Peeking under the Hood: How the Measurement Setup Influences the Video Streaming Behavior

Anika Schwind, Lea Janiak, Christian Moldovan, Florian Wamser, Tobias Hoßfeld
University of Würzburg, Institute of Computer Science
Würzburg, Germany
{anika.schwind | lea.janiak | christian.moldovan | florian.wamser | hossfeld}@informatik.uni-wuerzburg.de

***Abstract*—Global Internet video traffic will dramatically increase in the next years. With this rapid growth, interest in the application behavior of video streaming services and the resulting user experience rises. In particular, there is a need to understand the influence of system parameters on the streaming performance. Thus, several monitoring approaches have been developed which allow conducting automated measurements on a large-scale, for example, lightweight approaches for measurements running in the mobile networks or setups using a virtual frame buffer for servers without a display running in virtualized cloud environments. In some cases, the measurement hardware can totally be controlled, in other cases, there is no knowledge about parallel running services. In this paper, we answer the question whether and how much the measurement setup (e.g., virtualization, headless browsers, load on machines) influences the video streaming behavior. Therefore, we compare eight management setups with the ground truth (an end user watching a video on the own device) by evaluating the key performance indicators on application layer during video streaming, i.e. initial video playout delays and stalling. Our results reveal important insights: some of the common measurement setups heavily influence the measurements and must be avoided to collect reliable results.**

***Index Terms*—video streaming, monitoring, measurement, streaming behavior, measurement setup**

I. Introduction

Video Streaming has become one of the most popular Internet applications in the past decades. Video service providers want to optimize the application layer's Quality of Service (QoS) by optimizing their CDN architecture and their protocols. Internet service providers (ISPs) are interested in measuring their customers Quality of Experience (QoE). It is therefore necessary to measure a high number of videos and monitor the QoS and to characterize video streaming mechanisms. Virtual customers are often emulated in such measurement studies to investigate specific scenarios and to run multiple concurrent experiments. From a technical perspective, the virtualization may occur on several layers, e.g., operating system level, bare-metal hypervisors, hosted hypervisors. Virtual machines and containerization are good possibilities to run test beds since it is easy to set up the experiments' configuration.

In this paper, we investigate how the measurement setup and especially how limitations like CPU load affects the application layer QoS in HTTP video streaming for different virtualization methods. The most important QoS parameters in video streaming include the frequency and duration of stalling events, the average video quality [1], the initial delay [2], and the frequency of quality changes [3], [4]. According to user studies that are summarized in [5], the application layer QoS parameters stalling frequency and initial delay have the highest impact on the QoE in a video browsing scenario which is why we focus on them in this study. Our main research goal is to determine the impact of the virtualization, video rendering type, and the available computation resources on the performance of the video streaming application.

We conduct a measurement study to compare a non-virtualized system with a virtual machine and Docker with Xvfb, X window manager, and headless browser. We play a single YouTube video in different combinations of system configurations. Furthermore, we vary CPU load to investigate the influence on stalling, initial delay, and video quality.

The remainder of this paper is structured as follows. In Section II, we discuss technical background and related work. Section III describes our testbed and the design of the measurement study. In Section IV, we evaluate the results collected in the measurement study. Finally, we conclude our paper and give outlook to future work in Section V.

II. Background and Related Work

The ability to measure video streaming is important for ISPs and network operators to ensure that video content is rendered, transmitted, and received according to an appropriate quality standard. The challenge is to create a measurement setup that takes into account the most important quality parameters while not affecting the streaming behavior. Other criteria include the ability to realize the measurement in the cloud or using a virtual frame buffer or headless browser for servers without a display. The open question here is whether the respective measurement setup influences the streaming behavior.

According to [2], [6]–[9], the most important quality parameters for video streaming are perceptual factors such as waiting time and video quality as well as technical factors such as video codec or adaptation logic. Overall, the factors that a monitoring for video streaming should consider include video playback quality, playback interruptions (stalling), the level of adaptation (how drastically the perceived quality changes), the frequency of adaptation, and the initial delay. In the literature, solutions are proposed that make objective quality assessments by means of models [10], or application or network-layer monitoring solutions [11]–[15] that can be lightweight for

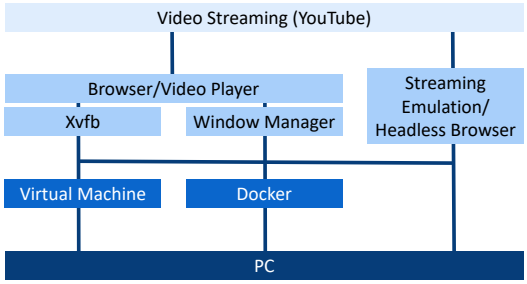

Fig. 1: Structure and composition of different measuring approaches for video streaming

TABLE I: Used measurement setups

Measurement Setup	Virtualization	Windowing System	Browser (Chrome)
PW	-	Window Manager	Standard
PX	-	Xvfb	Standard
PH	-	-	Headless
VW	VM	Window Manager	Standard
VX	VM	Xvfb	Standard
VH	VM	-	Headless
DX	Docker	Xvfb	Standard
DH	Docker	-	Headless

running in mobile networks or large-scale if virtualized in the cloud. This leads to different frameworks with different video streaming monitoring techniques.

The authors of [10] provide a comprehensive overview of approaches and standardization activities for IPTV assessment. In particular, objective assessment metrics are listed that are not subject of this study. YoMo [13], YoMoApp [11], YouSlow [12], and YouQ [14] measure streaming parameters on the application side directly without virtualization and partly without automation [11], [13]. Monitoring solutions deployed on the application side provide a reliable and accurate view of application layer quality parameters. While access to the client side or application provides insight into quality parameters, network operators usually have no access to such measurements and must limit themselves to network measurements. YOUQMON [15] estimates the streaming parameters in the network. [16] is an approach that can be started in the cloud and therefore uses virtualization. Virtualization has the advantage that multiple measurements can be easily started but has with the disadvantage of not knowing how much is being worked on in parallel on the whole system. All frameworks that use headless browsers [17], [18] have the advantage that, in addition to operating on servers without a screen, the measurement can also often be automated.

Figure 1 breaks down the measurement approaches and their methodology. Here, we assume that measurements are conducted on a PC. If virtualization is desired, a Virtual Machine or a Docker container can be run on top of it. In addition, for the three underlying systems (PC, VM, and Docker), it can be chosen if approaches with or without actual playback of the video content should be used (browser/video player vs. streaming emulation/headless browser). By using a browser/video player, the way in which it is rendered in the computer may also differ, for example using Xvfb [19] or native window manager. Xvfb is a window managers that renders browser or player video presentations in memory only, without requiring a real display for the measurement.

[16] uses Docker as a virtualization to monitor large scale QoE video streaming. [17], [18] uses Xvfb to evaluate video player performance or Spotify streaming. A comparison of different measurement approaches for Docker/VM, Xvfb/headless browser can be found in [21]–[23].

III. Methodology

To measure the impact of measurement setups on the behavior of video streaming services such as YouTube, a measurement testbed was build and eight different setups have been measured. Here, each of the measurement setups is used to run a video while the key performance indicators (KPIs) of the streaming are monitored on application layer. In addition, the testbed is used to limit the available CPU to further investigate the influence under stressed conditions.

This section gives an overview of the testbed components, the execution of the measurements, the collected data, and the way in which CPU limitations are achieved.

A. Testbed Components

In our testbed, eight different measurement setups can be compared. Table I lists all setups, starting with the abbreviation, followed by their used hardware, virtualization type, windowing system, as well es the type of the used Chrome browser. The setups starting with `P` run directly on the PC without a virtualization layer, respectively with a window manager (`PW`), Xvfb (`PX`), or a headless browser (`PH`). Similarly, setups starting with `V` use a VM. Lastly, for setups which start with `D`, Docker is used. For Docker, we did not measure a setup using a window manager as it is not common to use this combination. In the following, all measurement setups will be referred to using their abbreviations.

In later comparison, the measurement setup `PW` will be used as a baseline, since it's the normal setting for an end user watching a video. Thus, every deviation from its behavior can be interpreted as influence of the measurement setup on the application behavior.

1) Hardware and Underlying Software: The PC used as the basis for all measurement setups has a Intel® Core™ i5-7600 @ 3,50 GHz processor with four cores, a 512 GB SSD hard drive, and 16 GB RAM. The operating system Ubuntu 18.04.01 LTS is utilized. resolution of the monitor used in setups with a window manager is 1280x1024. For video streaming, the free browser *Google Chrome* (Version 68) is chosen. The browser automation tool *Selenium* is utilized to control Chrome during the measurements. For this, a *Chromedriver* (Version 2.41) is required.

2) Virtual Machine: The virtual machine *Oracle VM VirtualBox* version 5.2.10 for Ubuntu is used. The VM is configured to resemble the host PC as closely as possible to minimize potential sources of differences in the results. Mainly, the same operating system, software, and scrips are used.

3) Docker: The used *Docker* container is based on the container presented in the work of Schwind et al. [17]. Here, the Docker version 18.06.0-ce was used. Some alterations were made to this container to match the requirements of this work. Thus, Google Chrome is used as a browser instead of Mozilla Firefox and the option to run the browser in headless mode was added in addition to Xvfb.

4) Xvfb and Headless Browser: The display server *Xvfb* was used to simulate a physical screen in some of the measurement setups. In setups using a headless browser, the same Google Chrome version like in the PC measurements was used with the addition of running them using the headless option. The size of the virtual display for Xvfb and the headless browser were manually being set to the same size as the physical monitor (1280x1024) to exclude possible variations in the measurement runs.

5) CPU Limitation: To measure the performance of the different measurement setups with less available CPU, the CPU was limited to a constant value. For setups without virtualization or a VM, the *stress-ng* tool[1] is used, which stresses the CPU by performing resource intensive computations. To make sure, that the other applications have no access to the occupied resources, the niceness of the process is set to -20, which is the highest priority. Therefore, it is given the most CPU time.

Because of the way resource access is handled in Docker, even with a low niceness the *stress-ng* tool doesn't have priority over the processes running in the Docker container. Therefore, the access the container gets to the CPU has to be set by limiting its resources directly, by passing the flag `--cpus=<value>`, to make sure that the container is only able to access the desired amount.

B. Measurement Design and Collected Data

All measurements were run on the same machine during two weeks in October 2018. In total, we conducted 2,000 measurements in this time. For each of the eight setups we measured the performance under optimal conditions (no limitation), and limited resources, having a CPU stressed to 50 %, 75 %, 90 %, and 95 %. Each of these settings was repeated 50 times to get reliable results.

In our measurements, each setup was used to play a specific YouTube video[2]. The video has a duration of 60 seconds and is chosen because it is available in many resolutions, reaching from 144 p up to 4,320 p (8k), and because it does not include advertisements, which could distort the results. The video was played with the player set to "wide" mode, since in the "normal" mode it only takes up a fraction of the screen size. During the measurements, KPIs on application layer are monitored using an injected JavaScript file.

On the application layer, several key performance indicators (KPIs) are collected during the video playback. This includes the duration of the initial delay, as well as information about the number and the length of occurred stalling. As the JavaScript, which collects these information, can only be injected after the web page is partially loaded, it is possible that the starting time and the first seconds of the video cannot be monitored (hereafter referred to as monitoring delay). Thus, we linearly recalculate the starting time of the from the playback time. Here, we assume that in this starting phase, no stalling occur. In addition, the played out quality level is monitored. Here, the starting quality as well as the time on each quality level is considered. Connected to this, the number of adaptations and the quality levels, between which the switches occur, are compared. Additionally, on the hardware layer, the CPU usage of each setup is recorded, both directly on the host PC, as well as in Docker and the VM.

IV. Evaluation

In this section, the presented eight different measurement setups are compared on the base of their monitored KPIs on application layer. First, we focus on measurement results which were collected under optimal conditions. Afterwards, the influence of the CPU utilization on the different setups is evaluated.

A. Comparison under Optimal Conditions

To get information about the comparability of the results of different measurement approaches, we evaluated their KPIs on application layer and compared them to the normal end user setting. This setting is, in our example, a PC displaying the video with a standard window manager (here, setup `PW`). In this evaluation, we focus on initial delay, total stalling time (collective length of all occurred stalling), and the used quality layer during the video streaming.

Table II summarizes the most important finding by providing the mean of the initial delays, the standard deviation of the initial delays, mean quality, mean stalling time, and the standard deviation of the stalling time per measurement setup for optimal measurement conditions. The mean initial delay of the baseline setup (`PW`) is 0.60 s, while the delays for the other measurement setups varies from 0.54 s for Docker using Xvfb (`DX`) up to 1.21 s for VM using window manager (`VW`), which is more than twice as high than the initial delay of the baseline. The monitoring delay is very short (less than 1.5 s) and thus, it is negligible. Within each measurement setup, the standard deviation of the initial delay is very low and thus, no wide dispersion was measured within the measurement setups. To further investigate the initial delay and the differences between the measurement setups, we later have a more detailed look at the distribution of the initial delays. Looking at the played out quality, not many differences between the setups can be investigated. Here, the following quality layers were observed with the respective ordinal rank of the resolution in brackets:

[1] http://manpages.ubuntu.com/manpages/bionic/man1/stress-ng.1.html
[2] https://www.youtube.com/watch?v=RJnKaAtBPhA

TABLE II: Monitored KPIs under optimal conditions

Measurement Setup	Mean Init. Delay [s]	SD Init. Delay [s]	Mean Quality	Mean Stalling [s]
PW	0.60	0.07	4.98	0.00
PX	0.70	0.07	4.98	0.00
PH	0.73	0.11	5.00	0.00
VW	1.21	0.15	4.95	0.00
VX	0.86	0.13	4.92	0.00
VH	0.79	0.09	4.96	0.00
DX	0.54	0.05	4.94	0.00
DH	0.58	0.07	4.98	0.00

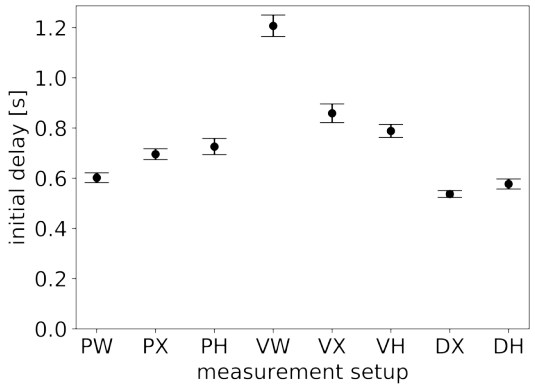

Fig. 2: Initial delay under optimal conditions

720 p (5), 480 p (4), 360 p (3), 240 p (2), and 144 p (1). As the mean quality for all setups is close to 5, most of the time videos were played out in 720 p. The same can be said about stalling. For all measurement setups under optimal conditions, no stalling occurred.

To evaluate the differences in the initial delays in more detail, Figure 2 shows their distribution. The x-axis presents the different measurement setups, while the y-axis shows the initial delays in second. For each setup, the 95 % confidence intervals are marked as well as the means. Even the longest initial delay, for VM using the window manger (VW), is short enough to no influence the users QoE, according to the work of Hoßfeld et al. [2]. Nevertheless, the initial delay of all setups, except for setup DH, significantly differs from the initial delay of the baseline measurement setup PW as the confidence intervals do not overlap. As the standard deviations for all setups are low, the difference to the initial delay of the baseline PW can be subtracted and thus, the results can be used for evaluation.

To summarize the comparison on optimal conditions, significant differences between the measurement setups can only be seen concerning the initial delay. To go more into detail, we wanted to have a look at other factors, which possibly influence the performance of the setups. Thus, we analyzed the CPU utilization of each setup in Figure 3. Here, the different measurement approaches are shown at the x-axis and ratio of

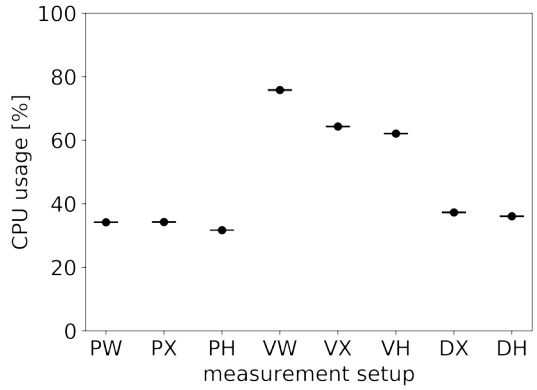

Fig. 3: CPU utilization under optimal conditions

CPU utilization on the y-axis. In all settings, the utilization was measured at the base machine (not within the VM or Docker Container). For each measurement setup, the 95 % confidence intervals are shown as well as the means per setup. For setups using no visualization, the CPU utilization is low, ranging between 31.27 % (PH) and 35.22 % (PX). Similar results can be seen for the Docker setups: Docker using a headless browser (DH) has a mean of 36.08 % and Docker using Xvfb (DX) a mean of 37.32 % CPU usage. A clear difference can be seen for the CPU utilization for setups using virtual machines. Here, the means were appreciably higher, ranging between 60.56 % for VM using a headless browser (VH) and 77.04 % for VM using a window manager (VW). For all setups, the confidence intervals are very low.

B. Comparison under Different CPU Utilizations

In the previous evaluation, we found that, on application layer, no significant differences are noticeable, except for the initial delay. Nevertheless, it was shown that the CPU utilization of the measurement setups heavily differed. In these measurements, no setup reached CPU utilization of 100 % as we used a well-equipped machine for our measurements. This fact can not be assumed for all measurement settings. If, for example, measurements are done in the field using small nodes, the available computing power can be relatively low. Likewise, if the used server or machine is not under full control of the conductor of the measurement, like on external servers, it is possible that other software runs beside the measurement, which can also reduce the available CPU. Thus, it is important to also have a look at the influence of the CPU utilization on the measurement setups. For that reason, we conducted additional measurement runs where we stress the CPU to 50 %, 75 %, 90 %, and 95 %. Here, again, we monitored the KPIs on application layer to compare the reliability of the measurement setups. We found that the limitation of the CPU leads to clear deviations in the KPIs from the normal use case for a end user presented in the previous section. Thus, in this section, the differences are highlighted and the usability for the measurement setups for given CPU utilization are evaluated.

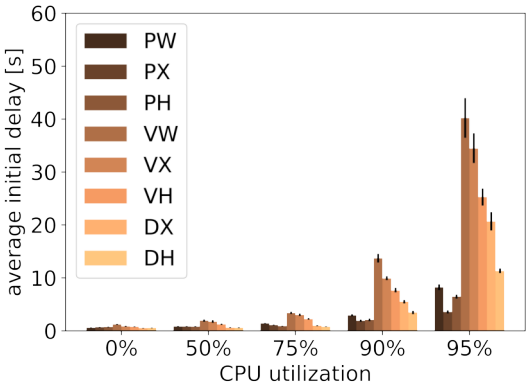

Fig. 4: Initial Delay at different CPU stress levels

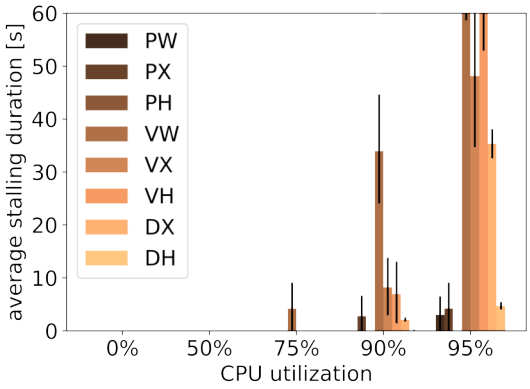

Fig. 5: Stalling at different CPU stress levels

For increasing CPU utilization, the monitoring delay increases drastically. Having an average monitoring delay of less than 1.5 s for all setups without additional CPU stress, it goes up to of 2.5 s for 50 %, 4.7 s for 75 %, 14.6 % for 90 %, and 37.5 % for 95 %. Here, for all stress levels, the measurement setups VW, VX, and VH reach the highest delays, having more than twice as high delays as the rest. This shows that high CPU utilization not only influences the streaming behavior, but also the accuracy of the monitoring tool. The system is heavily overloaded that is not possible to run all processes correctly. Consequently, the higher the CPU stress level, the more inaccurate the calculated values for initial delay and stalling. Nevertheless, we calculated these values and counted possible stalling in the monitoring delay period to the initial delay to estimate the influence of the setup on the streaming behavior.

The average initial delay can be seen in Figure 4. The x-axis shows the five CPU stress levels and the y-axis the average length of the initial delay in seconds. The colored bars represent the different measurement setups. On top of each bar, the 95 % confidence intervals are shown. Here, a clear trend is visible: With less CPU power available, the average initial delay increases, showing significant jumps for CPU stress level of 90 % and 95 %. For the measurement setups using a normal PC setting (PW, PX, and PH), as well as for the setup using Docker with a headless browser (DH), this increase can be subtracted if the CPU utilization is monitored. Here, the standard deviations are below 2 s and thus do not vary widely. This can not be said for the setups using a VM (VW, VX, and VH) and Docker using Xvfb (DX). In these setups, the results varied widely for each setting for stress levels of 90 % and 95 %, showing standard deviations of up to 56.41 s for VW, and thus, can not be subtracted. Thus, concerning the initial delay, measurement setups using a VM and setup DX are not usable on machines which have a high CPU utilization. For other setups, the influence on the initial delay can be subtracted if the CPU utilization is monitored during the measurement and the monitoring delay is small.

Comparing the used quality layers, no differences to the baseline can be noticed up to a CPU utilization level of 75 %. For a CPU utilization of 90 %, again the measurement setup using a VM and window manager (VW) differ from the rest. Here, the mean quality is 4.47, which means 480 p on average. For a CPU utilization of 95 %, all measurement approaches using a VM clearly behave different than the baseline use case. For VW the mean quality was 3.73, for VX 3.81, and for VH 3.60. For all setups, a mean quality of 480 p is measured, compared to 720 p (5) for the normal use case. Thus, little available CPU can highly influence the adaptation logic of the YouTube streaming and lead to distortion of the measured results, even without any changes of the bandwidth.

Having a look at the total stalling times, again an influence of the CPU utilization and the selected measurement setup is noticeable, as can be seen in Figure 5. Here, the x-axis shows the five CPU stress levels while the y-axis indicates the average stalling times in seconds. The colored bars represent the eight different measurement setups. As stalling times of more than the video duration are not acceptable, the y-axis is cut at 60 s. On top of each bar, the 95 % confidence intervals are shown. For a stress level of up 75 %, no stalling occurred, except for the VM using the window manager (4.21 s on average with a standard deviation of 16.77 s). This changes when looking at stalling times at the stress levels 90 % and 95 %. Here, for 90 % CPU stressed, only the measurement setups PW, PH, and DH showed no stalling. For all other setups, stalling, ranging between 2.10 s (DX) and 33.94 s (VW), are measured. Looking at the results with a 95 % stressed CPU, only for measurement setup PH no stalling occurred, all other setups show stalling. Furthermore, it must be considered that the total stalling times could be even higher, as the monitoring delay for high CPU utilization is very high. Stallings which occurred in this time were counted to the initial delay. In Addition, whenever stalling occurred, the standard variation was relatively high, starting from 2.27 s (DH) up to 56.41 s (VW). As the standard deviations show that even under the same settings, it is not clear that the same results can be measured for the same measurement setup, it is not possible to subtract this variation.

V. Conclusion

To measure the performance of video streaming services, several measurement setups are used by ISPs, streaming providers, and researchers in order to understand the influence of various system parameters. Therefore, approaches with and without actual application content playback and with and without virtualization are used for large-scale measurements. The question arises if the obtained measurement results are comparable with the application performance of a regular video streaming user.

To investigate the comparability and capability of different approaches, in this paper, we evaluated the influence of eight common measurement setups on the application behavior of video streaming. More specifically, we compared approaches using a normal PC, using a virtual machine (VM), and using a Docker container as basis. For each monitoring approach, we considered different ways to play a video: playback using a standard window manager, playback within Xvfb, as well as playback using a headless browser. In many settings the measurement hardware is not well equipped (e.g., for mobile measurements) or the machine is not completely under control of the conductor of the measurements (e.g., using a server in the cloud). Thus, in addition, we evaluated the influence of the CPU utilization on the streaming behavior for different setups.

For measurements running under *optimal conditions* (i.e. without CPU utilization from other processes), only minor differences were monitored in terms of the initial delay. Concerning stalling and the used quality layers, no differences were found for the setups. However, when having a look at the monitored results using different CPU stress levels, clear differences can be seen. Here, especially starting from a *CPU utilization of 90 %*, all measurement setups using a virtual machine as well as measurements using Docker combined with Xvfb are not recommended. Only caused by the limitation of the CPU, without changing the bandwidth or other settings, the streaming behaves significantly different to the normal use case, i.e. a regular user consuming a video. For high CPU loads, the initial delay increases, stalling occur, and the selected quality by the player is changed.

In summary, the measurement setup plays an important role for the reliability of the results. We conclude that wherever possible, virtual machines should be avoided. If virtualization is mandatory, we recommend using Docker in combination with a headless browser. In addition, for all measurements, the utilization of the CPU should always be monitored to ensure that the CPU is lightly utilized (e.g. $<50\,\%$) and does not influence the results.

References

[1] C. Alberti, D. Renzi, C. Timmerer, C. Mueller, S. Lederer, S. Battista, and M. Mattavelli, "Automated qoe evaluation of dynamic adaptive streaming over http," in *Quality of Multimedia Experience (QoMEX), 2013 Fifth International Workshop on*. IEEE, 2013, pp. 58–63.

[2] T. Hoßfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen, "Initial delay vs. interruptions: Between the devil and the deep blue sea," in *Quality of Multimedia Experience (QoMEX), 2012 Fourth International Workshop on*. IEEE, 2012, pp. 1–6.

[3] M. Zink, J. Schmitt, and R. Steinmetz, "Layer-encoded video in scalable adaptive streaming," *IEEE Transactions on Multimedia*, vol. 7, no. 1, pp. 75–84, 2005.

[4] L. Yitong, S. Yun, M. Yinian, L. Jing, L. Qi, and Y. Dacheng, "A study on quality of experience for adaptive streaming service," in *Communications Workshops (ICC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 682–686.

[5] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-Gia, "A survey on quality of experience of http adaptive streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.

[6] F. Wamser, P. Casas, M. Seufert, C. Moldovan, P. Tran-Gia, and T. Hossfeld, "Modeling the youtube stack: From packets to quality of experience," *Computer Networks*, vol. 109, pp. 211–224, 2016.

[7] R. K. Mok, E. W. Chan, and R. K. Chang, "Measuring the quality of experience of http video streaming," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE, 2011, pp. 485–492.

[8] T. Hoßfeld, M. Seufert, M. Hirth, T. Zinner, P. Tran-Gia, and R. Schatz, "Quantification of youtube qoe via crowdsourcing," in *Multimedia (ISM), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 494–499.

[9] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "Sdn-based application-aware networking on the example of youtube video streaming," in *Software Defined Networks (EWSDN), 2013 Second European Workshop on*. IEEE, 2013, pp. 87–92.

[10] A. Takahashi, D. Hands, and V. Barriac, "Standardization activities in the itu for a qoe assessment of iptv," *IEEE Communications Magazine*, vol. 46, no. 2, 2008.

[11] F. Wamser, M. Seufert, P. Casas, R. Irmer, P. Tran-Gia, and R. Schatz, "Yomoapp: A tool for analyzing qoe of youtube http adaptive streaming in mobile networks," in *Networks and Communications (EuCNC), 2015 European Conference on*. IEEE, 2015, pp. 239–243.

[12] H. Nam, K.-H. Kim, D. Calin, and H. Schulzrinne, "Youslow: a performance analysis tool for adaptive bitrate video streaming," in *ACM SIGCOMM Computer communication review*, vol. 44, no. 4. ACM, 2014, pp. 111–112.

[13] B. Staehle, M. Hirth, R. Pries, F. Wamser, and D. Staehle, "Yomo: a youtube application comfort monitoring tool," *New Dimensions in the Assessment and Support of Quality of Experience for Multimedia Applications, Tampere, Finland*, vol. 6, 2010.

[14] I. Orsolic, D. Pevec, M. Suznjevic, and L. Skorin-Kapov, "A machine learning approach to classifying youtube qoe based on encrypted network traffic," *Multimedia tools and applications*, vol. 76, no. 21, pp. 22 267–22 301, 2017.

[15] P. Casas, M. Seufert, and R. Schatz, "Youqmon: a system for online monitoring of youtube qoe in operational 3g networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 2, pp. 44–46, 2013.

[16] A. Schwind, M. Seufert, O. Alay, P. Casas, P. Tran-Gia, and F. Wamser, "Concept and implementation of video qoE measurements in a mobile broadband testbed." in *TMA*, 2017, pp. 1–6.

[17] A. Schwind, F. Wamser, T. Gensler, P. Tran-Gia, M. Seufert, and P. Casas, "Streaming characteristics of spotify sessions," in *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, 2018, pp. 1–6.

[18] D. Stohr, A. Frömmgen, A. Rizk, M. Zink, R. Steinmetz, and W. Effelsberg, "Where are the sweet spots?: A systematic approach to reproducible dash player comparisons," in *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 2017, pp. 1113–1121.

[19] "XVFB." [Online]. Available: https://www.x.org/archive/X11R7.6/doc/man/man1/Xvfb.1.xhtml

[20] "Xdummy." [Online]. Available: http://xpra.org/trac/wiki/Xdummy

[21] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2015, pp. 171–172.

[22] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon, and B.-J. Kim, "Performance comparison analysis of linux container and virtual machine for building cloud," *Advanced Science and Technology Letters*, vol. 66, no. 105-111, p. 2, 2014.

[23] A. M. Joy, "Performance comparison between linux containers and virtual machines," in *2015 International Conference on Advances in Computer Engineering and Applications*, Mar. 2015, pp. 342–346.