dashc: a highly scalable client emulator for DASH video

Aleksandr Reviakin
Dept. of Computer Science
University College Cork
Cork, Ireland
ar11@cs.ucc.ie

Ahmed H. Zahran
Dept. of Computer Science
University College Cork
Cork, Ireland
a.zahran@cs.ucc.ie

Cormac J. Sreenan
Dept. of Computer Science
University College Cork
Cork, Ireland
cjs@cs.ucc.ie

ABSTRACT

In this paper we introduce a client emulator for experimenting with DASH video. *dashc* is a standalone, compact, easy-to-build and easy-to-use command line software tool. The design and implementation of *dashc* were motivated by the pressing need to conduct network experiments with large numbers of video clients. The highly scalable *dashc* has low CPU and memory usage. *dashc* collects necessary statistics about video delivery performance in a convenient format, facilitating thorough post hoc analysis. The code of *dashc* is modular and new video adaptation algorithm can easily be added. We compare *dashc* to a state-of-the art client and demonstrate its efficacy for large-scale experiments using the Mininet virtual network.

CCS CONCEPTS

• CCS → Computing methodologies → Modeling and simulation → Simulation support systems → Simulation tools; CCS → Information systems → Information systems applications → Multimedia information systems → Multimedia streaming

KEYWORDS

DASH, video client emulator, video player, headless player, scalability, network performance

ACM Reference format:

A. Reviakin, A. Zahran, C. Sreenan. 2018. Open Dataset & Software Track. ACM Multimedia Systems Conference, *Amsterdam, Netherlands, June 12 – 15 2018 (MMSys'18)*, 6 pages.

1 INTRODUCTION

IP video traffic is growing at unprecedented rates and is expected to account for 82% of the total Internet traffic by 2021 [1]. In this context, video delivery systems should be designed to ensure satisfactory performance, even with significant growth in the volumes and numbers of users. Of special interest is the behaviour of DASH video players[1] when sharing a network bottleneck link, and implications for fairness, stability and quality of experience. MPEG DASH (Dynamic Adaptive Streaming over HTTP is the dominant standard for video streaming [7].

Network researchers and engineers have a requirement to be able to fully validate their DASH video solutions prior to deployment. The evaluation of large-scale networked systems can be conducted using physical testbeds. While offering a high degree of realism, this approach is usually expensive in both time and equipment costs, is difficult to maintain, and does not offer a controlled environment in which experiments can be confidently repeated. Reducing the cost of large-scale testbeds by using cheap computing nodes has been explored [8, 20]. For instance, Kleinrouweler et al. [8] run twenty DASH clients per Raspberry-Pi to create a six-hundred client experiment.

Recently, novel network emulation tools, such as Mininet [15], have evolved to enable implementation of realistic networks in a single machine. However, experimenting with a large number of resource-demanding applications, such as video, could saturate the available computing and memory resources. Hence, there is a need for developing a highly scalable DASH video client, suitably instrumented to support controllable experimentation of large-scale video systems.

In this paper, we present *dashc*, a new, light-weight DASH-compatible video streaming client, which has low system requirements and at the same time provides necessary data for analysis of the streaming sessions. *dashc* supports a set of published video adaptation algorithms (conventional [2], BBA-0, BBA-1, BBA-2 [3], ARBITER [4]). Our scalability test comparison of *dashc* and the state-of-the-art TAPAS tool [9] shows that for the same number of video clients, *dashc* uses ~20% less RAM and 2-3 times less CPU. To demonstrate *dashc* we evaluate it running with Mininet for 100 clients. To the best of our knowledge, there are no similar published experiments at that scale on a single machine.

In Section 2 we summarise the DASH standard and state-of-the-art players and tools. Section 3 explains the design and implementation of *dashc*, focusing on its inherent scalability,

[1] In line with convention, we use the terms "client" and "player" interchangeably.

extensibility and modularity. Section 4 uses *dashc* to explore the behaviour of many clients sharing a network bottleneck. Section 5 concludes.

2 BACKGROUND AND RELATED WORK

In the MPEG DASH standard a video file is split into many small segments of the same duration, usually 2/4/6/8/10 seconds each [7]. Every segment is encoded in a set of representations that are defined by the video resolution and average video bitrate. The structure of each media file is described in a MPD (media presentation description) file, an XML-compatible document. When a video client wants to play a movie it firstly downloads a corresponding MPD file and then the client's video adaptation algorithm is responsible for requesting the most appropriate representation for each segment throughout the playback.

Test tools related to DASH video streaming can be split into three categories, depending on whether a client decodes and displays video. Regular clients decode and display, e.g. GPAC [10], ExoPlayer [21]. Hybrid clients allow the user to select whether video is decoded/displayed, e.g. TAPAS [9]. Others do not support decode/display, e.g. AStream [11], *dashc*. Obviously, the players with no decoding/display have much lower system requirements. Tools vary also in regard to their degree of support for experimentation through logging, and extensibility to add new video adaptation algorithms.

A well-known state-of-the-art tool is TAPAS. The main goal of TAPAS is to allow researchers to focus on development of video adaptation algorithms and be able to easily reproduce experiments. TAPAS is written in Python. It was tested with two sets of DASH datasets. We note that TAPAS suffers from compatibility problems, being unable to parse DASH datasets (MPD file) generated by GPAC project [10]. The incompatibility is likely caused by the fact that the parser for MPD files was written before appearance of ISO/IEC 23009-1:2014[2]. AStream has similar compatibility issues.

In [13] the vertically integrated simulation framework for HTTP-based adaptive streaming applications was presented. The authors focused on three common issues, which appear in research on the topic of HTTP-based streaming technologies. The first is the use of unrealistic network environments. The second is the lack of comparability with other studies. The third is focusing on a narrow subset of possible parameter configurations. The framework has a client module, adaptation algorithm module and server module. The main purpose of the client module is a simulation of a playback: downloading a MPD file and requesting next segments, collaborating with an adaptation algorithm module in order to get the next representation index according to the chosen adaptation algorithm. The adaptation algorithm module implements a video adaptation algorithm. The server manages the connected clients and serves their requests. All of these modules are integrated into the *ns-3* simulator which supports many wireless and wired networks [22]. There is no real playback in this framework which means there is no need for actual video content, or to deal with scalability issues.

In [14] the authors described AdViSE, another vertically integrated emulation framework. AdViSE is made for automated testing of media clients. The backbone of this framework is Mininet. The Selenium server is used as a testing framework to automatically conduct experiments with available media players within a web browser. The system includes nine JavaScript media players. AdViSE consists of three servers: the first contains video content and MySQL database for storing performance measurements, the second one contains Mininet and the third one is used for a web management interface for configuring experiments and presenting real-time information. One of the features of this framework is to provide two quality of experience (QoE) metrics based on the number of stalls, total stalling time, stalling frequency, average duration of a stall and the video start-up time.

We can see that the described tools have been designed to address a number of different concerns. In contrast, in this paper we are focusing specifically on the scalability aspect of a standalone video player that is to be used for large experiments running on a single machine.

3 *dashc* DESIGN AND IMPLEMENTATION

Providing a convenient test tool for research has certain functional requirements. Our work is guided primarily by the need to facilitate experimentation involving large numbers of video clients. Thus the tool should have low system requirements – much lower than a commercial video client. On the one hand the number of available options should be comparable with a real player to have a real video session and on the other side it should be able to target the necessary research-related issues in a flexible way. One of the most important parts of any research tool is a data log file, the future tool should give as much available data as possible in an easy way for parsing by other tools and for ease of reading. Furthermore, the code structure of the future tool should be modular – easy to extend with new functionality and to ease tracking bugs.

Guided by these requirements, we designed *dashc*. The code base of *dashc* is very small. There are many different available parameters (full description in help, but omitted here due to space restrictions). If a video adaptation algorithm has certain default values, they will be applied (for example, BBA-2 uses 240 seconds buffer size). There are 3 main logical parts:

- buffer emulator
- adaptation algorithm
- logging system

The buffer emulator keeps track of the available playback buffer by constantly decreasing it like in the real video player and by increasing it when new segments arrive.

The adaptation algorithm makes a decision about the quality of the next requested segment.

The logging systems writes information about network performance and adaptation algorithm performance to a file.

2 http://dashif.org/conformance.html

The MPD-parser is written and tested with datasets from [16], which are compatible with the ISO/IEC 23009-1:2014 standard mentioned before. Additionally, *dashc* has support of the MPD format used with TAPAS tool.

The log system is implemented in a way to save as much potentially useful information for analysis as possible. The log file looks like a table with a header, each row contains information about every downloaded segments. The columns include next information:

- segment number starting from 1;
- arrival time of a segment in milliseconds;
- time spent for delivery of this segment in milliseconds;
- stall (freeze) duration, in milliseconds;
- representation rate of downloaded segment in Kbit/s (taken from MPD file);
- delivery rate of the network in Kbit/s (calculated as segment size divided by time for delivery);
- the actual bitrate of this segment (segment size in divided by the segment duration) in Kbit/s;
- segment size in bytes;
- buffer level after this segment was just downloaded, in seconds;

The log file can be parsed and applied as raw data for post processing. For example, the produced amount of data is enough for calculation of QoE metrics described in [17] and [18].

To facilitate ease of development and debugging of *dashc*, we selected OCaml. OCaml is a functional programming language with a static strong inferred type system. It is used in commercial companies and in academia, and it has an active community [5, 6]. OCaml has several key features for our project: it allows the development of real applications very quickly, the type system helps to avoid many bugs typical for the languages such as C/C++, and the native code compiler allows the generation of fast executable files. The package manager (opam) and the build system (jbuilder/dune) are also extremely convenient for usage.

The architecture of *dashc* is similar to a real video player, except for demuxing and decoding functions, which are omitted. *dashc* was successfully compiled and tested in Ubuntu x64 16.04.3/16.10/17.04/17.10 (as well as in Ubuntu 16.04.3 with Raspberry Pi 2/3).

4 EVALUATION

Our evaluation has two elements. Firstly it is to benchmark *dashc* against a competitive state-of-the-art player. This establishes the superiority of *dashc* in regard to scalability. Secondly, we take advantage of *dashc* to offer new insight into scenarios involving large numbers of clients.

For the evaluation we emulated a network using Mininet-WiFi (master branch from July 2017 [19]). All tests were made inside of the virtual machine created with VirtualBox 5.1.26 with 1 dedicated logical core (the host machine has Intel Core i7-6700HQ 2.6 Ghz Skylake with 4 physical cores or 8 logical), 4096 Mb of RAM (the host machine has 16 Gb of RAM), OS is Ubuntu 17.04 (kernel 4.10) – guest one and host one (with all available updates in August 2017), OCaml 4.05.0 version was used. We used the topology depicted in the Figure 1. Where caddy web server version 0.10 has a role of the web server [3], openvswitch is used as a switch [4], and video clients are *dashc* or TAPAS.

The data set "bbb-264" from the iVID project [16] was used (Big Buck Bunny movie encoded in x264 format, 10 representation levels, the lowest one has ~235kbps bitrate and 320x240 resolution, the highest one has ~4300kbps bitrate and 1920x1080 resolution). Only the first 5 minutes were used for playback.

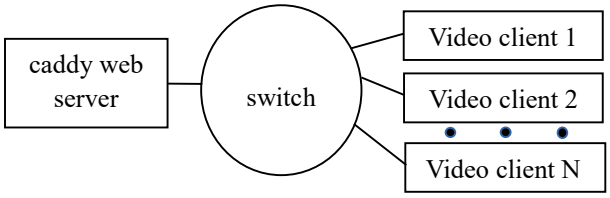

Figure 1. Network Topology inside the VM.

We decided to use the well-known state-of-the-art TAPAS tool for scalability comparison [5]. We had to apply numerous changes in the code in order to be able to run it with Python libraries (TAPAS dependencies) delivered in Ubuntu 17.04 repository (the similar issues most likely will come up with older versions as well). We used the TAPAS "fake" engine which does not decode the video. We modified the Conventional algorithm [22], in the TAPAS tool to make the operating configuration the same as in *dashc*: the buffer size was increased from 15 seconds to 60 seconds, the alpha coefficient was decreased from 0.2 to 0.1 in the conventional controller class, the inactive_cycle options (the number of the first segments which are downloaded in the lowest quality) was increased from 1 to 2. The conversion of the target rate to the representation level is made differently in the TAPAS tool, so we implemented the same logic in *dashc* (it can be easily turned on/off by a flag).

4.1 Scalability

To verify that the implementation of the conventional algorithm is the same in the TAPAS tool as in *dashc*, we made a couple of tests and compared the downloaded representation levels. In the first test the link between the switch and the host with a client had a static rate limit of 3 Mbit/s. The link between the host (web server) and the switch didn't have any limits. Results were the same for *dashc* and TAPAS. In the second test the topology was the same, however the bandwidth between the switch and the client host was changing every 20 seconds from 3 Mbit/s initially to 1 Mbit/s and vice versa. These regular changes in the rates were added to make experiments more realistic. The highest target achievable rate was equal to 2350 kbps (this is the rate from the MPD file), the real delivery rate during period of 3

[3] https://caddyserver.com/
[4] http://www.openvswitch.org/
[5] https://github.com/ldecicco/tapas

Mbit/s limitation was around 2.8 Mbit/s, the pattern of the requested representation levels was similar between TAPAS and *dashc*. For the scalability tests 1, 2, 3, 10, 20, 30, 40, 50 simultaneous video clients were tested. To record the CPU load and RAM usage we used /usr/bin/time application.

Figure 2 shows that the CPU load is 2-3 times lower during all experiments with *dashc* in comparison with experiments with the TAPAS tool. The RAM usage is lower by 20% when *dashc* was used in comparison with the TAPAS tool. During the experiment with 50 TAPAS clients the OS was much slower than usually. At the same time with 50 simultaneously running *dashc* the OS was almost as responsive as without running any experiments. In addition, we noticed that the swap file started to grow during the experiment with 50 TAPAS clients, which almost certainly does not guarantee an optimal performance.

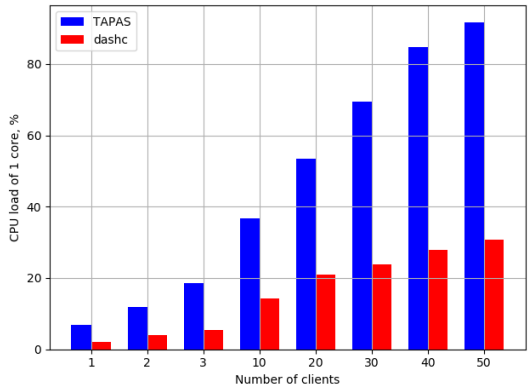

Figure 2. CPU load of 1 core.

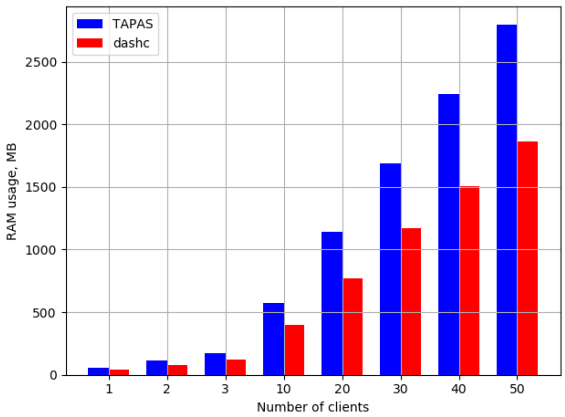

Figure 3. RAM usage.

The higher the CPU load and memory consumption in case of the TAPAS tool can be partially explained by its use of the Python language [9]. OCaml has a static type system and produces native binary code. This difference may not be critical if a small number of clients is used, on the contrary, the available computing resources deplete for large scale experiments.

4.2 Competing video clients

In this section, we evaluate streaming performance when a large number of clients share a link. These experiments take advantage of the fact that *dashc* can scale to many clients, and are included in the paper to illustrate the potential value of using *dashc*.

HTTP video streaming is often used for the investigation of competition between several video clients of a sharing limited link. A typical real example is family members in one house watching simultaneously different videos (on demand or live) would compete for a sharing bandwidth. In [12] authors focused on three performance problems: player's instability unfairness between players and bandwidth underutilization.

With low system requirements of *dashc*, we can evaluate large numbers of clients. Here we show 100 competing video clients. Experiments were made for 2, 4, 6, 8, 10, 20, 40, 80, 100 video clients, each experiment was repeated three times and then the average QoE was calculated as a final result (for 2, 4, 6, 8, 10 clients, 15 runs instead of 3 were used so as to ensure statistical accuracy). The link between the switch and the web server has a rate limit of (N*k), where N is the number of clients and for k 5 rates were used: 0.375, 0.75, 1.5, 3.0, 4.5 Mbit/s. In addition, the link between the switch and the web server has a constant one-way delay of 10 ms, 100 ms or 250 ms. The same movie "bbb-264" described in the scalability tests section was used (only the first 5 minutes were used for playback).

Initially we made our experiments with a default queue size (1000 packets) in the switch. This caused undesirable throughput restrictions, so we changed it to the recommended 1xBDP (bandwidth delay product) requirement. The precise value is rounded up to the closest divisible by 100.

Due to space restrictions, we present here only the most interesting results in our opinion (not the whole range of combinations of rate limits, delays, video adaptation algorithms). The further analysis was made based on four dimensions: rate limit, delay, number of clients, video adaptation algorithm.

Firstly, if we look at the average representation rate plot of any adaptation algorithm there is a clear differentiation between rate limits – the higher the rate limit the higher the representation rate. Secondly, for the 3.0 and 4.5 Mbit/s rate limits we can see the decline of average representation rate: the clearest decline can be noticed for ARBITER adaptation algorithm due to its nature to request higher representation levels on average than the BBA-2 algorithm. In Figure 4, we plotted results for ARBITER for 5 tested rate limits and 10 ms delay (results for 100 and 250 ms delays have similar trends to 10 ms delay ones).

The main differences between BBA-2 and ARBITER is that BBA-2 mostly relies on buffer size and future segment sizes, and is quite conservative – if the buffer is low it will request the low quality. ARBITER on the other side is a riskier algorithm. If it sees that the delivery rate is high and the variance of delivery rate in the past is low, it will take advantage of this situation and will request a higher video quality.

If we look at the average representation rates with the ARBITER adaptation algorithm in Figure 4, we can see a clear trend decreasing as the number of clients increases. This drop is noticeable for 3.0 and 4.5 Mbit/s rate limits. It can be partly explained by the discrete nature of average rates for different representation levels, the higher the representation level the bigger the gap between the average representation rate. Despite satisfying the 1xBDP requirement, we can see a negative effect of a large number of clients.

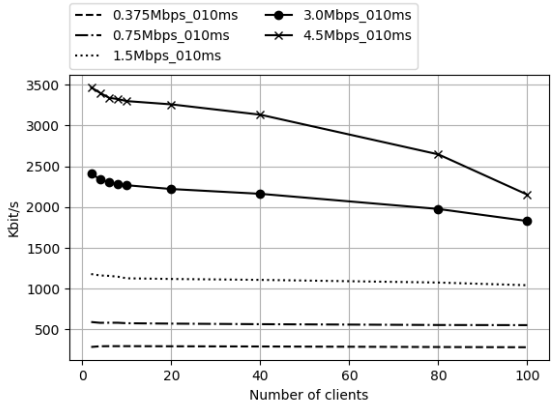

Figure 4. Average representation rate per session, ARBITER.

The other metric, which slightly changes with the rate limit, is the total stall duration. Only for the lowest rate limit of 0.375 Mbit/s the total stall duration has risen up to 0.95 average total stall duration per video client (exactly this situation happens with 40 simultaneous video clients) from 0.11 average total stall duration for 2 clients. The value 0.11, for example, means that in one of 30 video clients (15 runs with 2 video clients) had 3 stalls with a total duration of 3.341 second – by dividing 3.341 by 30 we get 0.11 average value. Investigation of per trace results showed that with the 0.375 Mbit/s rate limit, some clients can temporary get a much higher bandwidth share, up to 1 Mbit/s, meanwhile the other clients suffer from the lack of available bandwidth even to be able to support streaming with the lowest possible representation level.

The number of switches between different representation levels is much smaller for the lowest rate limit – 0.375 Mbit/s, which is expected, because with such a low rate limit an adaptation algorithm just doesn't have an opportunity to request higher levels. For 0.75 Mbit/s and bigger rate limits the number of switches doesn't have a clear dependency, because at these rates, the adaptation algorithm plays the main role in this parameter.

The delay parameter does not have any clear effect (clear trend) on any final result.

The Figures 5 and 6 show that the average number of switches scales with the number of clients. The same trend is observed across all tested clients with ARBITER and BBA-2. This increase can be explained by the fact that sometimes several of video clients get higher bandwidth share (sometimes quite suddenly) and the other get lower than average, which at the end triggers an adaptation algorithm to adapt. The results of the number of switches for all 5 tested rate limits and 100 ms delay with ARBITER and BBA-2 adaptation algorithms are shown in the figures 5 and 6 correspondingly. We can see in the Figure 5 that the most noticeable "rump up" in the number of switches happens between 2 and 20 video clients. The value is more stable for 20-100 clients.

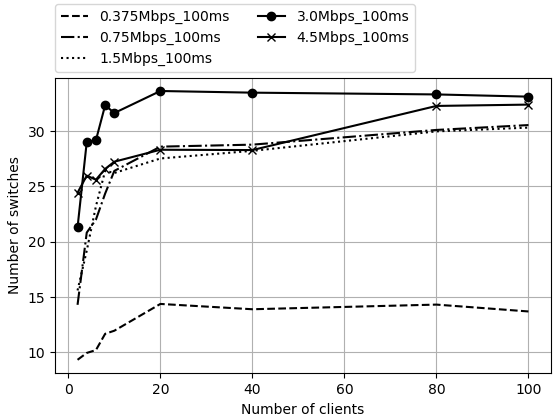

Figure 5. Average number of switches per session, ARBITER.

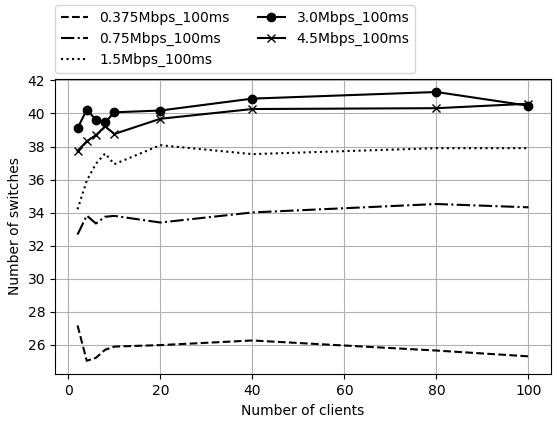

Figure 6. Average number of switches per session, BBA-2.

From Figure 6 we can make a conclusion that BBA-2 adaptation algorithm is not so sensitive to changes in bandwidth. The steady state of BBA-2 is based on the buffer level and reservoir size.

The low number of switches for 0.375 Mbit/s rate limit in Figures 5 and 6 can be explained by the fact that an adaptation algorithm just does not have enough capacity to switch frequently.

Figures 7 and 8 show the average representation rate for ARBITER and BBA-2 adaptation algorithms correspondingly. 250 ms delay and all 5 tested rate limits are shown.

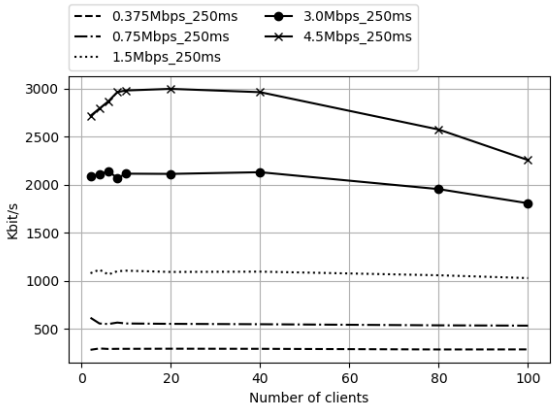

Figure 7. Average representation rate per session, ARBITER.

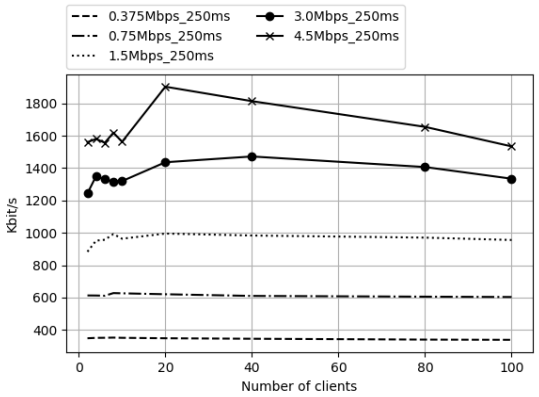

Figure 8. Average representation rate per session, BBA-2.

We can see that for 0.375 and 0.75 Mbit/s rate limits, BBA-2 adaptation algorithm requests slightly higher representation rates than ARBITER. With the rate limit of 1.5 both algorithms perform very similar. ARBITER perform better than BBA-2 algorithm when the rate limits are 3.0 and 4.5 Mbit/s.

CONCLUSIONS

We introduced *dashc* as a test tool for DASH video streaming. *dashc* is flexible in terms of available options, the code base is small and easy to extend. Of most significance is that *dashc* by design and implementation has low system requirements, allowing it to scale in experiments with large numbers of client. We benchmarked *dashc* against a state-of-the-art player, showing its scalability advantage. To illustrate its utility, we showed results from experiments using 100 video clients simultaneously on a single core machine. We believe that dashc will prove a valuable tool for use by network researchers and engineers. The code of *dashc* is available at https://github.com/uccmisl/dashc

ACKNOWLEDGMENTS

The publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number 13/IA/1892.

REFERENCES

[1] Cisco Visual Networking Index, 2016-2021. White paper, June 7, 2017.
[2] Z. Li, X. Zhu, at al. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. IEEE J. Sel. Area in Commun., vol. 32, issue 4, pages 719-733, April 2014.
[3] Te-Yuan Huang, at al. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. Proc. of the 2014 ACM conf. on SIGCOM, Chicago, Illinois, USA, 2014, pages 187-198.
[4] A. H. Zahran, C. J. Sreenan. ARBITER: Adaptive rate-based intelligent HTTP streaming algorithm. 2016 IEEE International Conference on Multimedia Expo Workshops (ICMEW). Seattle, WA, USA, 1–6.
[5] Teaching OCaml, https://ocaml.org/learn/teaching-ocaml.html (last access: February 2018).
[6] Companies using OCaml, https://ocaml.org/learn/companies.html (last access February 2018).
[7] T. Stockhammer. Dynamic adaptive streaming over HTTP –: standards and design principles. MMSys '11 Proc. of the second annual ACM conf. on Multimedia systems, San Jose, CA, USA, pages 133-144.
[8] J. W. Kleinrouweler, B. Meixner, P. Cesar, Improving Video Quality in Crowded Networks Using a DANE, ACM NOSSADAV'17 Proceedings of the 27th workshop on network and operating systems support for digital audio and video, pages 73-78, Taipei, Taiwan, June 20-23, 2017.
[9] L. De Cicco, at al. TAPAS: A Tool for rApid Prototyping of Adaptive Streaming algorithms. VideoNext '14 Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming, Sydney, Australia, pages 1-6.
[10] GPAC, https://gpac.wp.imt.fr (last access: February 2018).
[11] P. Juluri, V. Tamarapalli, Deep Medhi, "SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP," Communication Workshop (ICCW), 2015 IEEE International Conference on Communication Workshop, London, UK.
[12] S. Akhshabi, at al. What happens when HTTP adaptive streaming players compete for bandwidth?, NOSSDAV'12 Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video.
[13] H. Ott, K Miller, A. Wolisz. Simulation Framework for HTTP-Based Adaptive Streaming Applications. WNS3'17 Proceedings of the Workshop on ns-3, Porto, Portugal, June 13 – 14, 2017, pages 95-102.
[14] A. Zabrovskiy, at al. AdViSE: Adaptive Video Streaming Evaluation Framework for the Automated Testing of Media Players. MMSys'17 Proceedings of the 8th ACM on Multimedia Systems, Taipei, Taiwan, June 20 – 23, 2017, pages 217-220.
[15] Mininet. http://mininet.org (last access: February 2018).
[16] J. J. Quinlan, A. H. Zahran, C. J. Sreenan. Datasets for AVC (H.264) and HEVC (H.265) evaluation of dynamic adaptive streaming over HTTP (DASH). MMSys'16 Proceedings of the 17th International Conference on Multimedia Systems, Klagenfurt, Austria, May 10 – 13, 2016.
[17] Y. Liu, at al. Deriving and Validating User Experience Model for DASH Video Streaming. IEEE Transactions on Broadcasting, volume 61, issue 4, December 2015, pages 651 – 665.
[18] S. Petrangeli, at al. QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming, ACM Transactions on Multimedia Computing, Communications and Applications (TOMM), volume 12, issue 2, March 2016, article No 28.
[19] R. R. Fontes, at al. Mininet-WiFi: emulating software-defined wireless networks, Proceedings of the 11th International Conference on Network and Service Management (CNSM '15), pp. 384–389, Barcelona, Spain, November 2015.
[20] A. K. I. Remke, at al. Capabilities of Raspberry Pi 2 for Big Data and Video Streaming Applications in Data Centres, Proceedings of the 18th International GI/ITG Conference on Measurement, Modelling and Evaluation of Dependable Computer and Communication Systems (MMB & DFT 2016), pp. 183-198, Switzerland, April 2016.
[21] ExoPlayer, https://github.com/google/ExoPlayer (last access: March 2018).
[22] Ns-3 simulator, https://www.nsnam.org/ (last access: March 2018).