Emulation of Dynamic Adaptive Streaming over HTTP with Mininet

Anatoliy Zabrovskiy, Evgeny Kuzmin, Evgeny Petrov, Mikhail Fomichev
Petrozavodsk State University
Petrozavodsk, Russia
{z_anatoliy, kuzmin, johnp, fomichevm}@petrsu.ru

***Abstract*—Video streaming is becoming more and more popular technology for media content delivery over the Internet. Dynamic Adaptive Streaming over HTTP (DASH) allows delivering data streams to a user with the highest possible bit rate in varying bandwidth conditions which is particularly significant for hopping communication channels present in wireless networks. This paper investigates the delivery of media content over the Internet using MPEG-DASH technology within the network emulation environment Mininet connected to a real hardware client and a server from a real IP-network. We present Mininet settings that allow embedding this virtual environment into existing network infrastructure. The study shows that the bandwidth variation of a communication channel emulated in Mininet yields similar effects on streaming video as compared to experimental results obtained with a specialized hardware-software network emulator configured with the same channel characteristics. We conclude that Mininet can be considered as a practical tool for the emulation of video streams transmission employing Dynamic Adaptive Streaming over HTTP, as well as be used for the development of new adaptive control algorithms.**

I. Introduction

The technologies for streaming video delivery are continuously developing, primarily due to the need to keep up with the rapidly growing amount of video traffic on the Internet [1]. Recently, video streaming transmission over HTTP known as adaptive HTTP streaming has become widely deployed. Several software vendors like Adobe, Microsoft and Apple have long been using their own proprietary systems for content delivery over HTTP such as HTTP Dynamic Streaming [2], Smooth Streaming [3] and HTTP Live Streaming [4] respectively. The advantage of using HTTP is that the ordinary web servers with a caching capability can be used for streaming video.

HTTP Adaptive Bitrate Streaming (ABR) technologies allow a real-time adjustment for the delivered bit rate and video resolution depending on the varying bandwidth conditions, client's CPU usage as well as the type of an end device. Video files of different bit rates and resolutions should be adapted accordingly and uploaded to a server in advance or, in case of live streaming, generated by the server in a real-time mode. The decision regarding a bit rate switch is made on the client side by the media player using built-in adaptive control algorithms.

Dynamic Adaptive Streaming over HTTP (DASH), also known as MPEG-DASH, is the first bit rate adaptive HTTP-based solution which became an international standard in 2012 [5]. Currently, this standard is being widely applied, especially in live streaming video systems. YouTube and Netflix have started deploying MPEG-DASH [6], [7] which means that the format will play an important role in streaming.

In order to investigate new technologies for streaming video existing communication networks can be utilized which is not always convenient or even feasible. Thus, to overcome the aforementioned obstacle network emulators [8] are frequently used: one of them is open-source Mininet [9]. Mininet is capable of building realistic virtual topologies consisting of numerous network elements such as end hosts, switches, routers and communication links. In a nutshell, Mininet implements a concept of Software-Defined Networking (SDN). SDN is an emerging networking paradigm in which a control layer is implemented in software and separated from a packet forwarding plane [10]. It is expected that the future deployment of a new SDN concept [11] will allow administrators to control network services and hardware without having to know the intricate underlying functionality.

As we expect, the rapid development of new approaches to network establishment and maintenance together with technologies for media content delivery will eventually lead to their complementary utilization. Enabling such functionality would maximize the perceived quality of experience (QoE) while watching video. With this in mind, we decided to estimate the delivery efficacy of real MPEG-DASH traffic through Mininet. Hence, the main contributions of this paper are as follows. Firstly, we develop a methodology for setting Mininet virtual environment with bandwidth shaping functionality. Secondly, an experimental setup is presented which interconnects two parts: a virtual environment established with Mininet and a real IP-network. Finally, we compare the process of transmitting DASH content via Mininet and the real network infrastructure under a number of traffic shaping scenarios. Based on our findings we recommend using Mininet emulation environment as a handy tool for testing and evaluating the performance of adaptation logic for DASH-based multimedia streams. Moreover, Mininet is considered to be reasonably accurate for emulating DASH-content delivery in pre-set topologies as well as software-defined networks.

The rest of the paper is organized as follows. Related work is discussed in Section 2. Section 3 provides an overview of Mininet within the SDN paradigm. The research methodology and experimental setup are explained in Section 4. Section 5 presents the main results along with a relevant discussion. Finally, in Section 6 conclusions are drawn and future work directions are indicated in Section 7.

II. Related work

To start with, there exist several hardware-software network emulators which can be used for investigating the impact of channel characteristics on the delivery of DASH-based streams. For example, open-source WANem emulator [12], specialized Linktropy hardware [13] from Apposite Technologies, or Netem module [14] built into a Linux kernel. Furthermore, several studies have shown the applicability of network emulators for performance-wise estimation of varying communication channels. Muto and colleagues [15] evaluated the performance of MPEG-DASH over Content Centric Networks (CCNx) by considering such link parameters as bandwidth, delay and a packet loss ratio set in PacketStorm emulator [16]. In [17] authors conducted experimental analysis of dynamic adaptive streaming over HTTP in CCNx, where bandwidth shaping was achieved with Linux Traffic Control Program (*tc*) [18] and Hierarchical Token Bucket (*htb*) packet scheduler [19]. Additionally, the round-trip time (RTT) was controlled by Linux Network Emulator (Netem) [14]. It should be noted that some of the aforementioned emulators can be configured to support a traffic shaping environment (e.g. dedicated Linktropy 5500 hardware has such functionality). In essence, this feature allows setting a predefined scenario according to which bandwidth or other channel parameters would change over time.

Mininet can be used to achieve both goals simultaneously, namely setting up a network emulation environment and enabling traffic shaping for a specified topology. Lantz and colleagues [20] considered several tools for network emulation and pointed out that Mininet is a solution for rapidly prototyping large networks on constrained resources of a single laptop. In [21] it is claimed that Mininet provides a simple and inexpensive network testbed for developing OpenFlow applications in Software-Defined Networks. Recent work [22] used Mininet virtual network to evaluate the effect on adaptive streaming over HTTP when a bottleneck link was shared. However, the presented experiments did not take into account a bandwidth shaping scheme for the bandwidth variation of a communication channel.

In this paper we describe how to build a system for delivering DASH-content over the Mininet environment that comprises of a virtual topology connected to a real network. The proposed solution allows configuring and applying the bandwidth shaping scheme which we consider vital for the experimental analysis. Therefore, our approach leverages Mininet flexibility and scalability as compared to other works which used several instances to set up a virtual environment and tune channel parameters. Moreover, we compare the accuracy of Mininet emulation capabilities with specialized Linktropy equipment which provides insights whether the former can become an affordable platform for researching and developing systems for adaptive streaming.

III. Mininet overview

Mininet is designed in a way that follows a concept of Software-Defined Networking (SDN). The main difference between SDN and a traditional network (Fig. 1) is the separation between control and forwarding planes which results in more flexible and agile network configurations [23]. The control layer is usually represented by so-called network controllers such as NOX [24], ONIX [25] and Beacon [26] which are responsible for centralized control (e.g. a decision whether to drop, forward or buffer a particular packet) and network monitoring. The forwarding plane consists of various networking devices such as a switch, router and access point. Both layers communicate through some interface protocol such as OpenFlow [27]. A vital feature of SDN is its capability to be adjusted according to concrete user or application requirements after the network has been set. Therefore, utilization of SDN opens a wide range of opportunities for rich network configurations and domain specific optimizations.

In its core, Mininet uses a mechanism of Linux lightweight containers which allows configuring and running a virtual network within a single OS kernel [23]. In Mininet topology there are various instances that can be set such as communication links, hosts, network switches and controllers [9]. Mininet host behaves like a real computer. For example, it is possible to use SSH connections to remotely access a particular node and run real programs on it. Additionally, Mininet provides a capability to configure and tune numerous parameters of a communication channel such as throughput, delay, jitter, etc. To accommodate a large number of networking instances (e.g. hosts and switches) Mininet uses process-based virtualization within a single OS kernel [9]. On top of that Mininet supports user interaction (e.g. using *ping* or *traceroute* utility) with a virtual network environment by means of a command line interface (CLI). Furthermore, Mininet can be interfaced with real networks, for instance, wired or wireless LANs. Moreover, in order to install Mininet a laptop or a desktop running Linux is required. A straightforward and extensible Python API is provided for network establishment and development [9].

IV. Methodology and experimental setup

In this section we describe our methodology along with settings and network parameters used for DASH-based content delivery. The experimental setup that we developed (Fig. 2) consists of the following elements:

- Apache HTTP Server (package httpd-2.2.15-39.el6.centos.x86_64) width DASH content
- Mininet network emulator (version 2.2.1)
- Client PC (Windows 8 64-bit, Intel Core i7-4700MQ, RAM 8 GB) with MPEG-DASH media player
- Customized web-based management interface that we implemented
- MySQL RDBMS

Further we provide a full description of the core elements in our experimental setup. More rigorous and detailed information regarding the presented setting is given in Table I.

A. DASH content generation

The DASH content for all experiments has been generated using bitcodin service [28] by encoding file "Sintel.2010.720p.mkv" [29] (14 minutes 48 seconds long) into MPEG-DASH format with a wide range of different bit rates. We considered the following bit rates for video streams: 4.0

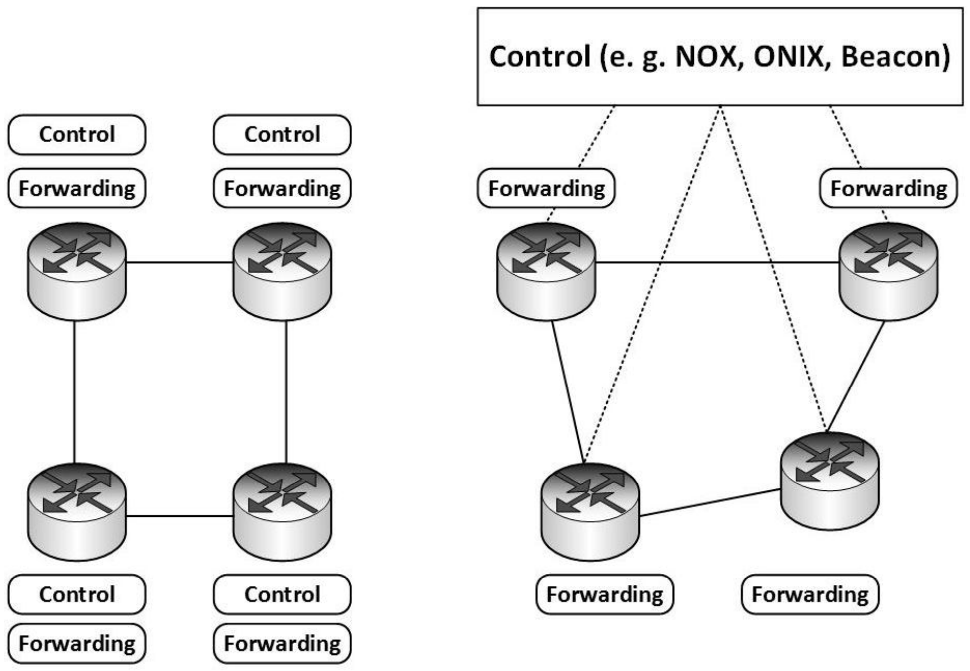

Fig. 1. Traditional network (left) vs. SDN (right)

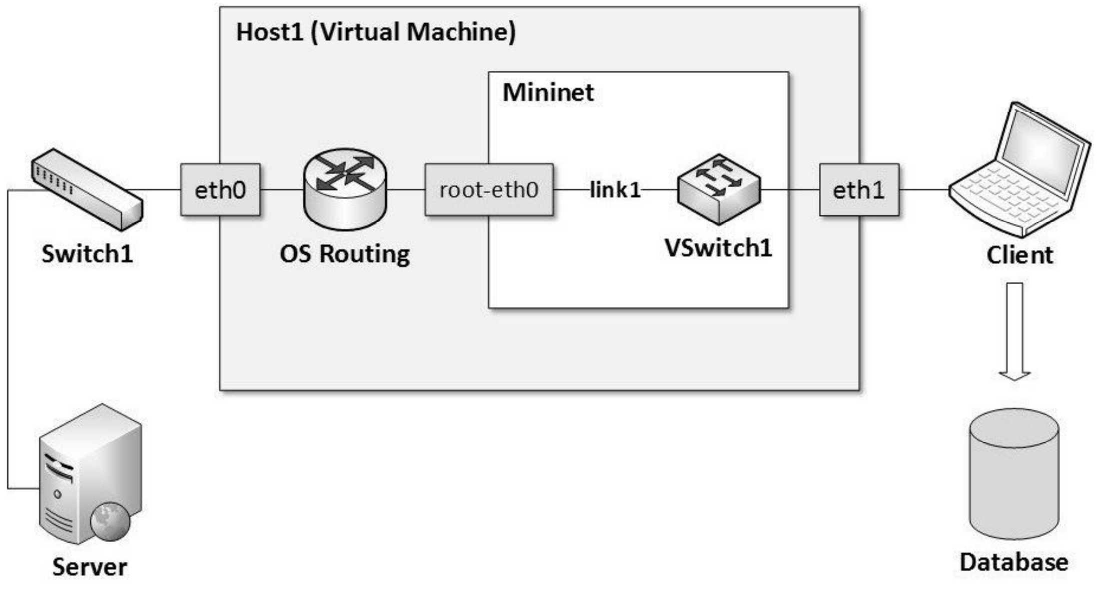

Fig. 2. Experimental setup

Mbps, 3.0 Mbps, 2.4 Mbps, 1.5 Mbps, 1.1 Mbps, 0.8 Mbps and 0.5 Mbps as well as a single audio stream with a bit rate of 128 Kbps. The obtained video and audio content was uploaded to Apache HTTP Server together with a MPD file. The content was transmitted between the web server and the client over HTTP 1.1 protocol.

B. Mininet and network parameters

In our experiments Mininet was running within a virtual machine (VM VirtualBox), launched on a computer with two network interfaces. Both network cards were configured in a bridge mode and enabled in VirtualBox. In fact Mininet could have been established on a real computer as well, which would produce the same results. Hence, we assume that the VM acts as a normal PC host, named *Host1*. In this setting one of the Ethernet interfaces (*eth0*) on *Host1* was connected to a physical network switch (*Switch1*). Additionally, a desktop running Apache HTTP Server (*Server*) was attached to the same *Switch1*. Correspondingly, *Server* and *Switch1* represent a real network in our experimental setup.

The virtual network that was created with Mininet Python

TABLE I. ELEMENTS OF EXPERIMENTAL SETUP

Name	Description
Server	Apache HTTP Server (package httpd-2.2.15-39.el6.centos.x86 64) with DASH content
Mininet	Network emulator (version 2.2.1)
Client	PC with MPEG-DASH media player and web-based management interface (Windows 8 64 bit, Intel Core i7-4700MQ, RAM 8 GB)
Web-based management interface	Our customized implementation (version 0.3)
Database	MySQL RDBMS
OS Routing	Linux kernel forwarding feature
Host1 (Virtual Machine)	Virtual-Box VM (version 4.2.16) with Ubuntu OS (version 14.04 LTS)
Switch1	Netgear JGS524E-100PES
Vswitch1	Virtual switch (mininet.node.OVSKernelSwitch Class)
link1	Link with symmetric TC interfaces configured via opts (mininet.link.TCLink at Mininet Python API)
root-eth0	root namespace node (mininet.node at Mininet Python API)
eth0	Host1 ethernet interface 0
eth1	Host1 ethernet interface 1

API comprised of a single virtual switch (*VSwitch1*) and a built-in default controller. By means of *Intf()* function [30] we attached another real hardware interface of *Host1* (*eth1*) to *VSwitch1* and connected a personal computer (*Client*) to the same interface. It should be noted that *Client* was assigned an IP address from the range of Mininet IP network. Therefore, in the presented setting *Client* is considered as a part of the Mininet virtual IP network. More concrete details regarding the network parameters of our experimental setup can be found in Table II.

A high-level overview of the developed setting can be given as follows. The main Python script is used to automatically create a virtual Mininet topology and interconnect it with a real IP network. Furthermore, with this program we can vary channel bandwidth within the virtual network according to a predefined scenario. Therefore, video streams destined for *Client* can be influenced and the corresponding effect estimated.

C. Interconnecting Mininet with the real network

The virtual Mininet topology was connected to the real network via a special node *root-eth0* created on *Host1* by our Python script with the following command: root = Node(root, inNamespace=False). Afterwards, this node was assigned an IP-address from the Mininet IP network and was used as a gateway address for *Client*. By means of *TCLink()* function [31] a link between *root-eth0* and a virtual switch (*VSwitch1*) was established as such: *link1* = TCLink(root, VSwitch1, bw=100, delay=0ms, loss=0, jitter=0ms). Thus, we obtained a communication channel that could be configured with different bandwidth values. Additionally, on *Host1* a special route was added to forward packets addressed to the Mininet IP network through the *root-eth0* instance. In order to enable routing on *Host1* we utilized a Linux kernel forwarding feature in the following manner: sysctl net.ipv4.conf.all.forwarding = 1. Eventually, network packets sent to the range of Mininet IP addresses were redirected by *Host1* to the Mininet network.

D. Mininet and bandwidth shaping

By utilizing Minievents framework [32] our program is capable of tuning channel characteristics (see *link1* above) between virtual and real networks at specified moments in time. To achieve this we created a configuration file in JSON notation (Fig. 3) which contained a list of events that defined time at which a particular event should occur and the properties of *link1*, in the current setting it is bandwidth.

TABLE II. NETWORK SETTINGS OF EXPERIMENTAL SETUP

Network (IP-address)	Network instance	Instance IP-address
Real IP Network (192.168.1.0/24)	Server	192.168.1.1/24
	eth0	192.168.1.254/24
Mininet IP Network (192.168.2.0/24)	root-eth0	192.168.2.254/24
	eth1	no address
	Client	192.168.2.1/24

```
[
 {
   "time": 0,
   "type": "editLink",
   "params": {
     "link": "link1",
     "bw": 1
   }
 },
   ...
 {
   "time": 61,
   "type": "editLink",
   "params": {
     "link": "link1",
     "bw": 3
   }
 },
   ...
]
```

Fig. 3. A sample of configuration file for bandwidth shaping

E. Client side

On the client side we opted for Bitmovin bitdash player [33] as a media player for the video streams. It was configured to send videoBitrate parameter which is a bit rate of the currently played video segment in kbps. VideoBitrate values were obtained through player's API provided by the developer. Throughout the whole experiment the aforementioned parameters were recorded each second and stored in a database. The media player was launched in Google Chrome browser (version 45.0.2454.93 m) with a buffer size set to 40 seconds which was a default value specified in the official documentation [34]. In the current setup it means that Bitmovin bitdash player starts downloading new segments as soon as the buffer level drops below 40 seconds.

V. EXPERIMENT RESULTS

The duration of each experiment was 120 seconds. We deemed this time length to be representative in terms of considered bit rates and, to the best of our knowledge, reflecting a typical application case. For all experiments, bandwidth values for the communication channel (see *link1* above) varied according to the predefined scenario shown in Fig. 4. As can be seen each 30 seconds the bandwidth changed in the following sequence: 1 Mbps, 2 Mbps, 3 Mbps and 1 Mbps. Such a pattern of bandwidth shaping inevitably caused the bit rate switch of various DASH-based streams described at the beginning of this section.

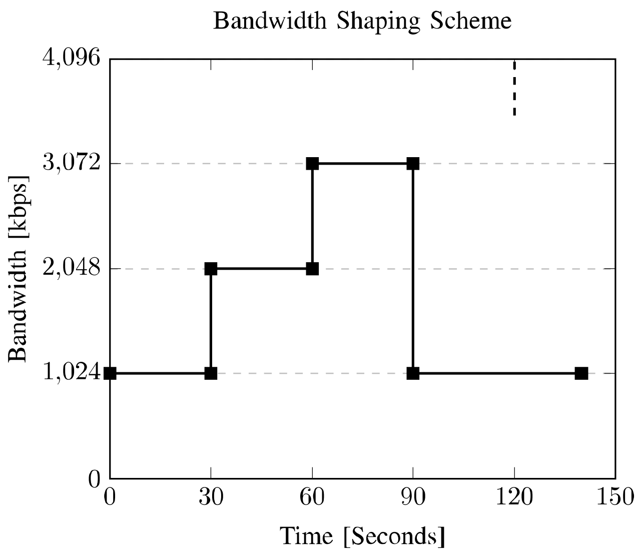

Fig. 4. Bandwidth shaping scheme of experiments

Overall, we conducted 50 experiments using the aforementioned experimental setup. For each experiment we obtained 120 samples of videoBitrate. All videoBitrate values acquired throughout the study were divided into four categories with 1500 samples in each. Therefore, resulting groups contained videoBitrate values related to one (out of four possible) bandwidth outcomes: 1 Mbps (from 1 to 30 sec), 2 Mbps (from 31 to 60 sec), 3 Mbps (from 61 to 90 sec) and 1 Mbps (from 91 to 120 sec).

To evaluate the relevance of the results obtained with Mininet, we repeated the same set of experiments with specialized Linktropy 5500 equipment. Fig. 5 depicts averaged videoBitrate values for both network emulators. Additionally, we compared experimentally acquired figures for videoBitrate groups within Mininet setting to similar categories obtained with Linktropy 5500 by applying Student's t-test for independent samples. That is, the first group of values resulted from Mininet experiments was compared to the first group from Linktropy 5500 and so on. Furthermore, we formulated a null hypothesis H_0 about the equity of two expectations. Proof of such hypothesis is based on normal distribution. Table III contains the obtained t_e values. All four empirical values t_e are less than Student's t-critical value under the chosen significance level ($p = 0.05$). Thus, the null hypothesis H_0 is accepted which means that the difference between average values from Mininet and Linktropy 5500 groups is insignificant under the above selected t-parameters.

TABLE III. STUDENT'S T-TEST RESULTS

Groups of Experiments	Bandwith	t_e
Group 1 (1-30 sec.)	1 Mbps	0,42
Group 2 (31-60 sec.)	2 Mbps	0,63
Group 3 (61-90 sec.)	3 Mbps	1,83
Group 4 (91-120 sec.)	1 Mbps	0,42

VI. CONCLUSION

In this paper we investigated how to deliver DASH-based content through Mininet environment which can be configured to interconnect both real and virtual networks. In addition, we presented results for a set of experiments aimed at studying the delivery of streaming video under varying bandwidth conditions. The aforementioned experiments were conducted using two different network emulators, namely SDN-based Mininet and specialized hardware-software Linktropy 5500. Our findings supported by statistical data analysis indicate that both solutions obtain comparable results. That is, establishing a bandwidth varying channel with both emulators yields similar effects with regard to stream bit rates that would be selected by the media player.

VII. FUTURE WORK

In our future research we are planning to incorporate more complex network topologies within Mininet environment. With this in mind, we could conduct more sophisticated experiments in the delivery of DASH-based content that would reflect large heterogeneous infrastructures typical for real-life scenarios. Furthermore, future endeavours aim to investigate the influence of various network impairments such as delay, packet loss, etc. on transmitted video streams. On top of that, it would be interesting to explore the frequency of bit rate switches in different network emulators. Additionally, we envision that Mininet can be viably used for emulating MPEG-DASH streams in Content Delivery Networks (CDN). A distinct advantage of Mininet is its suitability for large scale experiments where specified parts of the infrastructure can be assigned various network parameters.

Yet another research direction that we deem perspective is the development of tools which could simultaneously communicate with Mininet environment and APIs provided by the vendors of media players. Systems with such feature would be able to automatically conduct experiments for the delivery of MPEG-DASH content, taking into account different network impairments. Finally, having such functionality opens new opportunities for testing adaptive control algorithms implemented in current media players as well as developing new ones.

ACKNOWLEDGMENT

This work was supported by the Strategic Development Program of Petrozavodsk State University (2012-2016).

REFERENCES

[1] Cisco Visual Networking Index: Forecast and Methodology, 2014-2019 White Paper,
Web: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.

[2] HTTP Dynamic Streaming,
Web: http://www.adobe.com/ru/products/hds-dynamic-streaming.html.

[3] Smooth Streaming,
Web: http://www.microsoft.com/silverlight/smoothstreaming/.

[4] HTTP Live Streaming,
Web: https://developer.apple.com/streaming/.

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

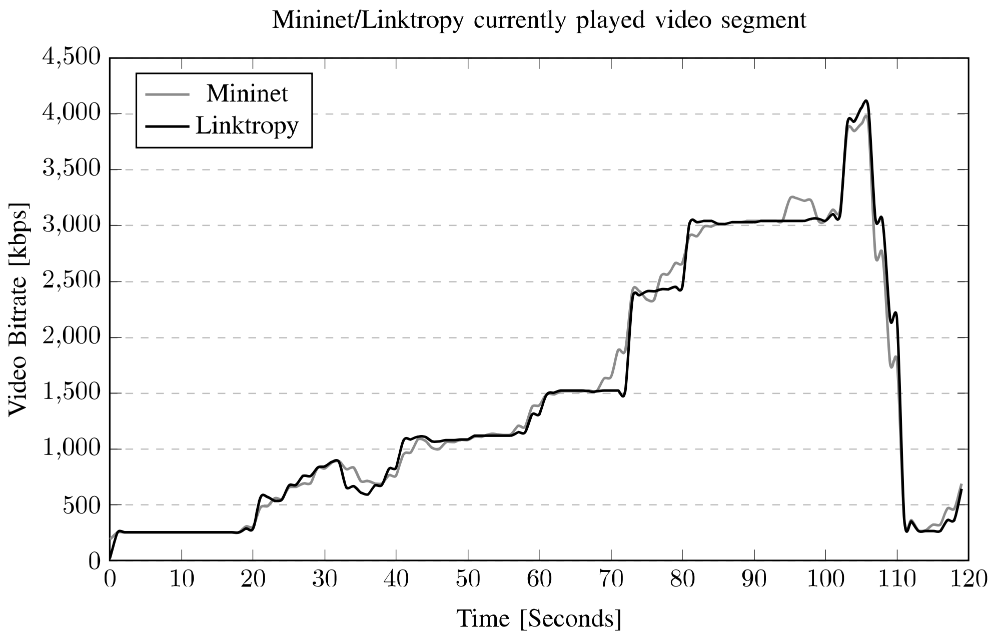

Fig. 5. Currently played video segment in kbps

[6] The Status of MPEG-DASH today, and why YouTube and Netflix use it in HTML5 and beyond,
Web: http://www.dash-player.com/blog/2015/02/the-status-of-mpeg-dash-today-and-why-youtube-and-netflix-use-it-in-html5/.

[7] Bitmovin turns hours into minutes by transcoding video fast using Google Compute Engine,
Web: https://cloud.google.com/customers/bitmovin/.

[8] Emulation,
Web: http://packetstorm.com/network-emulation/.

[9] Mininet Overview,
Web: http://mininet.org/overview/.

[10] Manar Jammal, Taranpreet Singh, Abdallah Shami, Rasool Asal, Yiming Li: "Software defined networking: State of the art and research challenges", *Computer Networks*, vol.72, 2014, pp. 74–98.

[11] Liyanage, M. and Gurtov, A. and Ylianttila, M., *Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture*. Wiley, 2015.

[12] WANem,
Web: http://wanem.sourceforge.net/.

[13] Linktropy 5510 WAN Emulator,
Web: http://www.apposite-tech.com/products/5510.html.

[14] Netem,
Web: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[15] T. Muto, Y. Wang, S. Awiphan, K. Kanai. J. Katto(UWA), "Evaluation of MPEG-DASH over CCNx over Different TCPs", Packet Video Workshop, Poster Session, San Jose, CA., December 13 2013.

[16] PacketStorm Communications: WAN Emulator and Simulator,
Web: http://packetstorm.com/.

[17] Stefan Lederer, Christopher Mller, Benjamin Rainer, Christian Timmerer, and Hermann Hellwagner, "An Experimental Analysis of Dynamic Adaptive Streaming over HTTP in Content Centric Networks", *in Proceedings of the IEEE International Conference on Multimedia and Expo 2013*, San Jose, USA, July, 2013.

[18] Linux Traffic Control,
Web: http://man7.org/linux/man-pages/man8/tc.8.html.

[19] HTB - Hierarchy Token Bucket,
Web: http://linux.die.net/man/8/tc-htb.

[20] B. Lantz, B. Heller, and N. McKeown. "A network in a laptop: rapid prototyping for software-defined networks", *In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 19:1–19:6.

[21] Chandan Pal, Veena S, Ram P. Rustagi and K.N.B.Murthy, "Implementation of Simplified Custom Topology Framework in Mininet", Computer Aided System Engineering (APCASE), 2014.

[22] J.J. Quinlan, A.H. Zahran, K.K. Ramakrishnan, C.J. Sreenan, "Delivery of adaptive bit rate video: balancing fairness, efficiency and quality", *Local and Metropolitan Area Networks (LANMAN), 2015 IEEE International Workshop on*, Apr. 2015, pp. 1–6.

[23] Brandon Heller, "Reproducible network research with high-fidelity emulation", Ph.D. Thesis, Stanford University, 2013.

[24] N. Gude, T. Koponen, J. Pettit, B. Pfa , M. Casado, and N. McKeown, "NOX: Towards an operating system for networks", *SIGCOMM Comput. Commun. Rev.*, vol.38, July 2008, pp. 105–110.

[25] T. Koponen, M. Casado, N. Gude, J. Stribling, Poutievski. L., M. Zhu, R. Ramanathan, Y Iwata, H. Inouye, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks", *In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, 2010.

[26] Beacon: a Java-based OpenFlow control platform,
Web: http://www.beaconcontroller.net/.

[27] The OpenFlow switch,
Web: http://www.openflow.org.

[28] Video Encoding Service for Adaptive Streaming,
Web: https://www.bitcodin.com/.

[29] Sintel movie,
Web: http://eu-storage.bitcodin.com/inputs/Sintel.2010.720p.mkv

[30] mininet.link.Intf Class Reference,
Web: http://mininet.org/api classmininet_1_1link_1_1Intf.html.

[31] mininet.link.TCLink Class Reference,
Web: http://mininet.org/api/classmininet_1_1link_1_1TCLink.html.

[32] Minievents framework,
Web: https://github.com/cgiraldo/minievents.

[33] Bitmovin bitdash player,
Web: http://www.dash-player.com/.

[34] Player Configuration,
Web: https://github.com/bitmovin/bitdash-developers/wiki/Player-Configuration.