Measuring DASH Streaming Performance from the End Users Perspective using Neubot

Simone Basso[1,2], Antonio Servetti[2], Enrico Masala[2], Juan Carlos De Martin[1,2]
[1]Nexa Center for Internet & Society [2]Control and Computer Engineering Department
Politecnico di Torino - corso Duca degli Abruzzi, 24 - 10129 Torino, Italy
{simone.basso, servetti, masala, demartin}@polito.it

ABSTRACT

The popularity of DASH streaming is rapidly increasing and a number of commercial streaming services are adopting this new standard. While the benefits of building streaming services on top of the HTTP protocol are clear, further work is still necessary to evaluate and enhance the system performance from the perspective of the end user. Here we present a novel framework to evaluate the performance of rate-adaptation algorithms for DASH streaming using network measurements collected from more than a thousand Internet clients. Data, which have been made publicly available, are collected by a DASH module built on top of Neubot, an open source tool for the collection of network measurements. Some examples about the possible usage of the collected data are given, ranging from simple analysis and performance comparisons of download speeds to the performance simulation of alternative adaptation strategies using, e.g., the instantaneous available bandwidth values.

Categories and Subject Descriptors

H.5.1 [**Multimedia Information System**]: Video; C.2.3 [**Computer Communication Networks**]: Network Operations

General Terms

Algorithms, Measurement, Standardization, Documentation

Keywords

Dynamic Adaptive Streaming over HTTP, DASH, Rate Adaptation, Dataset, Neubot, Network Measurement Tool

1. INTRODUCTION

Recent years have witnessed a strong shift towards media streaming based on the HTTP protocol. This trend significantly changes the traditional paradigm in which web content and streaming media were delivered using two different strategies. Web content was transferred using the HTTP protocol and huge investments have been made to ensure fast delivery of popular content to the end users, relying on caching proxies and generic content delivery networks (CDNs). Conversely, streaming media relied on the UDP and other protocols designed to control streaming applications, such as RTSP, which made it very difficult to benefit from the architecture and investments in generic CDNs for web content delivery unless they were designed and tailored specifically for the protocols in use.

MMSys '14, March 19 - 21 2014, Singapore, Singapore

DOI:http://dx.doi.org/10.1145/2557642.2563671.

However, starting from 2007 the HTTP protocol has also been used to deliver media by means of splitting it in small file chunks, so that each chunk could be seen as an independent web resource with its own URL. This allowed to benefit from the investments made to optimize web content delivery, while at the same time shifted the burden to manage the media transfer to the client. The client has to constantly monitor the download speed, a critical aspect in multimedia delivery, and to adapt to the varying network conditions to limit impairments on the user's quality of experience.

This technology, named in its infancy "adaptive HTTP streaming", allowed streaming media to take full advantage of the investments made in CDNs and web caching proxy architectures to optimize web content delivery. Thus, many solutions were developed independently from major companies: Microsoft with Smooth Streaming in 2008, Apple with HTTP Live Streaming in 2009, Adobe with HTTP Dynamic Streaming in 2010.

Although the basic idea is practically the same, those solutions were incompatible among them due to lack of standardization. This problem has recently been addressed by the MPEG Working Group which, in April 2012, ratified a new standard named Dynamic Adaptive Streaming over HTTP (DASH) [4]. Following the MPEG standard philosophy, it enables interoperability and fosters adoption by the industry, but it still allows for competition among different players which can optimize different aspects of the implementation.

For instance, the standard does not define the client rate-adaptation logic that is a fundamental element to efficiently cope with the varying network conditions but has no impact on interoperability.

This paper presents our architecture for testing different DASH implementations, in particular their client rate-adaptation logic, on a large number of clients connected to the Internet using different access technologies, from residential DSL to high-speed connections, from wireline to wireless (cellular) connections. The proposed architecture, at the same time, creates a dataset with all the measurements that can be used for further analysis and improvements by other interested researchers.

Built on top of Neubot [2], a Measurement Lab [7] tool developed by the Nexa Center for Internet and Society, we released a *dashtest* tool that is based on the *libdash* library

and the DASH dataset [5] and that periodically tests and records DASH streaming sessions from each Neubot client to a server in the Internet. These measurements provide insights into the performance of DASH streaming on a large scale and they may provide cues to improve the design of new rate adaptation logic algorithms in terms of quality of experience. Some examples are shown in this work.

The remainder of this paper is organized as follows. Section 2 presents the related work, the motivation for data collection and the intended use of the data set. Section 3 gives a detailed description of the architecture used to perform the tests and to collect the dataset, that is based on a tool called Neubot. The details on the format of the data collected and some characterizing statistics are presented in Section 4. Finally, Section 5 concludes the paper.

2. RATIONALE AND RELATED WORKS

A key feature of MPEG DASH is the ability to handle bandwidth variations during a streaming session. A pull-based system is described in which the client is responsible for selecting and requesting the most appropriate resources from the server. Therefore, the streaming logic completely resides on the client side, i.e., the client can vary the streaming media bitrate every time a new media segment is required.

To date most of the open source software implementations that support DASH are based on the *libdash* library that, however, provides only two basic adaptation algorithms: *manual adaptation* and *always lowest bitrate*. Other open source implementations such as Mozilla (https://www.mozilla.org) and VLC (http://www.videolan.org/) rely on a *long term average* bitrate adaptation algorithm where the target bitrate of the next media segment, to be downloaded, is computed as the average bitrate of the whole streaming session.

Besides these simple algorithms, the research literature presents several papers that address the issue of designing and comparing the logic of different adaptation algorithms in a number of network environments.

Liu et al. [6] propose an algorithm that use the ratio of the segment duration and the latest single segment fetch time to detect network congestion and spare network capacity. The technique is evaluated using the network simulator ns-2 in the presence of exponential and constant bit-rate background traffic. Other papers consider instead real word scenarios. Akhshabi et al. [1] experimentally evaluate the rate adaptation algorithms implemented in three major players (Smooth Streaming, Netflix, OSMF). Muller et al. [8] evaluate their DASH implementation on a vehicular network scenario and compare its performance with other proprietary systems.

In addition to the academic interest in DASH, HTTP-based adaptive streaming solutions are increasingly chosen by several companies as the delivery method for streaming media. A first live and large-scale demonstration occurred with the 2012 London Olympics that reported up to 1,000 concurrent viewers, then Wimbledon and Roland Garros followed. In the industry premium on-demand services are also reported to be based on HTTP streaming such as Netflix, LoveFilm, and Amazon Instant Video.

However, none of these implementations or studies released to the public their data on the experienced network conditions, the logic of the adaptation algorithms, the measured download rates. As a consequence, it is extremely difficult to evaluate and compare different DASH algorithms, and there is a significant lack of a common dataset that can be used for this purpose.

To the best of our knowledge the only public dataset for DASH has been released in 2012 by Lederer et al. [5] and it mainly consists of A/V material for video quality evaluation. It comprises several videos with a duration between 10 and 90 minutes, that are provided in different segment lengths as well as in different representations ranging from 50 kb/s up to several Mb/s.

Using that dataset as a starting point, we add our contribution to the study and evaluation of DASH adaptation logic (AL) and algorithms with the release of a companion dataset with a huge collection of network measurements from DASH streaming sessions on the Internet. The measurements are collected by means of a Neubot [2] module for DASH that is inspired by the protocol implementation of the *libdash* library and that periodically tests and records DASH streaming sessions from each Neubot client to a random server in the Internet.

As will be detailed in Section 3, Neubot is a research project based on a lightweight open source program that runs in the background on thousands of Internet clients and periodically performs transmission tests with test servers hosted by the distributed Measurement Lab platform. Transmission tests probe the Internet using various application level protocols (which now include DASH) and test results are published on the web allowing anyone to inspect and analyze the data for research purposes.

The dataset, containing all the measurements collected by the Neubot clients and the preliminary analysis presented in this paper, is available at (http://media.polito.it/mmsys14).

3. METHODOLOGY

In this section we describe the Neubot architecture, the integration between Neubot and Measurement Lab, the Neubot DASH module, and the *dashtest* adaptation logic.

3.1 The Neubot Architecture

Fig. 1 shows the Neubot architecture, which consists of the Neubot program, the configuration server, the discovery server, the test servers, and the collect server.

The Neubot program (henceforth, Neubot) is a piece of software, written in Python, that volunteer users install on their computers. Neubot runs in the background as a system

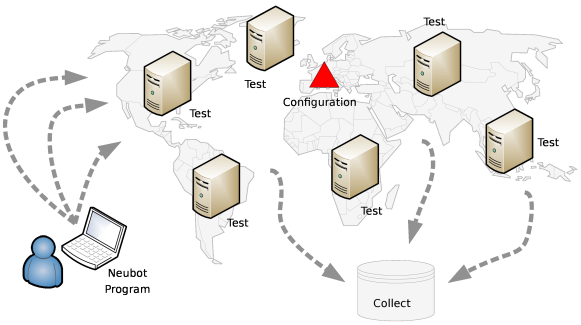

Figure 1: Neubot architecture. Instances on the users' computers are coordinated by the Neubot configuration server and connect to a test server to perform a network tests. Results are collected on the collect server.

service and every 30 minutes performs a network test (henceforth, test), by cooperating with the configuration server, the discovery server, and the test servers. In the following, we call an instance of the Neubot program running on the computer of a user 'the instance'.

Before running a test, the instance connects to the configuration server and invokes the Web API that returns the list of available tests[1]. Then, the instance selects one of the available tests (which we call the 'selected test'); the current implementation, in particular, picks the *dashtest* with higher probability than the other tests, because the *dashtest* is the current focus of our studies.

After the instance has selected a test, it connects to the discovery server and invokes the Web API that returns the address of a test server. By default such Web API returns the address of the closest server, but the instance may also request a random server[2].

Next, the instance connects to the test server address returned by the discovery server. The test server assigns to the instance a temporary token and inserts the instance into a wait queue. The instance has to wait for its turn to perform the test, which arrives when it reaches the front of the queue. At that point, the test server authorizes the instance to start the test, by passing to the instance an authorization token and other test parameters (e.g., in the *dashtest* case, the list of available representation bitrates).

During the test, the test server and the instance measure test-specific performance metrics. After the test, the test server and the instance share what they measured, and save the measurements results on their local disk.

Every day the collect server collects the results from the test-servers disks, and publishes them on the web.

3.2 Integration with Measurement Lab

The test servers are hosted by Measurement Lab (M-Lab), a distributed server platform that hosts open-source network-performance tools [3]. In practice, M-Lab provides a Linux-Vserver (virtual private server) in which Neubot is run in test server mode. As of October 2013, there are 124 M-Lab servers operating across 40 geographically-distributed sites around the globe.

M-Lab also provides Neubot with the functionalities of the discovery and the collect servers. The discovery server functionality is implemented by mlab_ns (the M-Lab Name Service), which is a Google App Engine application that returns the address of the closest M-Lab server or, optionally, of a random M-Lab server.

The collect-server functionality is implemented by the M-Lab data-collection pipeline, which every day fetches the results from the test servers and publishes them on the Google Cloud Storage service.

3.3 The Neubot DASH Module

The *dashtest* is implemented by mod_dash, a Neubot module that emulates the DASH streaming of a video resource composed of fifteen two-second segments. Each segment is available in one of the bitrate representations used by the DASH dataset (100, 150, 200, 250, 300, 400, 500, 700, 900, 1200, 1500, 2000, 2500, 3000, 4000, 5000, 6000, 7000, 10000, 20000 kb/s).

[1]Possibly tailored to the instance location and ISP.

[2]For the *dashtest*, which aims at probing many diverse network paths, the instance requests a random server.

Differently from a standard DASH implementation there is no media presentation description (MPD), but the test server sends the list of available bitrates to the instance just before the beginning of a test. Also, differently from a real DASH stream, the video payload is emulated by a sequence of random bytes.

The mod_dash module is composed of a mod_dash test server (HTTP-request handler) and a mod_dash client (henceforth, client). The client connects to the test server and requests the fifteen emulated-video segments using the adaptation logic that we describe in the following subsection.

3.4 The Dashtest Adaptation Logic

At the beginning, the client requests the first segment using the most-conservative bitrate representation, i.e., 100 kb/s. After the download of each segment the client computes the estimated available bandwidth (EAB) dividing the size of the segment in kbit by the download time in seconds. Subsequent segments are requested using possibly diverse representations that depend on the network conditions, i.e., the client selects the highest available representation bitrate that is lower than the EAB.

Note that all the segments are requested using the same persistent HTTP connection and, because the interval between two consecutive requests is in general smaller than the retransmit timeout, TCP should not exit the congestion avoidance state (unless, of course, network losses force the connection out out of such state).

This logic aims at selecting a representation bitrate that permits the download of the next segment in about two seconds (that is the emulated playout duration of the segment). However, when network conditions deteriorate and the *dashtest* estimates that too much bandwidth is being used, we adopt a mechanism to further reduce the EAB. We are concerned, in fact, that the *dashtest* could have a negative impact on the QoE of users' foreground flows.

The mechanism is implemented as follows and it is ispired by the the LEDBAT congestion control algorithm [9]:

```
if EDT > PLAY_TIME:
  REL_ERR = 1 - EDT / PLAY_TIME
  EAB = EAB + REL_ERR * EAB
  EAB = max(min_rep_bitrate,EAB)
```

in which EAB is the estimated available bandwidth, EDT is the elapsed download time, PLAY_TIME is the playout duration of the segment, and min_rep_bitrate is the minimum bitrate available in the DASH dataset.

As it will be detailed in Section 4.2, since the *dashtest* also records the channel throughput during the test, the measured data can be used to simulate the performance of more complex players (i.e., with different adaptation algorithms) in the same network conditions.

Name	last year	last 3 months
# of countries	146	108
# of Autonomous Systems (AS)	1744	1036
# of UUIDs	4060	2089
# of IP addresses	112371	29275
# of Different Locations	9613	4610
Median of # all tests per IP	12	11
Median of # *dashtest* per IP	7	7

Table 1: Dataset statistics as of Nov 11, 2013.

NAME	EXAMPLE	DESCRIPTION
uuid	7528d674-25f0-4ac4-aff6-46f446034d81	Random unique identifier of the Neubot instance, useful to perform time series analysis.
platform	linux2	The operating system platform, e.g. "linux2", "win32".
version	0.004016008	Neubot version number.
real_address	130.192.225.141	Neubot's IP address, as seen by the server.
internal_address	130.192.225.141	Neubot's IP address, as seen by Neubot.
remote_address	80.239.142.212	The server's IP address.
whole_test_timestamp	1382434858	Time when the test was performed (Unix epoch time).
srvr_data.timestamp	1382434858	Time when the test was performed on the server (Unix epoch time).
timestamp	1382434858	Time when the test was started (Unix epoch time).
clnt_schema_version	3	Version of the client schema.
connect_time	0.02469491958618164	RTT estimated by measuring the time that connect() takes to complete, measured in seconds.
iteration	14	Sequence number of the current donwload request.
request_ticks	1382434858.103292	Time when the request was performed on the server (Unix epoch time).
elapsed_target	2	Expected download duration, measured in seconds.
rate	20000	Segment representation rate for the current request, measured in kbit/s.
elapsed	0.5023031234741211	Time elapsed from the download request to the end of the download (i.e. download duration), measured in seconds.
received	5000131	Amount of bytes received from the server for the current request, measured in bytes.
delta_user_time	0.06000000000000005	Accumulated user time during a request, measured in seconds.
delta_sys_time	0.06000000000000005	Accumulated system time during a request, measured in seconds.

Table 2: *Dashtest* data format

4. DESCRIPTION OF THE DATASET

This Section briefly characterizes the measurements collected by the *dashtest* and discusses some insights that can be extracted from the measurements. These characterizations may be the foundation for further studies with real world measurements by researchers that aim at evaluating the performance of DASH streaming. Neubot counts about 1,000 users each day and a slightly higher number of IPs because some of the users connect their machine from both home and office locations with typically two different addresses. The number of tests run each day is about 20,000 and they comprise all the four implemented tests. Other statistics are presented in Table 1 and up to date information is available at (http://media.polito.it/neubot).

The Neubot DASH module performs about 10,000 tests each day involving more 1,000 IP addresses from about 100 countries and 1,000 autonomous systems.

A single *dashtest* lasts about 30 seconds because it performs the download of 15 two-second long segments at the available download bitrate.

Each test result is described with a number of properties that characterize the Neubot client, the server, the connection and each one of the segment downloads. The values that are recorded for each test are listed in Table 2.

The first set of values, from *uuid* to *remote_address*, describes the configuration of the Neubot client and it does not change with the type of test. The second set of values, from *whole_test_timestamp* to *connect_time*, describes the current *dashtest* and it is fixed among all the segment downloads in the same test. The field *srvr_data.timestamp* can be used, with the *remote_address* to uniquely identify each test. The third set of values, from *iteration* to *delta_sys_time*, characterizes the download of each segment.

After each test the client sends the results, in JSON format, to the server where they are stored. Then, once per day, the M-Lab data-collection pipeline fetches the results and publishes them on the Google Cloud Storage service. All the data collected by Neubot are available to the public through the Google Cloud Storage service without any restriction under a No Rights Reserved Creative Commons Zero Waiver.

All the Neubot raw data are organized into tarballs, which are grouped by the tool that generated the data, the date when the data was collected, and the server that collected the data. This means that each tarball contains all the data collected during a single day, by a single tool running on a single M-Lab server. For example, the tarball 20131008T000000Z-mlab1-lga01-neubot-0000.tgz contains the first 1GB of data collected by all the Neubot tests that were served by the M-Lab server mlab1-lga01 on October 8, 2013. The document at (https://code.google.com/p/m-lab/wiki/HowToAccessMLabData) describe how to access all the M-Lab data.

The Neubot DASH dataset contains the results of *dashtests* that are part of the tarballs for the months of November and December 2013, and January 2014.

4.1 Dataset Analysis

A deeper analysis of the data collected in the dataset allows to characterize the tests in terms of client geographic location, time of day at which they are performed and connection throughput.

The authorization to publish the *real_address* of the client, given by the Neubot users, allows us to determine the client country, region, city, latitude and longitude, and also the Autonomous System to which their IP belongs. For example, Fig. 2 shows the location of the clients in October 2013 as results from the IP geolocation information given by the free Geolite service provided by (http://www.maxmind.com). A more extended representation covering all the 10,000+ different IPs and the 2,000+ different locations from which Neubot test have been performed can be accessed on the web at the following link (http://media.polito.it/neubot/world).

Since Neubot is an always running background process

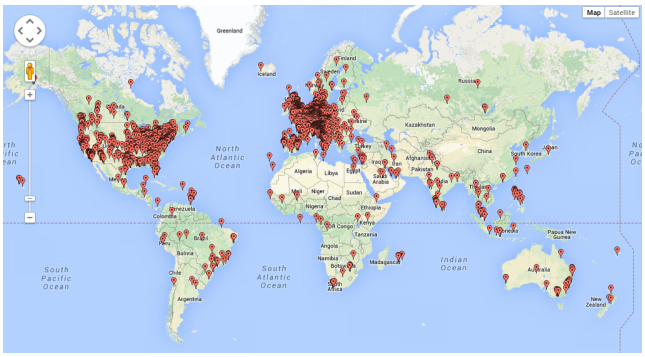

Figure 2: Geographical distribution of the Neubot clients in the world on the basis of their IP address. Figure refers to tests run in October 2013.

that schedules tests two times per hour, the dataset presents repeated tests per IP in a single day. The top graph in Fig. 3 maps the distribution of the tests during a day (November 2, 2013) for each one of the client IPs. IPs are sorted on the x axis as a function of the number of tests run in a day. The middle graph of Fig. 3 shows that more than half of the IPs run more than 7 tests a day (e.g., IP no. 470 run a *dashtest* at 2:30, 5:11, 6:50, 8:39, 12:32, 16:31, 17:02). The tests are almost equally distributed during the day as shown in the bottom graph of Fig. 3. The mean is 360 tests per hour.

Some insights into the characteristics of the *dashtest* results are shown in Fig. 4 that presents the histogram of the average download bandwidth measured by all the clients of a given Internet operator in the last year: values have been grouped for each IP for each day, then averaged and plotted in the histogram.

The graph clearly shows the typical download speed for that operator, which has a popular and cheap DSL offer at 7 Mb/s nominal speed. The sharp drop after that value is due to the fact that only few customers signed for more expensive contracts with higher nominal speeds. Moreover, note that the actual speed may be quite variable since it also depends on the network congestion at the time of measurement. This type of results is just a small sample of the information that researchers can extract from the dataset to investigate the potentiality of new techniques in the context of DASH streaming systems.

4.2 Example of DASH Simulation

In addition to the analysis of the collected data, dataset measurements can be used to drive simulations that test the performance of different rate adaptation algorithms on a given recorded trace. While the recorded trace is bound to the adaptation logic used for the measure, it is also able to provide information on the measured bitrate during the download of each segment. i.e., nearly every two seconds.

With the assumption that during each "sampled period" the connection bandwidth is constant and equal to the measured average bitrate, we can linearly interpolate the values between each point to produce a "download bitrate" suitable to run simulations.

Figure 5 shows an example in which the measured bitrates are used to draw the arrival time of each byte of the streaming session. With this information we can simulate the behavior of any specific adaptation logic.

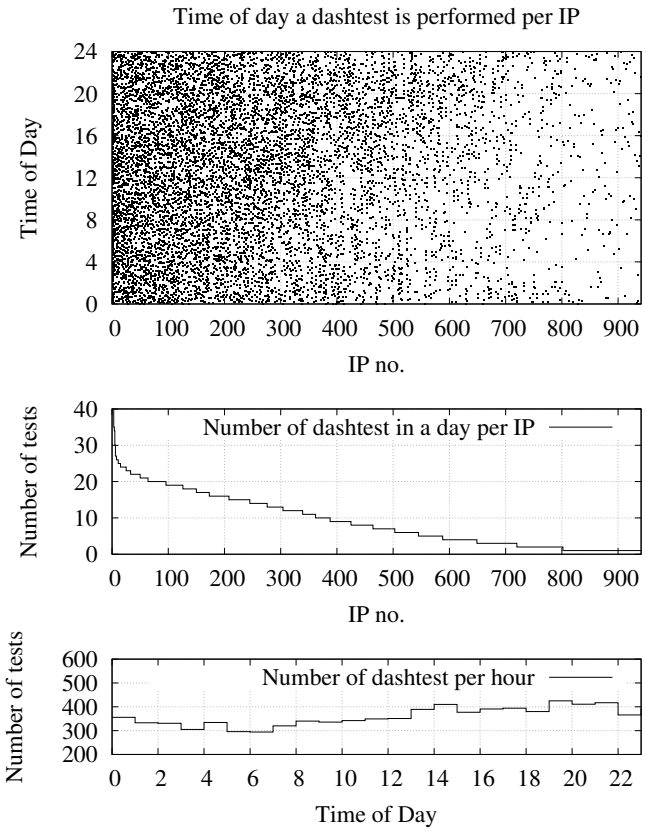

Figure 3: Distribution of *dashtests* during a day (November 2, 2013) for each client IP address. Each value on the x axis correspond to an unique IP. IPs are sorted on the x axis as a function of the number of tests run in a day. Each dot in the main graph represents a test from a given IP at a given time of day. The middle graph shows the number of tests per IP in a day and the bottom graph the number of tests per hour.

For testing purpose we implemented three adaptation algorithms named a) session average bitrate (SAB), b) last segment bitrate (LSB), and c) moving window average bitrate (WAB). SAB is the algorithm currently implemented in the VLC DASH plugin and requests the highest representation bitrate (HRB) which is lower than the average bitrate measured from the beginning of the streaming session. The LSB algorithm requests the HRB which is lower than the bitrate measured for the download of the last DASH segment. The WAB algorithm requests the HRB which is lower than

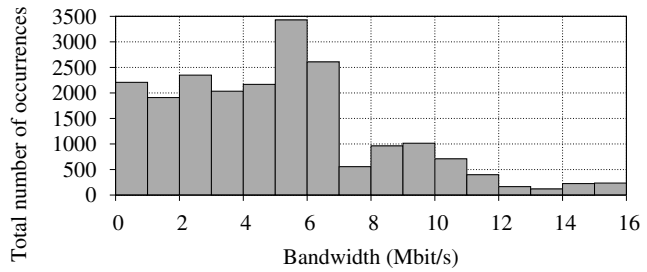

Figure 4: Number of occurrences of average download speed values over the last year (only clients connected using a given Internet operator).

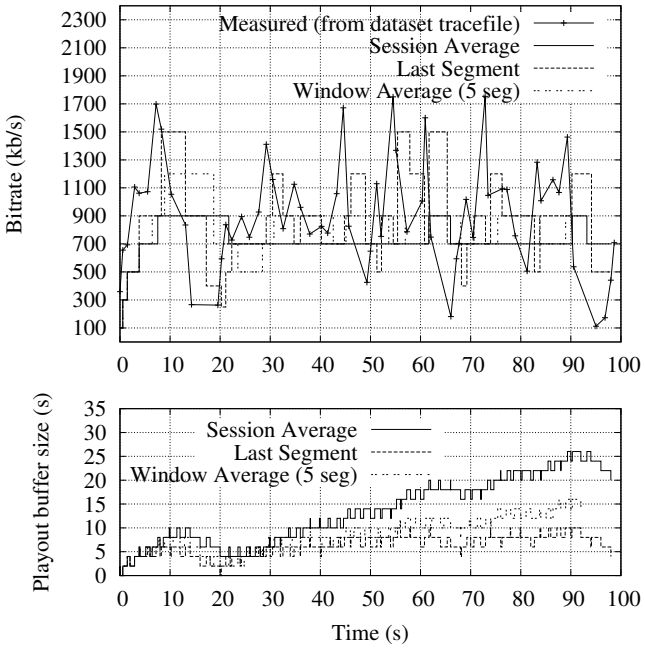

Figure 5: Simulation of the behavior of three different rate adaptation logic using the measured download bitrate in a particular trace of the dataset. The top graph shows the trace measured bitrate and the representation rate requested by the three simulated algorithms. The bottom graph shows the playout buffer level in the three cases.

the average bitrate measured on the last L segments.

The simulation tests these adaptation algorithms on a trace of 60 segments downloaded using a cellular connection. In the top graph, the bitrate measured by the *dashtest* is plotted together with the simulated representation bitrate requested by the three algorithms. Representation bitrates are as specified in the DASH dataset (i.e., 100, 250, 300, 400, 500, 700, 900, 1200, 1500 kb/s).

Here the SAB algorithm is able to adapt to the varying network bandwidth only during the beginning of the streaming session. After a while the weight given to the past measurements is too high to allow for rate adaptation and, even in presence of bandwidth fluctuations, SAB is unable to change the requested representation bitrate. The main advantage of this algorithm is that the user does not experience changes in the video quality.

On the contrary the LSB algorithm presents fast adaptation to the channel bandwidth with frequent and short changes in the requested representation bitrate. Often the changes are towards a high quality representation, in fact the overall video bitrate requested in the whole session is higher than in the SAB case. However, the risk to incur in freezes during the video playback is high. In fact, the bottom graph of Fig. 5 shows that, among the tested adaptation algorithms, this is the one that keeps the playout buffer closer to the minimum level.

The WAB algorithm is a simple compromise between the other two ones. The requested representation bitrate is computed using a moving average whose length can be chosen trading off fast adaptation for stable video quality (possibly as a function of the buffer level). A longer average window can produce a smoother throughput measurement, however it will also result in a slower rate adaptation behavior.

Note that this is only a small example of the possibilities offered by the collected data. A more complete assessment of different adaptation algorithms will be the focus of our future work.

5. CONCLUSIONS

This work presented a novel framework to evaluate the performance of rate-adaptation algorithms for DASH-based HTTP streaming. A number of network measurements have been performed by means of more than a thousand clients distributed in the Internet. The activity has been carried out using a DASH module built on top of Neubot, an open source tool for the collection of network measurements. Data are publicly available and they include the performance over time for each client. The paper also showed, by means of some examples, the usefulness of the dataset for various purposes, including HTTP streaming analysis and performance comparisons. We believe that this new dataset can strongly contribute to the rapidly growing research in the field of HTTP adaptive streaming.

6. REFERENCES

[1] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 157–168. ACM, 2011.

[2] S. Basso, A. Servetti, and J.C. De Martin. The network neutrality bot architecture: a preliminary approach for self-monitoring of Internet access QoS. In *IEEE Symposium on Computers and Communications (ISCC)*, pages 1131–1136. IEEE, 2011.

[3] C. Dovrolis, K. Gummadi, A. Kuzmanovic, S. D. Meinrath, and G. Tech. Measurement lab: Overview and an invitation to the research community. *ACM SIGCOMM Computer Communication Review*, 40(3):53–56, 2010.

[4] ISO/IEC DIS 23009-1. Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats, 2012.

[5] S. Lederer, C. Müller, and C. Timmerer. Dynamic adaptive streaming over HTTP dataset. In *Proceedings of the 3rd Multimedia Systems Conference*, pages 89–94. ACM, 2012.

[6] C. Liu, I. Bouazizi, and M. Gabbouj. Rate adaptation for adaptive HTTP streaming. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 169–174. ACM, 2011.

[7] Measurement Lab. http://www.measurementlab.net/.

[8] C. Müller, S. Lederer, and C. Timmerer. An evaluation of dynamic adaptive streaming over HTTP in vehicular environments. In *Proceedings of the 4th Workshop on Mobile Video*, pages 37–42. ACM, 2012.

[9] D. Rossi, C. Testa, S. Valenti, and L. Muscariello. LEDBAT: the new BitTorrent congestion control protocol. In *Proc. of 19th Intl. Conf. on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, 2010.