De-anonymizing Encrypted Video Streams

Master's Thesis

7 September 2018

Stefano Peverelli
pstefano@student.ethz.ch

supervised by
Prof. Ankit Singla & Melissa Licciardello

Systems Group
Department of Computer Science
ETH Zürich

Abstract

In the last recent years streaming services such as Netflix, Youtube, Amazon Prime Video, Hulu and others, have become the main source for video content delivery to the public. With the effort of private companies and of the AOM consortium [1], various coding formats and streaming techinques have been refined and have been vastely adopted by the industry. *Variable Bitrate Encoding* and *Adaptive Bitrate Streaming*, between others, are two techniques that increasingly often get coupled together to improve high quality streaming of media content over HTTP.

Netflix relies on first encoding a title at a *variable* bitrate, and streaming it using DASH *Dynamic Adaptive Streaming over HTTP*, an instance of Adaptive Bitrate Streaming originally developed by MPEG. In DASH each media file gets encoded at multiple bitrates, which are then partitioned into smaller segments and delivered to the user over HTTP. As of today, Netflix makes extensive use of DASH, by encoding each title in their catalogue at different quality levels, so that the user is served tailored-size content depending on network conditions, and its device capabiities [2].

DASH and *variable-bitrate-encoding*, have proven to be leaking potentially harmful data. Reed et Al. [3] have shown, how despite a recent upgrade in Netflix infrastructure to provide HTTPS encryption to video traffic, it is possible to recover unique *fingerprints* for each title, and match them against future traffic traces. In their study, they make use of adudump [4] a command-line program that can run on passive TAP devices [5] or on a live network interface, and uses TCP and ACKs sequences to infer the size of application data units *ADUs* transferred over each TCP connection in real time.

Our approach reiterates parts of Reed et Al.'s work, but cannot rely on their every assumption and discovery, due to constant changes that Netflix is integrating into their enconding and streaming algorithms. Moreover our intent is focused on finding out if, by analyzing coarse-grained traffic data, we are able to identify a video based on its *bitrate-ladder*.

TODO: *refine it and add results*

Contents

1 Introduction

According to the latest Cisco's VNI [6], video will account for 82% of all IP traffic in Europe by 2021; in addition, the overall IP traffic per person will triplicate from 13*GB* to 35*GB*. These forecasts clearly picture the growth of the streaming industry, posing, at the same time an important question on the present and future states of the final user's privacy.

As shown by Reed et Al. [3] anonimity of user's viewing activity is at risk. Not for the use that Netflix or other streaming services do of user's session data, but because of the risk of a man-in-the-middle attack *MITM* carried by an *evil* party that has control over the flow of packets over a network.

In particular, they have shown how the adoption of HTTPS to protect video streams from Netflix *CDN*s to user's end devices, does not hold against passive traffic analysis.

1.1 Motivation

The goal of this project is to replicate part of the work conducted by Reed et Al. and to investigate the possibility of identifying a Netflix stream solely based on the observed average bandwidth. This, follows from the intuition that *per-title encoding* embeds the nature and the complexity of video frames in a unique way, that may reveal the identity of the content being streamed.

1.1.1 Per-Title Encoding

In December 2015 Netflix announced [2] that it was introducing a new method to analyze the complexity of each title and find the best encoding recipe based on it. Their goal with the adoption of per-title encoding was to provide users with better quality streams at a lower bandwidth.

Before then, each title was encoded with a *Fixed Bitrate Ladder*; their pipeline returned a list of {*Bitrate, Resolution*} pairs that represented the sufficient bitrate to encode the stream at a certain resolution (Table 1.1), with no visible artificats.

Bitrate (kbps)	**Resolution**
235	320×240
375	384×288
560	512×384
750	512×384
1050	640×480
1750	720×480
2350	1280×720
3000	1280×720
4300	1920×1080
5800	1920×1080

Table 1.1: Netflix original's Fixed Bitrate Ladder.

This "one-size-fits-all" ladder, as reported, achieved good results in the encoded video's perceived quality (**PSNR** [7]) given the bitrate constraint, but, would not perform optimally under certain conditions. For instance, high detailed scenes with sudden changes of light, or rapid transitions of camera shots, would require more than 5800*kbps;* in contrast, more static frames, as in animated cartoons, may be encoded at higher resolutions mantaining the same bitrate level.

In summary they noticed how in certain cases, the produced encoding would either present some small artifacts (*e.g.* complex scenes), or waste bandwidth, (*e.g.* static, plain scenes). For this reason, they came up with per-title encoding.

In order to find the best fitting bitrate ladder for a particular title, there are several criterias that they took into account, the principal ones being:

- How many quality levels should be encoded to obtain a *JND* between each of them.
- Best {*Resolution, Bitrate*} pair for each quality level
- Highest bitrate required to achieve the best perceivable quality

As aforementioned, each title's perceived video quality, gets computed as a measure of *Peak signal-to-noise ratio*. The comparison is performed between the produced encode, upsampled to 1080*p*, and the original title in 1080*p*, and the best {*Bitrate, Resolution*} pair is assigned to that specific quality level, as depicted in Table 1.2.

In Figure 1.1, we can see the impact of per-title encoding on the original bitrate ladder: in order to achieve the same perceivable quality level (point **B** and **C**), it requires a lower bitrate to be encoded to (point **A**). Moreover, with around the same bitrate, one can see

Resolutions	Fixed Bitrate Ladder (kpbs)	Per-Title Bitrate Ladder (kbps)
320×240	235	150
384×288	375	200
512×384	560	290
512×384	750	
640×480	1050	
720×480	1750	440
720×480		590
1280×720	2350	830
1920×1080	3000	1150
1920×1080	4300	1470
1920×1080	5800	2150
1920×1080		3840

Table 1.2: Comparison between the two different approaches for the same title: note how different titles may have different numbers of quality levels. For each movie, the minimum number of quality levels gets computed to produce a just-noticeable-difference (JND), when switching bitrates during playback.

how per-title encoding can achieve a higher resolution compared the fixed case (point **A** and **D** respectively). It follows obviously that, holding to a high-quality stream while maintainig or lowering the used bandwidth is key: the end user will get same or better quality then before, at a lower bandwidth.

1.1.2 User's Privacy

As of 2018, data began the most valuable commodity on the planet [8], and with its value rising, society starts to question how to legislate to protect both industry's and user's rights.

In 2016, Netflix announced the introduction of HTTPS to protect the content being streamed to users. With the addition of TLS on top of HTTP, Netflix aim, was to avoid the risk of eavesdropping on unsecure connections, protecting themselves and users from third-party applications and governments potentially collecting viewer's data and streaming habits.

Avoiding deep packet inspection from potential eavesdropper certainly adds another layer of security, but given the narrow relationship between Netflix and ISPs, one might conclude that the chance that IP packets do not get inspected by ISPs (especially in the United States, after the reclassification of ISPs as Title I *Information Services* [9]), is still a matter of mere trust given to the platform-provider relationship.

Cases of user's data breaches as the Kanopy one [10], are a testimony of how easy would be for an attacker to extract information to the point at which users could become

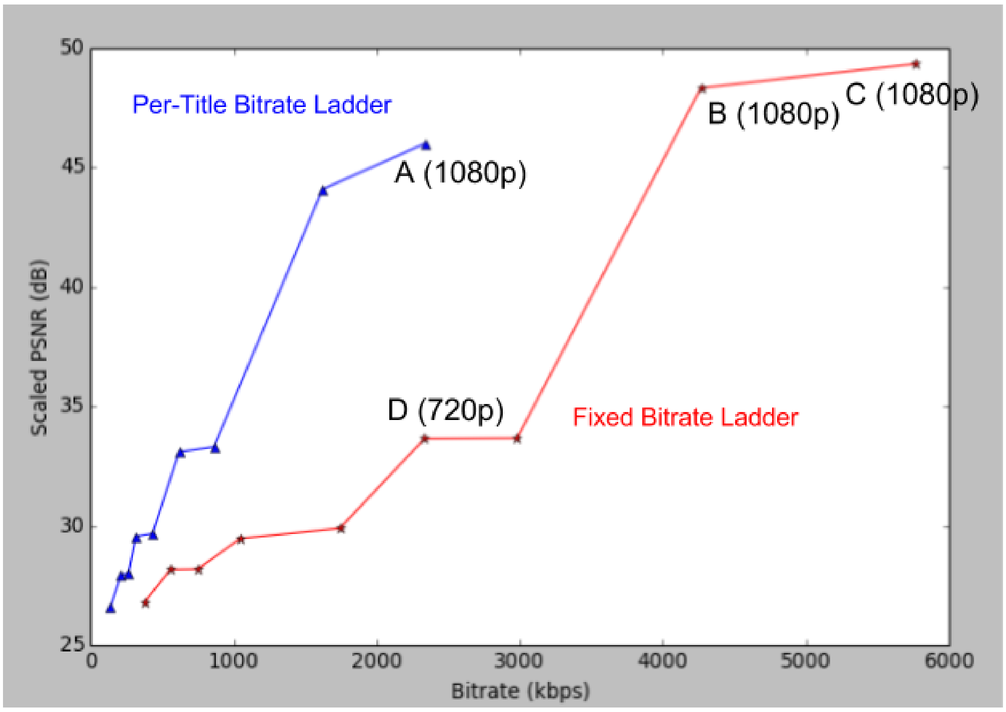

Figure 1.1: Difference between per-title vs. fixed bitrate ladders.

identifiable by the solely information the platform was collecting in their internal log files. The content of which, include between others: timestamps, geo-location data, client-device informations and IP addresses.

The ability to cluster people based on just their video-streaming habits, poses a potential threat for how the data could be processed and used by parties with access to it. One could imagine how government agencies could easily get in possession of sensitive information about the nature of the content a particular user is interested about, or how ISPs could profit from selling data profiles to advertisement companies that in turn would exploit their information to improve per-user recommendation algorithms.

Considering this trend, it is for us crucial to investigate how parties that can have access to transport-layer information, (more details in Chapter 3), could exploit per-title encoding to identify video traffic.

1.2 Related Work

TODO: add more

As previously mentioned, our work is mainly insipired by Reed et Al.'s [3] research paper, in which they presented a novel method to de-anonymize encrypted netflix stream in real-time with limited hardware requirements. Their system was able to identify a video using uniquely TCP/IP headers, by making use of `adudump`, a command-line program built on top of *libpcap* [11], a powerful C library for network traffic capture. They acquired for each video, metadata information with a tampering tool, and then matched adudump traces against with. The evaluation of their method revelaed that they could identify majority of videos by recording only 20 minutes of traffic each.

An earlier proof on how bandwidth analysis could reveal the content of ecnrypted traffic, was given by Saponas et Al. [12], in their "SlingBox-Pro" case study, that exposed how, by recording network traffic and producing and combining different trace levels, they were able to identify 98% of 40 minutes video traces.

Similar work has been conducted also by Moser [13], who analyzed how bitrate ladders could uniquely embed the identity of Netflix titles. In his work he further studies the impact of the aggregation of each video's segment bandwidth on the overall accuracy of his system.

1.3 Main Objective

The main goal of this project is to build a system that can manipulate network bandwidth and observe video traffic from Netflix, to verify the intuition that each title could be identified by its own bitrate ladder.

Furthermore we investigate and discuss possible countermeasures that streaming providers could adopt to preserve users privacy. A detailed explanation of our approach is presented in Chapter 3

1.4 Structure of this Report

In this section, we outline the contents and the structure of this report.

Chapter 1 introduced our motivation behind the project, presented featured work that influenced and inspire our approach, and listed the goals we would like to ultimately achieve.

Chapter 2 presents the attack scenario from a perspective of a malicious user that has compromised a node on the path between the client and the server), the infrastructure needed to acquire the data, the nature of the information that could be infered, and the consequences that might arise.

Chapter 3 shows our version of the attack, in a different context, but up to some extent, with similar conditions to the one depicted in Chapter 2. It also higlights the similiarities and differences from the featured approaches we followed.

Chapter 4 we evaluate our system, we present results, and discuss the relevance of such a method.

Chapter 5 tries to summarize and draw conclusions based on the claims made at the beginning of this report.

2

End-to-End Attack Scenario

Of great importance to our approach, is aquiring real-time network traffic, with downstream throughput being our only focus; here we discuss the potential scenarios that allow adversaries to obtain such information, and describe how, in particular ISPs, as described in Section 1.1.2, may be collecting personal data on behalf of goverments, or may independently decide to profile users to attract companies that in turn could target customers with ad-hoc advertising. What follow, can be considered a general scenario of an attacker that has compromised, or has gained access, of a node between the client's device and the server sending the video-stream.

2.1 Netflix-ISP relationship

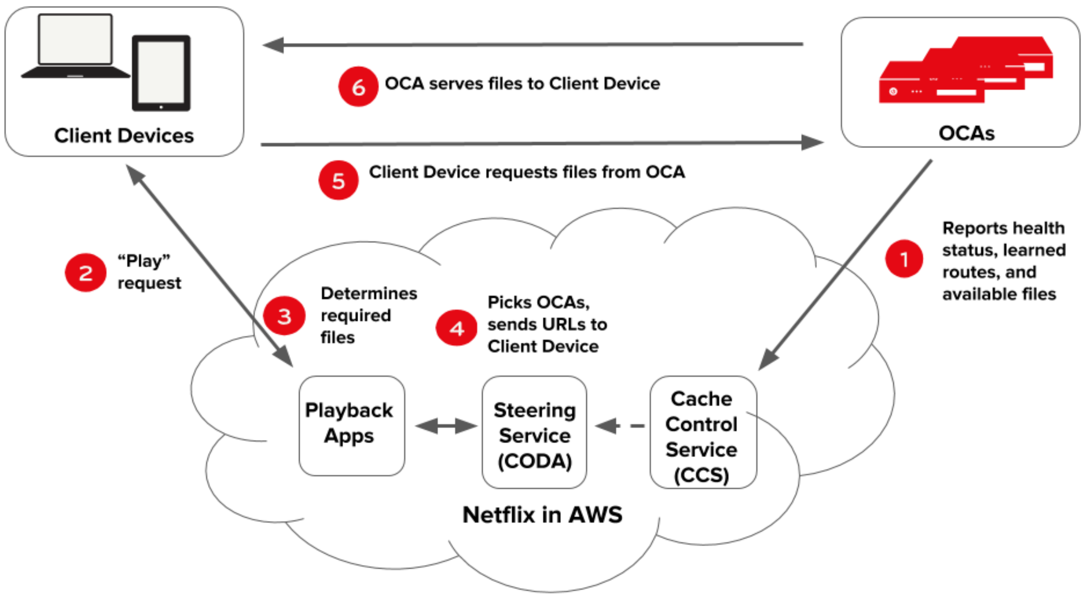

Figure 2.1: Playback process diagram. [14]

Above is illustrated the process of retrieving content whenever a user wants to play a Netflix title. OCAs *Open Connect Appliances* are purpose-built servers responsible to store and serve video files over HTTPS to client devices. OCAs as Netflix claims, do not store client data (DRM info or member data), but only report health metrics to Netflix monitoring services in AWS.

Whenever a play request is issued by a client device, playback application services in AWS determine which streaming assets are required to handle the streaming of the intended title

Then the steering service uses cache-control services to select the best OCAs depending on client characteristics and network conditions, generating URLs that are passed back to the client device, that is now able to "communicate" with the OCAs.

TODO: *insert an example of URL, and explain how ISP cannot associate a URL to a specific title in a trivial way*

Netflix partners with ISPs in the deployment of their CDN, following are presented two different policies:

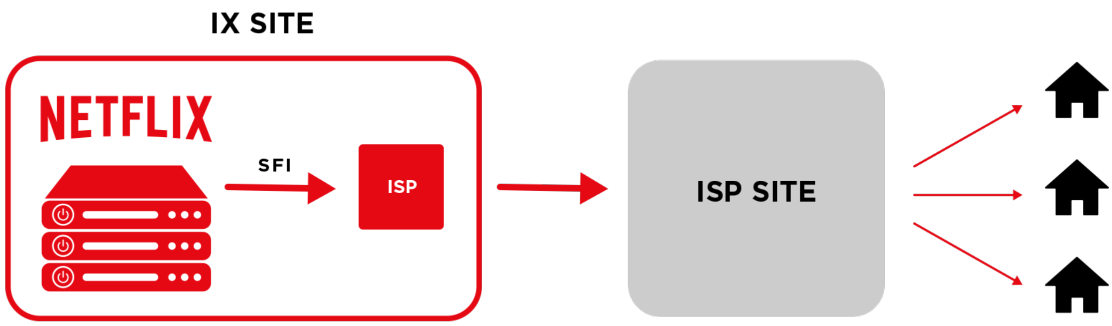

Figure 2.2: OCAs are installed within Internet Exchange Points (IXP), and interconnected to ISPs. [14]

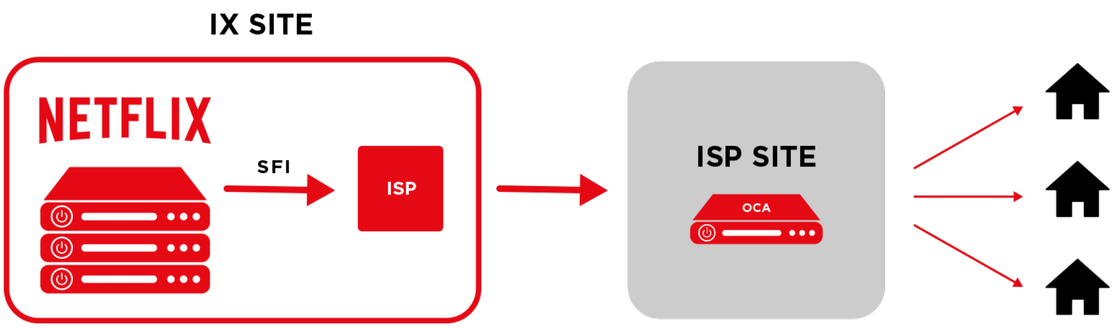

Figure 2.3: OCAs are directly deployed inside ISP networks. [14]

2.2 Attack Scenarios

Given the nature of modern Internet infrastructure, an adversary interested in eavesdropping a particular communication, only needs to compromise a node on the path the communication travels through. An *on-path* attacker could easily gain passive access to network and transport layers, and start capturing network traffic. This can include malicious or compromised Wi-Fi access points, routers, tapped network cables and ISPs.

Rather than just attacking physical devices, leaks of information could occur in network connections, in which an attacker physically close to the victim, could make use of a *Wi-Fi sniffer* to estimate traffic by capturing physical layer WLAN packets. Information may be encrypted by 802.11, and the sniffer may not take into account packet retransmissions at the session layer nor multiple TCP/IP flows on the same link, potentially causing noise on the observation. Despite so, Reed et Al. [15], have shown that it is still possible to estimate WAP-to-client throughput and use it to identify the content being streamed.

In addition to the above, *side-channel eavesdropping* can exploit information about the network structure, to saturate a link between the user and the server, and estimate fluctuations of congestion by sending probes remotely and observing queueing delays in routers [16].

We will now present the relevant phases and tools needed for an ISP to perform an attack, considering the ISP an *on-path* attacker.

2.3 Video Fingerprinting

In order to identify video stream traffic of a specific user, the ISP needs to build a database of video fingerprints to match against. To build such a database, the ISP needs to have access to a network interface, control its inbound bandwidth, (to get different levels of quality for each title), and capture incoming traffic passing through it.

In order to control the incoming bandwidth, the ISP could either limit it directly onto a generic L4 switch, or decide to connect any UNIX-like machine and throttle the throughput of its main ethernet interface. We will consider the scenario in which the ISP limits the bandwidth of the ethernet interface of the switch, (less noisy and more stable).

The value of the enforced bandwidth determines the quality (bitrate) of the content that will be captured. Assuming that each title has a unique bitrate ladder, the ISP should come up with an ad-hoc policy for every video to faithfully reconstruct the quality levels of it. While correct, a more viable approach to this problem, would be to just consider a range of bitrates capable of spanning the space of possible quality levels.

In our own version of the attack, presented in Chapter 3, we use the values in Table 2.1.

Bandwidth levels (Mbps)												
0.6	0.8	1.2	2	3.5	4.2	4.8	5.5	6.5	7.05	10	15	20

Table 2.1: Enforced bandwidth levels.

The ISP can now connect a UNIX-like machine to the switch's network interface with bandwidth limit *b*, and invoke `adudump` to infer the size of each *application data unit* ADU by processing TCP/IP packet header traces that generate *a-b-t* connection vectors [17].

2.3.1 The a-b-t *model*

The lifetime of a TCP endpoint that sends and receive data units is not only dictated by the time spent on these operations, but also by quiet times in which the TCP connection remains idle, waiting for upper layers to formulate new demands. Clearly, longer lifetimes may have a huge impact overall, due to the fact that resources needed to handle TCP state, remain reserved for a longer time. In contrast, ADUs that are sent within a short period of time, get aggregated by the TCP window mechanism, reducing the exchange of requests/responses between source and destination.

These ideas have been formalized into the *a-b-t model*, that describes TCP connections as sets of ADU exchanges and quiet times. The term a-b-t is descriptive of the building blocks of the model: *a-type* ADUs, sent from the connection initiator to the connection acceptor, *b-type* ADUs, sent from the responder back to the initiator, and quiet times *t*'s, during which, no data segments are exchanged.

The model has two different flavors depending on whether ADU interleaving is sequential or concurrent: the former is used for modeling connections for which only one ADU is sent from one endpoint to the other at a given point in time, (one endpoint will wait until the other has completed the transmission of the current ADU before sending a new one), while the latter is used for modeling connections in which both endpoints send and receive ADUs simultaneously.

As one can notice from a sample traffic trace from Netflix, the two endpoints behave in a sequential fashion, as shown by the `SEQ` flag in Listing 2.1.

```
ADU: 1565880580.152388 192.168.0.157.54924 <1 45.57.19.134.443 198837 SEQ 0.804088
ADU: 1565880581.321246 192.168.0.157.54924 <1 45.57.19.134.443 198760 SEQ 0.907568
ADU: 1565880582.688335 192.168.0.157.54924 <1 45.57.19.134.443 199407 SEQ 1.052862
ADU: 1565880591.912352 192.168.0.157.54974 <1 45.57.19.134.443 525046 SEQ 0.058709
ADU: 1565880592.168340 192.168.0.157.54968 <1 45.57.19.134.443 275568 SEQ 0.558654
ADU: 1565880592.168344 192.168.0.157.54974 <1 45.57.19.134.443 305115 SEQ 0.062028
ADU: 1565880592.172495 192.168.0.157.54982 <1 45.57.19.134.443 198735 SEQ 0.670224
```

```
ADU: 1565880592.697314 192.168.0.157.54974 <1 45.57.19.134.443 236474 SEQ 0.179115
ADU: 1565880592.820292 192.168.0.157.54982 <1 45.57.19.134.443 286547 SEQ 0.106913
ADU: 1565880592.820395 192.168.0.157.54968 <1 45.57.19.134.443 198530 SEQ 0.310506
```

Listing 2.1: Incoming traffic trace of Mulan, captured at 10Mbps (first 10 segments).

2.3.2 Inference

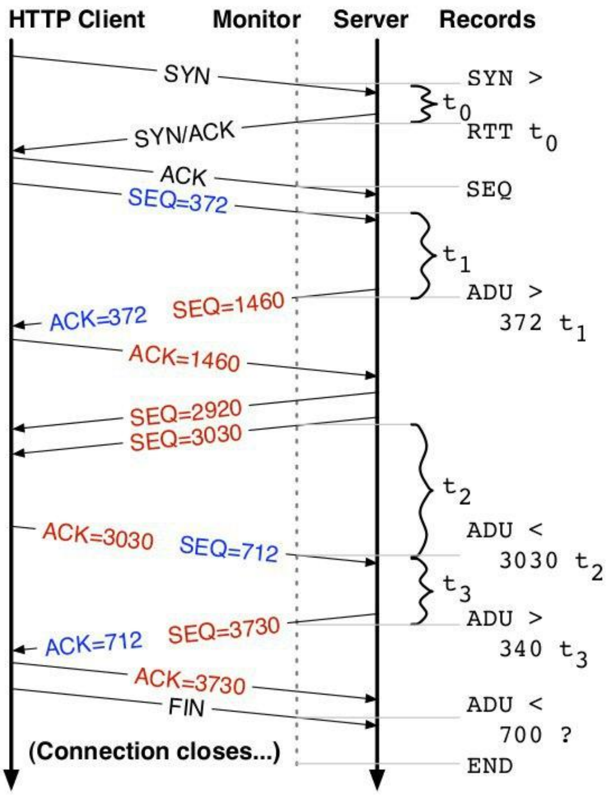

Figure 2.4: Detailed segment size inference in adudump.

Figure 2.4 shows an example of how Adudump's inference process works. The connection starts when the three-way handshake completes, (marked as a SEQ record). The monitor then records the time the first data segment is sent from the client. The server then acknowledges the previous request, and adudump infers the size of the completed ADU to be 372 bytes, generating a record with the ADU's size direction and think-time. Note how adudump generates no record until it infers that the ADU is complete, moreover the think times reported, are relative to the position of the monitor in the network, the farthest the monitor is from the server, the noisiest the measure of quiet times becomes. More on the effect of quiet times is described in Chapter 3.

2.3.3 Identifying Netflix traffic

In order to identify specific video traffic from a Netflix CDN, the ISP could either inspect the DNS response packets or the TLS handshake messages, and perform a DNS lookup for each video every time, or to maintain an updated list of IP addresses with mappings to the recorded traffic. Then all the ISP is left with, is a mixed trace of audio and video segments. Each Netflix audio segment represents about 16 seconds of playback, while every video segment corresponds to a 4 second playback. The ISP could now decide to either filter out audio segments by either thresholding on the size of ADUs, as done by Reed et Al (at 200 kilobytes), or to consider audio as part of the fingerprint of a title. The 200 kilobytes threshold represent a good heuristic when considering traces recorded at high bandwidth levels, in fact, audio segments sizes are around 198 kilobytes, whereas if considering the same title at a low level of bandwidth (less than 1 Mbps indicatively), one cannot distinguish audio and video segments by their size. For this reason, we consider an approach that does not filter out audio segments. In Chapter 3 we highlight the difference between the two methods.

2.3.4 Building a database of fingerprints

The ISP can now automate the streaming of Netflix videos, and build a database of such fingerprints. Here, the ISP is free to use a method of storage of choice, depending on the granularity of the features required to represent each title, and on the cardinality of the set of titles it wants to capture. In addition, as Netflix is constantly adding/removing titles from its library, the ISP is required to constantly keeping updated the database to guarantee that the attack can target newly added titles.

2.4 Capturing video traffic

In order to capture video traffic to match it against a database of collected fingerprints, it requires the ISP to be in possess of a generic L4 switch capable of mirroring traffic from a port to another one. Then any UNIX-like system that implements *libpcap* is suitable for capturing inbound traffic on the mirrored port, as presented below in Figure 2.5.

TODO: *modify this image*

In this scenario, the client is unaware of the fact that its traffic is being mirrored and recorded by the ISP, and there is no practical way for him to claim that. On the other hand, a user concerned with its privacy, may avoid such attack, as described in Section 2.4.3.

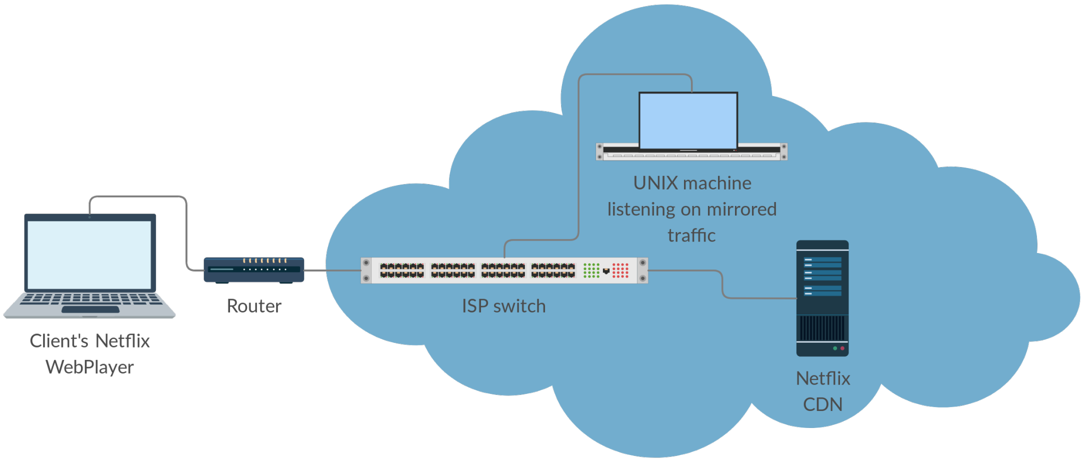

Figure 2.5: Traffic capture scenario.

2.4.1 *Data acquired by the ISP*

Captured traffic, as shown in Listing 2.1, apart from the size of each segment and quiet times, include timestamps of each ADU, and source and destination IP addresses. With this information, the ISP is able to reconstruct streaming habits of every targeted user (identifiable by the source IP address), by looking at what time he/she does usually connect to the platform, identifying the content of the stream (matching to fingerprints), and by analyzing how much time the user spends watching content. Thus the ISP can aggregate user's collected data and build a database of user profiles to use it for its own will. Data about a particular user is no longer in the solely hands of Netflix, as one could argue it should be, but it now resides in some ISP database, potentially exploitable and profitable, with any prior consent.

2.4.2 *Matching Captured traffic to fingerprints*

The ISP can now identify the captured trace by comparing it to the saved fingerprints. Based on the type of data structure used to store them, there are several methods to search and retrieve the best candidate for a given trace. We will not articulate this process here, but we rather present our own solution later in Chapter 3.

2.4.3 *Possible countermeasures*

In order to safely stream Netflix videos without the ISP being able to identify the content of them, a user particularly concerned about its privacy, (e.g. public figures, politicians, government agency employees), may decide to use a VPN. When using a VPN, the user, still needs the ISP to connect to the internet, but instead of having the ISP communicating directly with the desired resource, the ISP now "talks" with the VPN server. It is responsibility of the VPN server in fact, to establish a communication with the webpage the user wants to access. The key point is that the client and the VPN server establish a secure connection (VPN tunnel). Thus, as long as the user's VPN of choice, encrypts data transfer, either with IPSec's **Encapsulated Security Payload** or, in case of a remote-access VPN, with point-to-point based protocols such as L2F, PPTP or LT2P, the user is guaranteed that its traffic cannot be intercepted by the ISP. This is feasible as long as both the VPN software provides an automatic kill switch to avoid dropouts, and the online resource does not restrict access to users connected via VPN.

Netflix does not allow the use of VPN software to prevent users to get access to other countries catalogue, although certain VPN clients are able to bypass this check [18].

3 Approach

After having shown the perspective of an ISP in the role of an attacker, we will now present our contribution and practical approach to build a database of fingerprints, capture and identify video traffic of an unaware user over a compromised network.

We have built a system capable of manipulating the incoming bandwidth of a network interface, able to obtain fingerprints for a limited number of Netflix titles at various bandwidth levels, and reconstructed each video's bitrate ladder. We evaluate such a system by feeding it several test-captures of videos present in our fingerprint's database at *unseen* bandwidth levels.

3.1 Attack Scenario

Our own version of the attack, conversely to the one depicted in Figure 2.5, does not model the ISP as the adversary, nor does involve the ISP at all. We consider the case in which the attacker is a malicious user i, with access to the same network an honest client is using to stream a Netflix title. Attacker i has:

- either installed a passive TAP device on a LAN
- or gained control of the main switch of a LAN
- or compromised an AP over a public WiFi network.

TODO: *Illustrate the possible scenarios for an attacker*

For each of the aforementioned scenarios, the steps required for the attacker i to identify video traffic of another user, are the same: the first phase consists of recording fingerprints of each Netflix title, process them, store them in a persisent (and convenient for search and retrieval) data structure, while the second phase consists of exploiting the compromise device in the network to capture user's traffic and identify the content of the stream by querying the database of fingerprint.

3.2 Video Fingerprinting

Let T be the set of Netflix titles for Switzerland, we refer to n as the cardinality $|T|$, which is roughly 3500; consider now the set R as the set of bandwidth levels shown in Table 2.1, we refer to i as the cardinality $|R|$.

Consider now the cartesian product $T \times R$:

$$T \times R = \{(t_1, r_1), (t_1, r_2) \ldots (t_n, r_i)\}$$

and its cardinality:

$$|T \times R| = |T| \cdot |R| = n \cdot i \approx 45500$$

Due to time restrictions, we have decided to fix the size of the set T of titles to 100, in order to give a proof of concept on the feasibility of such an attack. According [3], we have also decided to bound the time of each video capture to 4 minutes of playback, as it has been shown to be sufficient in order to uniquely identify a video over more than 40000 titles.

3.2.1 Implementation overview

We have implemented a set of Python scripts to be able to:

1. Crawl the swiss Netflix catalogue to obtain a list of video IDs.
2. Manipulate the incoming bandwidth of an ethernet network interface.
3. Invoke `tcpdump` listening on the same network interface.
4. Instrument the browser to:
    a) Navigate to a specific title URL (identified by the title ID).
    b) Control the Netflix video player by injecting JavaScript code.
    c) Capture HAR metadata via a proxy.

Note that contrary to [3], in this phase we do not make use of adudump, instead, to record traffic, we use tcpdump [11]. The main reason behind this choice, is the fact that adudump has been conceived to reconstruct data segments sizes in an online fashion. By doing so, the time spent by adudump processing each segment, creates an overhead that in turn, results in noisy measurements. Adudump can work in an offline-fashion, simply by passing as input a `pcap` [11] file, that gets generated when invoking tcpdump. We have tested and analyzed this behaviour and visual evidence is presented in **??**.

TODO: *add plots of adudump's behaviour when invoked online vs on a .pcap file*

3.2.2 Crawler

The script responsible of crawling the Netflix catalogue is `crawler.py`. We use the `scrapy` Python library [19] to get a list of Netflix titles divided by genre. Note that, for the sake of simplicity, we have decided to work only with movies, as for TV series, we would have need to add checks due to the autoplay function of subsequent episodes in the viewing phase.

The resulting output of the script is a CSV file with the following structure:

```
ID         GENRE              TITLE

70115629, Family Animation, Despicable Me
70264803, Family Animation, Despicable Me 2
80096067, Family Animation, Ice Age: Collision Course
70220028, Family Animation, Hotel Transylvania
80121840, Family Animation, The Emoji Movie
70021636, Family Animation, Madagascar
70216224, Family Animation, Madagascar 3: Europe's Most Wanted
70213513, Family Animation, Brave
14607635, Family Animation, Mulan
```

Listing 3.1: Sample of crawled movies

Due to the fact that a movie can be labeled in more than one category, the resulting CSV have been filtered to include just one occurrence of each title.

```
cat netflix_titles/titles.csv | cut -d , -f1 | sort | uniq
```

Listing 3.2: Command to filter out unique IDs

3.2.3 Bandwidth Manipulation

In order to be able to manually control the bandwidth of the ethernet interface, we use `tcconfig` [20], a Python wrapper for the `tc` [21] Unix utility to configure traffic control in the kernel.

The script that throttles the bandwidth is `bandwidth_manipulator.py`, that invokes the command:

```
tcset --device <network_interface> --direction incoming --rate
    <bandwidth_rate>
```

Listing 3.3: Enforce a bandwidth rate on the specified interface

TODO: *tcconfig, plots of real bandwidth levels*

3.2.4 Tcpdump

The script that records the capture traffic is `capture.py`, it calls tcpdump as below:

```
tcpdump -i <network_interface> net 45 -w <output_file>
```

Listing 3.4: Listens for TCP/IP traffic on the specified interface

Note the usage of argument `net 45`. As Netflix OCAs IP addresses are of the form `45.XXX.XXX.XXX`, to simplify the process of identifying traffic from Netflix, we just use this regular expression for convenience. Steps required to carry out a more general way to achieve this have been described in Section 2.3.3.

3.2.5 Automated streaming with Selenium

In order to instrument the browser to automatically stream each title, we have developed a script that uses the `Selenium` library [?]. Selenium provides a `WebDriver` interface capable of controlling various browsers such as Chrome, Opera, Safari, Firefox and others. One must use the appropriate version of the WebDriver, according to the browser of choice and its version. For convenience of use, and support, we have chosen to work with GoogleChrome and its ChromeDriver interface.

In order to asses the quality of adudump's inference of video segment sizes, we compare the recorded traffic with HAR [?] metadata. The format of each HAR file is a json object containing content and session data acquired during the playback of the video. For this task, we use `Browsermob` [?], a webproxy built to work with Selenium. A more detailed explanation of the comparison between recorded traffic traces and HAR metadata is presented in **??**.

Furthermore, to speedup the capture of each video, we have installed an extension that is able to control HTML5 video speed playback.

We have implemented a class `netflix_browser.py` capable of:

1. Log in the user to its Netflix account
2. Navigate to a video url given a video ID
3. Start the proxy to record HAR metadata
4. Seek the video to 240 seconds (4 minutes) (avoiding entry credits)
5. Wait until the buffer of the video reaches 8 minutes (start at 240 seconds, ends at 480).
6. Stop the proxy and save the resulting HAR file

***TODO:** draw a diagram with the steps required to obtain the fingerprints* ***TODO:** insert listing specifying format of saved captures*

3.3 Post-processing

The script that handles post-processing of `.pcap` output files is `post_process.sh`. It is composed by three main functions.

The first one invokes adudump on the traces recorded with tcpdump, after that, it extracts the DNSs contained in the HAR metadata and performs a DNS lookup using Unix's `host` command. This step is required to get the complete list of IP addresses of the OCAs that served the video, so that we can filter all non-video traffic that tcpdump has captured.

The second function is responsible for generating the bitrate ladders of each title, and to save all fingerprints in a text file. Given a trace at a particular bandwidth level b, we compute the average bitrate as:

$$AVG_{bitrate} = 2 \cdot \frac{\sum_{i=0}^{n} adu_i}{n}$$

We average over all ADUs (sizes are in bytes), divide by 4 seconds (approximate lenght of a video segment), and multiply by 8 to get a measure in *bits/s*.

TODO: *clarify thresholding*

3.4 Capturing video traffic

TODO: *Explain how we capture and filter video traffic*

3.5 Identification

TODO: *Describe the identification process and the choice of data structures*

4

Evaluation and Analysis

5

Conclusions

Bibliography

[1] Wikipedia. Alliance for open media, 2019. URL https://en.wikipedia.org/wiki/Alliance_for_Open_Media.

[2] Netflix TechBlog. Per-title encode optimization, 2015. URL https://medium.com/netflix-techblog/per-title-encode-optimization-7e99442b62a2.

[3] Andrew Reed and Michael Kranch. Identifying https-protected netflix videos in real-time. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 361–368, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4523-1. doi: 10.1145/3029806.3029821. URL http://doi.acm.org/10.1145/3029806.3029821.

[4] Jeff Terrell, Kevin Jeffay, F. Donelson Smith, Jim Gogan, and Joni Keller. Passive, streaming inference of tcp connection structure for network server management. IEEE International Traffic Monitoring and Analysis Workshop, 2009.

[5] Wikipedia. Network tap, 2019. URL https://en.wikipedia.org/wiki/Network_tap.

[6] Business Insider Intelligence. Video will account for an overwhelming majority of internet traffic by 2021, 2019. URL https://www.businessinsider.com/heres-how-much-ip-traffic-will-be-video-by-2021-2017-6?r=US&IR=T.

[7] Wikipedia. Peak signal-to-noise ratio, 2019. URL https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.

[8] The Economist. The world's most valuable resource is no longer oil, but data, 2018. URL https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data.

[9] Wikipedia. Net neutrality in the united states, 2019. URL https://en.wikipedia.org/wiki/Net_neutrality_in_the_United_States.

[10] @xxdesmus. Kanopy.com leaking api and website access logs, 2019. URL https://rainbowtabl.es/2019/03/21/kanopy-data-leak/.

[11] tcpdump. URL https://www.tcpdump.org/.

[12] Scott Saponas, Jonathan Lester, Carl Hartung, Sameer Agarwal, and Tadayoshi Kohno. Devices that tell on you: Privacy trends in consumer ubiquitous computing. In *Proceedings of the 16th USENIX Security Symposium*. USENIX, August 2007. URL https://www.microsoft.com/en-us/research/publication/devices-tell-privacy-trends-consumer-ubiquitous-computing/.

[13] Florian Moser. Identifying encrypted online video streams using bitrate profiles, 2018. URL https://github.com/famoser/network-experiments.

[14] Netflix Open Connect. URL https://openconnect.netflix.com/Open-Connect-Overview.pdf.

[15] A. Reed and B. Klimkowski. Leaky streams: Identifying variable bitrate dash videos streamed over encrypted 802.11n connections. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 1107–1112, Jan 2016. doi: 10.1109/CCNC.2016.7444944.

[16] Xun Gong, Nikita Borisov, Negar Kiyavash, and Nabil Schear. Website detection using remote traffic analysis. In Simone Fischer-Hübner and Matthew Wright, editors, *Privacy Enhancing Technologies*, pages 58–78, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-31680-7.

[17] F Donelson Smith, Felix Hernández-Campos, Kevin Jeffay, and David Ott. What tcp/ip protocol headers can tell us about the web. volume 29, pages 245–256, 06 2001. doi: 10.1145/378420.378789.

[18] Nord VPN. URL https://nordvpn.com.

[19] Scrapy. URL https://docs.scrapy.org/en/latest/.

[20] tcconfig. URL https://pypi.org/project/tcconfig/.

[21] tc. URL http://man7.org/linux/man-pages/man8/tc.8.html.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

De-anonymizing encrypted video streams

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Stefano	Peverelli

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 07.09.2019	Stefano Peverelli

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.