ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bachelor's Thesis Nr. 214b

Systems Group, Department of Computer Science, ETH Zurich

Identifying encrypted online video streams using bitrate profiles

by
Florian Moser

Supervised by
Prof. Dr. Ankit Singla and Melissa Licciardello

February 2018 – July 2018

inf | Informatik
Computer Science

Abstract

In this work, we investigate how easily video traffic can be attributed to a video, therefore posting a threat to the privacy of users of video streaming services. Specifically, we investigate the effect of the different bitrate ladders per video introduced by the way netflix employs per-title optimization [1]. We replicate the attacks as shown by Reed [2] [3], and then show that the bitrate ladder makes the video traffic even easier to identify. Furthermore, we can also show that an identification of a video is possible using solely the bitrate ladder.

Contents

1 Introduction **5**
- 1.1 The privacy issue . . . 5
- 1.2 Motivation . . . 6
- 1.3 Contributions . . . 6
  - 1.3.1 Modified Cadmium player . . . 7
  - 1.3.2 Netflix crawler . . . 7
  - 1.3.3 HAR extractor . . . 7
  - 1.3.4 Attacker simulation . . . 7

2 Background **8**
- 2.1 Netflix . . . 8
- 2.2 Video compression . . . 8
- 2.3 DASH . . . 9
- 2.4 Variable bitrate encoding . . . 10
- 2.5 Previous work . . . 10

3 Methodology **12**
- 3.1 Attacker model . . . 12
  - 3.1.1 Attacker model used in experiments . . . 12
- 3.2 Approach . . . 13

4 Implementation **14**
- 4.1 Technologies used . . . 14
- 4.2 Our scripts . . . 15
  - 4.2.1 Python libraries . . . 15
  - 4.2.2 Netflix extension . . . 16

5 Results **17**
- 5.1 Creating a proof of concept . . . 17
- 5.2 Capture traffic . . . 19
- 5.3 Analysis . . . 20
  - 5.3.1 Simulating an attacker . . . 21
  - 5.3.2 Other observed behaviours . . . 22

6 Results **42**

7 Discussion **43**
- 7.1 How to enhance video streaming privacy . . . 43
- 7.2 Further work . . . 43
  - 7.2.1 Collect more evidence . . . 44
  - 7.2.2 Modify DASH . . . 44

A Algorithms **45**
A.1 Package existence algorithm 45

1 Introduction

70% of the worldwide consumer internet traffic is generated by video streaming [4]. In this work, we try to give more answers how easy it is to attribute traffic to a certain video. First we will introduce privacy, and then further motivate our research. We then present shortly our contributions.

1.1 The privacy issue

Privacy is defined as "the quality or state of being apart from company or observation" at Merriam-Webster [5]. While privacy should not need any additional motivation (it is seen as such fundamental to society that it has its own article in the human rights convention [6]), it seems as it is not seen as important as it should be by the stakeholders, which is the only way its neglectance can be explained.

stream providers While some constrains of video streaming can not be influenced easily by the streaming providers (like video compression and end device constrains), they control the streaming technology used client side. With some small changes to these algorithms, identification of videos could be rendered much more difficult, with sacrificing little to no bandwidth or user experience. Specifically, the naive approach described above which simply remembers exact video segment sizes could be countered by applying random padding to the video segments.

users A study in from 2018 in Germany showed that only 39% the questioned users would pay a service if in turn it would not sell their private information [7]. An example on how this view expresses itself may be the Cambridge Analytica incident, where personal information coming from facebook had been sold [8]. While it did provoke a public outslash [9], a large-scale exodus of users or customers did not occur [10].

future development While the Cambridge Analytica scandal might not hurt facebook financially, it had to take a lot of bad press and the firm in question even had to go out of business. Furthermore, it made worldwide news how badly protected privacy might had been used to influence a democratic election. One can hope that both users and firms start to realize how important privacy is, and that it will play a larger role in future decisions. New regulatory approaches focusing on data protection could be helpful such as the GDPR [11] which helps the user take informed decisions, and forces firms to declare how personal data is used.

1.2 Motivation

We want to confirm or discard our intuition that one can determine the video being watched solely by observing the average bandwidth used by a user at a certain available bandwidth. Our intuition follows from the fact that Netflix uses per title optimization rather aggressively [1].

consequences of identifiable traffic If video traffic can be easily attributed to the respective video, mass-surveillance of users of video streaming services is feasible. Furthermore, ISPs could discriminate users depending on their video streaming habits. To prevent public outslashes due to a lack of privacy or ISP discrimination, large streaming providers would need to develop new or adapt existing stream technology in the interests of their users and themselves.

per-title optimizations as a new danger But another trend seems to emerge that is dangerous for privacy: More aggressive ways of optimizing per title might expose even more information about the movie being watched than before. Per-title optimizations optimize the tradeoff between video quality and file size for each video separately. While this development can be motivated from a usability standpoint, if this measure is employed aggressively and effectively, identification of a video could be possible with even less information about the traffic than ever before.

encryption does not solve the problem While today most video traffic is encrypted [12], identification of such has been shown to be feasible. Even a naive approach, which simply remembers the TCP steam sizes, is powerful enough to allow an identification as shown by Reed [3]. The follows from the fact that video segment sizes are very different, due to a combination of:

- **video encoding properties** such as the different archived compression factors depending on the video content
- **constraints of end devices** such as a fixed resolution set
- **used streaming algorithms** which focus solely on delivery

relevance of our approach If the video can be identified solely by average bandwidth used, even a privacy enhanced streaming algorithm which does not leak segment size information would then not prevent identification anymore: No matter how it times or sets up the requests, it always has to transfer the full video file in a bounded time window to archive its target to play the video.

1.3 Contributions

We shortly present our contributions which may be used for similar work or to replicate our results.

1.3.1 Modified Cadmium player

Cadmium is the video player used by Netflix. It is not open source, but its code is distributed to the users as a minified [13] JavaScript file [14]. The minification process makes large parts of the file unreadable and hard to understand. However, we were able to modify it as needed by our crawler.

feature list We exposed the following helpful methods and properties from the player which we used for our crawling and verification scripts.

- enable/disable specific video quality profiles
- list available bitrates
- enforce a specific bitrate
- set the video buffer size
- detect if the video started playing
- check if the video is actively playing
- jump to a specific point in time of the video

keyboard shortcuts Additionally, we were able to find the following useful keyboard shortcuts supported by the player:

- Ctrl + ⇧ + Alt + d or Ctrl + ⇧ + Alt + q to list current properties of audio/video, buffer size and other state information
- Ctrl + ⇧ + Alt + s for a menu allowing us to manually set bitrates of audio/video and the download server
- Ctrl + ⇧ + Alt + l which opens the log console containing event information from the player

1.3.2 Netflix crawler

Using Selenium [15], BrowserMob proxy [16] and a Netflix extension [17], we created a crawler written in Python which captures HTTP requests/responses and saves their header information in HTTP Archive (HAR) [18] files [19]. It requires a valid username/password pair of a Netflix account, and the ids of the to be captured videos of Netflix.

1.3.3 HAR extractor

We provide scripts which parse the HAR files, and put interesting data into an sqlite database [20], prepared for performant retrieval [21] [22].

1.3.4 Attacker simulation

We present a script which performantly simulates an attacker, relieving the researcher from time intensive and imprecise experimenting. The researcher first chooses, how much knowledge the attacker has, which is compromised of captured video traffic. Then the scripts plots how many movies match to the knowledge the attacker was provided with [23]. We will go into this process in great depth in section 5.3.

2 Background

We will introduce some background which is necessary to understand the motivation behind our work, and the intuition behind our adopted approaches.

2.1 Netflix

Netflix [24] is a popular video streaming service from California, USA. In March 2018, they had approximately 131 millions of users [25]. We chose Netflix for our work because it employs aggressive per-title optimization [1].

2.2 Video compression

A video is just a sequence of pictures, but would be very inefficient to save it as such. There are multiple ways which allow to reduce the video size substantially, besides simply reducing the resolution. We will shortly describe two approaches, the lossy encoding and a form of compressing a sequence of images.

lossy encoding Most video encodings are lossy, which means a single frame is not saved exactly, but only approximately. If applied correctly, this allows to discard a large number of bits while to the human eye the compressed image still looks very similar to the original one.

sequence compression Besides applying compression to a single image, one can also compress sequences. While a few images of the sequence are chosen to be fully preserved, most images are however only preserved indirectly: Only the difference to one of the fully preserved image is preserved, therefore saving a lot of space if the two images in question are similar.

H.264 These two described approaches and many more are used by H.264 [26]. All these lossy compression techniques can be applied more or less aggressively (independent of the resolution). H.264 is the encoding of choice by DASH [27] and therefore used by Netflix [1].

filesize of videos depends on its content This implies that, depending on the video content and the chosen parameters at encoding time, the filesize differs heavily for the same video length, between different videos and even if both segments are part of the same video.

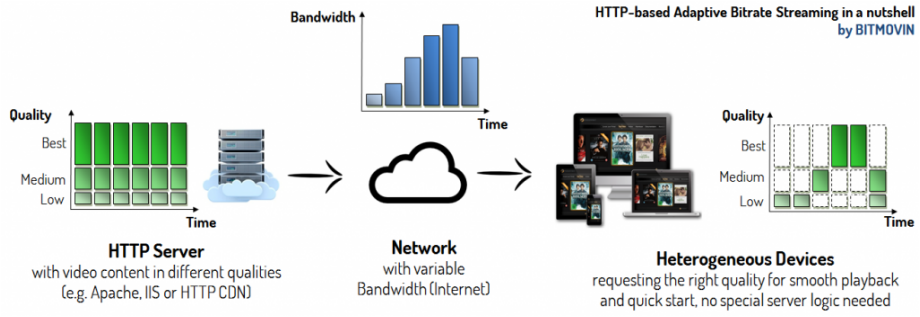

Figure 1: Adaptive bitrate streaming: The client chooses the right quality for smooth playback.

2.3 DASH

DASH (Dynamic Adaptive Streaming over HTTP) is an adaptive streaming standard [27]. It takes care of the transmittal over the network and decoding of the video material while other parts of the video are already playing. A short overview is given in figure 1 [28].

adaptive video playback It can deal with different streams of a single video (which offer this video at different qualities at respectively different file sizes), delivering the user the most appropriate version as given by the network conditions and other factors. This happens in the best case even without interrupting the playback.

manifest file A DASH player needs a manifest file which describes how to retrieve the different available streams per video [27] [29].

video segments The video is retrieved in the form of segments, which are played one after the other. The DASH algorithm might vary the stream for each segment, continuously adapting to changing conditions.

implementation There is a client reference implementation freely available [30], but DASH leaves a lot of details open for developers.

Netflix observations We observed that the Cadmium player starts with a low quality stream for the first few segments to be able to start the video fast, and if the network conditions are good, then chooses higher quality streams for the further segments. Further, we observed that each segment is retrieved with its own HTTP request, and is 5 seconds long.

2.4 Variable bitrate encoding

One has two ways to decrease the filesize of a specific video: Either employ more aggressive compression (possibly leading to visible effects due to the lossy compression) or reduce the resolution (possibly leading to pixilation).

adapt compression to the specific video We have already seen that, through the way the video compression works, some videos can inherently be compressed better than others. Small changing landscape movements can use aggressive compression without the user noticing much. They can be heavily compressed, and therefore, even if they are encoded at high resolutions, the resulting file is of an acceptable size. If other videos were compressed that much, for example fast changing fighting scenes, one would however see noticeable defects (called "artifacts") in the rendered image. These fast paced movies, with heavily changing images, may have to be encoded with a less aggressive compression, resulting in a bigger file size. To counter that effect, which means to reach the same file size as the landscape scene, the resolution may be reduced appropriately.

Netflix's approach Netflix described their per-title optimization approach as follows: First, they fixed the available resolutions to some sensible defaults. Then, for each title and resolution, they aim to find the most appropriate respective bitrate such that the compression produces acceptable results while keeping the bitrate as low as possible. These video-specific available bitrates are called its bitrate ladder [1]. While determining this bitrate ladder per video is computationally expensive (as different variants have to be created and analyzed), it potentially pays off if the video is watched often.

2.5 Previous work

Reed and his colleagues have presented approaches to identify Netflix videos based on observed traffic.

wireless identification In Reed-Klimkowski (2016) [2], they present a crawler to create fingerprints of Netflix videos (around 50), and then showed than one can accurately identify wireless traffic resulting from watching one of these videos.

large video database In Kranch-Reed (2017) [3], they present a crawler to create fingerprints of Netflix videos, a huge database of captured fingerprints, and an algorithm to identify what video is currently watched by the target. They demonstrate that most videos even in their huge collection have unique fingerprints.

video segments used Their attacks look at the segment sizes of the videos, which are most of the time unique due to the properties of the compression

algorithms and the way Netflix applies them.

placed on the transport layer They use special tools which can attribute TCP packets to a HTTP request [31] as they measure on the transport layer. As we look directly at the HTTP request, we do not need to replicate this part of their work.

encrypted traffic used As both of their described attacks work on the encrypted packages, it seems as the HTTP/TCP overhead is neither big nor random enough to make the identification process infeasible.

3 Methodology

We present how we structured our work to archive our result.

3.1 Attacker model

We assume an attacker which can observe the network and influence the bandwidth, but not break cryptography. It is similar to the Dolev-Yao attacker model [32] as we listen to the network and limit the bandwidth to specific levels (which can be implemented by continuoulsy sending a specific amount of dummy traffic). We argue that our attacker is even stronger, as he can not modify existing messages, and will not try to deceive the user with forged messages.

a very realistic model Nearly any participant of a network fits this attacker model: Global adversaries such as ISPs, government or non-government agencies and local adversaries. As long as the package sizes and their timings of the victim are physically measurable by the attacker (as with wireless or LAN cables), and he can limit the bandwidth somehow (with a switch or dummy traffic), our attack is feasible.

3.1.1 Attacker model used in experiments

To make our experiments easier to implement and execute, we use a stronger attacker which can perform a man in the middle attack. We use this only to be able to look at the HTTP request URL (to identify video traffic, something otherwise possible using IP filters and similar) and the HTTP response body size (to avoid TCP package calculations as done by [31]). This allows us also to reuse large parts of the scripts for both the capture and the attack. Through the work done by Reed [2] [3], we are confident that any results archived with decrypted packets (which do only use the information as described above) are generalizable to encrypted packets.

packages Our attacker therefore deals with packages: HTTP response body sizes, corresponding to a single video/audio segment.

aggregation Our attacker can also aggregate packages: An aggregation of 2 means each pair of directly following packages is combined to a single bigger package, an aggregation of 3 means a triplet is combined, etc. This is done continuously, so 2 packages result if 4 packages are aggregated at level 2. We use the aggregation levels as an approximation of only using the bandwidth (dividing the aggregated package by the time they cover results in the used bandwidth).

n-packages We refer to aggregated packages as n-packages, for n determining how many packages it consists of.

3.2 Approach

As a first part of the work we replicate the results of Reed [3] with our strong attacker model. This gives us confidence in the results of Reed and in our captured data. Then, we perform an analysis on the crawled video data to archive our results.

4 Implementation

Multiple approaches were used during the creation of our work. However, we will only present the most promising ones here. A detailed and complete documentation of our process is provided in the form of our GIT repository [33].

4.1 Technologies used

We present the used technologies and libraries which could be helpful to continue research in this area.

tc `tc` [34] is a tool for linux which allows to limit the bandwidth to specific levels using different configurable approaches (drop a certain percentage of packages, drop all which do not fit inside the given window, ...). We used a Python wrapper (first `tcconfig` [35], then our own) which simplified the development of an automatic crawling / attack script.

HAR files HAR files [18] document a HTTP request, including but not limited to request time, request specification and response information like received body size. We used the url to attribute a HTTP request to a specific video, and the response body size to get the video segment size.

Browsermob Browsermob [16] is a tool which creates a local proxy and then captures HTTP traffic passed over that proxy. In our experiments, we configured our browser to use this proxy, and could then analyze all HTTP traffic sent and received while surfing as we normally would. Browsermob needs to employ a man in the middle attack [36] with its own (self-signed) certificate to be able to capture the HTTP traffic. Browsermob is controlled by sending commands to a specific port at `localhost`.

Cadmium player The Cadmium player is used by Netflix to implement DASH and control the video playback. We used a modified version to be able to capture specific bitrates, get infos about the currently running video, and speed up otherwise lengthy captures by multiple factors.

Chrome Chrome [37] is a web browser developed by Google, and supported by Netflix. We used Chrome because its control can be automated and one can avoid the certificate errors introduced by the Browsermob proxy by running Chrome with the `--ignore-certificate-errors` switch.

Chrome extensions Chrome allows the user to install extensions which can execute arbitrary JavaScript code and block content. For our approach, we used an existing Netflix extension [17] with our modified Cadmium version. The extension essentially blocks the Cadmium JavaScript request of Netflix and replaces it with its own Cadmium JavaScript file. On every Netflix page we visit

with the attached extension, we can then use the functionality provided by our modified Cadmium.

Selenium Selenium [15] is able to control browsers. It has access to basic Chrome features (like navigation or lifetime events), and to its JavaScript runtime, enabling us a two-way communication between our modified Cadmium.

SQlite SQlite [20] is a database which needs no setup or running services and saves the entire state to a single file. It is therefore trivial to setup and to share, while still providing most advantages of a database. The SQL queries allowed a more complex, faster and easier analysis of the data than it would have been possible by using only Python. By creating additional indexes on queried columns (namely on the size column), and splitting the content into multiple tables (replacing certain WHERE clause conditions), the performance could be further increased.

4.2 Our scripts

We implemented some own Python libraries which we used in our concrete experimentation scripts, fit to be reused in other projects.

4.2.1 Python libraries

Our capture and evaluation scripts are written in Python. While the scripts differ, most of them use some common classes we have written as part of this work. We will present these here as they can be reused in future experiments.

bandwith_manipulator.py [38] Using `tc`, the available bandwidth on the device running the script can be controlled.

browser_proxy.py [39] Using the Browsermob proxy, we implemented a wrapper which abstracted away the details of the Browsermob implementation. It can return or directly save the active capture and start a new one.

config.py [40] We created a collection of random movies and some other configurations to coordinate the different scripts.

har_analyzer.py [41] To better extract information from the HAR files, this script exposes a `HarEntry` per request with its corresponding response.

mouse.py [42] Able to click and move the mouse, it allows to continue to automate tasks when JavaScript is suddenly of no help anymore.

netflix_browser.py [43] Using Selenium, this class controls the Chrome browser. It can navigate to videos, ensure they are played correctly, speed up their capture and enforce specific bitrates. Concretely, it creates a Chrome instance, configures it to use the Browsermob proxy, attaches the Netflix browser exten-

sion, and then navigates to our target video. If the user is not yet logged in, it attempts to authenticate using provided credentials. Cookies of successful login attempts are cached locally, and reapplied in the next session to avoid having to login too often.

4.2.2 Netflix extension

We build a Netflix extension [14] which exposes otherwise not accessible features of the Cadmium browser to the public scope. We have commented all interesting parts of the file with the `//#` line prefix.

5 Results

We present the experiments we executed and results we archived in a primarily logical, and mostly chronological order. We shortly describe drawbacks of selected approaches we did not pursue further. For each experiment, for which we present data or plots, we prepared a folder in our repository with the exact state at the time we conducted these experiments, and with ready-to-use scripts (if configuration we can not share is provided, such as Netflix credentials).

5.1 Creating a proof of concept

With Selenium controlling Chrome, the Netflix extension and `tc`, we did our first experiments.

measure video size In some given HAR file, we differentiated the HTTP video/audio request from the other requests (like CSS and js), by looking at their request link. Removing subdomains and `GET` parameters, a video/audio request link is of the form `https://nflxvideo.net/range/1200-6237` (as intended by DASH [27], the video file is stored as a continuous block on the server, and the client side algorithm requests the bytes it needs). As the audio segments were also part of this single binary we needed to separate it from the video. Motivated by plots such as figure 2, we filtered audio segments out using the formula $0.185 * (1024 * 1024) < package\_size < 0.195 * (1024 * 1024)$: 0.185 and 0.195 to include generously all audio packages, and because the plot is in megabytes, we multiply those two end points with $(1024 * 1024)$. The filtered data plotted then looks like as presented in figure 9.

confirmation that available bandwidth impacts the measured video size We then, first manually using `tc`, then later using `tcconfig`, restricted the bandwidth available on the test machine, and observed the behaviour of the video playback. We noticed that the Netflix player handled decreasing bandwidths badly (crashing completely or failing to fetch new packages), therefore all further experiments which modified the bandwidth started at the lowest one to be measured, and then only ever increased it.

first Cadmium modification tested We then tested with a simple crawler [44] whether the modifications we did on the Cadmium JavaScript had effects on the video size. This indeed was the case, but the size was not always the same for the same Cadmium quality setting as it can be seen in Table 1.

movie id	quality	traffic
80018499	1	618.8MiB
80018499	1	510.2MiB
80018499	2	1.1GiB
80018499	3	1.4GiB
80018499	3	2.5GiB
80018499	4	4.9GiB
80111448	1	196.9MiB
80111448	1	231.6MiB
80111448	1	196.3MiB
80111448	1	229.7MiB
80111448	1	255.3MiB
80111448	1	32.6MiB
80111448	1	194.9MiB
80111448	2	141.0MiB
80111448	2	140.2MiB
80111448	2	24.8MiB
80111450	1	157.6MiB
80111450	1	29.6MiB
80111450	1	180.1MiB
80111450	1	13.5MiB
80111450	1	157.6MiB
80111450	1	158.3MiB
80111450	2	105.4MiB
80111450	2	105.4MiB
80111451	1	198.7MiB
80111451	1	199.4MiB
80111451	1	198.7MiB
80111451	2	192.8MiB
80111452	1	171.4MiB
80111452	1	171.4MiB
80111452	1	171.4MiB
80111452	2	125.0MiB
80111452	2	302.2MiB
80111452	3	302.2MiB
80111452	4	302.2MiB

Table 1: measured traffic at different quality levels

We then found out that we only enabled/disabled quality profiles, profiles such as low quality, high quality, or similar. For example, the unmodified Cadmium player only enables the highest quality profile if it detects either Chromium OS[1] or Edge[2]. But each profile still contained several bitrates. This meant, that even with only a single active quality profile, the player did still adapt to different bitrates depending on its network conditions. This was a side effect which would have made it difficult to perform an accurate analysis over the video segment sizes, which is why we looked again at an approach using `tc` .

bandwidth enforcement with tc To be able to run longer captures [45] we added capabilities to our setup to set the allowed bandwidth with `tc` without user input. The results of the following experiments were more convincing, but still had a lot of noise. The following plots visualize measured video traffic on the y axis against the enforced bandwidth by tc at the x axis, for a single video in figures 3 and 4 and then a graph which accumulates over all measured videos in figure 5.
Possible reasons why the the measurements were inaccurate include that the used testing network was not as reliable as expected, and other irregularities in the streaming algorithm when adapting to new network conditions. We therefore shifted our attention back to modify the Cadmium player to fully suit our needs.

[1] A operation system developed by Google. `https://www.chromium.org/chromium-os`
[2] A browser developed by Microsoft `https://www.microsoft.com/en-us/windows/microsoft-edge`
[4] this is a testvideo probably intended to be used by Netflix engineers
[6] Captain America: Civil War
[8] the specific videos can be looked up by appending the respective number to the url `https://www.netflix.com/watch/`

successful Cadmium modifications We managed to modify Cadmium to enable all quality profiles on our device, and then to enforce a specific bitrate if needed: We found a hidden menu which provided a way to select the bitrate, and used its mechanisms to automate the selection process. After that milestone, we were able to capture video segment sizes of a specific bitrate, respectively capture video segment sizes of all bitrates of a video.

5.2 Capture traffic

We created a crawler which can collect video segment sizes of all bitrates of a video, either for the full video length or within a specified time range [46]. The movies were randomly chosen by hand using the recommended video on the landing page of Netflix.

setup The capture script first starts the browser proxy, then the Netflix browser and lastly advises the user to move his mouse to the center of the browser window. Afterwards everything works automatically: For each video, it navigates to the url and ensures the video plays. Then it gathers the possible bitrates for said video, and for each of the bitrates plays the specified part of the video while capturing the HTTP requests and saving them to disk in the HAR format.

small but long capture For an initial analysis, and to replicate the results archived by Reed [3], we run the crawler on a small set of movies (6) for the full video length each. We used the capabilities of our Cadmium player to shorten the time needed for the capture: Repeatedly after a configured time has passed, the player would advance the video position by some seconds, thereby reducing playback time while still ensuring that the player has load all video segments. This helped us to reduce the capturing time by a factor four, taking the acceptable risk that some segments were missed if the pointer was moved too fast.

big but short capture Our biggest capture, to be used in the detailed analysis 5.3, run on a total of 45 movies. It captured 5 minutes of each video (at each bitrate), enough to simulate an attacker. It did not speed up the capture, as we wanted to ensure that all segments of the 5 minutes are captured.

database creation After the capture script finished, we used another script to transform the important information from the HAR files to a database. We used different schemas and indexes depending on the later usage, but what they all had in common was a table for the captures (declaring captured movie and bitrate) and at least one other table with the packages (minimally described by its size and a pointer to the capture it is a part of).

listening We created a script which tries to find out the movie currently playing in our browser. Once the Selenium powered browser is running, the tester can put in any movie which is also contained in the database, and after a

short time the script is able to output the correct playing movie. This is a variant of the attack as described by Reed [3], which we successfully recreated and therefore confirmed. Without executing systematic tests of this procedure we have observed that after as little as 20 seconds the correct matches were determined.

5.3 Analysis

We performed analysis on the big data set to be able to evaluate possible views of an attacker [47]. First we will present some overall properties of the dataset we have collected. Secondly, we will show our evidence that using n-packages from different bitrates makes the identification of videos easier than using only n-packages from the same bitrate. Thirdly, we show that an identification using only the different bitrates (and hence the bandwidth usage) is possible.

bitrates We visualize the different bitrates of our captured movies in figure 6. Only two movies have the same bitrate ladder; those are 70098331[9] and 80018499[10] with bitrates at 120, 235, 375, 560, 750, 1050, 1750, 2350 and 3000.

n-package sizes We look at the different n-package sizes contained in the collection. Figure 7 gives us a first hint that our collection is very heterogeneous. We confirm our intuition with a boxplot which shows huge differences in n-package sizes in figure 8.

n-package size per movie We look at 1-package sizes of a single movie in figure 9 and compare them in figure 10. In figure 11 and 12, we look at 10-packages and 28-packages respectively. The observed relative difference in sizes decreases as expected with an increasing aggregation level. Depending on the bitrate, a different number of n-packages are plotted: This is because the capture simply waits for 5 minutes, without modifying the Cadmium fetch logic. So depending on the network condition and the chosen bitrate, the number of fetched packages may differ slightly. A second reason is because we need to filter out the audio packages, our simple heuristic sometimes also removes video packages, for bitrates where those two types of packages are of similar size.

counting package collisions We now count how many packages exactly match over different movies in figure 13. Note that the collision percentage decreases with the aggregation level; this is because the higher aggregation level implicitly remembers the ordering of its containing packages.

[9] Babylon A.D., an action/adventure/Sci-Fi movie from 2008

[10] this is a testvideo probably intended to be used by Netflix engineers

5.3.1 Simulating an attacker

We simulate two different attackers on our dataset; one using n-packages from a single bitrate, the other using n-packages from different bitrates.

choosing how many packages to look at We need to choose *package_by_bitrate*, a factor determining how many n-packages the attacker is allowed to look at for his attack. We choose the number three for all the following experiments unless otherwise described; because it gives us an acceptable performance of our evaluation and a realistic attack scenario. If we were to increase this number, the accuracy would increase too as it can be seen in figure 5.3.1. As the increase in accuracy is more or less uniformly, we assume that the exact number does not influence our conclusions much, which are based on relative numbers.

attack algorithm We execute our attack algorithm as follows:

1. query the database for x n-packages of movie m at bitrate b.
2. distort each n-package with a delta $0 \leq \delta < 1$ to a range with $start = (1 - \delta) * exact\_size$ and $end = (1 + \delta) * exact\_size$.
3. construct a query requiring the existence of a different n-package for each range as determined in the second step.[11]
4. retrieve the set of matching movies per query, and intersect them; the resulting set contains m and zero or more other movies (the "collisions").

our attackers We use two different attackers in our experiments: We use a `naive attacker` which is not aware of the bitrate ladder of a movie. He only knows about n-packages of a single bitrate. We also created a `bitrate attacker`, which is allowed to know the same number of n-packages like the naive attacker, but chooses those from different bitrates. We track how high we can increase a factor *delta* until the best guess for a movie m does consist of more than just one movie.

simulating the bitrate attacker For each movie m, he executes the attack algorithm multiple times: For each bitrate b of that movie, for $x = package\_by\_bitrate$ n-packages each. The result of the b sets of movies from the b executions of the algorithm are intersected, and determine the final best guess of that attacker.

simulating a naive attacker For each movie m, he executes the attack algorithm only a single time: For a selected bitrate b, for $length(available\_bitrates) * package\_by\_bitrate$ n-packages (to ensure that the naive attacker looks at the same number of n-packages than the bitrate attacker). The resulting movies of that single attack algorithm execution then form his best guess.

bitrate vs naive attacker We observed the accuracy of the bitrate attacker is higher compared to the naive attacker as it can be seen in figure 15.

[11] Our used algorithm determining this is only an approximation, but fit for purpose. We describe it in the appendix A.1 closer.

identifying using only bitrate profiles To identify a video using only its bitrate profile (which is another way of saying his bitrate ladder) we may only use the average bandwidth for our attack. We already have all the tools available that we need to perform such an attack: We use our bitrate attacker, set *package_by_bitrate* to 1, and use a high aggregation level to approximate the bandwidth. The result can be seen in figure 16, and clearly shows that an identification based on solely the bitrate profiles under the testing conditions is possible, and even advantageous compared to the naive attacker. While this result is only an approximation (because it is not shown over the full video size), we argue that it is realistic (as the attacker only needs to capture a full minute of video at each bitrate) and has shown itself to be practically feasible to archive for us (as we save us the hassle of capturing 100's of hours of video traffic more).

trend of increasing aggregation levels In figure 17 we show how the accuracy of the bitrate attacker changes when aggregating higher n-packages. This is interesting as it allows us to estimate how the effectiveness of our attack would change using the average bandwidth over a longer time span, up to the point our limited capture database can show us. One can see in the plot that with each higher aggregation level, the accuracy also increases, with the increase being less extreme with each higher aggregation level. We expect this trend to continue at higher aggregation levels, possibly even to the point that the attack starts to lose precision again.

5.3.2 Other observed behaviours

Over the course of this work, we also looked at other factors which might influence our attackers. We will provide a short description of the two most interesting results.

low count of collisions The amount of collisions per movie are mostly distributed as expected: Generally more movies have a low number of collisions than a high one, as it can be seen in the figure 18. The few high collisions can be explained by movies which have only a few bitrates, therefore need to match less n-packages. What can also be seen in this figure is that even at high *delta* there are still movies with very low collision numbers.

aggregation decreases collisions If we execute the same attack we observed that with increasing aggregation levels we get increasingly better results (nearly by a factor 10 at aggregation level 10) as one can observe in figure 19. The few visible inconsistencies (higher aggregation but lower accuracy) can be explained due to the randomness generated when aggregating different following packages. The general trend that a higher aggregation level leads to higher accuracy can be explain due to the way we use SQL in our simulation: we lose any information about the ordering of the packages. This makes our attack clearly less powerful than it needs to be. We forgo an detailed analysis what happens if the ordering of packages is also taken into account as even without this condition we can already provide enough evidence that the bitrate ladders helps with the identification

of a movie.

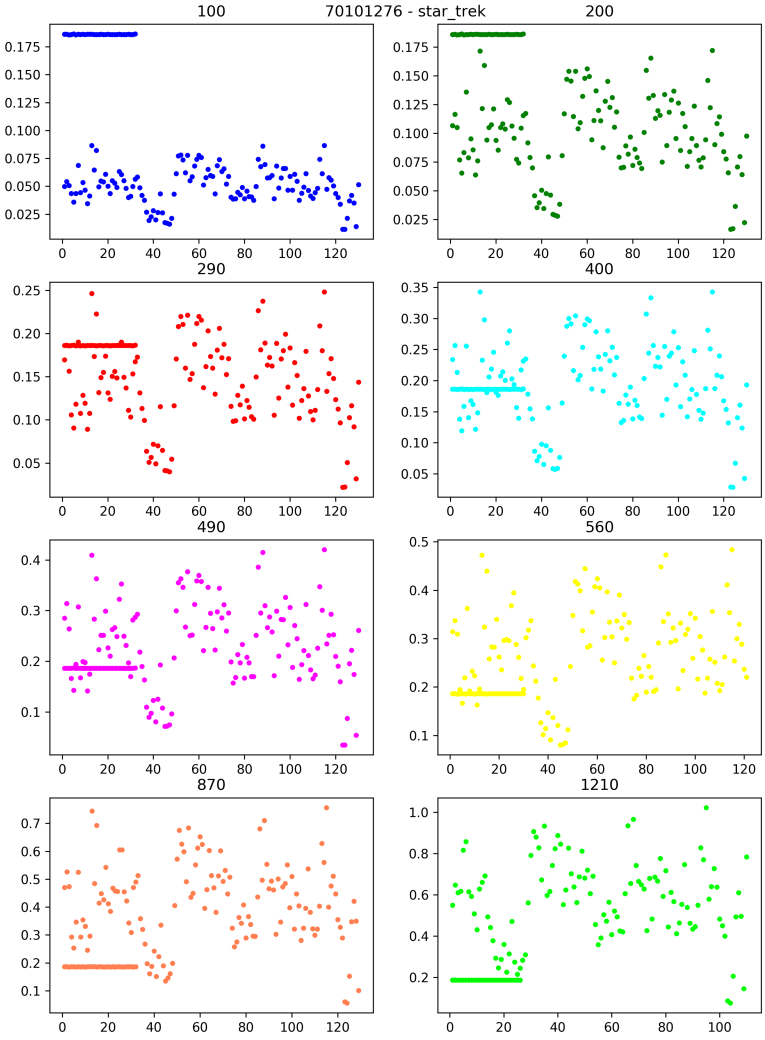

Figure 2: The size of 1-packages of star trek without audio filtering. Each subplot plots the 1-packages of a single bitrate in kilobits/second (denoted on top of the plot, for example 100 for the plot top left). The 1-packages are ordered by their capture time, each x-axis value corresponds to a single 1-package. The y-axis denotes the size of the 1-package in megabytes.

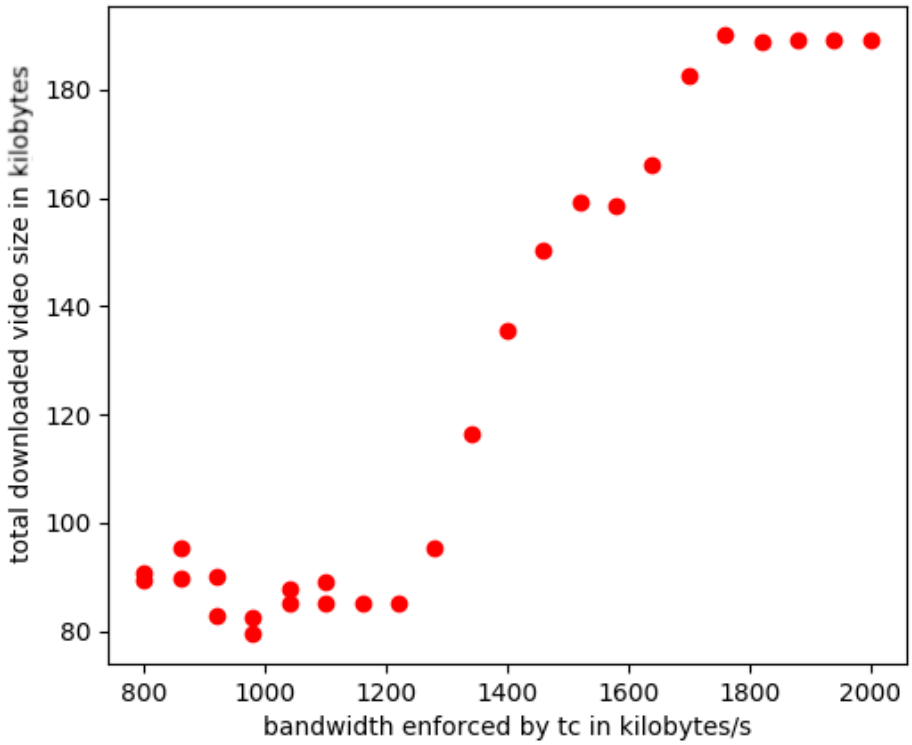

Figure 3: Measured bandwidth usage of video 80018499[4] at specific enforced maximum bandwidths.

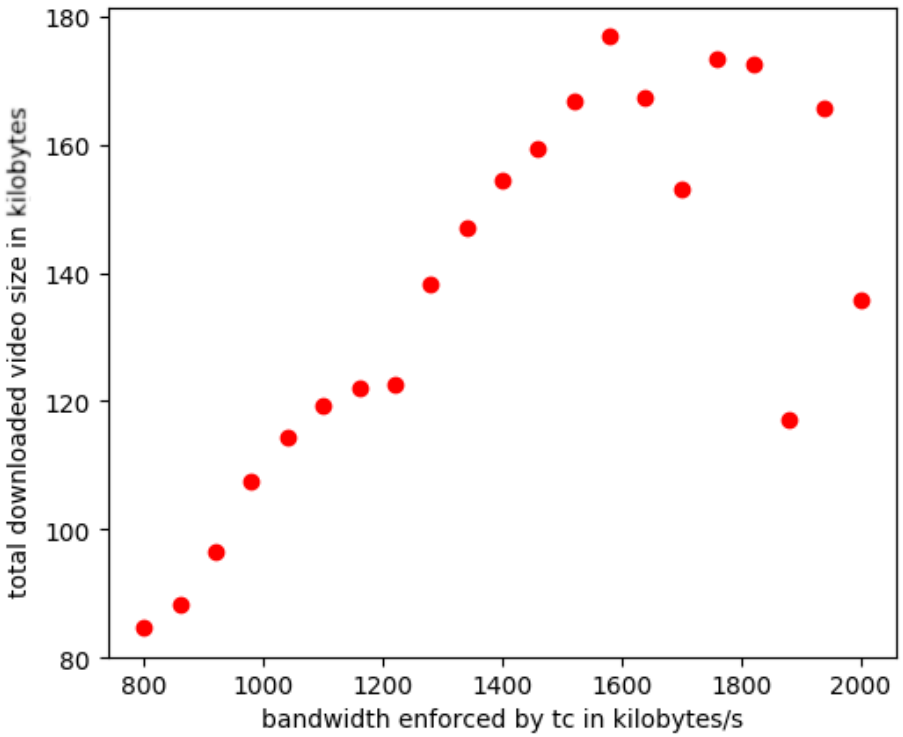

Figure 4: Measured bandwidth usage of video 80088567[6] at specific enforced maximum bandwidths.

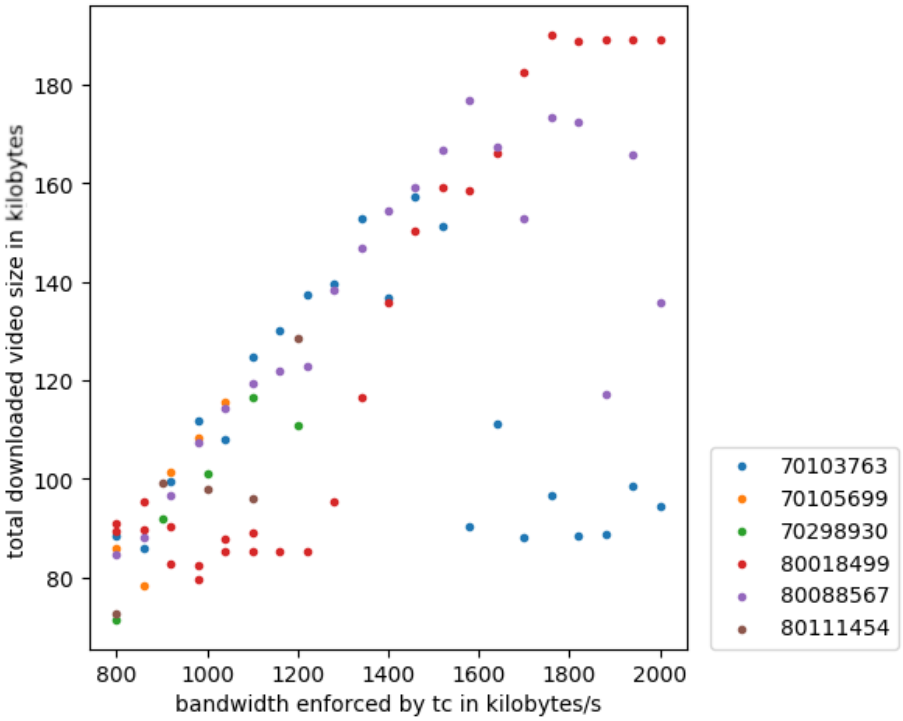

Figure 5: Measured video download size in kilobytes of all tested videos at a specific, by `tc` enforced, maximum bandwidths[8].

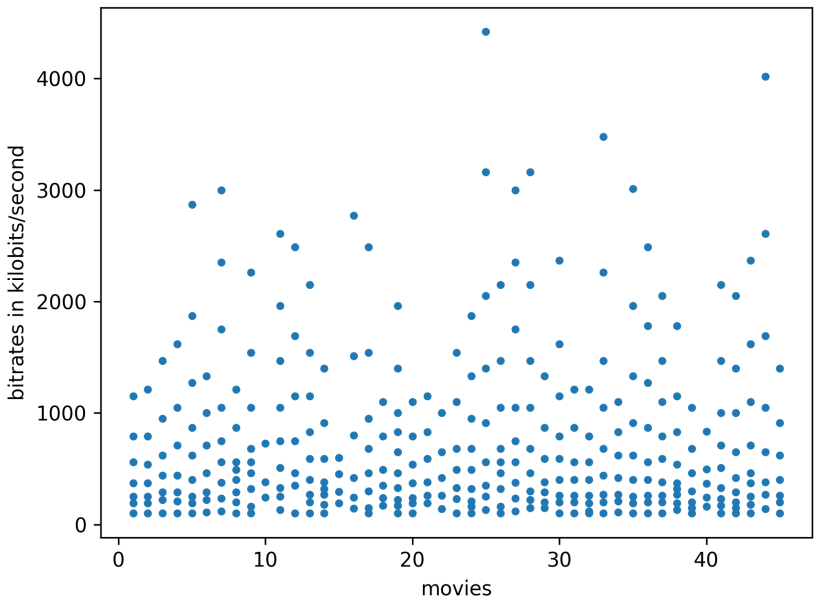

Figure 6: Bitrate ladders of captured videos. Each x-axis value represents a single movie; the corresponding y-values are the bitrates of that movie (hence its bitrate ladder).

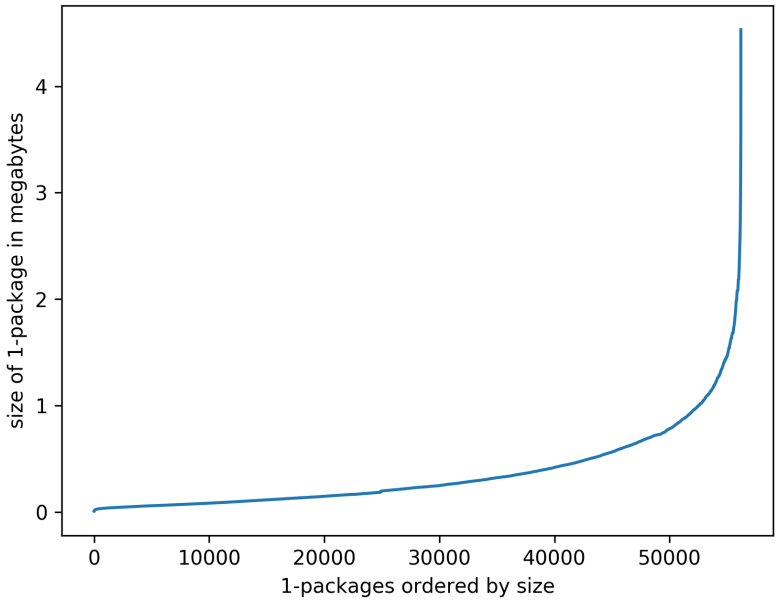

Figure 7: The size of ordered 1-packages (packages at aggregation level 1).

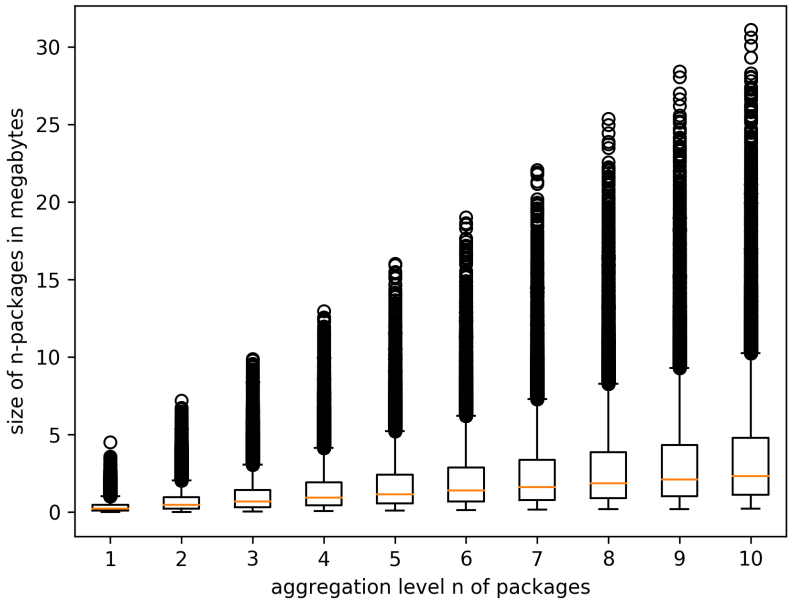

Figure 8: Boxplots of n-packages for different aggregation levels n.

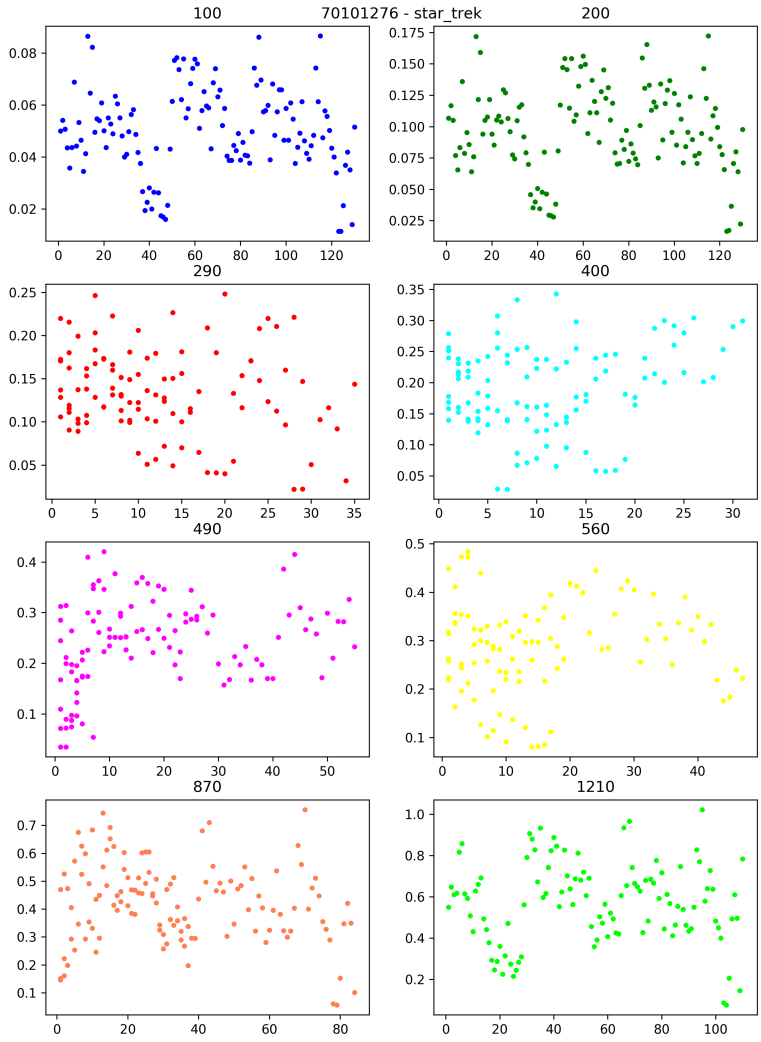

Figure 9: The size of 1-packages of star trek. Each subplot plots the 1-packages of a single bitrate in kilobits/second (denoted on top of the plot, for example 100 for the plot top left). The 1-packages are ordered by their capture time, each x-axis value corresponds to a single 1-package. The y-axis denotes the size of the 1-package in megabytes.

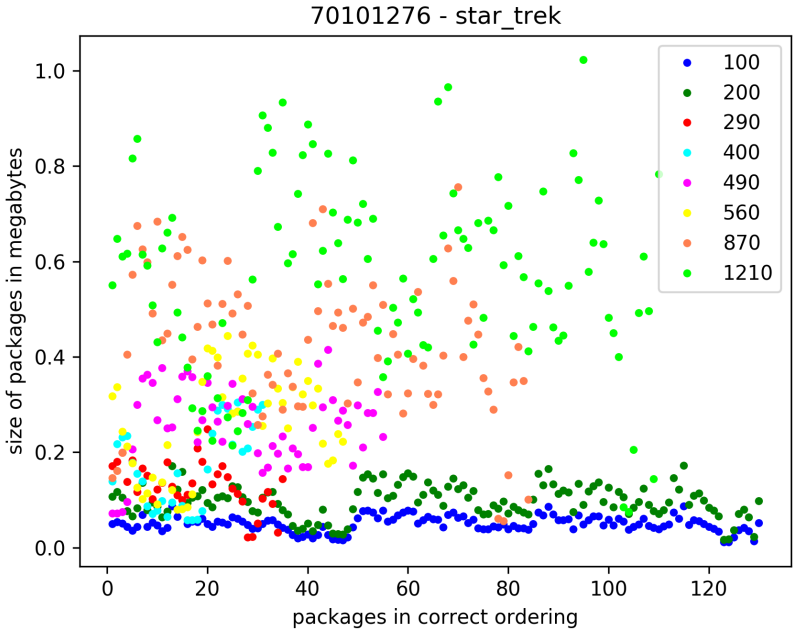

Figure 10: 1-packages of star trek of all different bitrates.

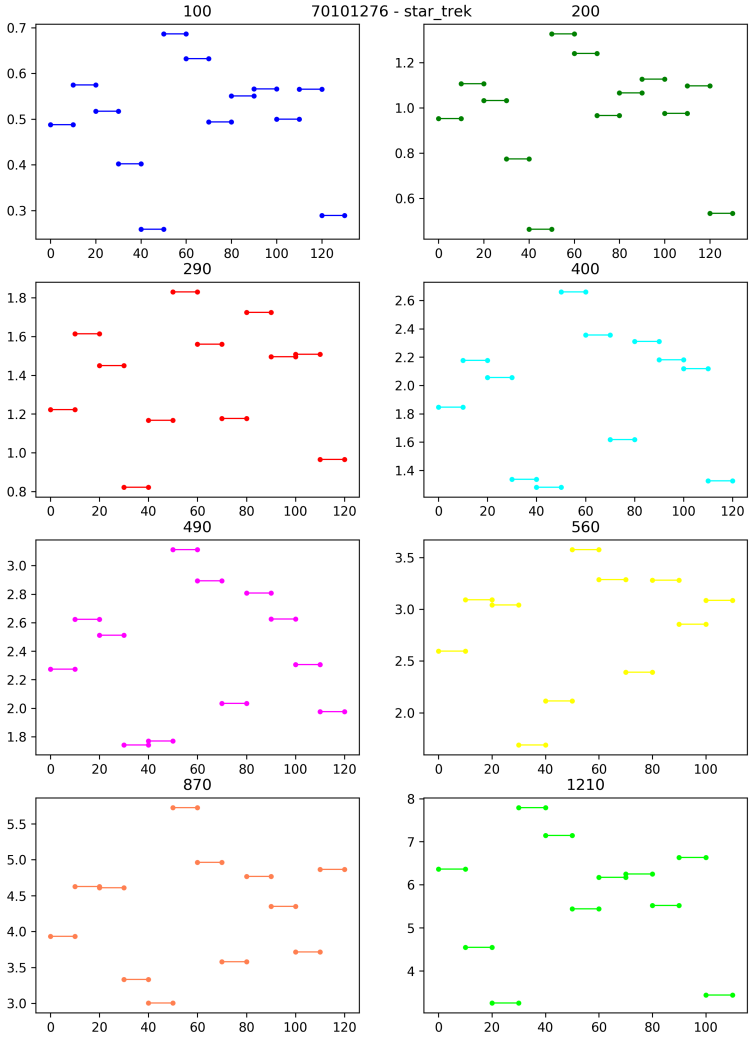

Figure 11: 10-packages of star trek. Each subplot plots the 10-packages of a single bitrate in kilobits/second (denoted on top of the plot, for example 100 for the plot top left). The 10-packages are ordered by their capture time, each x-axis value corresponds to a single 10-package. The y-axis denotes the size of the 10-package in megabytes.

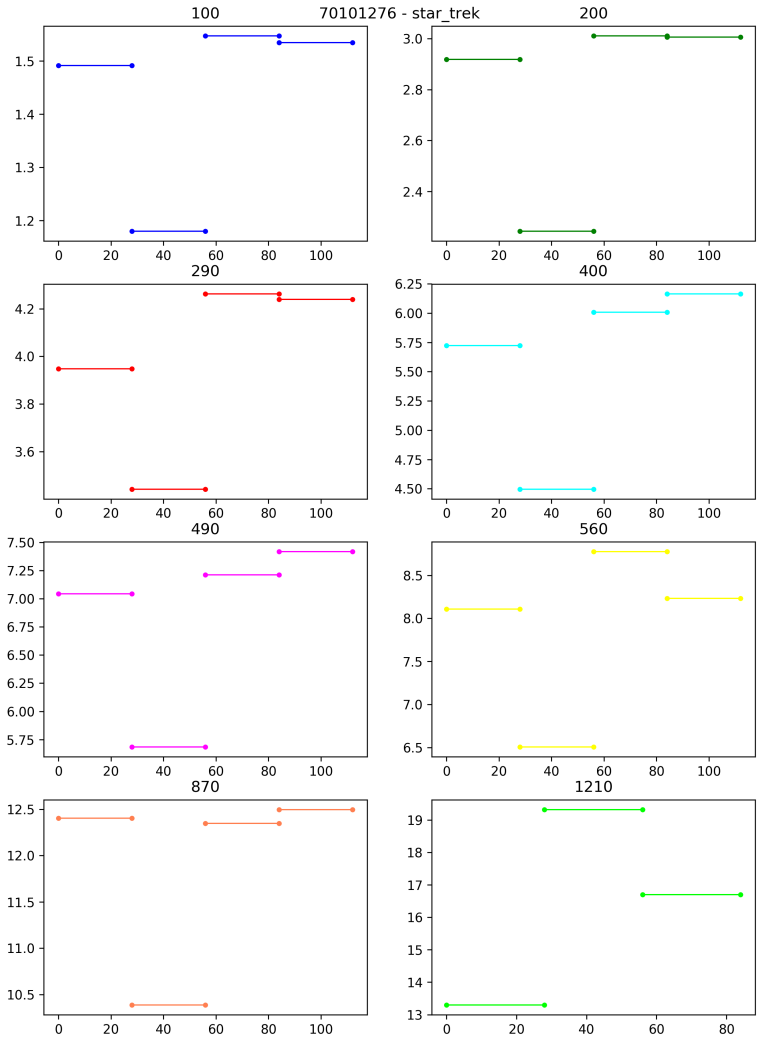

Figure 12: 28-packages of star trek. Each subplot plots the 28-packages of a single bitrate in kilobits/second (denoted on top of the plot, for example 100 for the plot top left). The 28-packages are ordered by their capture time, each x-axis value corresponds to a single 28-package. The y-axis denotes the size of the 28-package in megabytes.

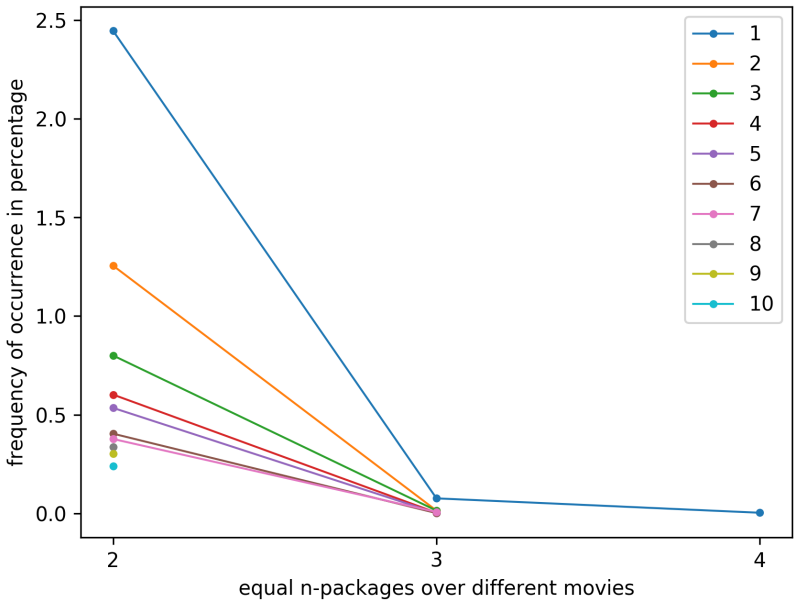

Figure 13: Equal n-packages over different movies and how often this happens. Each color represents a different tested aggregation level.

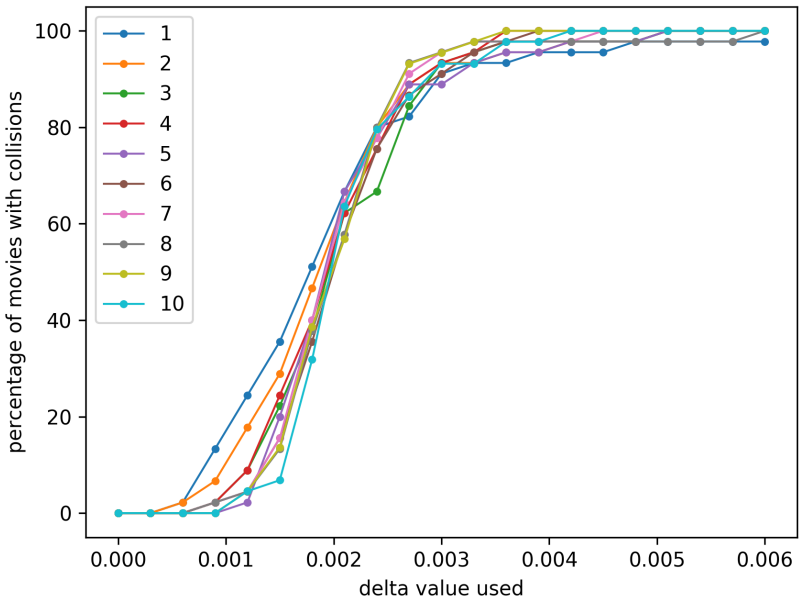

Figure 14: Changes in accuracy for different *package_per_bitrate* values.

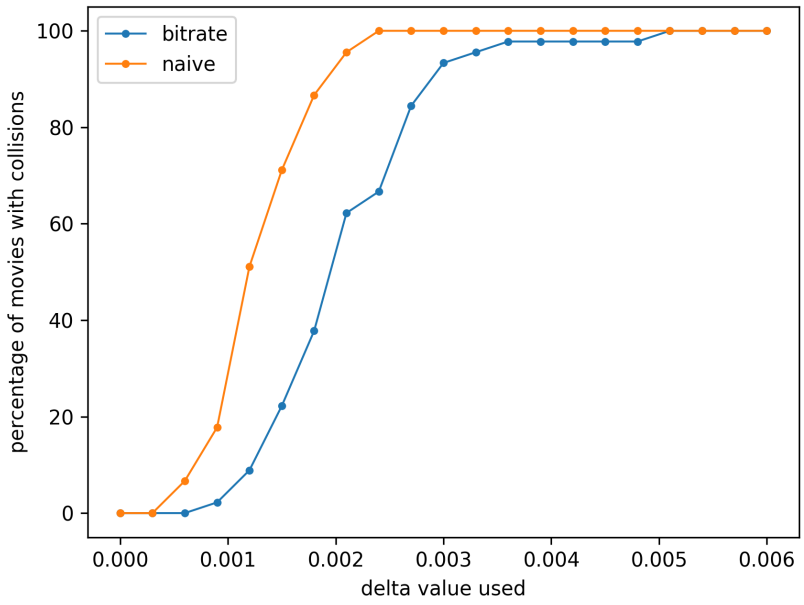

Figure 15: Naive vs bitrate attacker effectiveness. With $package\_by\_bitrate = 3$, both attackers used the same number of 1-packages.

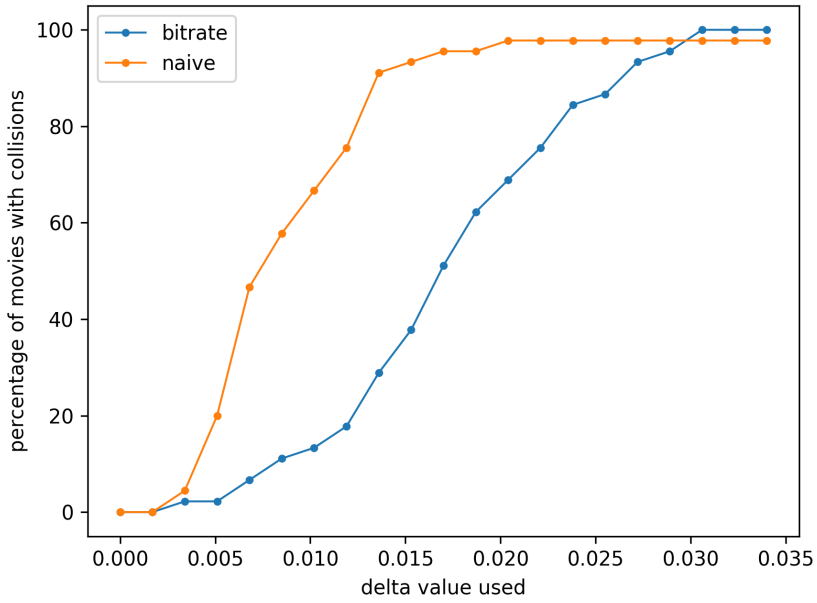

Figure 16: Naive vs bitrate attacker effectiveness using an approximation of measured bandwidth. With *package_by_bitrate* = 1, both attackers used the same number of 10-packages.

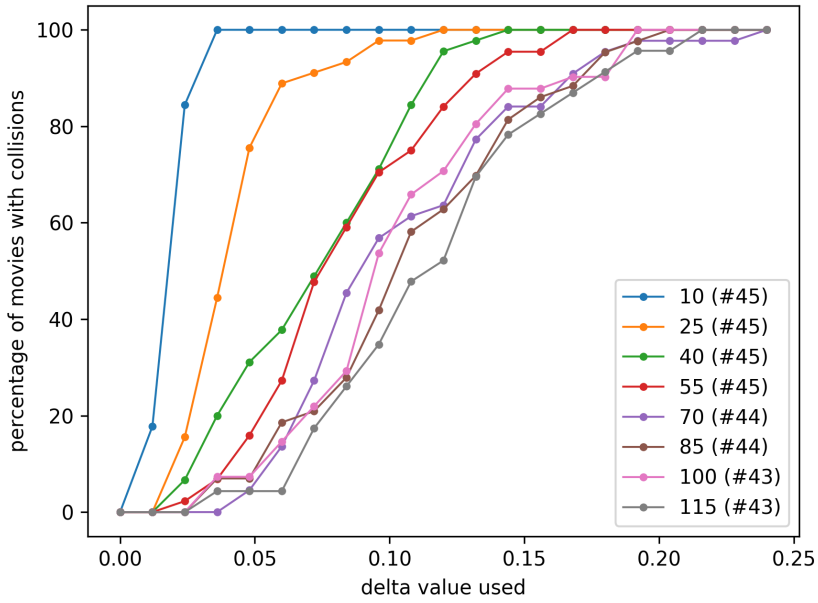

Figure 17: Bitrate attacker effectiveness with *package_by_bitrate* = 1 over different aggregation levels (n-packages). The first number in the legend denotes the used aggregation level, the second number the amount of movies we were able to test at that specific aggregation level (to be able to test the movie we needed at all bitrates enough packages to aggregate to the specific aggregation level. Due to limitations of our capture database this stopped being possible for the first movies at an aggregation level of 70, for most it stopped at around 130).

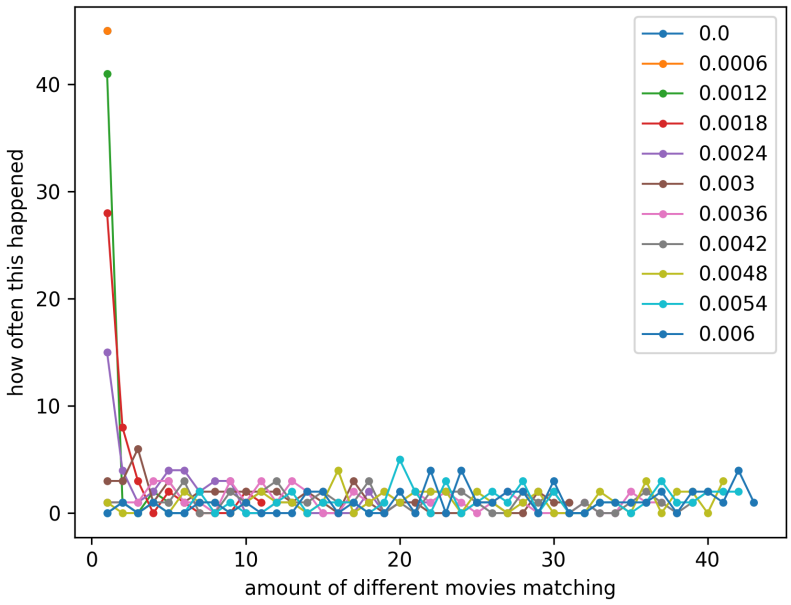

Figure 18: How often collisions (different matching movies) of a certain size occurred with $0 \leq \delta \leq 0.006$ using 1-packages.

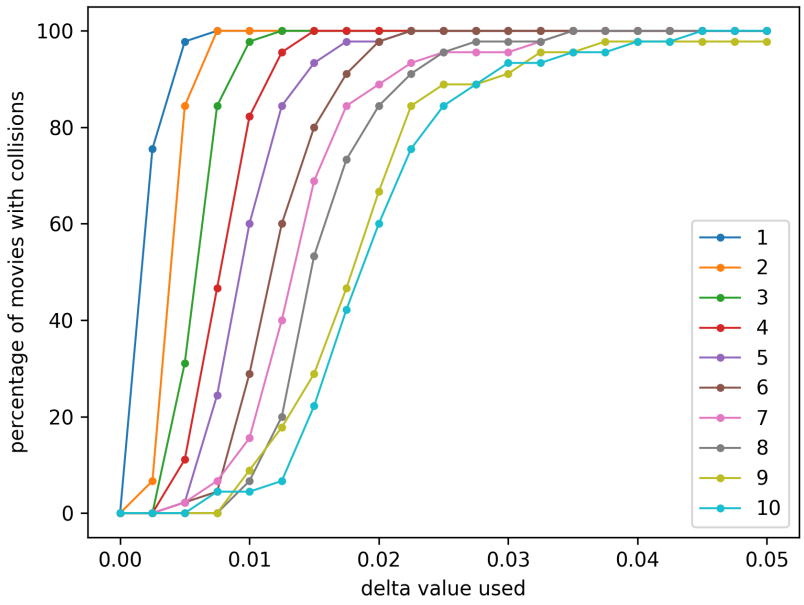

Figure 19: Bitrate attacker collisions with *package_by_bitrate* = 3 for different aggregation levels.

6 Results

We confirm the results already published by Reed [3], and provide evidence that identification of a video with using its bitrate ladder is feasible.

video segment sizes are nearly unique This has already been shown by Reed and has been confirmed by this work. What was still surprising to us is how heterogeneous the segments sizes really are.

the bitrate ladder makes a movie easier identifiable As shown in figure 15 and 16 the attacker has a clear advantage when he uses video segment sizes from different bitrates rather than from a single one.

the bitrate ladder makes a movie identifiable As shown in figure 16 the attacker using only a single aggregated n-package can still identify the videos.

7 Discussion

We could show in this work that identification of videos is made easier by the per-title encoding and its resulting nearly unique bitrate ladder as done by Netflix. While our analysis is not extensive in the terms of the number and length of the videos captured, we strongly think the approach would also work with a larger sample set.

7.1 How to enhance video streaming privacy

It seems clear to us that video streaming, and in particular DASH, has a privacy problem. We present some possible solutions any video streaming provider may implement:

random padding One could randomly pad each video segment. This would counter any attack which works with the exact package sizes, and would probably make other attacks less accurate.

fetch same sized segments A better solution without wasting any bandwidth would be to always fetch same size segments. This should be easily doable in JavaScript with another layer of indirection: Upon requesting another video segment, the player creates a few same sized requests until he has all the data needed for the next segment (possibly more). Already now the video binary file is stored as a continuous file, and the player decides by himself which byte ranges to retrieve [27], serverside this should therefore not need any changes at all.

fix bitrates over multiple videos To solve the issue arising with the per-title bitrates one needs to make deeper changes into the way videos are encoded. To get video streaming services to abandon the advantages of the per-title encoding is probably an illusional target, as both their infrastructure and their users profit greatly from it. A practical approach however would be to sacrifice the fixed resolution set requirement, and instead fix the bitrates. For each bitrate and video, a pair of resolution-compression parameters could be worked out. While preserving the privacy of the video traffic, streaming services and its users could still benefit from a title-specific encoding, while only losing the fixed resolutions set property of their video collection.

7.2 Further work

Some future work comes to mind, which can be divided in two conceptual approaches.

7.2.1 Collect more evidence

One could continue the path of presenting further evidence that video streaming has a privacy problem.

acquire a larger dataset Currently the presented work has not a large data set. We believe however that it shows a trend which could be further confirmed with a larger data set. The current approach of crawling the videos is however not feasible to be used with thousands of videos. Either Netflix itself provides an extensive data set, or the crawler must be enhanced. One potential way of enhancing it would be to take advantage of the initially transmitted manifest: While it is encrypted by Netflix, the JavaScript necessarily needs to decrypt it to be able to use it: Therefore a decrypted version must exist client-side. There are projects which have already archived to reverse engineer this process, their approaches are probably also suited to be used as parts of the crawler [48].

lower algorithms space/time requirements We could provide an intuition that aggregating packages preserved some kind of order information of the packages. It is therefore feasible to assume that high aggregation levels would be enough to identify videos, which would need less space and less expensive query computations than a naive approach where simply all packages are saved. It would be interesting to find out how much the space/time requirement can be lowered while still archiving acceptable results.

develop algorithm which takes order into account As we have already explained in our work, our algorithm does not take ordering into account. We believe this to be a difficult task: While it is clearly solvable using a bruteforce approach, an efficient implementation may be challenging.

7.2.2 Modify DASH

Another path is to try to mitigate the privacy issues. A possible approach would be to modify the existing reference JavaScript dash clients [30] video request component. Clever algorithms which randomize or normalize requests may improve privacy, and their implementation could be tested for feasibility. If such a project would be successful, the maintainers of the DASH reference client may be open for such an improvement.

A Algorithms

A.1 Package existence algorithm

The problem we need to solve (requiring different n-packages which match different criterias) can be reformulated as follows:

task *For n sets s_i, choose one entry $e_i \in s_i$ such that for all $0 < i \leq n$, $0 < j \leq n$ and $j \neq i$ it holds that $e_i \neq e_j$. Return true if such a selection is possible, and $false$ otherwise.*

example A correct algorithm would return *true* for input $s_1 = \{1, 2\}, s_2 = \{1, 2\}$ and for input $s_1 = \{1, 2\}, s_2 = \{2\}$, but would return $false$ for $s_1 = \{1\}, s_2 = \{1\}$ and $s_1 = \{1, 2\}, s_2 = \{1, 2\}, s_3 = \{1, 2\}$. One can assess that this is not an easy problem with the possible input $s_1 = \{1, 2\}, s_2 = \{1, 2\}, s_3 = \{1, 2, 3\}$: The result of the algorithm must be *true*, but if it takes a wrong selection in s_3 (say 1 or 2) it may assumes that the answer is $false$.

our implementation We need to use SQL in our implementation to be able to simulate our attacks in a sensible time frame. We use an approximation which under certain circumstances produces a wrong result, but we believe this to happen rarely, and believe our algorithm to be as accurate as needed for our conclusions. Furthermore, as we only look at relative results, and both attacks we execute use the same algorithm, the failures introduced should not influence our relative result much.

mapping the task to our specific problem In our SQL, we need to retrieve movies which match our given n-package ranges (determined by the given n-package and the chosen *delta*). We do not want however that the same n-package is taken for different ranges, we want that each n-package range is fulfilled by a different n-package. The sets s_i from above are therefore determined by a n-package range, the elements e_i correspond to a single n-package.

our algorithm Our algorithm works as follows:

1. Create a subquery for each range which retries the closest n-packages to our given n-package, within the given n-package range and at most n (the total number of n-packages given and therefore executed subqueries and to-be-found n-packages).
2. Take the $UNION$ of the resulting n-packages of all subqueries.
3. If the resulting set is of size bigger or equal to n we accept the movie, else we reject.

weakness of our algorithm The algorithm produces the wrong results if smaller sets and a bigger one like $s_1 = \emptyset, s_2 = \emptyset, s_3 = \{1, 2, 3\}$ are chosen as input. To solve this issue directly in SQL we would need to recursively perform

our checks till $n = 1$ over all pairs/triplets/... of our subqueries.

assessing our approximation We think the loss in accuracy is acceptable compared to the speedup archived in the development and the execution of the simulations. Furthermore, as we only draw conclusions from relative results of two attackers using that same algorithm, possible advantages/disadvantages may cancel out.

References

[1] M. M. J. D. C. D. R. Anne Aaron, Zhi Li, "Per-title encode optimization." [Online]. Available: https://medium.com/netflix-techblog/per-title-encode-optimization-7e99442b62a2

[2] A. Reed and B. Klimkowski, "Leaky streams: Identifying variable bitrate dash videos streamed over encrypted 802.11 n connections," in *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*. IEEE, 2016, pp. 1107–1112.

[3] A. Reed and M. Kranch, "Identifying https-protected netflix videos in real-time," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. ACM, 2017, pp. 361–368.

[4] "Cisco visual networking index: Forecast and methodology, 2016–2021," September 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[5] M. Webster, "Privacy." [Online]. Available: https://www.merriam-webster.com/dictionary/privacy

[6] "Artikel 12 human rights." [Online]. Available: https://www.humanrights.ch/de/internationale-menschenrechte/aemr/text/artikel-12-aemr-schutz-freiheitssphaere-einzelnen

[7] pwc.de, "Vertrauen in medien 2018." [Online]. Available: https://www.pwc.de/de/technologie-medien-und-telekommunikation/pwc-studie-vertrauen-in-medien-2018.pdf

[8] C. C. Matthew Rosenberg, Nicholas Confessore, "How trump consultants exploited the facebook data of millions." [Online]. Available: https://www.nytimes.com/2018/03/17/us/politics/cambridge-analytica-trump-campaign.html

[9] S. F. Matthew Rosenberg, "Facebook's role in data misuse sets off storms on two continents." [Online]. Available: https://www.nytimes.com/2018/03/18/us/cambridge-analytica-facebook-privacy-data.html

[10] H. van Rinsum, "Facebook unter beobachtung: Das blieb vom datenskandal." [Online]. Available: https://www.internetworld.de/online-marketing/facebook/facebook-unter-beobachtung-blieb-datenskandal-1550354.html

[11] EU, "General data protection regulation (gdpr)." [Online]. Available: https://gdpr-info.eu/

[12] Sandvine, "Global internet phenomena report," December 2015.

[13] "what is minification." [Online]. Available: https://blog.stackpath.com/glossary/minification/

[14] "Modified cadmium." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/tools/netflix-1080p-1.2.9/cadmium-playercore-2018-03-17.js

[15] "Selenium." [Online]. Available: https://www.seleniumhq.org/

[16] "Browsermob proxy." [Online]. Available: https://github.com/lightbody/browsermob-proxy

[17] "Netflix 1080p extension." [Online]. Available: https://github.com/truedread/netflix-1080p

[18] "Har 1.2 spec." [Online]. Available: http://www.softwareishard.com/blog/har-12-spec/

[19] "capture.py." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/3_video_capture/capture.py

[20] "Sqlite." [Online]. Available: https://www.sqlite.org/about.html

[21] "create_listen_db.py." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/4_analysis/create_listen_db.py

[22] "create_plot_db.py." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/4_analysis/create_listen_db.py

[23] "plot_attacker.py." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/4_analysis/plot_attacker.py

[24] netflix, "About netflix." [Online]. Available: https://media.netflix.com/en/about-netflix

[25] dpa AFX, "Netflix-aktie schießt auf rekordhoch: Netflix glänzt mit kräftigem umsatz- und gewinnplus." [Online]. Available: https://www.finanzen.net/nachricht/aktien/boom-haelt-an-netflix-aktie-schiesst-hoch-netflix-glaenzt-mit-kraeftigem-umsatz-und-gewinnplus-6100873

[26] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.

[27] Dash-Industry-Forum, "Dash industry forum - about." [Online]. Available: https://dashif.org/about/

[28] C. Mueller. [Online]. Available: MPEG-DASH (Dynamic Adaptive Streaming over HTTP)

[29] ISO, "Mpeg-dash schema files." [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-DASH_schema_files/

[30] Dash-Industry-Forum, "Dash.js reference implementation." [Online]. Available: https://github.com/Dash-Industry-Forum/dash.js

[31] J. Terrell, K. Jeffay, F. D. Smith, J. Gogan, and J. Keller, "Passive, streaming inference of tcp connection structure for network server management," in *International Workshop on Traffic Monitoring and Analysis*. Springer, 2009, pp. 42–53.

[32] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, Mar 1983.

[33] "Network experiments." [Online]. Available: https://github.com/famoser/network-experiments

[34] "Tc." [Online]. Available: https://wiki.archlinux.org/index.php/advanced_traffic_control

[35] "Tc." [Online]. Available: https://github.com/thombashi/tcconfig

[36] "Man in the middle." [Online]. Available: https://us.norton.com/internetsecurity-wifi-what-is-a-man-in-the-middle-attack.html

[37] "Chrome." [Online]. Available: https://www.google.com/chrome/

[38] "bandwidth_manipulator.py." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/python_libs/bandwidth_manipulator.py

[39] "browser_proxy.py." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/python_libs/browser_proxy.py

[40] "config.py." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/python_libs/config.py

[41] "har_analyzer.py." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/python_libs/har_analyzer.py

[42] "mouse.py." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/python_libs/mouse.py

[43] "netflix_browser.py." [Online]. Available: https://github.com/famoser/network-experiments/blob/publication/python_libs/netflix_browser.py

[44] "Experiment 1: First cadmium modification." [Online]. Available: https://github.com/famoser/network-experiments/tree/publication/1_first_cadmium_modification

[45] "Experiment 2: Automate bandwidth enforcement." [Online]. Available: https://github.com/famoser/network-experiments/tree/publication/2_automate_bandwidth_enforcement

[46] "Experiment 3: Video capture." [Online]. Available: https://github.com/famoser/network-experiments/tree/publication/3_video_capture

[47] "Experiment 4: Analysis." [Online]. Available: https://github.com/famoser/network-experiments/tree/publication/4_analysis

[48] "Netflix plugin for kodi 18." [Online]. Available: https://github.com/asciidisco/plugin.video.netflix