Leaky Streams

Identifying Variable Bitrate DASH Videos Streamed over Encrypted 802.11n Connections

Andrew Reed, Benjamin Klimkowski
Dept. of Electrical Engineering and Computer Science
United States Military Academy at West Point
West Point, New York, USA
{andrew.reed, benjamin.klimkowski}@usma.edu

***Abstract*—In recent years, Dynamic Adaptive Streaming over HTTP (DASH) has become the primary method to deliver video on the Internet, with Netflix currently leading the industry. Thus, any method to determine the content of a wireless Netflix stream presents a potential privacy concern for the entire DASH industry. In this paper, we demonstrate that it is possible to identify the Netflix video being streamed over an encrypted 802.11n connection with high accuracy in less than five minutes. Moreover, our technique works in scenarios where it is difficult to capture data frames due to a wireless access point's use of enhancements such as beamforming and multi-input/multi-output transmission.**

Keywords—privacy; traffic analysis; wireless networks; dynamic adaptive streaming over HTTP

I. INTRODUCTION

Yearly reports from Sandvine [9, 10] continually indicate that Dynamic Adaptive Streaming over HTTP (DASH) services account for a large proportion of traffic to households in North America, with Netflix alone comprising 34.9% of all downstream traffic in the most recent report. As such, Netflix presents a prime target for an adversary that wants to determine the content being watched by a given household. In this paper, we assume that the adversary is a wireless eavesdropper and that the target household uses a wireless access point (WAP) with encryption (e.g. WPA2). We also assume that an eavesdropper has only a limited amount of time to monitor a given WAP (e.g. less than 5 minutes) before drawing attention.

Despite these challenges, we show that an eavesdropper can accurately identify the Netflix video being streamed by (i) estimating WAP-to-client throughput from the *client's* Block Acknowledgements (BlockAcks) and then (ii) matching this throughput data to a video fingerprint. Since our technique does not require access to frame payloads, it renders encryption moot. Furthermore, since BlockAcks are often transmitted via 802.11a/g, our technique is effective in environments where the eavesdropper is unable to capture data frames due to a WAP's use of 802.11n enhancements such as beamforming and multiple-input/multiple-output (MIMO) transmission.

Although our paper focuses on Netflix, our methods are applicable to any DASH service that exhibits the same characteristics as Netflix. We have made our code and test data available at [4]. The rest of the paper is organized as follows.

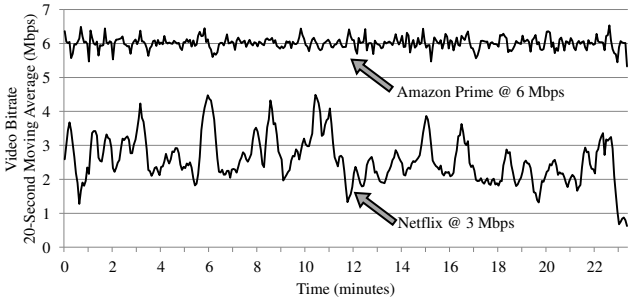


Fig. 1. A comparison of the average bitrate between the same video (*Curious George*, season 1, episode 1) on different DASH services.

Section II provides an introduction to DASH. Section III details our method for constructing a video database and Section IV describes our method for capturing wireless data. Our method for identifying a Netflix video is explained in Section V and evaluated in Section VI. Section VII offers countermeasures to thwart our technique and Section VIII reviews related work.

II. BACKGROUND

A DASH video is first encoded at various bitrates, or quality levels, and then each encoding is segmented into equal length time slices. These video segments are then served from HTTP web servers, typically via content distribution networks (CDNs). During playback, a DASH client will continually gauge the available bandwidth to determine which quality level of each segment should be requested. If bandwidth is limited, a DASH client will request segments from a lower bitrate. As bandwidth improves, the client will begin to request segments from a higher bitrate.

Netflix currently provides browser-based streaming with Microsoft Silverlight [8]. Netflix encodes each video at 235, 375, 560, 750, 1050, 1750, 2350, and 3000 kbps and then segments them into four second video segments and 16 second audio segments. The individual encodings are made available as separate .ismv files, a format which is based on MPEG4. As defined in the MPEG4 specification [5], each .ismv contains

The views expressed herein are those of the authors and do not reflect the position of the United States Military Academy, the Department of the Army, or the Department of Defense.

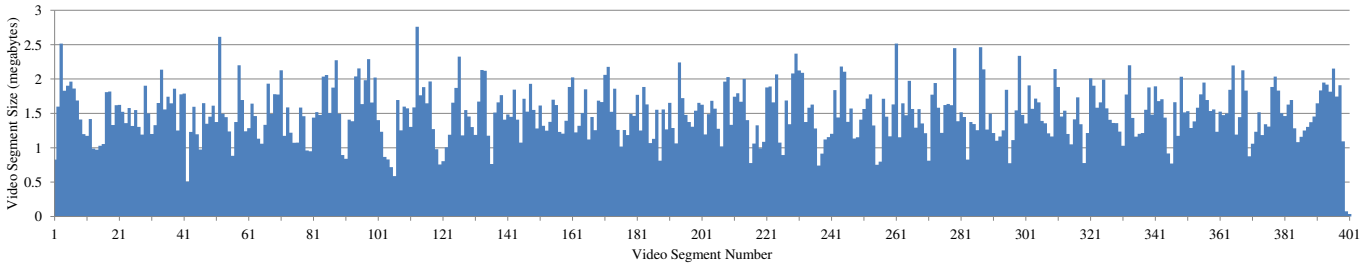


Fig. 2. Video fingerprint for *Room on the Broom* (3000 kbps encoding).

metadata at the beginning of each file which details how the video is segmented. Specifically, the metadata's segment index box (sidx) lists the sizes (in bytes) for each individual video segment of an encoding. The Netflix Silverlight player uses this information to generate byte range requests when requesting each segment. Thus, one of the first actions performed by the Silverlight player when streaming a video is to retrieve the metadata from each encoding, thereby allowing the player to switch bitrates at any time during the stream. Currently, the metadata of an .ismv is neither encrypted nor protected by Digital Rights Management (DRM), but DRM does encrypt the actual video content.

Netflix's Silverlight player seeks to fill a four minute buffer as quickly as possible. As long as the buffer remains full, the player will request one video segment each time a video segment in the buffer is consumed, resulting in requests being sent every four seconds (on average). This behavior is referred to as *steady state* playback [6].

In the course of our research, we have discovered that Netflix uses variable bitrate encoding (VBR) to a greater extent than other services, as shown in Fig. 1. It is Netflix's extensive use of VBR, coupled with DASH's steady state mode, which makes Netflix videos identifiable via throughput analysis.

III. Creating the Video Database

In this section, we describe the process to obtain Netflix fingerprints and then present the method that we developed to turn fingerprints into a searchable database of individual two-minute windows. This database will be queried by the identification algorithm in Section V.

A. Acquiring Fingerprints

Critical to the success of our technique is the ability to create an authoritative database of video fingerprints, where each fingerprint lists the sequence of video segment sizes for each quality level of a video. Conveniently, the information required to create a fingerprint is provided by each .ismv's metadata (sidx). Thus, the task is reduced to (i) obtaining an .ismv's metadata and (ii) parsing its sidx to obtain the sequence of video segment sizes.

We provide a script at [4] that performs these steps with minimal effort required by a researcher. The script requires that a researcher capture the first few seconds of a video with the Firefox add-on Tamper Data [13], during which time the Silverlight player sends HTTP GET requests for each quality level's metadata. The script (i) extracts the metadata URLs, (ii) re-downloads all metadata, and (iii) calculates segment sizes from sidx information. With this script, a researcher can create the fingerprints for a video's multiple encodings in 2-3 minutes. A sample fingerprint is depicted in Fig. 2.

B. Window Storage and Retrieval

From each fingerprint we extract every 30-segment sliding window and store them individually in a six-dimensional (6D) kd-tree [1]. Each window is stored with the following key:

- **1st Dimension: Total Size.** This is the total amount of data contained in the window (unit = bytes).
- **2nd – 6th Dimensions: Data Allocations.** We divide the two minute window into five 24-second slices and calculate the percentage of the entire window's data contained in each slice (no units; these are numbers between 0.0 to 1.0).

When we run the identification algorithm, outlined in Section V, this 6D key allows us to perform a range search for candidate windows that are roughly the same size (1st dimension) and shape (2nd – 6th dimensions) as a given wireless capture. Fig. 3 depicts the key for a sample window from *Legally Blonde*. Since this movie is 1440 segments long (96 min), each encoding results in 1411 sliding windows. Thus, *Legally Blonde's* eight encodings will yield 11,288 individual windows.

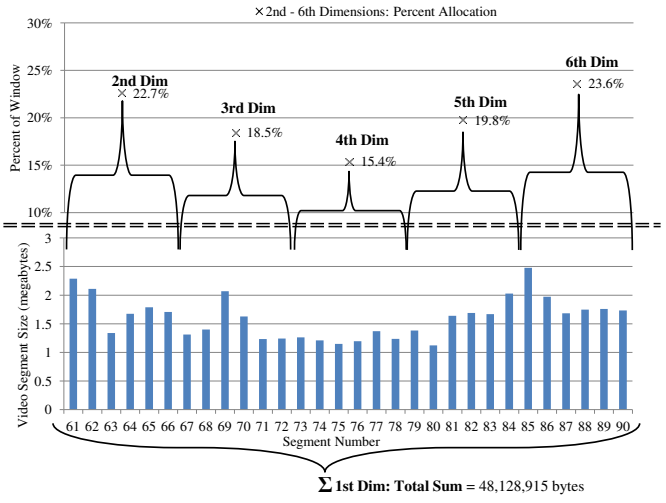


Fig. 3. Key construction for segments 61 through 90 of *Legally Blonde* (3000 kbps encoding). This window's 6D key is: [48128915, 0.227, 0.185, 0.154, 0.198, 0.236].

IV. Data Acquisition

In this section, we describe our method to capture and process wireless traffic to support identification.

A. Estimating WAP-to-Client Throughput

Although the most direct method to calculate WAP-to-client throughput is to log the sizes of all data frames sent to the client, it can be difficult to capture data frames in 802.11n environments. For instance, if a WAP uses beamforming, then the eavesdropper may have trouble finding a location that provides sufficient received signal strength. Alternatively, if the WAP and client support MIMO transmissions, then the eavesdropper's capture device must support the same number of spatial streams employed by the WAP and client.

To demonstrate the effects of MIMO incompatibility when capturing, the wireless capture in Fig. 4 represents a snippet of a Netflix stream as transmitted by an ASUS RT-N66U 3x3 wireless router to a Windows 7 laptop equipped with an Intel Centrino Advanced-N 6200 2x2 adapter. This capture was produced by a laptop running Kali Linux using a TP-LINK TL-WN722N 1x1 USB adapter. Notice that Fig. 4 contains no data frames. This is due to the 1x1 USB adapter's inability to capture data frames that are sent over multiple spatial streams.

Although the capture in Fig. 4 contains no data frames, it *does* contain control frames, namely request-to-send (RTS), clear-to-send (CTS), and BlockAck control frames. Further inspection of the Radiotap headers for these frames in Wireshark reveals that they were sent via 802.11g (Wireshark lists them as *pure-g*), i.e. they were *not* sent via 802.11n MIMO. Thus, since BlockAcks are easier to capture, we leverage them to estimate throughput. We do so by (i) assuming that a BlockAck's starting sequence number correlates to the receipt of the data frame with the same sequence number, (ii) inserting missing data frames between the received BlockAcks, and (iii) assuming that all data frames carry a full-size packet (i.e. 1500 bytes), which equates to 1460 bytes of application layer data. Fig. 5 depicts our approach.

B. Data Aggregation

Before attempting to match the throughput data to a window in the kd-tree, we must first aggregate the frame sizes into four second chunks. In order for a match to be found, however, the four second chunks must be aligned with the times at which the Netflix player requested video segments during steady state playback. If these chunks are misaligned, then data frames from adjacent video segments will be incorrectly combined, resulting in a false negative.

To achieve proper alignment, we first sum frames into 250 ms bins, with the first bin starting upon receipt of the first captured frame. These bins are stored in a deque that contains a moving window of the most recent 480 bins (i.e. two minutes of throughput data). As the moving window advances, the current deque of 480 bins is consolidated into a sequence of 30 chunks that is then processed by the identification step. Thus, every two minutes' worth of throughput data will be checked using 16 different alignments.

Time	Source	Dest.	Info
05:42.5028	WAP	Client	RTS
05:42.5028	Client	WAP	CTS
05:42.5028	Client	WAP	Block Ack Start: 3290
05:42.5028	WAP	Client	RTS
05:42.5028	Client	WAP	CTS
05:42.5028	Client	WAP	Block Ack Start: 3295
05:42.5028	WAP	Client	RTS
05:42.5053	Client	WAP	CTS
05:42.5053	WAP	Client	RTS
05:42.5053	Client	WAP	CTS
05:42.5079	Client	WAP	Block Ack Start: 3297

Fig. 4. Sample snippet from a wireless capture of Netflix traffic.

Time	Sequence Number	App. Layer Data
05:42.5028	3290	1460
05:42.5028	3291	1460
05:42.5028	3292	1460
05:42.5028	3293	1460
05:42.5028	3294	1460
05:42.5028	3295	1460
05:42.5053	3296	1460
05:42.5079	3297	1460

Fig. 5. Estimating WAP-to-client throughput using the capture depicted in Fig. 4.

Note that these 30 chunk sequences will initially represent both video *and* audio throughput. For browser based streaming, the Netflix player will request a single, 16-second audio segment for every four video segments. These audio segments are encoded at a constant bitrate (CBR) of 64 kbps and they average 135 kB in size. Since our technique is unable to detect when these audio segments were received, we simply subtract 33.75 kB from each four second chunk in order to approximate the removal of audio data from the capture. Removing audio data allows us to place tighter restrictions on the 1st dimension when conducting the kd-tree range search, described in Section V-A.

V. Identification Algorithm

A. Stage 1 – Retrieve Candidate Windows

The purpose of Stage 1 is to generate a shortlist of candidate windows that exhibit similar throughput characteristics as the capture window. This is done by creating a 6D key for the capture window and then conducting a range search of the kd-tree for all candidate windows where the 1st dimension and the 2nd through 6th dimensions are within a predetermined set of thresholds from the capture window. The results returned by the range search are then sent to Stage 2.

B. Stage 2 – Report Matching Windows

The purpose of Stage 2 is to determine which, if any, of the candidate windows match the capture window. To do this, we perform the following two steps for each candidate window:

1) Ignore Outliers: Even during steady state playback, it is sometimes the case that a video segment will be received outside of its expected four second slot (either early or late). When this occurs, it will create an abundance of data in a

neighboring slot and a drought of data in its own slot, thereby creating two incorrect throughput measurements. To account for this, we allow for two such occurrences by ignoring the *four* "worst" pairs of segments (by percentage difference) from the candidate and capture windows, reducing the number of segments under consideration to 26 pairs. When the candidate and capture windows are plotted as an *x,y* scatter plot, this step has the effect of removing four of the outliers.

2) Calculate Pearson's Correlation: The second step is to compute Pearson's correlation (*r*) between the remaining 26 points in the candidate and capture windows. If Pearson's *r* is above a given threshold, then the candidate window is considered a match. A matching window will be reported to Stage 3 with the following information: the title of the source video, the starting segment number of the window, and the start time of the capture window that it matched.

C. Stage 3 – Determine Video from Reported Matches

Stage 3 receives the reported matches (the full 30-segment window) from Stage 2 and stores them. From here, Stage 3 is equipped with two modes that it can use to identify a video: *fast mode* and *slow mode*.

1) Fast Mode: Upon the receipt of each new match, Stage 3 conducts an "all pairs" test to see if there is a pair of windows from the same source video that (i) overlap by no more than 25 segments and (ii) whose separation in the source fingerprint is congruent to the timestamp difference between their corresponding capture windows. Once such a pair is found, the video is declared and the algorithm terminates early. Since the second window in the pair must begin at least five segments after the first window began, the earliest declaration cannot occur before 2 minutes and 20 seconds into the wireless capture.

This mode is modeled after the final step performed by the audio search service Shazam [14]. Whereas our technique requires only a single pair of correlated matches, [14] requires multiple temporally-aligned matches from a single audio track. Should we find that our method generates a high number of false positives as the fingerprint database grows, one potential improvement would be to require multiple correlated matches, as in [14].

2) Slow Mode: Occasionally, the algorithm fails to make a fast mode declaration prior to the end of the wireless capture. In this case, the algorithm reverts to a majority voting scheme. For each potential candidate video, Stage 3 calculates the number of distinct windows matched and declares that the video with the most distinct matches is the video being watched.

VI. Evaluation

To support the evaluation of our technique, we created two sets of fingerprints: Dataset A, which consists of 50 movies, and Dataset B, which consists of the first season of *House of Cards*. When loaded into the kd-tree, Dataset A yields 584,776 windows and Dataset B yields 77,048 windows.

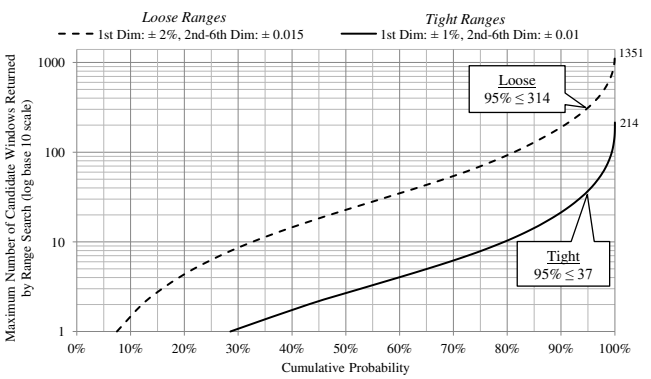


Fig. 6. Cumulative probability of Stage 1 shortlist sizes when performing a range search of Dataset A.

A. Assessing Stage 1's Filtering Capability

The purpose of Stage 1 is to present only the most relevant candidate windows to Stage 2, thereby improving query efficiency. As such, our first test is designed to gauge the ability of Stage 1 to filter the video database for candidate windows. For this test, we built a database from Dataset A and proceeded to conduct a range search for each window in this same database. For each search, we recorded the number of windows returned in the shortlist. Since each search represents a window that is also in the database, every search will return at least one result. This test was conducted twice: once with a set of *loose* ranges and once with a set of *tight* ranges. The *loose* range search was conducted with ±2% for the 1st Dimension and ±0.015 for the 2nd through 6th Dimensions. The *tight* range search was conducted with ±1% for the 1st Dimension and ±0.01 for the 2nd through 6th Dimensions. Fig. 6 depicts the results as an inverse cumulative distribution function.

For the *loose* range test, 7.4% of all range searches returned one result (i.e. the same window used for the search), 95% of all range searches returned no more than 314 candidate windows, and no search returned more than 1351 candidate windows. For the *tight* range test, 28.5% returned one result, 95% returned no more than 37 candidates, and no search returned more than 214 candidates. In other words, all *tight* range searches returned less than 0.037% of the database and all *loose* range searches returned less than 0.23% of the database.

B. Full System Assessment

In order to evaluate our technique in a "real world" setting, we streamed 25 movies from Dataset A using the same hardware that generated Fig. 4 within a residential home. We placed the wireless router and Kali Linux laptop on the first floor of the home, and the Windows 7 laptop was placed on the second floor. The router was configured to use 2.4GHz channel 11. The home has a 30 Mbps downstream and 10 Mbps upstream Internet connection.

The order and selection of the 25 movies were determined by shuffling the movie titles and streaming the first 25 listed. For each movie, we first skipped to a random scene in the middle of the movie and then allowed the stream to reach steady state playback before beginning the wireless capture on

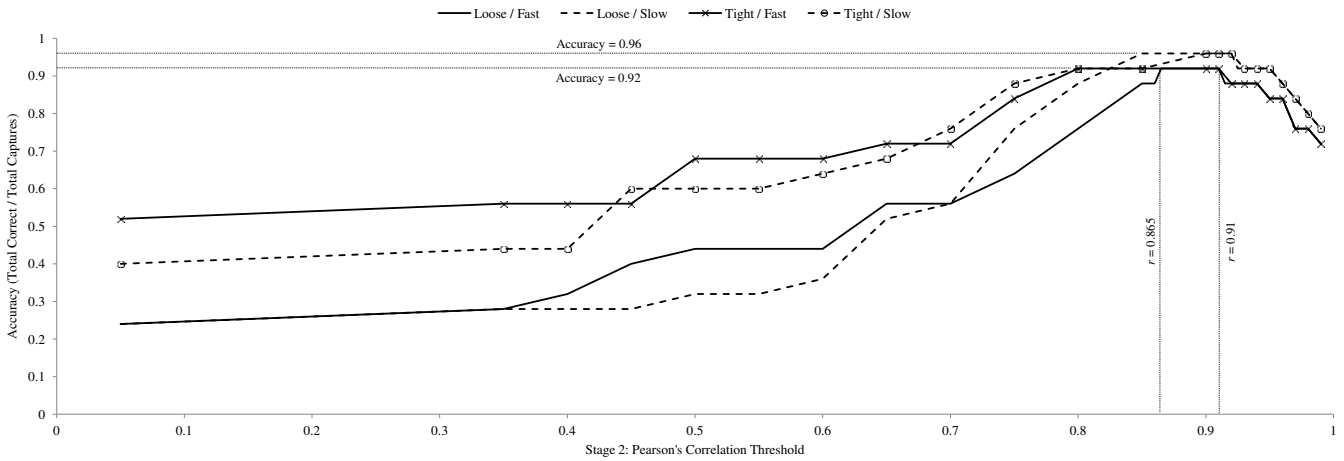


Fig. 7. The effect of increasing Stage 2 thresholds on accuracy.

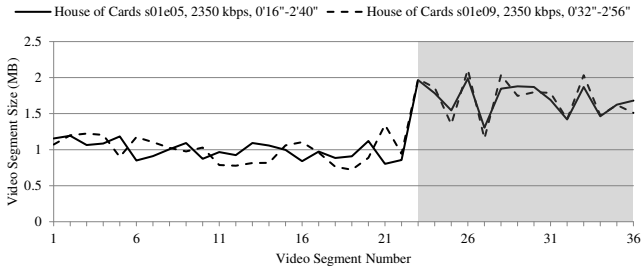
the eavesdropper laptop. Each capture was limited to exactly five minutes via the Linux `timeout` command.

We then ran a series of tests on the 25 capture files using a variety of thresholds to assess the technique's performance at each stage. Specifically, we tested Stage 1 using the *loose* and *tight* range searches across increasing thresholds for the Stage 2 Pearson's correlation. These tests were done for both of Stage 3's modes. For each permutation of thresholds and modes, we ran our program on the 25 captures and calculated its overall accuracy (total number of correct identifications / 25).

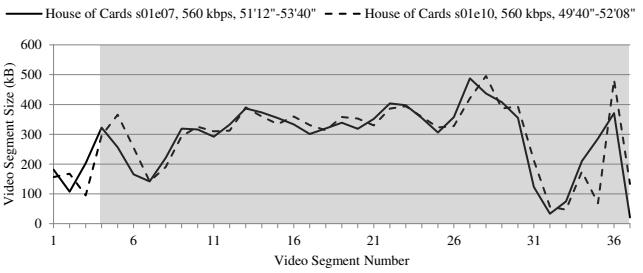
Figure 7 summarizes our results. Over all of the permutations of thresholds, Pearson's correlation (Stage 2) had the greatest overall effect on accuracy. As the threshold for Pearson's correlation rises, Stage 2 becomes an increasingly important component of the technique's accuracy. At optimal ranges for Pearson's, there is no difference in accuracy among the Stage 1 thresholds. In other words, *loose/fast* and *tight/fast* converge to the same results, and *loose/slow* and *tight/slow* converge to the same results.

Peak accuracy for both fast and slow modes is reached between $r = 0.865$ and $r = 0.91$, with slow mode achieving an accuracy of 0.96 and fast mode achieving an accuracy of 0.92. Accuracy then wanes as Pearson's r is increased beyond 0.91. At higher values for r the technique begins to reject correct windows as the program fails to allow for variation that arises from factors such as network conditions and the inaccuracies of our throughput estimation technique.

While the effect of Stage 2 is significant, tuning Stage 1 thresholds is important. Indeed, we see that moderate accuracy can be achieved even at low thresholds for Pearson's when using a *tight* range search, particularly when coupled with Stage 3's fast mode. This performance is due to the combination of Stage 1's ability to filter the database for relevant windows and fast mode's requirement that the offset between two candidate windows in a fingerprint must equal the time difference in the capture file. Moreover, a *tight* range search would be advantageous since it sends fewer candidate windows to Stage 2, thereby reducing the execution time of the program.



(a) Fingerprint similarity due to the *House of Cards* opening credits, depicted in gray (each episode begins with a unique scene before playing the credits).



(b) Fingerprint similarity due to the *House of Cards* end credits, depicted in gray.

Fig. 8. Scenes within *House of Cards* where identical footage results in similar segment sizes.

Across all of our tests, our technique never identified a single window from the *Star Trek Into Darkness* capture with even the lowest of thresholds for Stages 1 and 2. While this issue could be due to a failure of our algorithm, it is more likely due to a faulty capture (e.g. if the Netflix stream did not maintain steady state playback for a sufficient length of time). In the future, we plan on capturing redundant samples over different time periods to better identify and analyze the effects of network conditions on our identification algorithm.

C. Analysis of Identical Footage

When our technique is used to compare the fingerprints themselves, it reveals an interesting phenomenon: identical footage can produce strikingly similar segment sizes across videos. To demonstrate this effect, we conducted a pairwise

comparison between every fingerprint in Dataset B (excluding comparisons between the same episode) and identified sub-regions where our technique could not distinguish between the different videos.

For this test, we set Stage 1 to use a *tight* range search, we set Stage 2 to a minimum *r* of 0.91, and we used Stage 3's fast mode for the final declaration. Of the 4,992 fingerprint comparisons, two match-ups produced indistinguishable sub-regions, depicted in Fig. 8. In both instances, the confusion is a direct result of similar segment sizes produced by either the opening credits or the end credits. Interestingly, Wang [14] notes that Shazam produces false positives under similar scenarios (e.g. when a song samples audio from another track).

VII. Countermeasures

There are several potential countermeasures to our inference method. First, our method is only effective on videos that exhibit a high degree of segment size variation between windows, thus any service that uses either (i) VBR encoding within rigid bitrate constraints or (ii) CBR encoding will be resistant to our attack. That being said, it is doubtful that DASH service providers would be inclined to modify their encoding parameters, as any change to video quality might have a negative effect on user satisfaction.

An alternative, then, is to alter steady state playback so that video segments are not requested at fixed intervals. The naïve approach would be to simply introduce random offsets between requests. Li et al. present a rate adaptation algorithm in [6] that tailors its request rate in order to better gauge the available bandwidth. This technique both eschews the standard form of steady state playback and improves streaming performance.

VIII. Related Work

There is an extensive body of work on traffic classification based solely on data sizes and timing, of which [2], [3], [7], [12], and [15] represent only a small sample. Although our technique is focused on HTTP traffic, as are [3], [7], and [12], we are able to exploit characteristics of DASH that are not present in other HTTP applications, namely that (i) authoritative data sizes can be easily obtained, (ii) objects are requested sequentially, and (iii) timing is predictable during steady state playback.

Most similar to our work is that of Saponas et al. [11], who present a Discrete Fourier Transform (DFT) based approach to identify the content being streamed by a Slingbox Pro. In general, their technique required wireless captures that ranged from 10 to 40 minutes. Although Saponas et al. implored both the signal processing and security communities to confront the privacy issues related to VBR-based streaming, our work demonstrates that these issues have not been resolved.

IX. Conclusion

We have demonstrated that it is possible to identify Netflix videos streamed over encrypted 802.11n connections with greater than 90% accuracy in less than five minutes. In other words, despite encryption at the link layer and application layer (DRM), it is possible to eavesdrop on a user's video stream. Since our technique only relies on the interplay between VBR encoding and steady state playback, we believe that this presents a potential privacy concern for the entire DASH industry, and we echo previous authors' concerns regarding the inherent weakness of VBR to throughput analysis. We have made our code and data available to the research community.

As part of our future work we plan to create additional captures over various network conditions. The larger dataset will enable us to better characterize the accuracy of our algorithm and its ideal thresholds, better analyze the deleterious effects of network conditions on steady state playback, and determine what the effects are, if any, of varying the DASH client bandwidth. Furthermore, to account for windows that are naturally indistinguishable (e.g. opening credits of the same series) we plan to create an additional field in the database to flag these windows. We can then add logic to the algorithm that will continue to capture if the sniffed traffic is potentially one of the flagged windows. Another area of consideration is the development of a threshold for the slow detection method that will determine the minimal number of windows that should be collected in order to make an accurate claim, i.e. if the threshold is not met, then the slow method would declare that the video does not exist in its dataset.

References

[1] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol.18, no. 9, Sep. 1975.

[2] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *ACM CCR*, vol. 36, no. 2, pp. 23-26, Apr. 2006.

[3] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," *ACM CCS*, pp. 605-616, 2012.

[4] GitHub Repository, https://github.com/andrewreed.

[5] ISO/IEC 14496-12:2012, http://standards.iso.org/ittf/PubliclyAvailableStandards/c061988_ISO_IEC_14496-12_2012.zip.

[6] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 719-733, Apr. 2014.

[7] M. Liberatore and B. N. Levine, "Inferring the source of encrypted HTTP connections," *ACM CCS*, pp. 255-263, 2006.

[8] Microsoft Silverlight, https://www.microsoft.com/silverlight.

[9] Sandvine Report: Netflix and YouTube Account for 50% of All North American Fixed Network Data, https://www.sandvine.com/pr/2013/11/11/sandvine-report-netflix-and-youtube-account-for-50-of-all-north-american-fixed-network-data.html.

[10] Sandvine Report: Netflix Dominates (Still), Amazon Instant Video Growing, https://www.sandvine.com/pr/2014/11/20/sandvine-report-netflix-dominates-still-amazon-instant-video-growing.html.

[11] T. S. Saponas, J. Lester, C. Hartung, S. Agarwal, and T. Kohno, "Devices that tell on you: Privacy trends in consumer ubiquitous computing," *USENIX Security Symposium*, pp. 55-70, 2007.

[12] Q. Sun, D. R. Simon, Y. Wang, W. Russell, V. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," *IEEE Symposium on Security and Privacy*, pp. 19-30, 2002.

[13] Tamper Data, https://addons.mozilla.org/en-us/firefox/addon/tamper-data.

[14] A. L. Wang, "An industrial-strength audio search algorithm," *ISMIR*, pp. 7-13, 2003.

[15] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," *JMLR*, pp. 2745-2769, 2006.