live; both with millions of live streamers. For example, Twitch has 8 million unique streamers each month that (combined) attract over 31 million daily viewers [1]. While the platforms also serve videos on-demand, many users prefer to watch a live stream as this offers them first-viewer advantage and they can interact with the streamer and other viewers [2].

Not all streams are the same and the streams we choose to watch can therefore reveal information about our thoughts, opinions, and interests. Online streaming services also contain many controversial streams, including streams expressing strong opinions or political views. Being able to identify viewers of these streams can reveal privacy-sensitive information about individuals. For example, a government may perform mass surveillance to identify potential protesters or users with specific opinions. Or, advertisers and political campaigns may try to identify people with particular interests or political biases to target with advertisements or (mis)information [3].

To protect users' privacy, the popular streaming services (and most other websites [4], [5]) today use HTTPS. The use of HTTPS ensures end-to-end encryption and prevents somebody monitoring the internet traffic to view the HTTP requests and data delivered in clear text. However, as we show here, the use of HTTPS does not prevent an adversary from determining what streams a user is watching.

In this paper, we present the first fingerprinting attack against Twitch and live streaming services. Similar to fingerprinting attacks against other services, the attacker is assumed to passively monitor the encrypted network traffic of one or more users.[1] By creating fingerprints for streams of interest and comparing these with the observed traffic patterns, our attack can successfully identify what streams a user is watching. The attack is also the first fingerprinting attack leveraging the chat messages of live streaming services as a side channel. The use of the chat streams allows us to (1) interact with the streams and (2) both passively and actively fingerprint Twitch streams (and other live streaming services) even though their video content is delivered and encoded using Constant Bit Rate (CBR) encoding [8]. This provides us with a unique advantage over previous fingerprinting research of video streams [9]–[12], which have focused on identifying on-demand videos based on their Variable Bit Rate (VBR) encoding.

While also the chat messages are encrypted, our results show that there is sufficient diversity in the traffic patterns generated by these chat streams to identify what streams different viewers watch with high accuracy. In particular, we use extensive measurement-based experiments (§II) to show that the use of two features that are easily extracted from the packet sequences associated with the chat messages (i.e., the packet sizes and their relative timings) are enough to successfully distinguish between different live streams. Here, we show that users can be identified on the fly without the need to break any encryption, target any security flaws, or cooperate with a streaming provider. An adversary only needs to monitor the victim users' connections and some candidate streams that it wants to determine if the users are watching.

Our experiments show that high overall accuracy can be achieved through passive monitoring (§III) but that the accuracy may be somewhat lower for channels with less activity (§IV). For these streams, we demonstrate how much the attacker can improve its accuracy by adding messages of different sizes (§IV). We also show that the attack still is possible when multiple streams are delivered over the same connection and when the network conditions are poor (§III).

[1]For example, an adversary can wiretap the user's connections, sniff wireless packets over the air, or compromise/collaborate with an ISP. As seen in the past [6], [7], governments can also force ISPs to collaborate by establishing new laws and regulations.

978-3-903176-47-8

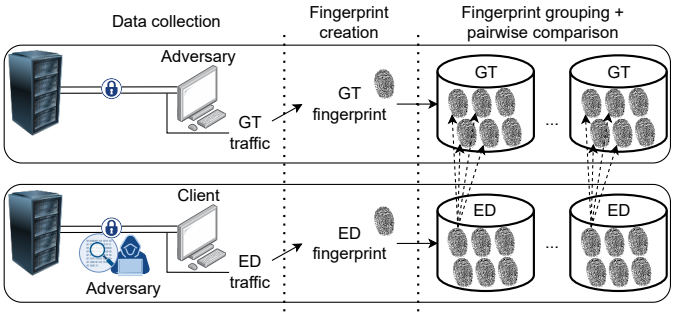
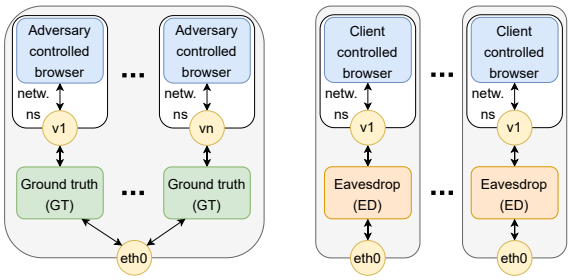


Fig. 1. Framework overview



(a) Local streams (b) Client streams

Fig. 2. Data collection overview

In our analysis, we focus on Twitch streams. It is the most popular live streaming service, offer more streams than other live streaming services, and its API allows us to easily scale our data collection to a large number of streams. We have also validated that the methodology works on YouTube Live.

The large number of channels and high diversity in activity level observed on Twitch (§IV) and other live streaming services present some important tradeoffs regarding which and how many channels an adversary may monitor. To provide insights into the accuracy that attackers using different strategies for selecting target channels may have, we study how the accuracy and activity level differ between different Twitch channels and compare some baseline strategies (§IV).

Finally, we study countermeasures and their effectiveness (§V). While the use of a VPN decreases the classification accuracy, the reduction is far from large enough to mitigate the attack. With the streaming providers themselves having very limited incentive to provide any countermeasures to obfuscate traffic patterns (instead this would result in a bandwidth and latency costs for them), we therefore present a novel client-based fingerprinting timing countermeasure, and a VPN-based countermeasure with padding to provide users further protection when watching video streams with chat messaging functionalities. These countermeasures are demonstrated to be effective, especially when used in combination.

Outline: We first present our attack and evaluation framework (§II), and a measurement-based evaluation of the attack (§III). Next, we study the impact of the channel popularity (§IV), present and evaluate countermeasures (§V), and discuss related work (§VI). Finally, conclusions are presented (§VII).

II. Attack and Evaluation Framework

Our framework to demonstrate and evaluate the effectiveness of the fingerprinting attack is split into two tracks: one for handling and creating Ground Truth (GT) fingerprints, and one for Eavesdropped (ED) fingerprints. The GT fingerprints represent those created over the adversary's network for comparison and include metadata such as labels specifying the associated stream. In contrast, the ED fingerprints are based solely on the encrypted network traffic delivered to the user being identified/attacked.[2] Figure 1 shows our two tracks split into four parts: (1) Data collection, (2) Fingerprint creation, (3) Fingerprint grouping, and (4) Fingerprint comparison/identification. We next describe each part.

Data collection: To classify a user's live stream, we first collect, extract, and label encrypted network traces. Figure 2 shows an overview of our data collection. We use a local server (Figure 2(a)) to crawl n live streams from an adversary-controlled browser and collect data for GT fingerprints. In parallel, we crawl the same n streams on n separate cloud servers and networks (Figure 2(b)) to represent streams from a client-controlled browser. Due to our bandwidth and server limitations, we collect data on four different streams in parallel (n = 4). In practice, an adversary (e.g., government or ISP) can easily circumvent these limitations, listen to many more streams, and stay updated on the current streams.

Although Twitch performs load balancing and can send chat messages for two users from separate servers, we isolate the messages using IP addresses and reverse DNS lookup. On Twitch, we note that (1) video and chat data are delivered over separate TCP sessions from different IP addresses. (2) All IP addresses delivering chat messages resolve to an AWS Compute server in region *us-west-2* (regardless of our geographical location), shown by the resolved hostname *ec2-[ip].us-west-2.compute.amazonaws.com*. (3) Clients make requests using the Internet Relay Chat (IRC) and WebSocket Secure (wss) protocols to URL *irc-ws.chat.twitch.tv*. (4) By periodically resolving the request URL, we are able to obtain a complete list of IP addresses delivering Twitch chat messages. This list is then used to separate the session related to the chat. As discussed later, one can also use the packet size distribution when IP addresses are not available (e.g., due to VPN).

On YouTube Live, we again observe different TCP sessions and IP addresses being used for chat and video data, where HTTPS is used to periodically poll for new chat messages. We were not able to find a clear pattern using DNS, but instead obtained the list of currently used chat servers by periodically crawling live streams and observing IP addresses.

Our framework uses Selenium [13] for crawling, network namespaces [14] to isolate traffic on our servers, and the Twitch API [15] to obtain streamers and their metadata (e.g., number of viewers). We use tcpdump [16] to capture packets forwarded between the server's network interface (eth0) and the virtual interface (v1, ..., vn) of the network namespace.

Based on the stream popularity (discussed in §IV), our crawling process begins with obtaining all online streams with 100 or more viewers. We exclude the unpopular streams (with

[2]For the purpose of evaluation – and evaluation only – we also label the ED fingerprints. In practice, an adversary can only label GT fingerprints.

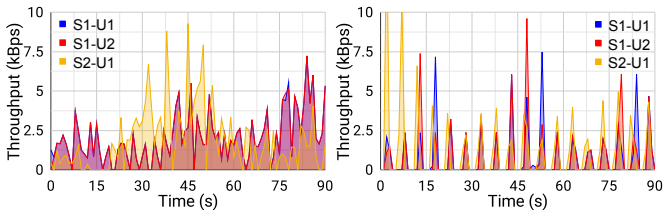
less than 100 viewers) to avoid crawling the long tail and instead focus on the more popular streams as these attract most of the viewers. After crawling through the list (approximately 1,500 to 3,500 streams on Twitch depending on time), we filter those streams containing no messages during our data collection. We collect 90 and 100 seconds of chat stream data for each ED and GT, respectively. As we later discuss, we collect 10 seconds less data for ED fingerprints to consider potential offset. To ensure that the ED window is within the GT window, we start the ED data collection approximately 5 seconds after the corresponding GT data collection. We repeat the process until a total of 10,000 successful fingerprint pairs (i.e., 20,000 fingerprints) have been created.[3] Over our eight experiments, we collected in total 140,000 fingerprints, resulting in 3,700 hours of labeled streaming traffic (each fingerprint is on average 95 seconds long).

Fingerprint creation: Using the collected traffic traces, we create a fingerprint for each user and stream. Figure 3 shows example traces for Twitch chat and YouTube Live chat. We see that two users watching the same stream have similar network patterns, while the patterns differ for users on different streams. Chat messages are bundled and sent every five seconds for YouTube Live, while they are sent immediately on Twitch. Therefore, when creating fingerprints, we choose our sampling rate as 1 and 5 seconds for Twitch and YouTube Live, respectively. A fingerprint is simply defined as the number of bytes observed within each such interval for some time period (e.g., 90 seconds) and their respective order.

Fingerprint grouping: To allow large-scale evaluation, we store the fingerprints and group them before performing pair-wise comparisons. In our default case, we group the created fingerprints into ten buckets of 1,000 pairs each, and then move half of the ED fingerprints to another bucket. As a result, each bucket contains 1,000 ED and 1,000 GT fingerprints, out of which 500 are matching pairs. The choice of bucket size did not impact the average F1-score, although a larger bucket size decreased the standard deviation. For example, with our default settings (described later), bucket sizes of 100, 500, and 1,000 all resulted in an F1-score of 0.97 but with standard deviations of 0.0241, 0.0090, and 0.0077, respectively. The purpose of the bucketing is to obtain a mix of matching and unmatching comparisons, similar to what is seen in practice, and to allow confidence calculation over ten sample sets.

Fingerprint comparison/identification: Before describing our fingerprint identification algorithm, let us first define the distance between two fingerprints ED and GT. Here, we use a variation of the edit distance [17] in which we allow insertion, deletion, and substitutions of data volumes in the fingerprint samples ED_i and GT_i, where $0 \leq i < m$ is the sample index (m=90 for ED; m=100 for GT). Our algorithm also accounts for the two samples being offset by some distance (discussed later) and spill between individual samples due to

[3]We note two exceptions. (1) For the multistream experiment, we collect approximately 5,000 ED and 10,000 GT as an ED fingerprint can contain two streams combined. (2) For our campus-based VPN, we collect only 2,000 GT/ED pairs due to time and rate limitations.



(a) Twitch. Distances: 0.1, 63, 63 (b) YouTube. Distances: 0.8, 8, 7

Fig. 3. Example network traces for three clients on Twitch and YouTube Live chat. The downlink throughput shows that users (U#) watching the same stream (S#) exhibit similar network patterns. Edit distances calculated for S1-U1 to S1-U2, S1-U1 to S2-U1, and S1-U2 to S2-U1, respectively.

jitter. (An example of spill can be observed in Figure 3(b) between S1-U1 and S1-U2 in the 3rd and 4th peak.) For the spill, we also note that the jitter of two users typically do not differ by more than half a second. This ensures that spill at most happens between neighboring sample points. To illustrate this, we generate 10,000 chat messages on our stream and observe the delivery times to two geographically separated users. Figure 4 shows the empirical Cumulative Distribution Function (CDF) for the differences in message delivery times. As shown, 99% of all messages have a delivery time difference of less than 0.2 seconds and 99.9% less than 0.4 seconds.

To account for spill between data samples, we define three alternative cost distances for each sample. In the case there is no spill we denote the difference in data volume between two fingerprint samples $M_i = |GT_i - ED_i|$. For spills to the left, we use a cost term $L_i = |(GT_{i-1} + GT_i) - (ED_{i-1} + ED_i)|$ and for spills to the right we use a cost term $R_i = |(GT_i + GT_{i+1}) - (ED_i + ED_{i+1})|$. We then calculate the cost C_i for the corresponding edit distance operation as follows:

$$C_i = \begin{cases} \frac{M_i}{GT_i} \times \frac{N_{ED}}{A_{ED}}, & \text{if } M_i \leq \min(L_i, R_i) \\ \frac{L_i}{(GT_{i-1}+GT_i)/2} \times \frac{N_{ED}}{A_{ED}}, & \text{if } L_i \leq \min(M_i, R_i) \\ \frac{R_i}{(GT_i+GT_{i+1})/2} \times \frac{N_{ED}}{A_{ED}}, & \text{if } R_i \leq \min(M_i, L_i), \end{cases}$$

where $10 \leq N_{ED} \leq 90$ is the length of ED, $1 \leq A_{ED} \leq 90$ is the activity (number of non-zero samples) in ED, and $A_{ED} \leq N_{ED}$. The scaling $\frac{N_{ED}}{A_{ED}}$ is used to give a fair comparison between fingerprints of different activity; the less activity in the fingerprint, the more important we consider each data volume. Again, the edit distance of two fingerprints is defined as the combined cost of transforming one fingerprint to the other, with C_i being the cost of transformation operation i. The use of insertion and deletion operations in the edit distance allows us to account for network jitter and change of delays.

Now, consider the potential offset $0 \leq j \leq 10$. Even with live streaming, users can have considerably different delivery times of the "live" streams. Using video stats on Twitch and the Stats for Nerds feature on YouTube, we find that the live latency is usually below 10 and 30 seconds, respectively. The chat latency can also be further offset from the video stream. Therefore, when comparing two fingerprints on Twitch, we use a sliding window between the GT and ED stream, and calculate the edit distance for each candidate offset (up to 10 seconds in our experiments, as per construction of ED and

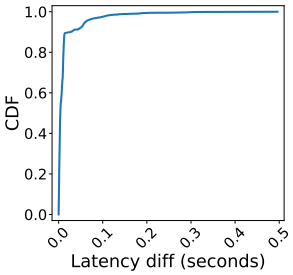
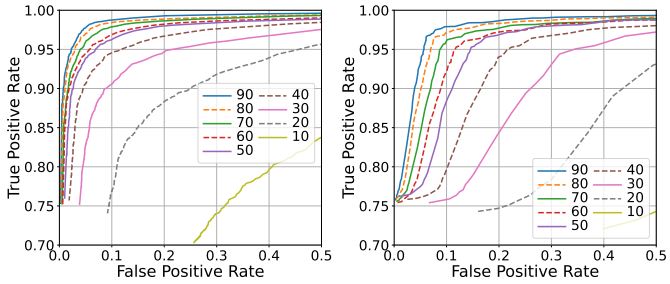


Fig. 4. Message delivery difference between users



(a) Relative classifier. Best F1-score: 0.966 ($N_{ED} = 90$, $\mu = 2.0$)

(b) Absolute classifier. Best F1-score: 0.953 ($N_{ED} = 90$, $\lambda = 3100$)

Fig. 5. Performance results for different eavesdropped length N_{ED} (seconds)

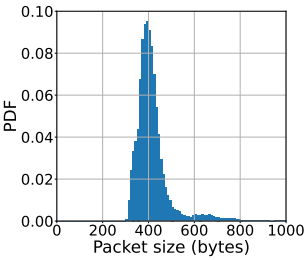


Fig. 6. Twitch chat packet size distribution

GT fingerprints). The final distance between two fingerprints is the lowest possible edit distance obtained for any offset.

Figure 3 shows example distances between three users (both on Twitch and YouTube Live). Here, the distance from two users watching the same stream (i.e., S1-U1 vs. S1-U2) is significantly smaller compared to for two users on different streams (i.e., S1-U1 vs. S2-U1 and S1-U2 vs. S2-U1).

Having explained how we calculate the distance between any pair of fingerprints (in practice simultaneously collected), we are now ready to explain the fingerprint identification step. For every bucket, we calculate the distance of each ED fingerprint to all GT fingerprints in that bucket. For a given ED, this gives us a vector of distances $d = \{d_1, d_2, ...d_{1000}\}$, where d_1 is the shortest distance.

Using our vector of distances, we next classify the ED fingerprint using two classifiers: relative based ($d_2/d_1 > \mu$) and absolute based ($d_1 < \lambda$). If the value is above/below a given threshold, we classify the ED fingerprint as a match with the GT fingerprint of distance d_1. For example, an ED fingerprint with distances of $d = \{20, 180, 185, ...\}$ to other GT fingerprints, and threshold examples of $\mu = 2.00, \lambda = 10$, we consider ED to be a match with only the relative-based classifier ($\frac{d2}{d1} = \frac{180}{20} > 2.00 = \mu$, but $d1 = 20 \not< 10 = \lambda$).

III. Performance Results

Relative vs. absolute classifier: As expected, the relative classifier performs better than the absolute classifier. This is demonstrated in Figure 5, where we show ROC curves for both classifiers for different eavesdropping durations. Here, each curve shows the tradeoff between the true positive rate (TPR) and the false positive rate (FPR). For the relative classifier, a greater threshold value μ results in a smaller TPR and FPR, while a greater threshold value λ for the absolute classifier leads to a larger TPR but at the cost of a larger FPR. Although the differences decrease between the two classifiers as the eavesdropping duration increases, the relative classifier consistently performs better. The differences are also visible when calculating the F1 scores. For example, for the 90-seconds duration, we obtain F1-scores of 0.966 and 0.953 when using the best threshold values of $\mu = 2.0$ and $\lambda = 3100$, respectively. Due to its superior performance, we focus on the results with the relative classifier in the remainder of the paper.

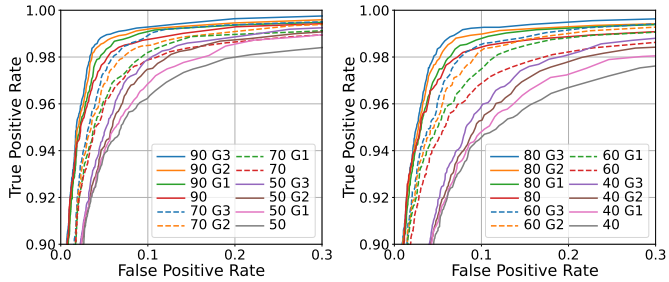
Diminishing improvements: From Figure 5(a) we also observe diminishing improvements as the durations increase. This suggests that most of the advantages can be achieved even with short traces (e.g., 60-90 seconds in duration). In addition to allowing classification of victims only watching the stream for a short time, this allows an adversary to improve their confidence by collecting several GT fingerprints for victims watching the same stream for a longer time. Finally, we note that our distance calculation is much faster for shorter traces.

Classification of low activity streams: While an attacker passively eavesdropping for 90 seconds can achieve an F1-score up to 0.966, we notice upon closer inspection of the failed classifications that these streams in most cases see little chat activity (e.g., less than five messages in 90 seconds). These streams are difficult to classify since there often are several other chat streams with low activity to which we also observe small distances (especially when taking the minimum distance over several offsets). To address this, an adversary can interact with the stream and actively generate messages as the streams are public. This will lead to specific traffic patterns for all users watching the stream. In practice, an adversary can monitor the streams of interest and adapt the amount of generated traffic needed to achieve a desired attack accuracy.

While we look closer at the impact of inserting messages in §IV, we present some initial experiments here in which we add 1-to-3 messages of lengths matching those observed on Twitch. Figure 6 shows the overall message size distribution we observed on Twitch (across users and messages). In our case, we could generate messages between 368 to 867 bytes, as messages are not compressed and consist of between 1 to 500 characters (maximum allowed size) plus some bytes of user-specific metadata. Using a discrete uniform distribution, we randomly generate (1) a packet of size 368 to 867 bytes, and (2) a data volume index. We then add the generated packet size to the corresponding index before identifying the stream. In practice, an adversary can use the packet size distribution to further distinguish from other sizes and use a different weight for these packets/markers when identifying the stream.

Figure 7 shows the ROC curve for these experiments and note that we increase the F1 score from 0.966 to 0.974 when adding three messages (labeled G3) to our default case. In §IV we show that the low activity streams (the majority) in isolation see much greater improvements.

Multiple streams: As one user could be watching several streams, or several users in the same household or organization



(a) Length 50, 70, 90 (seconds). F1-score: 0.974 (90 G3, $\mu = 2.0$)

(b) Length 40, 60, 80 (seconds). F1-score: 0.968 (80 G3, $\mu = 2.0$)

Fig. 7. Performance results when generating/writing 1-3 messages in the chat

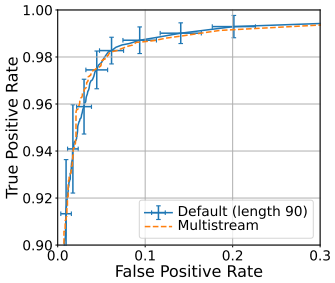


Fig. 8. Multi-stream experiments

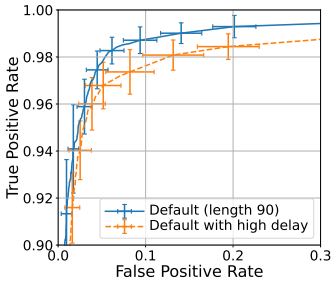
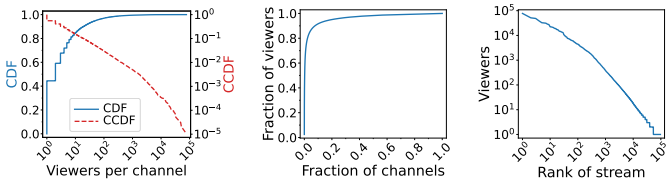


Fig. 9. High-delay experiments

can share IP addresses and network paths, an adversary must be able to distinguish and separate different connections delivered over the same infrastructure. To cover this scenario, we modified the client-side to include two streams and eavesdropped on the network path containing the mix of these streams. We consider a true positive if one of the two streams is identified. Figure 8 shows (1) the ROC curve for the default case (length 90 in Figure 5(a)) with its standard deviation, and (2) the default case but with two streamers over the same network path. The results show that the adversary can separate and identify both eavesdropped streams and achieve similar scores compared to the case with only one stream on the path (e.g., the values are well within the standard deviations).

Network conditions: While our experiments are conducted under good network conditions (high bandwidth, low latency), we argue that the results are more broadly applicable. An attacker (e.g., the ISP) with access to the network traffic has knowledge of the network conditions and can quickly adapt the attack to fit the current environment. For example, packet losses, retransmissions, and congestion can be detected and considered when creating/comparing fingerprints. Furthermore, as users watching a stream can fulfill the bandwidth requirements for video streaming (orders of MB/s), we expect these users to have sufficient bandwidth and network conditions to receive chat messages (orders of kB/s).

The main metric of interest is instead the end-to-end delay. To measure the effects of a high latency, we increase the connection RTT using tc. Here, we add normally distributed packet delays with a mean of $200ms$ and a standard deviation of $20ms$. Figure 9 shows the results. Here, the high-delay connections see somewhat worse performance. The main reason for this is the high degree of packet reordering and retransmissions that occur due to latency variations. Despite



(a) CDF + CCDF (b) Concentration plot (c) Rank plot

Fig. 10. Distribution of Twitch viewers per channel

poor network conditions, we can still achieve a high F1 score (modest decrease from 0.966 to 0.960 in default case).

Cloud provider: Our performance results do not appear to depend on the choice of cloud provider on which we place the adversary. While our primary data collection is performed from a research-based cloud service (WARA-Common), we validated the performance on both AWS and Microsoft Azure. In both cases did we receive scores well within the standard deviations. Our tested cloud instances had a similar capacity and were geographically separated in Europe.

IV. Stream Popularity and Channel Selection

We next look closer at the impact of stream popularity and channel selection strategies used by the adversary.

Channel popularity: We first present a basic characterization of the number of viewers per Twitch channel. Here, we include data for all channels observed on Nov. 7, 2021 (rather than only those that we monitored). Figure 10 shows (a) the CDF and Complementary CDF (CCDF), (b) the concentration plot, and (c) the rank plot of the distribution. The CCDF shows that the distribution is heavy tailed, suggesting that a small subset of the streams is responsible for the majority of the viewers. The concentration plot confirms that the views per channel indeed follows the Pareto principle, with the top-10% streams being responsible for 90% of all viewers. Finally, the straight-line behavior seen in the tail of the rank-plot shows that the long tail of less popular streams is Zipf-like.

Impact of channel popularity: Figure 11 shows the F1-scores obtained for streams of different popularity. Here, we show results (with 95% confidence intervals) for both our default case, using only passive measurements, and when also inserting 1, 2, 4, or 8 additional messages during data collection of a fingerprint (labeled G1-G8). We see that the accuracy of the streams with less than 200 viewers is significantly lower than those with more viewers but that the accuracy is significantly improved by inserting just a few extra messages. Figure 12 shows the corresponding results when streams are grouped based on activity level (defined as the number of non-zero samples during data collection). Here, the F1-score of a stream with five or fewer non-empty samples improve from 0.90 to above 0.97 by adding only 2 messages, and to above 0.99 by adding 8 messages. This clearly shows that a resourceful adversary can target channels of any popularity.

We also note that the most popular streams can already be classified with high accuracy without extra messages. An adversary focusing on the most popular streams therefore does not need to add messages and would easily catch most users. For example, as extracted from actual viewer distributions

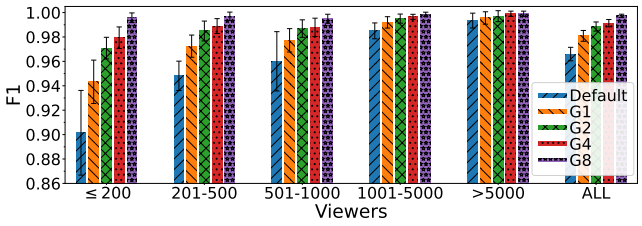


Fig. 11. F1-scores (including 95% confidence intervals) for channels with different number of viewers with and without additional messages ($\mu = 2.0$)

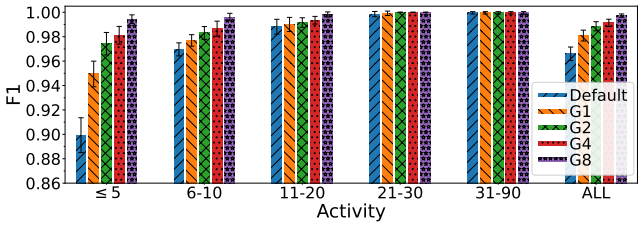


Fig. 12. F1-scores (including 95% confidence intervals) for channels with different activity level (defined as number of non-zero volume samples excluding added messages) with and without additional messages ($\mu = 2.0$)

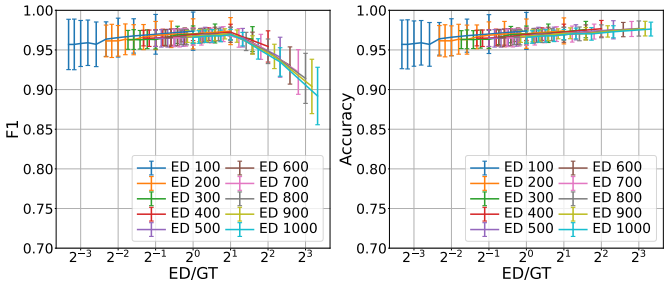
TABLE I
TOTAL PERCENTAGE OF STREAMS AND VIEWERS ASSOCIATED WITH EACH POPULARITY CATEGORY OF CHANNELS

Viewers per stream	$\leq$ 200	201-500	501-1000	1001-5000	>5000
Streams (%)	98.24	0.91	0.35	0.41	0.09
Viewers (%)	22.77	8.59	7.48	26.78	34.38

(Figure 10) and summarized in Table I, the streams with more than 1,000 viewers are responsible for 61% of the viewers but only 0.5% of the streams. In contrast, the channels with at most 200 viewers are responsible for 98% of the streams but only 23% of the viewers. Clearly, an attacker targeting the masses can focus on a few channels and do not need to add extra messages, whereas an adversary targeting niche streams may benefit more from interacting with the stream.

Relative ED to GT ratio: When first selecting the fraction of EDs with a matching GT in our default experiments (i.e., 50%), we considered the case of an adversary targeting the most popular streams looking at random users. For example, by creating GTs only for the streams with more than 1,000 users, we would expect 61% of the ED observations (made by random users) to match one of the GTs. We note that the actual number of unique EDs observed may still be much larger than twice the number of GTs. In fact, if monitoring all clients, the number of unique EDs would be 200 times larger than the number of GTs. Yet, an ED-to-GT ratio of close to 1 best captures the performance we would see for such adversary.

An adversary may of course select to create GTs for other subsets of channels, collect more/less GTs, and/or target other user groups than a random subset (e.g., it may have certain subsets of people it expects are more/less likely to watch certain channels). All these aspects may increase or decrease the fraction of observed EDs for which the adversary has a matching GT. To capture the impact of this, Figure 13 shows the F1-score and accuracy when we vary the ratios of EDs and GTs (for different number of EDs). Before discussing the results, we first provide some examples for the two



(a) F1-score (b) Accuracy

Fig. 13. Performance results with different ratios of ED and GT ($\mu = 2.0$)

extremes. In the left-most cases (ED$\ll$GT), the attacker has significant monitoring resources and is able to create GTs for significantly more streams than it observes (and the adversary will typically have created a matching GT). In contrast, in the right-most case (ED$\gg$GT), an example attacker (e.g., ISP or government) may performs large-scale surveillance on a large, diverse/general group of users to identify who watch a small subset of niche streams. While we observe the best F1-score near a ratio 2, with F1-scores decreasing (and standard deviations increasing) as the ratio increases, the accuracy is high throughout (as we still do a good job identifying true negatives when the ratio is large). These results suggest that the attack is most successful when the attacker knows something about its candidate pool of users (e.g., that a random user is more likely to watch a popular stream or that a group of users is more likely to watch certain niche content) but can still achieve high accuracy also in other cases.

V. COUNTERMEASURE PERFORMANCE

We next present, deploy, and evaluate five countermeasures against our fingerprinting attack. All countermeasures can be deployed by any user and require no modification or cooperation from streaming or network providers. The countermeasures span from the use of VPNs to a novel technique that uses TCP flags to modify packet timings and sizes.

A. Countermeasures

(1) Campus VPN and (2) OpenVPN: As a first countermeasure, we tunnel the user's traffic through a campus-based off-the-shelf VPN, a service similar to what we expect a regular user to use. Second, we deploy a VPN based on OpenVPN on a geographically separated cloud server and compare this to the off-the-shelf VPN. When extracting packet sizes, we assume that an adversary is aware of the VPN being deployed and deduct an approximate overhead per packet. For our VPNs, we observe this overhead to be 82 and 76 bytes, respectively. Due to the original IP address being hidden, we are unable to separate TCP sessions delivered over a VPN. An attacker can instead use the packet size distribution to extract chat messages. As Twitch uses CBR, the video packet sizes are fixed and can be deducted from the data volumes. Occasionally, we must account for changes in streaming quality due to DASH. However, the approach is affected by background traffic. Interesting future work therefore includes evaluating the effect of different amounts of background traffic.

(3) Client timing: Next, we alter the timing of packets sent from Twitch's server using a novel client-based countermeasure based on the TCP receive window. By sending TCP Zero Window packets to Twitch's servers, we can control the flow of incoming packets. Our client-based algorithm uses two randomized parameters t_z and t_n for the number of seconds to announce a zero and normal receive window size, respectively. When a new packet is received from the server, a client sends a TCP Zero Window acknowledgment packet to the server if t_n seconds have passed since the last zero window packet. Following every zero window packet, the client sends a TCP Window Update packet after t_z seconds containing the normal receive window size, notifying the server that the client is ready to receive data. The server then sends a burst of packets containing the data it has buffered during the silent period.

Figure 14 shows an example of the throughput varying over time with and without deploying the client-side timing countermeasure. Here, we deploy the client timing using t_z=5 seconds and t_n=2 seconds. During the silent period, where the client announces a zero receive window size, we see that the chat messages are buffered at the server to later be sent in a burst when a normal window size is received. Since packets can be in transit from the server when a client announces a zero window, packets can still arrive during the silent period.

(4) OpenVPN + padding: As our client-based countermeasure can only modify the packet timing, we study the effectiveness of using padding on packets by tunneling the traffic through our OpenVPN. Figure 15 shows an overview of our padding countermeasure. Here, a client connects to a video streaming server through a VPN server, and the full path is encrypted with HTTPS. Using libnetfilter-queue [18] on the VPN server, we append padding to packets carrying chat messages to the client such that the packet reaches an MTU of 1,500 bytes. On the client-side, the padding gets removed when extracting the original packet from the VPN connection.

By using a VPN with padding, an adversary observing packets between the VPN server and the client cannot use packet sizes nor original transport headers as a feature to distinguish between various chat messages. We note that the VPN server ideally should be placed outside the adversary's region (e.g., another country), such that the traffic between the VPN and video streaming server is not visible for an attacker. For example, the VPN server could be hosted on the same infrastructure as the video streaming server to minimize the attack surface. In the case of Twitch, owned and hosted by Amazon, AWS could hence be a good host.

(5) OpenVPN + padding + client timing: Finally, we tunnel the client's traffic through our OpenVPN and deploy both the padding and the client-based timing countermeasure.

B. Example results

Figure 16 shows the performance results of our presented countermeasures. As expected, our OpenVPN countermeasure performs similar but slightly worse than the campus-based off-the-shelf VPN. This is due to the OpenVPN being deployed in a controlled environment with little noise and background

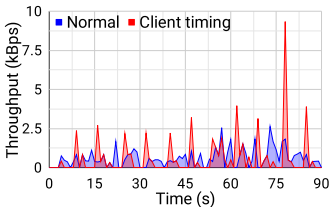


Fig. 14. Example of client-based timing countermeasure ($t_z = 5, t_n = 2$)

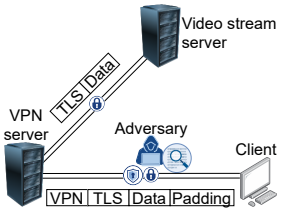


Fig. 15. Overview of countermeasure using OpenVPN and padding

traffic. By using the campus-based VPN, the best F1-score decreased from 0.966 to 0.810. For OpenVPN, we observe a decrease to 0.826. While the use of VPN offers some small protection, it is clear that users are still vulnerable to the attack.

For our client-based timing countermeasure, we present example results using $t_z = 5, t_n = 2$. For an active stream, the traffic will alternate between approximately 5 seconds of silence and 2 seconds of transmission, with the beginning of the transmission containing a burst of buffered packets. Here, we see that the client-based countermeasure is more effective than the VPNs, achieving a best F1-score of 0.637.

When deploying OpenVPN together with padding, the classification performs much worse, achieving an F1-score of only 0.152. When adding the client timing on top of OpenVPN and padding, we eliminate both packet timing and size aspects. In this case, the attack is almost entirely prevented.

In our evaluation, we assume an adversary that can observe all packets associated with a connection. An alternative countermeasure to those presented here is to reduce the amount of traffic that can be observed from a vantage point. By routing packets over multiple network paths (e.g., over different ASes) we can limit the data an adversary can collect. Interesting future work includes evaluating our approach when an adversary can only create fingerprints using parts of the traffic.

C. Impact of timing parameters

Running the experiments needed for a single parameter setting is very time consuming, as our evaluation methodology requires 10,000 pairs of fingerprints to be collected for each configuration. We therefore opted to use trace-driven simulations when evaluating the effectiveness of our client-based countermeasure together with other timing parameters. Here, we used our dataset for the default case (90 seconds) and the packet size distribution to simulate the number of bytes in transit at each time. We evaluate our simulated model in Figure 17, where we show results (with standard deviations) for $t_z = 5, t_n = 2$ with both our real-world collected dataset and the corresponding simulations. As shown, our simulation model produces results well within the standard deviations.

Figure 18 shows the simulated results for other parameters. As expected, a larger silence period (t_z) decreases the accuracy of the attack at the cost of the data freshness. We also note that our default choice ($t_z = 5, t_n = 2$) provides relatively good protection. However, as seen in Figure 16, the best protection is to combine this with VPN and padding.

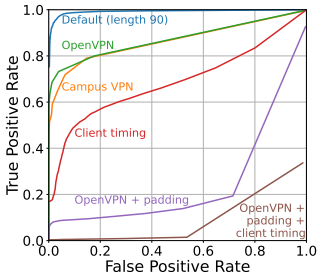


Fig. 16. Performance impact of various countermeasures

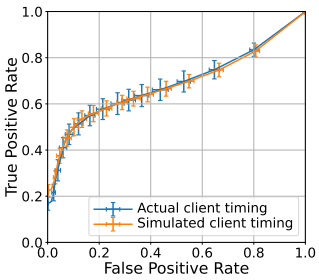
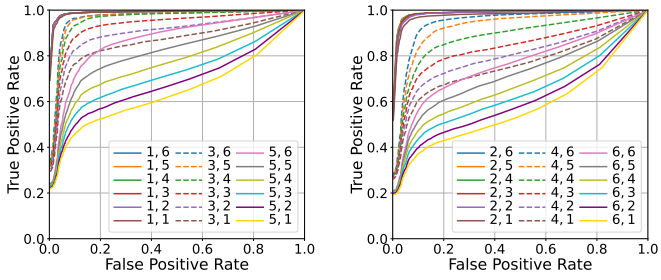


Fig. 17. Comparison of actual vs. simulated client-based countermeasure with parameters $t_z = 5, t_n = 2$



(a) $t_z = \{1, 3, 5\}, t_n = \{1..6\}$ (b) $t_z = \{2, 5, 6\}, t_n = \{1..6\}$

Fig. 18. Performance impact of various client-based countermeasure timing parameters

D. Ethical considerations

Our experiments did not compromise the privacy of any real-world viewers or streamers. Data were only collected on public live streams. We did not perform the attack on any real users, only our own streams. Finally, we only manipulated our own chat streams when evaluating countermeasures.

VI. Related Work

Website fingerprinting: Previous works have studied fingerprinting attacks and encrypted traffic analysis in various contexts. In website fingerprinting, an adversary tries to classify visited websites based on the encrypted traffic. Website fingerprinting is a well-studied area, with various attacks [19]–[22] and countermeasures [23]–[27] proposed throughout the years. Recent advancements in machine learning have also led to models utilizing various classification algorithms and deep neural networks to analyze encrypted network traffic [28]–[36]. While some of these approaches may be applicable on Twitch as well, they are not designed for the live streaming or live chat context but typically target (more) static webpages.

Video streaming: While some have studied live streaming services and their performance [37], [38], others have proposed frameworks to identify encrypted traffic on various streaming platforms [9]–[12], [39]. Reed and Kranch [10] use TCP/IP headers of HTTPS packets to identify streaming videos from Netflix. Schuster et al. [11] leverage information leakage from the VBR used with DASH to fingerprint and identify YouTube, Netflix, Amazon, and Vimeo videos delivered over HTTPS. Gu et al. [12] perform a similar study and utilize VBR to fingerprint video streams. Li et al. [39] identify YouTube video flows from encrypted traffic, but do not match these and distinguish between videos. While many have studied video streaming, none of these works focus on live streaming. Furthermore, no prior work has performed fingerprinting on Twitch videos. A possible reason is that Twitch uses CBR, making the problem non-trivial. Here, we utilize a side-channel (i.e., chat messages) to passively identify streams. In addition, this allows us to interact with the stream and modify viewers network traffic to create more distinct fingerprints.

Message fingerprinting: Previous works have focused on identifying encrypted messages in the context of messaging applications [40]–[43]. For example, Bahramali et al. [43] study instant messaging applications and perform encrypted traffic analysis on chat messages to identify administrators and group members. The authors also study potential countermeasures for altering packet timing and sizes. While their work is similar to ours, they consider a different problem, address different challenges, and we note several fundamental differences between chat messages of video streaming and instant messaging applications. For example, chat messages of live video streaming such as Twitch are smaller and occur in higher frequency. While a message flow on Twitch can consist of several messages per second, the messages on a messaging application are typically more spread out and can consist of larger transmissions (e.g., images, video, and other files).

Other fingerprinting areas: Fingerprinting attacks have also been applied to many other areas, including to identify voice traffic [44]–[46], mobile applications [47], [48], devices [49], user actions [50], and LTE traffic [51].

While many previous works have studied fingerprinting attacks and encrypted traffic analysis in various contexts, we are the first to present and evaluate such attack on chat messages of live video streaming.

VII. Conclusion

We have presented and evaluated the first fingerprinting attack on Twitch's live streaming service. Our attack uses the encrypted chat messages as a side channel and allows us to identify what streams a user is watching with high accuracy despite the video stream being encrypted and Twitch using CBR. To evaluate the attack, we used large-scale measurement experiments capturing 140,000 fingerprints (3,700 hours of labeled data). Our results show that high accuracy can be achieved by passively eavesdropping only for a short time (e.g., 90 seconds) and that the accuracy can be increased even further by actively introducing additional messages into streams with few active viewers. Implications of the stream popularity distributions and different adversarial attacks are discussed using the actual stream popularity distribution observed on Twitch. Finally, we have presented and evaluated countermeasures, with results suggesting that a client can best protect itself through a combination of countermeasures (e.g., VPN with padding and client-side manipulation of packet timings). Our findings demonstrate that the naive use of end-to-end encryption (e.g., via HTTPS or VPNs) is not sufficient to protect users' privacy.

Acknowledgment

The authors are thankful to our shepherd Suranga Seneviratne and the anonymous reviewers for their feedback. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] Twitch, "Press center," https://www.twitch.tv/p/press-center/, 2022.
[2] M. Hu, M. Zhang, and Y. Wang, "Why do audiences choose to keep watching on live video streaming platforms? An explanation of dual identification framework," *Computers in Human Behavior*, 2017.
[3] E. Zeng, M. Wei, T. Gregersen, T. Kohno, and F. Roesner, "Polls, clickbait, and commemorative $2 bills: Problematic political advertising on news and media websites around the 2020 U.S. elections," in *Proc. ACM IMC*, 2021.
[4] J. Aas, R. Barnes, B. Case, Z. Durumeric, P. Eckersley, A. Flores-López, J. A. Halderman *et al.*, "Let's encrypt: an automated certificate authority to encrypt the entire web," in *Proc. ACM CCS*, 2019.
[5] Google, https://transparencyreport.google.com/https/overview, 2022.
[6] R. S. Raman, A. Stoll, J. Dalek, R. Ramesh, W. Scott, and R. Ensafi, "Measuring the deployment of network censorship filters at global scale," in *Proc. NDSS*, 2020.
[7] R. S. Raman, L. Evdokimov, E. Wurstrow, J. A. Halderman, and R. Ensafi, "Investigating large scale HTTPS interception in Kazakhstan," in *Proc. ACM IMC*, 2020.
[8] Twitch guidelines, https://help.twitch.tv/s/article/broadcast-guidelines.
[9] A. Reed and B. Klimkowski, "Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11n connections," in *Proc. IEEE CCNC*, 2016.
[10] A. Reed and M. Kranch, "Identifying HTTPS-protected Netflix videos in real-time," in *Proc. ACM CODASPY*, 2017.
[11] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and the burst: Remote identification of encrypted video streams," in *Proc. USENIX Security*, 2017.
[12] J. Gu, J. Wang, Z. Yu, and K. Shen, "Walls have ears: Traffic-based side-channel attack in video streaming," in *Proc. IEEE INFOCOM*, 2018.
[13] Selenium, https://www.selenium.dev/, 2022.
[14] "ip-netns(8) - linux manual page," https://man7.org/linux/man-pages/man8/ip-netns.8.html, 2022.
[15] Twitch, "API documentation," https://dev.twitch.tv/docs/api/, 2022.
[16] Tcpdump, https://www.tcpdump.org/, 2022.
[17] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, 1966.
[18] Netfilter, https://netfilter.org/projects/libnetfilter_queue/, 2022.
[19] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Proc. IEEE S&P*, 2002.
[20] X. Cai, X. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proc. ACM CCS*, 2012.
[21] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit fingerprinting attacks: Passive deanonymization of tor hidden services," in *Proc. USENIX Security*, 2015.
[22] D. Hasselquist, M. Lindblom, and N. Carlsson, "Lightweight fingerprint attack and encrypted traffic analysis on news articles," in *Proc. IFIP Networking*, 2022.
[23] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail," in *Proc. IEEE S&P*, 2012.
[24] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *Proc. ESORICS*, 2016.
[25] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *Proc. USENIX Security*, 2017.
[26] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko, "Trafficsliver: Fighting website fingerprinting attacks with traffic splitting," in *Proc. ACM CCS*, 2020.
[27] D. Lu, S. Bhat, A. Kwon, and S. Devadas, "Dynaflow: An efficient website fingerprinting defense based on dynamically-adjusting flows," in *Proc. ACM WPES*, 2018.
[28] M. Nasr, A. Bahramali, and A. Houmansadr, "DeepCorr: Strong flow correlation attacks on Tor using deep learning," in *ACM CCS*, 2018.
[29] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale," in *Proc. NDSS*, 2016.
[30] T. Wang, "High precision open-world website fingerprinting," in *Proc. IEEE S&P*, 2020.
[31] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proc. ACM CCS*, 2018.
[32] S. Bhat, D. Lu, A. H. Kwon, and S. Devadas, "Var-CNN: A data-efficient website fingerprinting attack based on deep learning," in *PETS*, 2019.
[33] S. E. Oh, N. Mathews, M. S. Rahman, M. Wright, and N. Hopper, "GANDaLF: GAN for data-limited fingerprinting," in *Proc. PETS*, 2021.
[34] C. Wang, J. Dani, X. Li, X. Jia, and B. Wang, "Adaptive fingerprinting: website fingerprinting over few encrypted traffic," in *Proc. ACM CODASPY*, 2021.
[35] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet fingerprinting: More practical and portable website fingerprinting with N-shot learning," in *Proc. ACM CCS*, 2019.
[36] T. Wang and I. Goldberg, "On realistically attacking tor with website fingerprinting," in *Proc. PETS*, 2016.
[37] M. Laterman, M. Arlitt, and C. Williamson, "A campus-level view of Netflix and Twitch: Characterization and performance implications," in *Proc. SPECTS*, 2017.
[38] S. Keshvadi and C. Williamson, "An empirical measurement study of free live streaming services," in *Proc. PAM*, 2021.
[39] F. Li, J. W. Chung, and M. Claypool, "Silhouette: Identifying YouTube video flows from encrypted traffic," in *Proc. ACM NOSSDAV*, 2018.
[40] S. E. Coull and K. P. Dyer, "Traffic analysis of encrypted messaging services: Apple imessage and beyond," *ACM CCR*, 2014.
[41] K. Park and H. Kim, "Encryption is not enough: Inferring user activities on KakaoTalk with traffic analysis," in *Proc. WISA*, 2015.
[42] S. Keshvadi, M. Karamollahi, and C. Williamson, "Traffic characterization of instant messaging apps: A campus-level view," in *Proc. IEEE LCN*, 2020.
[43] A. Bahramali, R. Soltani, A. Houmansadr, D. Goeckel, and D. Towsley, "Practical traffic analysis attacks on secure messaging applications," in *Proc. NDSS*, 2020.
[44] X. Wang, S. Chen, and S. Jajodia, "Tracking anonymous peer-to-peer VoIP calls on the internet," in *Proc. ACM CCS*, 2005.
[45] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing Skype traffic: when randomness plays with you," in *Proc. ACM SIGCOMM*, 2007.
[46] J. S. Atkinson, M. Rio, J. E. Mitchell, and G. Matich, "Your WiFi is leaking: Ignoring encryption, using histograms to remotely detect Skype traffic," in *Proc. IEEE MILCOM*, 2014.
[47] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler: Towards automatic fingerprinting of Android apps," in *Proc. IEEE INFOCOM*, 2013.
[48] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Trans. on Information Forensics and Security*, 2017.
[49] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Dependable and Secure Computing*, 2005.
[50] G. Rizothanasis, N. Carlsson, and A. Mahanti, "Identifying user actions from HTTP(S) traffic," in *Proc. IEEE LCN*, 2016.
[51] K. Kohls, D. Rupprecht, T. Holz, and C. Pöpper, "Lost traffic encryption: fingerprinting LTE/4G traffic on layer two," in *Proc. WiSec*, 2019.