MODELING, IDENTIFYING, AND EMULATING DYNAMIC ADAPTIVE STREAMING OVER HTTP

Andrew C. Reed

A thesis submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science.

Chapel Hill
2014

ABSTRACT

Andrew C. Reed: Modeling, Identifying, and Emulating
Dynamic Adaptive Streaming over HTTP
(Under the direction of Jay Aikat)

As HTTP-based streaming video applications have grown to become a major source of Internet traffic, and as the new ISO standard Dynamic Adaptive Streaming over HTTP (DASH) gains industry acceptance, researchers need the ability to both (i) study real-world viewing data and (ii) emulate realistic DASH streams in network experiments. The first effort is complicated by the fact that researchers are often restricted to anonymized, header-only (i.e. payload-truncated) traces. The second effort is difficult since the process of encoding videos for DASH results in numerous large files and since popular videos are subject to restrictive copyright law.

In this thesis we present our work towards developing a model for DASH traffic and show how the model can be applied to identify specific DASH videos in anonymized, header-only traces. We also present our solution for emulating DASH using compact representations of both DASH services (e.g. Netflix and Amazon) and videos.

To my wife, Megan, and sons, Robert and Luke.

ACKNOWLEDGEMENTS

My advisor, Jay Aikat, has been a source of both motivation and inspiration throughout my thesis research. From topic identification through my thesis defense, she has given me her time, energy, and advice every step of the way. Any student would do well to seek her mentorship.

My committee members, Professors Kevin Jeffay and Ketan Mayer-Patel, have provided me with invaluable insight that has undoubtedly raised the quality of my research. Moreover, their contributions to my work began even before I had found a topic, as I had a class with each of them during my first semester of graduate school. Their excellent instruction provided me with a solid foundation for success.

This thesis would not have been possible had it not been for earlier work by Dr. Jeff Terrell. His program, *adudump*, is an outstanding program that should be in every network researcher's toolkit. I am also very thankful to Professor Don Smith for introducing me to both *adudump* and Hadoop; taking his course on datacenters was quite the fortuitous decision.

I thank the Global Environment for Network Innovations (GENI) for their support of my research. With their assistance, I was able to travel to the IEEE International Conference on Network Protocols (ICNP) in order to present my work-in-progress. This was a truly rewarding experience that I will never forget.

I am very grateful to David Ronca, Netflix's Manager of Encoding Tools, for taking the time to respond my email. His reply was exactly the insight that I needed to develop the programs for fingerprinting and profiling Netflix videos. These programs have saved me an enormous amount of time, as I had previously been creating fingerprints and profiles using a very inefficient method.

Above all, I thank my wife, Megan. All of my accomplishments, both in school and in my Army career, are a result of her unwavering love and support. In a world of criticism and doubt, it is a wonderful thing to know that she is in my corner.

Andrew C. Reed

Major, U.S. Army

April 2014

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADU	application data unit
CDN	content distribution network
CPU	central processing unit
DAG	data acquisition and generation
DASH	Dynamic Adaptive Streaming over Hypertext Transfer Protocol
DSL	digital subscriber line
EWMA	exponentially-weighted moving average
GB	gigabyte
Gbps	gigabits per second
GENI	Global Environment for Network Innovations
GHz	gigahertz
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISO	International Organization for Standardization
ISP	Internet service provider
KB	kilobyte
Kbps	kilobits per second
LCS	longest common subsequence
MB	megabyte
Mbps	megabits per second
PC	personal computer
PCAP	packet capture

RTT	round-trip time
TCP	Transmission Control Protocol
URL	uniform resource locator
USC	United States Code
VBR	variable bitrate
vCPU	virtual central processing unit
VM	virtual machine
VoIP	Voice over Internet Protocol
XML	Extensible Markup Language

CHAPTER 1: INTRODUCTION

Yearly reports from Sandvine [18,19,20] 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 accounting for 31.6% of all downstream traffic in the most recent report. As a relatively new method for streaming video, DASH is fertile ground for networking research.

1.1 DASH Overview

A prototypical DASH video is first segmented into equal length time slices and encoded at various bitrates, or quality levels. These video segments are then served from content distribution network (CDN) nodes over HTTP. During playback, DASH clients continually gauge the available bandwidth to determine which quality level of each segment should be requested. If bandwidth is limited, DASH clients will request segments of lesser quality, resulting in smaller application data units (ADUs). When more bandwidth is available, DASH clients will instead request higher quality segments, resulting in larger ADUs.

1.2 Research Problems

While DASH provides industry with an effective means to serve content to users, its use presents network researchers with several challenges. We classify these challenges as they relate to the tasks of *identification* and *experimentation*.

1.2.1 Identification

Since DASH video appears as standard HTTP traffic, it is hard for researchers to isolate and study DASH streams, especially if they are restricted to anonymized, header-only (i.e. payload-truncated) traces. Conversely, this means that DASH traffic has the potential to skew studies of "normal" browsing activity if it is not removed from a trace.

1.2.2 Experimentation

Since DASH segment sizes and buffering activity will vary based on network conditions, captured DASH streams should not be replayed in network experiments. The use of actual video, though, is made difficult by large storage requirements and restrictive copyright law.

1.3 Thesis

Due to DASH's discretization of movies into a series of segments which are fetched sequentially over a TCP connection that is often persistent, we hypothesize that these research problems can be solved by focusing on the ADUs in a DASH connection. Therefore, in an effort to advance the networking community's ability to conduct DASH research, we propose the following thesis:

> Application data unit (ADU)-level analysis of captured Dynamic Adaptive Streaming over HTTP (DASH) streams will enable us to develop a model of DASH traffic that can be leveraged to identify DASH source IP addresses in anonymized, header-only traces. Furthermore, an ADU-centric representation of DASH videos will enable us to design a lightweight, highly-configurable, distributed DASH emulator.

1.4 Contributions

This thesis makes the following contributions.

- A method for identifying specific DASH videos in anonymized, header-only traces that can be leveraged to obtain DASH source IP addresses.
- A method for emulating DASH using compact representations of both DASH services (e.g. Netflix and Amazon) and videos.

1.5 Deliverables

We have developed the following open source, non-copyrighted programs in support of this thesis.

- An offline Hadoop program that analyzes *adudump* logs and reports HTTP connections that match DASH video fingerprints.
- A Java-based DASH emulator that generates realistic, adaptive network traffic using DASH video profiles.
- Java-based utilities that create fingerprints and profiles of Netflix videos for use in the previous two programs.

1.6 Outline

In Chapter 2 we introduce and describe several technologies that our work is built upon. In Chapter 3 we present our ongoing work towards developing a model of DASH traffic that can be leveraged to identify specific videos. In Chapter 4 we present our solution to emulating DASH traffic and describe how researchers can use our solution in DASH client tests and network experiments. Finally, Chapter 5 suggests possible future work and concludes the thesis.

CHAPTER 2: BACKGROUND

2.1 tmix

Our inspiration for a compact representation of DASH videos stems from *tmix*, which Hernández-Campos et al. present in [12] as a means to generate traffic using *a-b-t* connection vectors derived from network traces. The *a-b-t* model, as presented by Hernández-Campos in [11], assumes that the ADU sizes (*a*, *b*) and the inter-exchange times (*t*) are constant for each exchange in a TCP connection between two network endpoints. Thus, *tmix* creates synthetic workloads by replaying the sequence of *a-b* exchanges using dummy payloads that are separated by *t* intervals.

For the purpose of generating realistic DASH traffic, however, the *a-b-t* model is inadequate as any given video segment size *b* and its associated interval *t* are driven by network conditions. More specifically, for each time slice of a video, a DASH client is allowed to choose a video segment from among the available bitrates. Since this decision is based upon the client's running estimate of the available bandwidth during a given playback, it would be inaccurate to simply replay the trace of a DASH stream in an experiment.

Despite its flaws, the *a-b-t* model offers valuable insight into the nature of DASH traffic. Indeed, the *a-b-t* model can be augmented so that each *b* is a *set* of sizes that represent the options for a given video segment, as opposed to a single size that represents the choice that a client made for a particular playback. We therefore use the *a-b-t* model as a starting point for our representations of DASH services and videos.

2.2 adudump

Terrell et al. present *adudump* in [23] as a means to passively monitor the performance of servers. *adudump* aids in this task by analyzing TCP/IP headers and generating *a-b-t* connection vectors for every exchange in a TCP connection, and it does so in one pass. Additionally, *adudump* will accept input from either a PCAP file on disk or directly from a network interface (e.g. Ethernet interface or DAG card). Thus, *adudump* is an ideal preprocessor for our programs as it does not require access to HTTP headers or payloads and its output can be used for offline analysis of HTTP connections.

2.3 Hadoop

Hadoop, originally designed by Yahoo! and now maintained by Apache, is a

> …framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. [4]

In a Hadoop program, *Mappers* iterate through a data set (e.g. a large text file spread across numerous servers) record-by-record (i.e. line-by-line) and emit programmer-defined *key-value* pairs to *Reducers*. All of the *values* for a given *key* will be sent to a single Reducer that then performs a programmer-defined task on the *set* of *values*. These *sets* of *values* are bundled as Java iterators (i.e. collections that provide a method for traversal over their data).

As Hadoop natively supports text files as data sets, it is well-suited for storing and processing *adudump* logs. Additionally, our programs are simplified by Hadoop's *key-value* paradigm: by defining the *key* to be the IP address and port pairing (*local_IP.port* + *remote_IP.port*) of an HTTP connection, we can use Hadoop to consolidate all of the ADUs from a single HTTP connection into an iterator that a Reducer can process in isolation from all other connections.

CHAPTER 3: MODELING AND IDENTIFYING DASH

Although DASH traffic accounts for a significant amount of downstream traffic in North America, it is difficult for researchers to study DASH "in the wild" as privacy considerations often dictate the use of anonymized, header-only traces. In this chapter, we present two models of DASH that, when used together, show promise towards the goal of identifying DASH video without requiring access to HTTP payloads.

3.1 Related Work

Amann et al. [3] identify DASH connections from network traces by inspecting the URL in each HTTP GET in order to find requests for files that end in either an Apple HTTP Live Streaming file type or a Microsoft Smooth Streaming file type (*.ts* and *.ism*, respectively). This approach will not work if a researcher is restricted to IP and TCP headers.

Similar to the problem of classifying anonymized, header-only traces, though, is that of classifying encrypted traffic. Such approaches typically involve the development of a reference data set that can be compared to those characteristics of captured traffic that survive encryption. For instance, White et al. [24] demonstrate that the series of packet lengths from an encrypted VoIP call can be reassembled into phonemes in order to determine the content of the call by leveraging a data set of previously encoded phonemes.

Closer to our intended goal, Saponas et al. [21] present an approach for identifying the video being streamed from a Slingbox by (i) calculating 100ms throughput samples during the stream, (ii) extracting features from the continuous samples with a Discrete Fourier Transform, and (iii) matching the query to a previously captured video trace. Unlike [21], however, DASH videos do not translate well to the model of a continuous signal, as they consist of several encodings (i.e. signals) that can be arbitrarily interleaved.

Instead, our technique is based on the website fingerprinting attack presented by Cai et al. in [5]. This attack matches a website's fingerprint (i.e. a stored trace of packet sizes) to a new trace by calculating the Damerau-Levenshtein edit distance required to transform the fingerprint into the new trace. We find that our task is much simpler than [5] for two reasons: (i) the loading of a webpage is less structured than the sequential playback of a DASH movie and (ii) we intend to trace ADUs as opposed to many thousands of individual packets, thereby reducing the complexity of the comparisons.

3.2 Approach

Our technique for identifying DASH videos in anonymized, header-only traces leverages two models of DASH: (i) a generic model of a DASH connection that can be used to identify *potential* DASH traffic and (ii) a model of an individual DASH video that can be used to identify *specific* content. We describe the models in further detail below.

3.2.1 Model of a DASH Connection

Our model of a generic DASH connection is based on these four properties of prototypical DASH traffic:

1. During steady state playback, outbound ADUs (which represent the HTTP GETs for successive video segments) are sent at regular intervals roughly corresponding to the length of each segment.
2. The sizes of the outbound ADUs exhibit low variance due to the URL naming convention for segments.

3. Since video segments represent a constant length of the movie, the maximum size for any inbound ADU will be limited by (i) a video service's highest offered bitrate, (ii) the duration of each video segment, and (iii) the degree to which a service allows the bitrate to vary (i.e. variable bitrate encoding, or VBR).
4. During steady state playback, new video segments are requested as buffered segments are consumed, and thus the average inbound data rate will be roughly equivalent to the bitrate of the movie.

3.2.1.1 Gathering Baseline Data for the Connection Model

In order to obtain baseline data for the connection model, we used *adudump* to trace the network traffic of a Windows 7 virtual machine (VM) while the VM streamed a single Netflix movie at a time. We captured four movies across four quality levels using the three supported web browsers (Internet Explorer, Firefox, and Chrome), for a total of 48 traces. The lengths of the movies used were 7 min, 17 min, 46 min, and 110 min. In addition to the 48 Netflix-only traces, we traced over 45 hours' worth of household Internet activity containing no Netflix traffic.

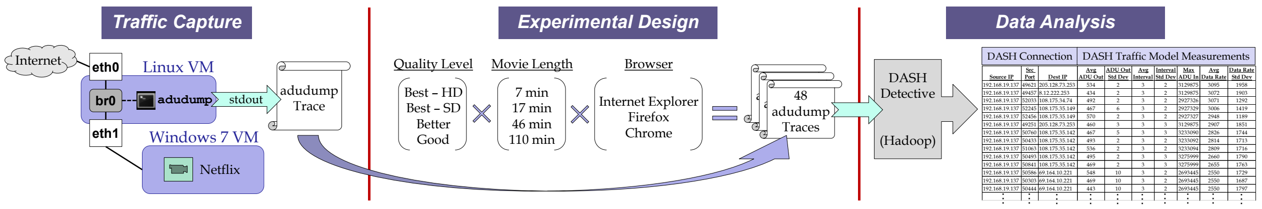
3.2.1.2 Data Analysis

We used a Hadoop program to calculate the statistics relevant to the four DASH traffic properties for each DASH connection in the baseline traces. These statistics are as follows:

- **Duration.** Length of the connection.
- **Average ADU Out.** The average size of all ADUs sent from the local IP and port to the remote HTTP server.
- **ADU Out Standard Deviation.** Standard deviation for the above.

- **Average Interval.** The average time between ADUs sent from the local IP and port to the remote HTTP server.
- **Interval Standard Deviation.** Standard deviation for the above.
- **Max ADU In.** The largest ADU sent by the remote HTTP server to the local IP and port.
- **Average Data Rate.** The average data rate for the ADUs received from the remote HTTP server (calculated using 10 second interval averages).
- **Data Rate Standard Deviation.** The standard deviation of the 10 second interval averages.

Figure 3.1 depicts our capture and analysis workflow.



DASH Connection			DASH Traffic Model Measurements						
Source IP	Src Port	Dest IP	Avg ADU Out	ADU Out Std Dev	Avg Interval	Interval Std Dev	Max ADU In	Avg Data Rate	Data Rate Std Dev
192.168.19.137	49621	205.128.73.253	534	2	3	2	3129875	3095	1958
192.168.19.137	49457	8.12.222.253	434	2	3	2	3129875	3072	1903
192.168.19.137	52033	108.175.34.74	492	2	3	2	2927326	3071	1292
192.168.19.137	52245	108.175.35.149	467	6	3	2	2927329	3006	1419
192.168.19.137	52456	108.175.35.149	570	2	3	2	2927327	2948	1189
192.168.19.137	49251	205.128.73.253	460	3	3	3	3129875	2907	1851
192.168.19.137	50760	108.175.35.142	467	5	3	3	3233090	2826	1744
192.168.19.137	50433	108.175.35.142	493	2	3	2	3233092	2814	1713
192.168.19.137	51063	108.175.35.142	536	2	3	2	3233094	2809	1716
192.168.19.137	50493	108.175.35.142	495	2	3	3	3275999	2660	1790
192.168.19.137	50841	108.175.35.142	469	2	3	3	3275999	2655	1763
192.168.19.137	50586	69.164.10.221	548	10	3	2	2693445	2550	1729
192.168.19.137	50303	69.164.10.221	469	10	3	2	2693445	2550	1687
192.168.19.137	50444	69.164.10.221	443	10	3	2	2693445	2550	1797
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 3.1: Capture and analysis workflow.

3.2.1.3 Measured Statistics for the Connection Model

Table 3.1 lists the measured statistics for the Netflix streams. As an indication of the connection model's strength, the baseline streams could have been identified, with no false positives from the non-Netflix traffic, by using a filter based on the ranges in Table 3.1.

Statistic	Min	Max
Average ADU Out (B)	433	570
ADU Out Standard Deviation (B)	1	10
Average Interval (s)	1	4
Interval Standard Deviation (s)	2	3
Max ADU In (B)	481,107	3,275,999
Average Data Rate (Kb/s)	469	3,095
Data Rate Standard Deviation (Kb/s)	174	2,145

Table 3.1: Baseline statistics for Netflix's DASH connection model.

3.2.2 Model of a DASH Video

Figure 3.2 depicts the video segment sizes for three separate viewings of the same video at the same quality level. From Figure 3.2 we see that the sizes of the underlying video segments remain constant across multiple playbacks of a video. Given this observation, we model a DASH video as the sequence that is formed by taking the video segment sizes for each bitrate encoding and interleaving them into a single, sequential ordering (which we refer to as a *fingerprint*). For example, a video with three Segments across two Bitrates would have a fingerprint of the form

$$\{\ \mathbf{S_1B_1}.\text{size},\ \mathbf{S_1B_2}.\text{size},\ \mathbf{S_2B_1}.\text{size},\ \mathbf{S_2B_2}.\text{size},\ \mathbf{S_3B_1}.\text{size},\ \mathbf{S_3B_2}.\text{size}\ \}.$$

Thus, the sizes of the video segments for *any* straight-through playback (i.e. no "rewinding" or "re-downloading") of a DASH video will be a subsequence of the video's fingerprint.

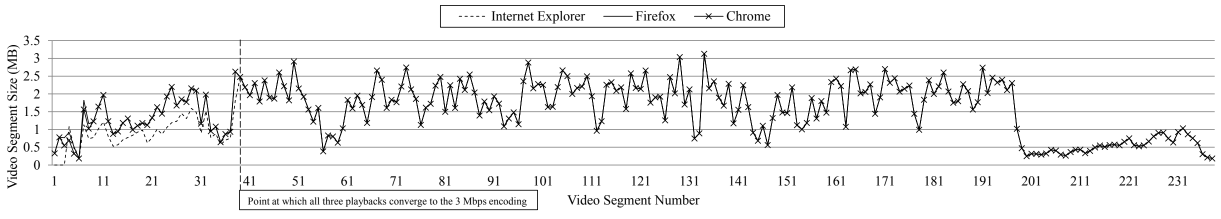


Figure 3.2: Inbound ADU (i.e. video segment) sizes for three separate playbacks of *Legend of the Boneknapper Dragon*.

These fingerprints can be created for Netflix videos with *Fingerprinter*, a Java-based utility that we make available at [9]. *Fingerprinter* first takes the XML manifest of a Netflix video and parses it to find the URLs for each bitrate encoding of a video. *Fingerprinter* then downloads the header of each encoding, from which it can calculate the sequence of segment sizes for each bitrate. These sequences are then interleaved to form a fingerprint that is output to stdout. With each fingerprint, *Fingerprinter* includes both the title of the video and the ordered list of available bitrates (in Kbps), thereby allowing a researcher to de-interleave the fingerprint, if needed.

3.3 Implementation

To identify DASH videos from anonymized, header-only traces, we use *dashid*, an open source, non-copyrighted Hadoop program which we make available at [7]. *dashid* takes both an *adudump* trace and a data set of DASH video fingerprints as input and returns a list of the HTTP connections in the trace that match a fingerprint. In this section, we provide a brief overview of *dashid* and we describe the process by which it identifies videos.

3.3.1 *dashid* Overview

As mentioned in Section 2.3, we define each key in *dashid* to be the concatenation of an *adudump* record's *source_IP.port* field and the *remote_IP.port* field so that each reduce task will receive all of the ADUs for an HTTP connection in a single iterator. We augment this key with a timestamp field so that Hadoop can perform a secondary sort on the keys, thereby ensuring that each reduce task will receive the ADUs of a connection pre-sorted in chronological order.

3.3.2 Identification Steps

3.3.2.1 Step 1: Filtering HTTP Connections

For every HTTP connection in the *adudump* trace, the reduce task performs one pass over the iterator of ADUs and (i) calculates the statistics listed in Section 3.2.1.2 and (ii) builds a chronologically-ordered list of the inbound ADU sizes. Once the iterator has been processed, the reduce task checks the measured statistics to verify if the HTTP connection fits the DASH connection model. If the connection fits the model, then the list of inbound ADU sizes is processed by the next step; otherwise, the list is discarded and the HTTP connection is ignored.

A limitation of this approach is that it does not account for the possibility that a DASH client may choose to either (i) pipeline GET requests via HTTP pipelining or (ii) aggregate segments using byte range requests. Either of these methods will result in larger than anticipated inbound ADU sizes, which might cause *dashid* to incorrectly identify a DASH stream as a non-DASH connection. We have observed neither of these behaviors in Microsoft Silverlight.

3.3.2.2 Step 2: Matching to a Video Fingerprint

For each HTTP connection that fits the DASH connection model, *dashid* compares the list of inbound ADU sizes to each fingerprint in the supplied data set and calculates the longest common subsequence (LCS) between the two. The fingerprint that yields the "longest" LCS is reported in *dashid's* output as a match for the HTTP connection. A minimum threshold for the LCS can be established to minimize the occurrence of incorrect matches.

3.4 Identifying CDN IP Addresses

The task of identifying CDN IP addresses consists only of compiling a list of all the HTTP servers from *dashid's* output, as these servers represent streaming video content servers. A researcher can then mark all connections in the original *adudump* trace that involved a server from the compiled list as a video stream.

3.5 Ongoing Work

As of this writing, we have yet to test *dashid*'s performance and accuracy against a large trace. From our observations, we see that on-campus Netflix streams will connect to a limited subset of IP addresses. Thus, we intend to trace traffic on the campus-wide link and anonymize the trace with a key for which we know the anonymized versions of these Netflix CDN IP addresses. We can then run *dashid* on the trace and verify that the technique described in Section 3.4 yields IP addresses from our ground truth list.

CHAPTER 4: EMULATING DASH

With the continuing popularity of DASH-based services such as Netflix, even the smallest enhancements to a particular DASH client's networking components (e.g. its *buffering*, *bandwidth estimation*, *bitrate selection*, and *CDN selection* algorithms) could improve the viewing experience for a significant number of viewers. Ideally, these modifications should be tested against a broad selection of videos under a variety of network conditions to ensure that any changes will improve playback for the majority of DASH streams. Moreover, experiments that seek to study the effects of DASH traffic on network performance, particularly bandwidth-constrained home networks, should include a variety of videos, as the load that a stream places on the network will largely depend on the video's encoding parameters.

To illustrate the role of video data in a DASH experiment, Figure 4.1 depicts the video bitrates (as 4-minute moving averages) for the Netflix version of *The Avengers* and both the Netflix and Amazon versions of *The Hunger Games*. We present the data for the highest quality encoding of each video, which for Netflix is 3 Mb/s and for Amazon is 6 Mb/s.

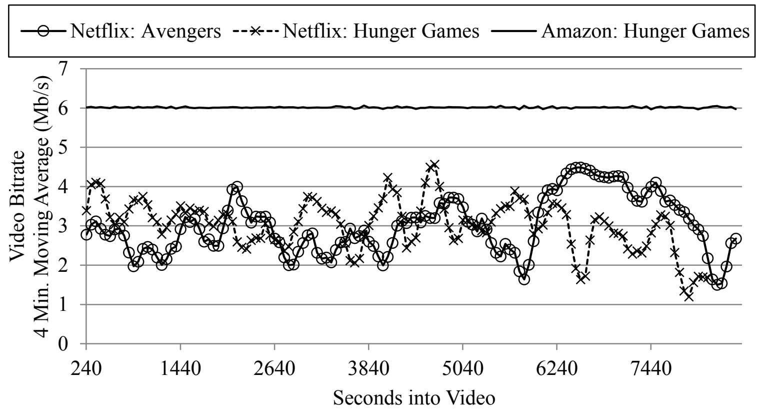


Figure 4.1: A comparison of the average bitrate between different videos from the same DASH service and between the same video on different DASH services.

By comparing videos from differing services, we see that a service's encoding parameters will drive the decisions that are made when designing its streaming client. For instance, since Netflix encodes its videos at a variable bitrate (VBR), its client requires almost 66% more available bandwidth than a target bitrate in order to stream at that quality level [13] (notice that the final action scene from *The Avengers* results in a 14 minute period where the average bitrate exceeds 4 Mb/s). Although we do not have data for the Amazon client, we expect that it would require significantly less headroom in order to stream a given bitrate, as its videos deviate little from their target bitrates.

From Figure 4.1 we also see that the bandwidth estimation logic of a DASH client will be presented with an added challenge if it must compete with a highly variable stream. Should a client begin streaming while a competing stream of *The Avengers* is at the dip just prior to the final action scene, the new client might select an unsustainably high bitrate. This sort of interaction would not be captured in experiments which use videos encoded at constant bitrates.

Although there are clear benefits to using a variety of videos in both DASH client tests and network experiments, there are two significant problems with the use of actual video:

- **Copyright law.** While the "fair use" clause of U.S. copyright law (17 USC § 107) permits the reproduction of material for research purposes, it would not be deemed "fair use" for a researcher to make the DASH encodings of a copyrighted video available wholesale to the networking community. Thus, if other researchers wanted to reproduce the results of an experiment that used a copyrighted video, they would have to re-encode the same version of the source material according to the exact settings used in the original experiment. For this reason, the overwhelming majority of content that users *actually watch* is unlikely to be used in DASH experiments.

- **Storage requirements.** Encoding videos for DASH requires substantial amounts of disk space. For instance, the combined total for all of the quality levels of the Netflix versions of *The Avengers* and *The Hunger Games* is approximately 21 GB. In storage-constrained virtual laboratories such as the Global Environment for Network Innovations (GENI) [10], even a small data set of DASH videos would be infeasible to use in an experiment.

To overcome these problems, we present a novel technique to reproduce the segment sizes of DASH videos using a format that requires minimal storage and which does not violate copyright restrictions. The main contributions of this chapter are:

- We present a method to represent a DASH service and its video library using a combination of small text files (which we refer to as *profiles*) that can be served as static files on a standard web server. These profiles enable a researcher to modify the settings of a DASH service per experiment by simply changing a line in the service-wide profile (e.g. define new bitrate encodings for all movies).
- We present an open source DASH client emulator called *dashem* [6] that uses these profiles to generate realistic DASH traffic.
- Included with *dashem* are a service-wide profile for both Netflix and Amazon Instant Video, as well as video profiles for ten Netflix videos and two Amazon videos.

We limit the scope of our technique to replicating those DASH services that use fixed-duration video segments and fixed-duration, constant-bitrate audio segments (e.g. Netflix and Amazon). Additionally, we do not attempt to replicate the *exact* buffering or rate-selection algorithms of a specific client. Instead, we have kept *dashem's* code minimal, while still implementing basic functionality, so that other researchers can easily modify it in order to test their own algorithms.

The rest of the chapter is organized as follows. Section 4.1 details how we reduce a DASH service and its videos to a set of profiles and Section 4.2 describes how *dashem* uses these profiles for traffic generation. Section 4.3 presents the results of tests designed to validate our technique and to gauge *dashem's* scalability. Section 4.4 details a sample experiment and Section 4.5 further discusses how experimenters can use *dashem* for their research. Section 4.6 reviews related work and Section 4.7 concludes the chapter.

4.1 Approach

4.1.1 DASH Traffic Observations

Our method of representing a DASH service and its video library as a collection of profiles is based on these two observations of DASH traffic from Netflix and Amazon:

- *We find that repeated playbacks of a video at the same bitrate encoding yield identically-sized, identically-ordered application data units (ADUs) received by the client*. Notice in Figure 3.2 that the ADU-level traces of separate playbacks of the Netflix video *Legend of the Boneknapper Dragon* are indistinguishable once they converge to the 3 Mbps encoding. This is a direct result of the fact that these services' DASH streams are comprised of individual video segments that are requested sequentially by the client.
- *We find that the segment sizes for a lower bitrate X are well-approximated by multiplying the segment sizes of the highest bitrate Y by the constant ratio X:Y*. Figure 4.2 compares the actual segment sizes of the 1750 and 560 Kbps encodings of *Legend of the Boneknapper Dragon* to their approximated sizes if the segment sizes of the 3000 Kbps encoding are multiplied by a constant ratio. For the 1750 Kbps encoding, the estimated sizes are only off by an average of 61.4 KB per segment. Moreover, the entire sum of

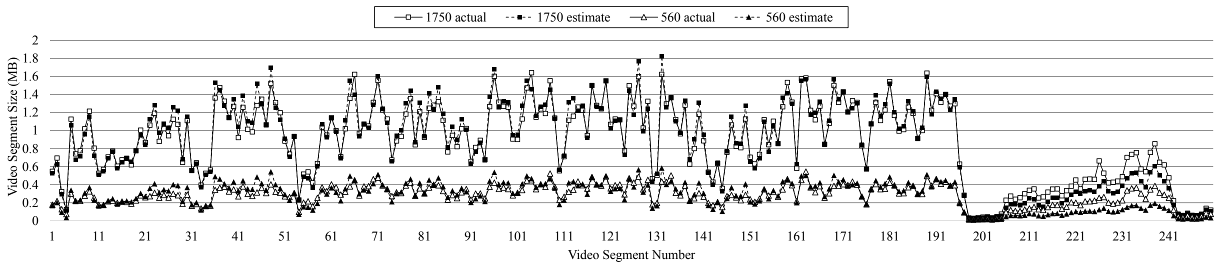


Figure 4.2: Actual segment sizes vs. estimated sizes for the 1750 and 560 Kbps encodings of *Legend of the Boneknapper Dragon*.

data generated by the 1750 Kbps estimate is only off by 0.7% (218.3 MB estimate / 219.9 MB actual).

As Netflix and Amazon both use Microsoft Silverlight for browser-based streaming, we do not generalize our current findings beyond these two services.

4.1.2 Profiles

Given our observations, we model an individual video as the sequence of video segment sizes for the highest bitrate encoding, from which the lower bitrate encodings can be approximated using ratios that are consistent throughout an entire service. We also note that, as a result of our scope from Section 1, all other DASH traffic properties (e.g. video segment duration, audio segment duration, and audio segment size) will be constant for an entire service and thus we do not model these separately for each individual video. We therefore use the following profiles to represent a DASH service and its video library.

4.1.2.1 Service-Wide Profile

Figure 4.3 depicts our service-wide profile for Netflix. Its rows are as follows:

- **Row 1: Video Profile Bitrate.** This indicates the bitrate (in Kbps) at which all of the video profiles for a service were derived.

- **Row 2: Encoding Levels.** These indicate, in descending order, the bitrate encoding levels of a DASH service as percentages relative to the value from row 1. For Netflix, this equates to 3000, 2350, 1750, 1050, 750, 560, 375, and 235 Kbps.
- **Row 3: Video Segment Duration.** The length of video (in seconds) contained in each video segment.
- **Row 4: Video-to-Audio Ratio.** The number of video segments per audio segment. For Netflix, this equates to 16 seconds of video per audio segment.
- **Row 5: Audio Segment Size.** The average size (in bytes) per audio segment. For Netflix, this equates to an audio bitrate of 64 Kbps, which we have calculated as an average across the audio segment sizes that we have observed, which range from approximately 131,500-135,800 bytes.

```
3000
100 78.333 58.333 35 25 18.666 12.5 7.8333
4
4
135100
```

Figure 4.3: Service-wide profile for Netflix.

Note that rows 1 and 3 are dictated by a service's library of video profiles. A researcher is free, however, to change the values of any other row. For instance, prepending the value 200 to row 2 of the Netflix service profile will "create" a 6000 Kbps encoding for every video in the Netflix library. Similarly, removing the value 100 will "delete" every 3000 Kbps encoding.

4.1.2.2 Video Profiles

Video profiles are text files that list the sequence of video segment sizes for the highest bitrate encoding, resulting in files of almost negligible size. For example, the Netflix profiles for

The Avengers and *The Hunger Games*, at 16.6 KB and 15.9 KB apiece, require significantly less storage than the full encodings of these two videos, which totaled to 21 GB. We provide a utility for creating profiles of videos at [17]. We also include 12 sample profiles (10 Netflix, 2 Amazon) with *dashem* at [6].

4.1.2.3 Profile Design Considerations

While we could have designed our profiles to store all segment sizes for all bitrates (or even to use a video's manifest directly whenever the manifest contains segment size information), it is our opinion that the benefit of absolute segment accuracy across a service's available bitrates does not outweigh the ability to create new bitrates for an experiment. In other words, since *dashem* recreates the segment sizes of the various bitrates based on row 2 of the *service-wide profile*, a researcher gains the ability to make any number of new bitrates available for an experiment. These will, of course, be approximations of what the bitrate encodings *might* have been, but we feel that the relatively minor loss in per-segment accuracy is worth the added extensibility.

4.2 Implementation

We generate traffic from profiles with *dashem*, an open source, non-copyrighted DASH client emulator that we developed in Java and which runs from a command line. In this section, we describe the prototypical experimental design and we provide a brief overview of how *dashem* works.

4.2.1 Prototypical Experimental Design

dashem allows a researcher to design an experiment that replicates a standard DASH architecture consisting of (i) a central server whose primary function is to redirect DASH clients to (ii) content distribution networks (CDNs) which then provide the actual content.

4.2.1.1 CDNs

In our prototypical design, a single HTTP server is used as a stand-in for an entire CDN. Here, the internal behavior of the CDN is *not* being modeled. Instead, a traffic controller is used to emulate the aggregate characteristics of a client's connection to the CDN, thereby allowing a researcher to test a CDN-selection strategy. Additionally, each HTTP server hosts only a single 13MB dummy file. Section 4.2.2.2.1 details how *dashem* generates traffic from this single file.

dashem allows CDNs to be grouped into notional geographic regions using text files called *CDN lists*. We use *notional* in this context as these "geographic regions" are simply divisions in the experimental design that allow a researcher to test *dashem* under a variety of network conditions based on regional statistics. Experimenters are free to define any number of regions according to whichever naming scheme suits their needs.

4.2.1.2 Central Server

The central server is an HTTP server that hosts the *service-wide profile*, the *video profiles*, and the *CDN lists* for a DASH service.

4.2.1.3 Example Experimental Design

Figure 4.4 depicts an example experimental design that uses all of the components of the prototypical design. It is similar to how a researcher would visually arrange an experiment in a GENI test bed using the Flack GUI. In this example, the instances of *dashem* being run on the *North Region Clients* PC will be redirected by the central server to the *North Region* CDNs and thus their DASH streams will be shaped by the traffic controller between them. Likewise, the instances of *dashem* in the *South Region* will request segments from the *South Region* CDNs and be subjected to different network conditions than the "viewers up north".

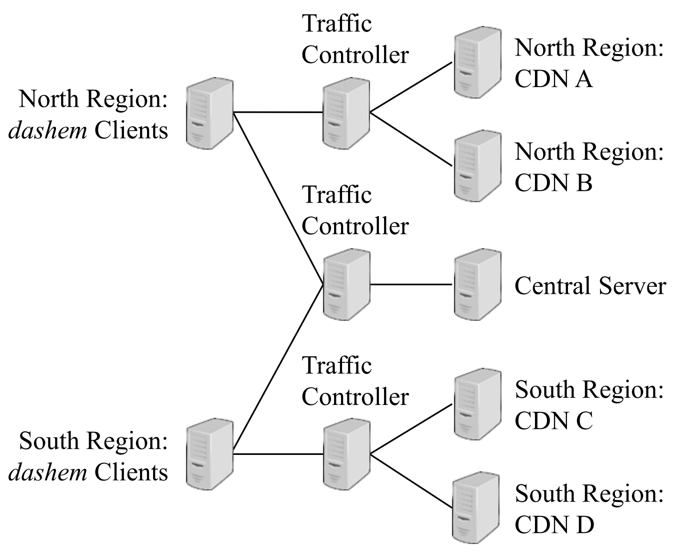


Figure 4.4: Example *dashem* experimental design.

4.2.2 *dashem* Overview

4.2.2.1 Startup

dashem takes the following command line arguments:

- **Central server address.** Either the IP address or domain name of the central server.
- **Service.** The DASH service to use for the given instance. This allows a single central server to host profiles and CDN lists for any number of services.
- **Region.** The notional geographic region for the given instance.
- **Video Title.** The name of the video profile to stream.
- **Account Name.** Account names are used in logs and can be used by an experimenter to create unique identifiers for each instance of *dashem*.

An instance of *dashem* begins by connecting to the central server and downloading the (i) *service-wide profile* for the indicated service, (ii) the *CDN list* for the indicated region, and (iii) the *video profile* for the indicated video title. The CDNs are ranked using a random shuffle that is seeded by the account name, similar to how Netflix ranks CDNs [1]. *dashem* then starts a streaming thread and a watching thread which communicate via a synchronized buffer that tracks the number of buffered video segments.

4.2.2.2 Threads

4.2.2.2.1 Streaming

The streaming thread connects to the highest ranked CDN and begins to request audio and video segments, with audio segments conforming to rows 4 and 5 of the *service-wide profile*. Video segment requests start with the lowest bitrate and are then based on a continual estimate of the available bandwidth. Bandwidth is sampled with each video request based on the time taken to download the size of the segment, and the average is taken as an exponentially-weighted moving average (EWMA) across all bandwidth samples. Note that the current bandwidth estimate is multiplied by a cushioning factor of 60%, similar to the Netflix client [13].

For each new video segment request, the highest bitrate encoding supported by the "cushioned" estimate of the available bandwidth is requested. The choice of bitrate encodings is determined from rows 1 and 2 of the *service-wide profile*. The streaming thread only increments the buffer upon receipt of a video segment; audio segments are effectively "extra work" that must be requested prior to their correlated video segments. The streaming thread will continue to request segments until the buffer is full, at which point it enters a steady state where new segments are requested as buffered segments are consumed by the watching thread. *dashem* uses the buffer's *wait()* and *notifyAll()* methods to ensure that the streaming thread runs only when necessary.

At *dashem's* core is its method for generating dummy traffic. As mentioned in Section 4.2.1.1, each CDN hosts only a single dummy file. Thus, to generate dummy segments from this file, *dashem* leverages HTTP/1.1 range requests. For each new segment request, the streaming thread sends an HTTP GET for the dummy file with the *Range* header set to a range of bytes equivalent to the size of the next segment, with the size being calculated as described in Section 4.1.1.

4.2.2.2.2 Watching

The watching thread serves to consume video segments from the buffer and to log playback status. Playback starts once the buffer has reached a minimum fill and playback will continue as long as the watching thread is able to consume a segment from the buffer. After successfully decrementing the buffer, the watching thread sleeps for the duration of a video segment, as dictated by row 3 of the *service-wide profile*. If the buffer is empty when the watching thread attempts to consume a segment, the watching thread will log that playback has paused and will then call the buffer's *wait()* method. Playback will resume if the buffer reaches the minimum fill, or if all segments have been received.

4.2.2.3 Tuning *dashem*

The *length* of the buffer, *minimum fill* to start/resume playback, bandwidth estimate EWMA *smoothing constant*, and bandwidth estimate *cushioning factor* are set to the default values of 240 seconds, 12.5%, 0.125, and 60%, respectively. Each of these is a constant in *dashem* and can be changed by an experimenter to alter *dashem's* behavior.

4.2.2.4 *dashem* Design Considerations

dashem lacks many of the "real world" strategies that a true DASH client would exhibit, for example:

- **Bitrate selection.** Akhshabi et al. [2] note that the Smooth Streaming player makes small bitrate transitions and they hypothesize that this is to provide a better viewing experience by avoiding drastic changes in video quality.

- **CDN selection.** Adhikari et al. [1] report that the Netflix client will switch CDNs whenever the current CDN can no longer support the lowest video bitrate. *dashem*, on the other hand, makes a CDN selection and sticks with it, regardless of network conditions.

The fact that *dashem* lacks these capabilities (and more) is a design decision on our part. It is our opinion that *dashem* implements *enough* functionality so as to demonstrate the effectiveness of our profiles. Additional capabilities would likely complicate the *dashem* source code, making it difficult for other researchers to implement their own strategies in *dashem*.

4.3 Evaluation

For the tests described in this section, we used the Netflix service-wide profile and the *Legend of the Boneknapper Dragon* video profile. We created two Ubuntu virtual machines (VMs) in VMware Workstation on a PC with a quad core Intel Core i7 1.73 GHz CPU and 14 GB of memory:

- **VM1.** On this VM we ran the instances of *dashem*. It was allocated 6 vCPUs (i.e. 6 of the 8 threads available on the i7) and 4 GB of memory.
- **VM2.** On this VM we installed Apache HTTP Server and configured it to act as both a *dashem* central server and a CDN by creating a single *CDN list* with VM2's IP address as the sole entry. This VM was allocated 2 vCPUs and 4 GB of memory.

Both VMs were connected to the same virtual network. Round-trip time (RTT) on this virtual network was less than 1 millisecond. A larger, more realistic RTT could have been emulated by adding delay with a tool such as *dummynet* running on either VM2 or on a bridge acting as a traffic controller, as depicted in Figure 4.4. We did not, however, add delay in this manner as we used alternative means to shape bandwidth, which we describe in the following subsections.

4.3.1 Validation

In this section, we do not conduct a head-to-head comparison of *dashem* against an actual DASH client since, as stated in Section 4.2.2.4, it is not our goal to produce an emulator that replicates the exact functionality of a specific client. Instead, we aim to provide the community with a tool that can be relied upon to provide the same feedback as a real DASH client and, as such, we validate our technique by demonstrating that *dashem* produces realistic DASH traffic from profiles.

For this test, we ran a single instance of *dashem* while varying the link rate of VM2's network adapter using the following rates: 5400, 3180, 1900, and 1000 Kbps. In order to enhance the visual distinction between bitrates in our results, we used a version of the Netflix service-wide profile that was reduced to the 3000, 1750, 1050, and 560 Kbps encodings by removing values from row 2 of the profile. Thus, the link rates were selected as they will each yield "cushioned" bandwidth estimates that are slightly above the available bitrates. We did not add delay to the link as we wanted to ensure that the link rates would yield comparable data transfer rates between VM1 and VM2.

Figure 4.5 depicts the video segment sizes generated by *dashem* during the test playback, as well as the actual segment sizes for the 3000, 1750, 1050, and 560 Kbps encodings. Vertical bars denote when the link rate changed and grey arrows denote when *dashem* responded by switching to a new bitrate. Compared to the actual sizes, the sizes produced by *dashem* are off by an average of 41.7 KB per segment, and the entire sum of data generated by the test is off by

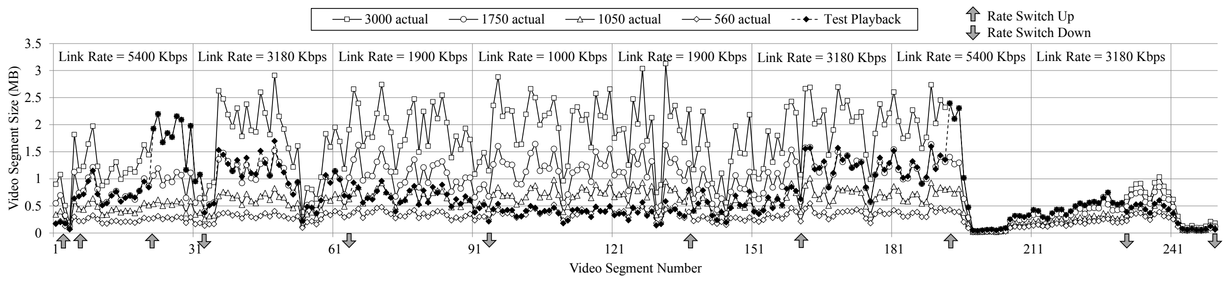


Figure 4.5: Video segment sizes generated by *dashem*.

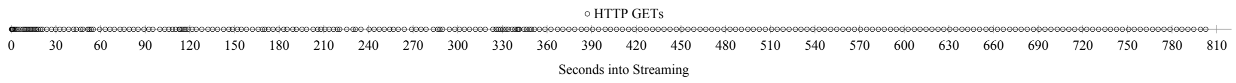


Figure 4.6: Timeline of video segment requests.

943.4 KB (0.53%). Notice that *dashem* accurately reproduces the segment sizes for the 3000 Kbps encoding, a direct result of our method for generating video profiles.

Figure 4.6 depicts the times at which *dashem* requested video segments. As expected, there is buffering activity at the beginning of the test when the buffer is empty, and there is additional buffering throughout the first half of the test as we lower the link rate. In the latter half of the test, during which time we steadily increase the link rate, video segment requests are sent at 4 second intervals, indicative of steady state playback.

These results demonstrate that (i) *dashem* responds to varying network conditions as if it were a “real” DASH client and (ii) that the use of our profiles allows *dashem* to reproduce the segment sizes of a DASH video’s various encodings with a high degree of accuracy. Based on these results, we believe that researchers can feel confident in using *dashem* to implement and test new DASH networking strategies.

4.3.2 Scalability

To gauge *dashem’s* ability to support large scale experiments, we measured VM1’s resource utilization while we conducted a test with 80 concurrent instances streaming the 3000 Kbps encoding. For this test, the link rate of both VM1's and VM2's network adapters were set to 1 Gbps, and a maximum throughput of approximately 900 Mbps was measured by running *iperf* for 90 seconds. Here, rather than emulate RTT on the link between VM1 and VM2, each instance of *dashem* was started with an instance of *trickle* [8], a program that allows an experimenter to limit the upload and download bandwidth consumption of *dashem*. Such a program would be useful in a real experiment to shape the available bandwidth *per instance* of *dashem* according to the average (video-watching) household bandwidth for a given geographic region. Thus, our results indicate the resource utilization that a researcher could expect if an experiment were to use both *dashem* and *trickle*.

For this test, *trickle* was set to 6 Mbps for both upload and download bandwidth, thereby allowing each instance of *dashem* to maintain the 3000 Kbps bitrate. In order to ensure that each instance reached a steady state as soon as possible, *dashem's* buffer length was set to 12 seconds and row 2 of the Netflix service-wide profile was reduced to just the 3000 Kbps bitrate.

To obtain resource utilization statistics, we started the Linux monitoring tool *sar*, part of *sysstat* [22], on VM1 and set it to poll the system every second. We allowed *sar* to run for 20 seconds and then began one instance of *dashem* every 3 seconds. Figure 4.7 depicts VM1's inbound bandwidth, CPU, and memory utilization throughout the test. The results for inbound bandwidth and CPU utilization represent 4-second moving averages of the data reported by *sar*, whereas the results for memory utilization represent the actual 1-second polls. The fact that Figure 4.7 shows a sustained inbound bandwidth of well over 240 Mbps (i.e. 80 instances at 3000 Kbps each) is to be expected, as periods of *Legend of the Boneknapper Dragon* reach sustained bitrates of over 4 Mbps.

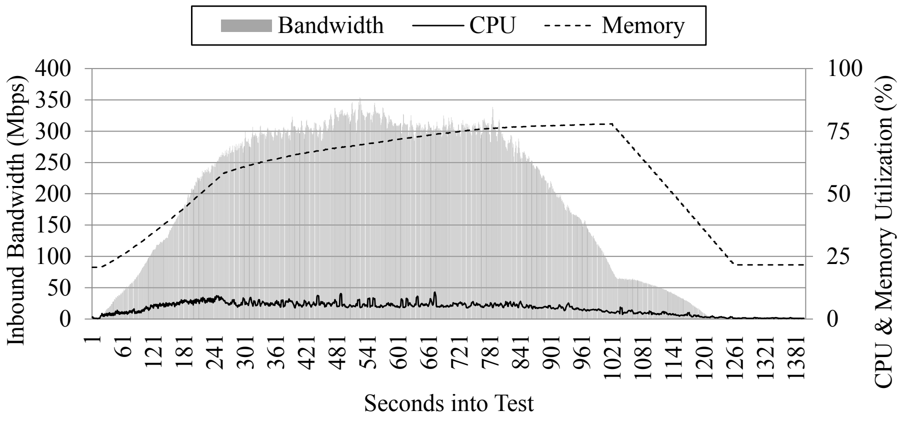


Figure 4.7: Resource utilization for 80 *dashem* instances.

We believe that these results, which depict very light CPU utilization and modest memory usage, show that *dashem* is efficient enough to support large scale experiments running on average hardware platforms.

4.4 Sample Experiment

To illustrate how *dashem* can be used for DASH research, we present the results of an experiment that investigates the "downward spiral effect" described by Huang et al. in [13]. In this experiment, the scenario being modeled is that of a North American household streaming two different videos simultaneously.

4.4.1 Setup

For this experiment we used the same VMs from Section 4 and set the link rate from VM1 to VM2 to 1 Mb/s, and from VM2 to VM1 to 6 Mb/s. These rates were chosen as they are the upload and download bandwidths for a mid-tier DSL plan from a large North American ISP. We also added 80ms delay on VM2 via the Linux command *tc* in order to emulate a realistic RTT. Once configured, we ran *iperf* for 90 seconds in both directions, which indicated that the maximum throughputs were 960 Kb/s upstream (VM1 to VM2) and 5700 Kb/s downstream (VM2 to VM1).

We used a script to start a playback of *The Hunger Games* followed 4 minutes later by a playback of an episode of *Curious George* (which has a duration of 24 minutes). Once *Curious George* completed, *Hunger Games* continued to stream for another 7 minutes before it was terminated. This script was executed twice: once using the Netflix service-wide profile and the Netflix versions of the videos, and once using the Amazon service-wide profile and Amazon versions of the videos. Throughout the remainder of this section, we refer to these tests according to the source data used (i.e. *Netflix test* and *Amazon test*).

Since the goal of this experiment was to investigate the effect that video data has on a DASH client's ability to estimate available bandwidth and to make bitrate selections, we wanted to ensure that *dashem* was able to choose from the same bitrates in both of the tests. This was accomplished by (i) modifying row 2 of the Amazon service-wide profile to match Netflix's video bitrates and by (ii) halving the value of row 5 of the Amazon service-wide profile to match Netflix's audio bitrate (the default Amazon profile is set to an audio bitrate of 128 Kbps). Figure 4.8 depicts this modified version of the Amazon service-wide profile.

```
6000
50 39.167 29.167 17.5 12.5 9.333 6.25 3.917
2
1
16545
```

Figure 4.8: Modified service-wide profile for Amazon.

Given (i) *iperf's* estimate of the total downstream bandwidth, (ii) the available video bitrates, and (iii) the fact that *dashem* uses a default throughput cushion of 60%, the optimal split between the two video streams occurs when both clients select the 1750 Kbps encoding. The second best split occurs when one client selects 2350 Kbps and the other selects 1050 Kbps.

4.4.2 Results

Figures 4.9 and 4.10 depict the bitrate selections made by *dashem* for each test. Notice that, while the Netflix test managed to reach the second best split towards the end of *Curious George*, neither test resulted in the optimal split. Furthermore, neither of the *Hunger Games* streams returned to 3000 Kbps after *Curious George* completed. Instead, the Netflix test alternated between the 1750 and 2350 Kbps encodings and the Amazon test remained at 1050 Kbps.

Based on the results of [13], we see that this suboptimal behavior is attributable to poor bandwidth estimation during steady state playback, as each video segment request will restart from TCP slow start. To confirm this effect, Figure 4.11 shows the bandwidth estimates reported by the *Hunger Games* clients after *Curious George* has ended. As expected, bandwidth estimates are correlated with video segment sizes, with only the largest of the Netflix segments yielding relatively accurate estimates.

It is clear, then, that the streams in the Amazon test had collapsed under the "downward spiral effect". Furthermore, due to Amazon's choice of a short segment duration (i.e. small segments) and lack of segment size variability, the bandwidth estimates for the *Hunger Games* playback consistently underestimated the available bandwidth, thereby preventing *dashem* from selecting a higher quality once *Curious George* had ended. Conversely, we see that Netflix's choice of a longer segment duration (i.e. larger segments) contributed to more accurate bandwidth estimates; however, Netflix's reliance on VBR encoding resulted in fluctuating periods of small and large segments which led to oscillating bandwidth estimates.

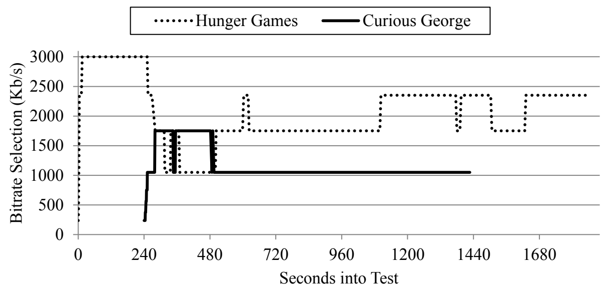


Figure 4.9: Bitrate selections for the Netflix test.

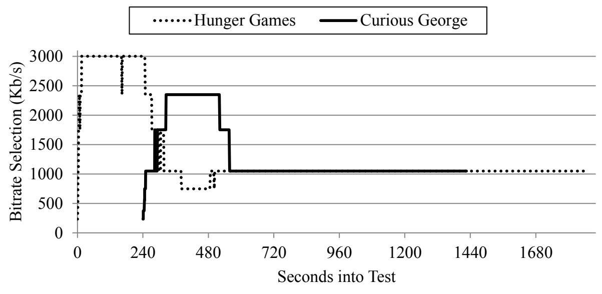


Figure 4.10: Bitrate selections for the Amazon test.

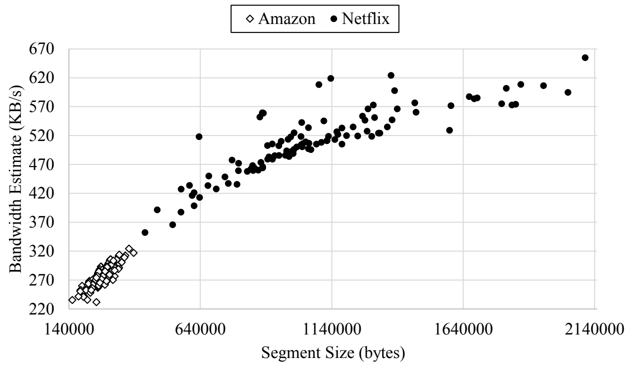


Figure 4.11: *Hunger Games* bandwidth estimates (as a function of video segment size) following the completion of *Curious George*.

4.5 Discussion

Given the results of Section 4.4, it should be apparent that the experiment could be extended by testing a variety of *bandwidth estimation* and *bitrate selection* algorithms in order to determine which strategies will provide the best playback for both streams. Indeed, this is exactly the sort of experiment for which *dashem* was designed: *multiple videos* x *multiple service configurations* x *multiple network conditions* x *multiple networking strategies*. However, we do not seek to limit *dashem's* usage to just this scenario. For instance, from Section 4.2 we see that, given its low resource requirements, multiple instances of *dashem* could be used to generate bulk traffic for the purpose of either testing a network's ability to carry DASH streams or to provide background DASH traffic for an experiment. In general, we believe that *dashem* is a suitable, if not preferred, alternative to a full-featured DASH client in any experiment where the actual rendering and displaying of video is unnecessary.

Furthermore, we encourage researchers to create and share profiles with the community. These profiles need not represent actual services and videos. In fact, we are intrigued by the prospect of identifying patterns of segment sizes which might prove difficult for a DASH client's buffering or rate-selection algorithms. If such patterns exist, then they could be distributed as *benchmark profiles* against which new algorithms are tested. Additionally, DASH encoders could be designed to avoid producing these pathological patterns.

4.6 Related Work

The problems associated with using actual video for DASH experiments are best exemplified by the work of Lederer et al. [14], which represents an initial effort to provide the networking community with a common DASH data set, currently consisting of 7 videos, as well as an open source DASH encoder. Since their data set is publicly available on the Internet, it is limited to videos that are under the Creative Commons license and to videos that the authors have been given permission to distribute. As for the sizes of the videos in the data set, even the shortest available video *Big Buck Bunny*, which is only 10 minutes long, would require 1.8 GB of disk space to store all of the encodings. Although each video's manifest could be used in lieu of the video itself using the same technique that *dashem* uses to generate dummy segments, a researcher would still be limited to the selection of publicly available videos.

Huang et al. [13] developed a custom Netflix client to test various modifications to the rate-selection algorithm in an effort to eliminate the tendency of the native client to underestimate the available bandwidth in the presence of competing flows. Their client functioned by reusing session tokens to replay previously watched movies from the same CDNs that the native client used. Thus, while their technique allowed them to compare the performance of their client against that of the native Netflix client under similar conditions, it would be difficult for a researcher to scale the technique to support multiple simultaneous clients streaming a variety of movies. Additionally, the use of actual CDNs to provide the video segments introduces outside network conditions to an experiment.

Simulated clients have been developed by Liu et al. for ns2 [16] and by Lederer et al. for OMNeT++ [15]. In [16], it appears that the authors do not use actual video, but instead model the size of video segments as a constant function of the segment's bitrate and duration. Such an approach will fail to account for the effects of VBR encoding as depicted in Figures 1 and 12. In [15], the authors use a single video from their previously discussed DASH data set [14].

4.7 Conclusion

We have presented a novel technique to reduce a DASH service and its video library into a set of small text files, called profiles, which can be shared without violating copyright law. We have also presented a DASH client emulator, called *dashem*, which demonstrates how these profiles can be used to generate traffic that accurately reproduces the segment sizes of a DASH video's many encodings. By using the techniques described in this chapter, researchers can leverage a DASH service's vast video library to provide data for their own experiments.

CHAPTER 5: CONCLUSION

5.1 Suggested Future Work

As the goal of this thesis is to advance the state-of-the-art in DASH research, we hope that researchers use our tools and techniques to explore all facets of DASH. In this section, we list but a few ideas for potential future work.

5.1.1 Identification

The following research areas are extensions of our work outlined in Chapter 3.

- *Study the behavior of DASH clients "in the wild".* Since each video fingerprint includes the list of bitrates in their interleaved order, a researcher should be able to reconstruct the bitrate transitions made by a DASH client during a playback that was captured in a trace. This information could be used to better understand how commercial DASH clients adjust the video quality level of a stream in response to network conditions.
- *Assess DASH traffic's effect on network performance.* A network manager could maintain historical logs of DASH traffic on the network and determine its growth over time. This sort of analysis could aid in future capacity planning and network design.
- *Online identification.* Although we have yet to assess the performance of *dashid* when using a large dataset of fingerprints, we expect that its use of LCS comparisons will not be fast enough to support online identification of DASH streams. We invite researchers to explore more efficient methods of identification that might support an online program.

5.1.2 Experimentation

The following research areas are extensions of our work outlined in Chapter 4.

- *Android port.* Since *dashem* is written in Java, we expect that it can be used to create an Android application, thereby allowing researchers to test new DASH networking strategies on mobile devices such as phones and tablets.
- *GENI experiments.* One of our primary motivations for developing *dashem* was to enable large-scale DASH research in virtual laboratories such as GENI. As such, we encourage the networking community to develop and share RSpecs that replicate a variety of DASH-oriented architectures, ranging from individual home networks to residential neighborhoods.
- *Classroom instruction.* We believe that *dashem* is well-suited for classroom instruction on DASH, as its source code is relatively straightforward and contains no video-related modules. Thus, it should be easy for students to modify *dashem's* logic and experiment with various networking strategies.

5.2 Summary

In this thesis, we have presented our work towards improving the state of DASH research. Although our goal of DASH identification is a work-in-progress, we believe that the technique described in Chapter 3 shows promise. Furthermore, we have presented a method to emulate "real world" DASH videos from popular services that we believe will aid researchers in conducting realistic experiments.

REFERENCES

[1] Vijay Kumar Adhikari et al., "Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery," in *IEEE INFOCOM 2012*, 2012, pp. 1620-1628.

[2] Saamer Akhshabi, Ali C Begen, and Constantine Dovrolis, "An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP," in *ACM MMSys 2011*, 2011, pp. 157-168.

[3] Nathalie Amann, Ali Gouta, Dohy Hong, Anne-Marie Kermarrec, and Yannick Lelouedec, "Large Scale Analysis of HTTP Adaptive Streaming over Mobile Networks," in *IEEE International Symposium and Workshops on a World of Wireless, Mobile, and Multimedia Networks (WoWMoM '13)*, 2013.

[4] Apache Hadoop. [Online]. http://hadoop.apache.org/

[5] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson, "Touching from a Distance: Website Fingerprinting Attacks and Defenses," in *ACM Conference on Computer and Communications Security (CCS '12)*, 2012, pp. 605-616.

[6] dashem GitHub Repository. [Online]. https://github.com/andrewreed/dashem

[7] dashid GitHub Repository. [Online]. https://github.com/andrewreed/dashid

[8] Marius A. Eriksen, "Trickle: A Userland Bandwidth Shaper for Unix-like Systems," in *USENIX 2005 Annual Technical Conference*, 2005, pp. 61-70.

[9] fingerprinter GitHub Repository. [Online]. https://github.com/andrewreed/fingerprinter

[10] Global Environment for Network Innovations. [Online]. http://www.geni.net

[11] Félix Hernández-Campos, "Generation and Validation of Empirically-Derived TCP Application Workloads," University of North Carolina at Chapel Hill, Doctoral Dissertation 2006.

[12] Félix Hernández-Campos, Kevin Jeffay, and F. Donelson Smith, "Modeling and Generating TCP Application Workloads," in *IEEE BROADNETS 2007*, 2007, pp. 280-289.

[13] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari, "Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard," in *ACM IMC 2012*, 2012, pp. 225-238.

[14] Stefan Lederer, Christopher Müller, and Christian Timmerer, "Dynamic Adaptive Streaming over HTTP Dataset," in *ACM MMSys 2012*, 2012, pp. 89-94.

[15] Stefan Lederer, Christopher Müller, and Christian Timmerer, "Towards Peer-Assisted Dynamic Adaptive Streaming over HTTP," in *IEEE International Packet Video Workshop 2012*, 2012, pp. 161-166.

[16] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj, "Rate Adaptation for Adaptive HTTP Streaming," in *ACM MMSys 2011*, 2011, pp. 169-174.

[17] profiler GitHub Repository. [Online]. https://github.com/andrewreed/profiler

[18] (2012, November) Sandvine Global Report: Internet Data Usage up 120 Percent in North America. [Online]. https://www.sandvine.com/pr/2012/11/7/sandvine-global-report-internet-data-usage-up-120-percent-in-north-america.html

[19] (2013, November) Sandvine Report: Netflix and YouTube Account for 50% of All North American Fixed Network Data. [Online]. https://www.sandvine.com/pr/2013/11/11/sandvine-report-netflix-and-youtube-account-for-50-of-all-north-american-fixed-network-data.html

[20] (2011, October) Sandvine's Fall 2011 Global Internet Phenomena Report Indicates a Race for Services and Quality Delivery. [Online]. https://www.sandvine.com/pr/2011/10/26/sandvine-fall-2011-global-internet-phenomena-report-indicates.html

[21] T. Scott Saponas, Jonathan Lester, Carl Hartung, Sameer Agarwal, and Tadayoshi Kohno, "Devices That Tell on You: Privacy Trends in Consumer Ubiquitous Computing," in *USENIX Security Symposium*, 2007, pp. 55-70.

[22] SYSSTAT. [Online]. http://sebastien.godard.pagesperso-orange.fr/

[23] Jeff Terrell, Kevin Jeffay, F. Donelson Smith, Jim Gogan, and Joni Keller, "Passive, Streaming Inference of TCP Connection Structure for Network Server Management," in *IEEE International Traffic Monitoring and Analysis Workshop 2009*, 2009, pp. 42-53.

[24] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose, "Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on fon-iks," in *32nd IEEE Symposium on Security and Privacy*, 2011, pp. 3-18.