SDQ: Enabling Rapid QoE Experimentation using Software Defined Networking

Lyndon Fawcett[1], Mu Mu[2], Matthew Broadbent[1], Nicholas Hart[1], and Nicholas Race[1]

[1]School of Computing and Communications, Lancaster University, UK
[2]School of Science and Technology, The University of Northampton, UK

***Abstract*—The emerging network paradigm of Software Defined Networking (SDN) has been increasingly adopted to improve the Quality of Experiences (QoE) across multiple HTTP adaptive streaming (HAS) instances. However, there is currently a gap between research and reality in this field. QoE models, which offer user-level context to network management processes, are often tested in a simulation environment. Such environments do not consider the effects that network protocols, client programs, and other real world factors may have on the outcomes. Ultimately, this can lead to models not functioning as expected in real networks. On the other hand, setting up an experiment that reflects reality is a time consuming process requiring expert knowledge. This paper shares designs and guidelines of an SDN experimentation framework (SDQ), which offers rapid evaluation of QoE models using real network infrastructures.**

***Index Terms*—Software Defined Networking; Quality of Experience; Experimentation; Adaptive Streaming**

I. Introduction

High quality video streaming accounts for a large portion of today's Internet traffic [3], with HTTP adaptive streaming (HAS) established as the predominant method of streaming content across networks to heterogeneous devices. On its own, HAS offers improvements in Quality of Experience (QoE), as the video delivered is adaptive to the prevailing network conditions [8]. However, when other devices share the same network, fluctuations can occur [5], [6], [9]. Software Defined Networking (SDN) has opened up a wealth of opportunities for improving QoE [16], [8]. This is achieved through a greater level of control and awareness of the behaviour of the underlying network. However, one of the current issues within the QoE community relates to the popularity of simulation environments, particularly for the evaluation of QoE models [21] [17]. Simulations can overlook the effects of network protocols, client programs, or other real world factors and ultimately can lead to models not behaving as anticipated in real networks [7]. On the other hand, setting up an experiment environment that reflects real network configurations is a time consuming process requiring expert knowledge. In this paper we go beyond the use of SDN to simply improve QoE, but consider how SDN can in fact support the rapid experimentation and testing of QoE models within a real network environment, thus addressing the aforementioned issue. We consider OpenFlow as a means to supporting innovation in QoE research in much the same way that it was originally conceived as a mechanism to enable researchers to innovate within networks.

In this paper, we present *SDQ*, a framework that uses SDN to provide an experiment automation and network enforcement eco-system for the design and validation of QoE models with real network infrastructures and clients. SDQ orchestrates the components of the experiment, including the networks and the nodes, and interfaces them with a QoE model. This is achieved through a framework that controls the OpenFlow protocol which is used to connect and configure the network environment. Moreover, it controls the virtualisation infrastructure, such as OpenStack, which configures the nodes. SDQ passes information and control from the network and the nodes to the QoE model so that it can process and enforce changes that need to be made in the network. This paper highlights some of the challenges of integrating QoE models using hardware SDN equipment and shares experiences in realising a SDN experimentation testbed. We also demonstrate the benefit of SDQ using an existing QoE model [17].

II. Background and Related work

HTTP Adaptive Streaming (HAS) is widely adopted for video distribution. Using MPEG-DASH as an example, media content is encoded into *representations*, each of which is a version of the content prepared in *segments* using different encoding specifications. The adaptation logic (between representations) is often determined at the player, with decisions based upon criteria such as estimated throughput [12] and buffer occupancy [10].

There has been significant effort placed in improving the QoE of adaptive video streaming. One solution uses cross-layer interaction between TCP and HTTP to provide better adaptation metrics [9]. Tian and Liu [22] used throughput-prediction algorithms to reduce video rate fluctuations. Mansy et al. [15] studied DASH's bursty nature and proposed adjusting DASH's buffering behaviour to keep the size of the client's receiver window low. Huang et al. introduced a buffer-based approach to rate adaptation to reduce the re-buffer rate in HAS streaming [10]. A client-side rate-adaptation algorithm for HAS is also introduced in [13]. Most of the aforementioned work focuses on optimising the network efficiency or the QoE on individual media streams. Without coordination between HAS clients, TCP-based resource allocation can lead to unfairness at the user level [17] and severe fluctuations in multi-stream environments [13]. With the increasing number of HAS streams, it is essential to orchestrate network resource consumption through 1) a better understanding of the user-level requirements of user applications, and 2) a transparent network resource allocation service in content networks.

Software Defined Networking [23], through the decoupling of the control plane from the packet forwarding plane, allows network services such as load balancing and context aware resource allocation to be realised and automated as part of the service delivery chain [16]. This has led to a significant body of research investigating the use of SDN for QoE assessment and improvement. Liotou et al. exploit the SDN global resource view and complementary QoE metrics to assure the desired performance for OTT applications in LTE networks [14]. The intelligent use of network data to facilitate the optimal use of network resources for QoE provisioning in the context of 5G network architecture is discussed in [4]. Jarschel et al. look at how YouTube streaming can benefit from SDN-based application-aware networks [11]. Nam et al. proposed a SDN application to monitor streaming flows in real time, dynamically changing the routing paths using MPLS traffic engineering for better user experiences [19]. We first demonstrated the feasibility of a SDN-assisted video quality management framework in [8]. Through fairness modelling using video quality, switching impact and cost efficiency as the impact metrics [18], a scalable resource allocation model *UFair* was introduced. This is designed to improve delivered video quality and user-level fairness between HAS media streams [17]. In particular, UFair seeks to reduce the frequency of adaptations over a group of clients, and moderate individual clients' choice of stream bandwidth, to the benefit of other clients. UFair operates by monitoring network status and "capping" resources on individual media streams, with the assumption that media clients can adapt their bandwidth utilisation in response to network constraints. Therefore, resource allocation can be achieved transparently in the network.

QoE models are often proven using programmatic simulation. Simulations overlook some networking and client aspects, which result in models not working as expected in real networking environments [7]. They are used because they can be created in a short space of time, adapted quickly, and run rapidly, all on a single device. In contrast, to test a QoE model in a real-world environment requires multiple switches and even more clients, all of which consume more resources and time than a simulation. The network emulator Mininet is often considered as an appropriate solution to the aforementioned problems; it is capable of running at large scale, uses a real networking stack, and can execute client programs [1]. However, at the moment it is limited by the underlying capabilities that the software switch (Open vSwitch) provides. In particular, Open vSwitch is missing some of the extended features of OpenFlow 1.3 that are important for QoE, such as *metering* (rate-limiting of flows). This highlights the need for a system that can balance these requirements: an SDN and virtualisation environment that provides the realism of real world experimentation whilst offering the benefits in agility and low cost associated with simulation environments.

III. SDQ Framework

In the following section, we describe the requirements and architecture of the *SDQ Framework*; a harness to be used in aiding rapid deployment and orchestration of experiments. As such, the architecture and experimentation environment has to fulfil the following requirements:

- *Experiments Close to Practice and at Scale*: To provide both realism and scale, the environment should encompass both physical and virtual elements.
- *Software Defined Dynamic Manipulation of Available Bandwidth Within the Network*: To match real-world network topologies, the link speeds must be rate limited. For QoE enforcement, dynamic configuration of rate limiting also needs to be applied to specific flows.
- *Configurable Clients*: The client's configuration should be changeable (automated based on test manifests) between experiments.
- *Rapid Repeatability of Experiments in a Clean Environment*: Ensure that no residual effects are left over from previous experiments. This may not be straightforward when the experiment involves hardware equipment.
- *Generic Framework*: The system should offer control and statistics from physical networks and any virtual infrastructure.

The SDQ framework orchestrates the virtual and physical network infrastructure using SDN to assist the execution and statistical data gathering of network based QoE experiments. It consists of a three layer architecture: the top layer contains components provided by the researcher including the test manifest and application/user-level functions such as a QoE module. The middle layer contains the SDQ orchestrator which interfaces with, and includes, the infrastructure managers. The bottom layer contains the network and virtualisation infrastructure where the experiments are deployed. The following is a breakdown of the components shown in Figure 1:

Test Manifest: describes the experiment in a JSON format. It includes each of the clients' IP addresses, the networks each is attached to, the virtual machine image to be used, network emulation requirements, and timestamps for automated tests.

QoE Model: an interchangeable component which communicates with the SDQ orchestrator (described below) through an RPC (Remote Procedure Call) interface providing information about resource allocation of one or more media flows. Additionally, information is sent back in regards to the current throughput at different points in the network using OpenFlow's *meter statistics* and *flow statistics* messages.

SDQ Orchestrator: handles communication between all of the components. It includes two subcomponents, the *virtual infrastructure manager (VIM)* and *network infrastructure manager (NIM)*. The VIM controls the virtualisation infrastructure through a RESTful API, it launches and configures experiment nodes with information from the test manifest. At the end of the experiment it resets the test environment, removing networks and virtual machines it instantiated, so that the

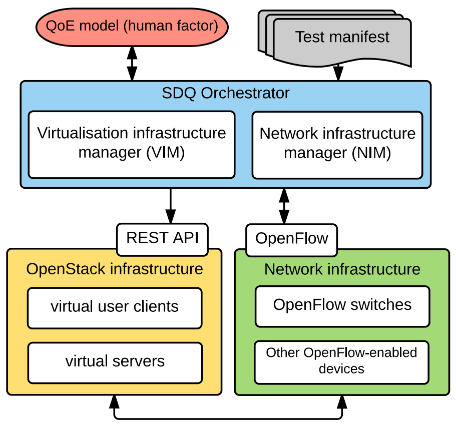

Fig. 1. SDQ framework

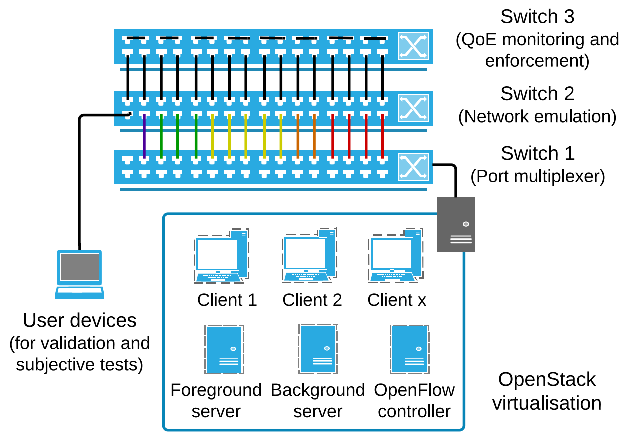

Fig. 2. Experiment environment

environment is ready for the next experiment. The NIM controls the network infrastructure and consists of an OpenFlow controller containing a metering and monitoring application. It installs meter flow mods on request from the QoE model and provides information from the network including current throughput of flows and switches.

The virtualisation infrastructure (through integration with OpenStack) instantiates experiment nodes and exposes them to the network. The network infrastructure creates connections between nodes and switches and provides a platform for QoE-based flow enforcement. Section IV provides further details regarding the configuration of each of these elements.

IV. Experimentation Environment

This section describes the virtualisation and network infrastructure used to create the environment for SDQ.

Our objective is to create network and virtualisation infrastructures that are controllable through the SDQ framework[1]. Ultimately, these need to resemble real world deployments. To illustrate the advantages of SDQ, we highlight a potential deployment scenario for evaluating QoE. This consists of a number of households connected to the Internet using consumer ADSL connections. They all share a common DSLAM, which is connected to the wider Internet over a restricted shared link. The video servers are located at the remote end of this connection. The home routers and the local DSLAM are also under SDN control to provide link emulation and QoE enforcement functionalities.

To realise the topology described above we create the experimentation environment shown in Figure 2. It consists of a large cluster of generic servers, three OpenFlow-capable switches, a user device, and local connectivity between each. This provides an environment for conducting a wide range of experiments involving the interactions between network endpoints (e.g., servers, clients, and middle-ware), and network elements (e.g., routers, switches and physical network links).

[1]https://github.com/LancasterNetworking/SDQ

In the following subsections, we outline the process undertaken to establish our experimental environment so that readers can replicate our experiments.

A. Virtualisation Infrastructure

At the core of the virtualisation infrastructure is an OpenStack installation. This provides the means of building and connecting virtual machines (VM). The OpenStack installation is standard, with one main modification: VLAN trunks are used to break-out network interfaces from virtual machines. These are then mapped one-to-one to exclusive physical interfaces on a switch. This is an essential feature for our experimentation as it allows each client to be directly assigned to a physical port on an SDN controlled switch.

We refer to this process as *port-multiplexing*, as it allows an Ethernet switch to implement remote physical interfaces for virtualised machines. The mechanism for this is based on the use of VLANs to carry VM traffic onto the switch responsible for realising this port-multiplexing. The configuration is such that each VM is allocated an exclusive OpenStack (Neutron) network, whose definition uses the *segmentation ID* parameter to specify the actual VLAN tag to be used on the trunk link between hypervisor host and port-multiplexing switch. Corresponding settings are needed in OpenStack and the underlying network configuration.

A typical experiment requires 10s to 100s of OpenStack configuration elements, most of which are replicas of a basic template. Building topologies by hand, either via the *Horizon* GUI, or using command line, is tedious and prone to error; the ability to both build and destroy test topologies rapidly and consistently is essential both for replicating, revisiting or extending experimental results and for sharing the test environment with other workers. As such, we developed an orchestration tool titled *MiniStack*[2], which takes topology description files and builds and boots fully connected experimental topologies. The tool is written in Python and uses the OpenStack REST API.

[2]https://github.com/hdb3/ministack

B. Network Infrastructure

The network infrastructure used in this example consists of two OpenFlow v1.3 capable switches (Switch 2 and 3 shown in Figure 2) with metering support. In our facility, we use Hewlett Packard Enterprise's HP3800 switches, as they fulfil both of these requirements. However, other compliant switches could be used instead. The HP3800 also hosts other important capabilities, such as the ability to flexibly partition a single physical switch into a number of virtual switches. Each of these is a complete, distinct OpenFlow instance. This too is outside of the scope of OpenFlow, but is a feature present on a number of devices available on the market. This partitioning feature is vital in achieving the scale required in experimentation without incurring the associated cost.

It is important to note that the metering feature required in our experimentation is not available in any production ready software-based switches (at time of writing), such as Open vSwitch. Similarly, the OpenFlow specification is not specific on how this metering functionality should be implemented. For example, the support in the switches used in our experimentation use a rapid discard policy. This simply drops packets once a threshold is exceeded. The alternative technique would involve queuing packets. Enabling these behaviours to be configured allows for the greatest flexibility. However, it would require additional functionality to be implemented on the devices and subsequently described within the protocol specification. The lack of specification could result in different metering behaviour between OpenFlow devices.

The network infrastructure is controlled by the NIM: a Ryu [2] controller application *OpenFlow Bandwidth*[3] that provides a REST/JSON API to issue requests for bandwidth management and monitoring using simple intuitive JSON defined messages. Ryu was selected because of its support for extended OpenFlow 1.3 features and support for various hardware switches. For the application, a typical request would be to report the current network traffic level for a port or previously defined flow. An example command would be to define a flow (using a source/destination IP pair), and request that the flow be limited to certain level (defined in Mbps). The response to this command includes a unique identifier which can be used in subsequent requests for traffic data.

Overall, the combination of features, programmability, and openness provided by OpenFlow greatly assist us to fully realise QoE applications in real-world networks. Furthermore, creating a solution around vendor-specific interfaces has limited applicability; with OpenFlow, we can create a generic solution that should work across a multitude of vendors and within a variety of scenarios. This is particularly important when scalability and interoperability are core requirements, as is the case with this work.

V. QoE Model Experimentation

This section describes how the SDQ experimental environment (described in Section IV) was used to evaluate the UFair QoE model [17], which has previously been evaluated via means of a simulator. When transforming this QoE model into a real networked implementation, we came across a number of issues. During early testing HAS clients were not receiving sufficient bandwidth to reach the stream quality that the model was aiming to achieve. Following analysis, the cause was determined to be an assumption made in the simulator that if meters were set to a specific bandwidth then clients would receive exactly that. In addition, the simulator did not consider packet header sizes, the meter dropping policy, and that HAS chunks can vary in size. These were factors that were easy to overlook when creating a simulation. In order for the model to work as originality intended, bandwidth headroom was introduced when applying metering to ensure that the clients could achieve the desired stream. In the remainder of this section, we describe in more detail the experimental setup, the experiments performed and the results of doing so.

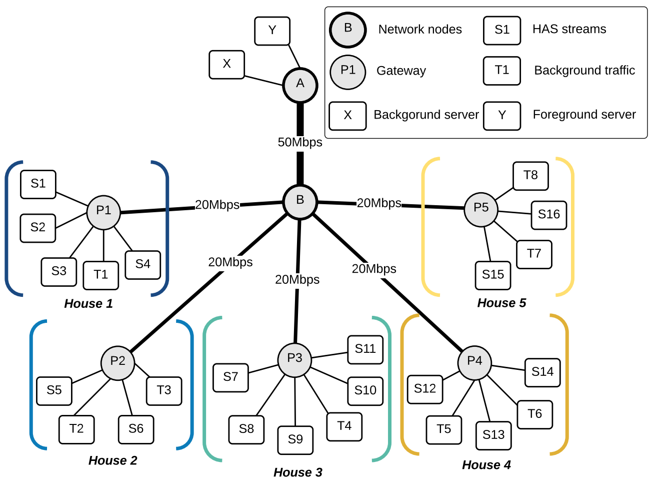

Fig. 3. UFair experiment topology

A. Experiment topology

The topology used for the evaluation of UFair is shown in Figure 3, consistent with the original simulated environment. It represents a tiered multi-household network, in which each household contains 4-6 hosts, all of which are connected to a gateway. This gateway is then connected, along with other gateways in the topology, to an aggregation switch (switch B). Over another hop (towards switch A), a foreground (serving DASH content) and a background server act as endpoints for their respective traffic types.

To simulate a potential home network environment where a household has limited bandwidth, and the link shared between houses is also limited, the emulation of network link characteristics is used. Through SDQ, the links between switch B and the gateway switches are limited to 20Mbps. Similarly the link between B and A is restricted to 50Mbps. The sum of the connectivity available to household links is 100Mbps, and is greater than the link between B and A. This results in a situation whereby there is more demand than there is supply

[3]https://github.com/biirdy/openflow-bandwidth

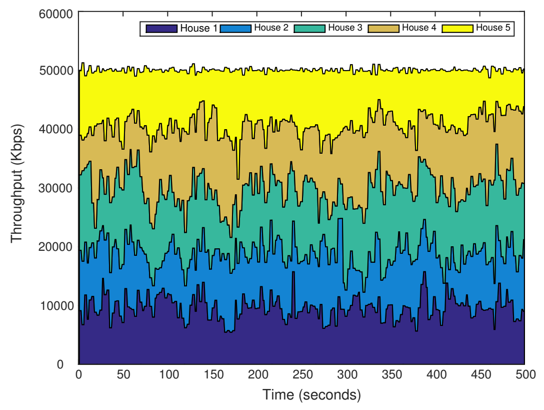

Fig. 4. No QoE model across households

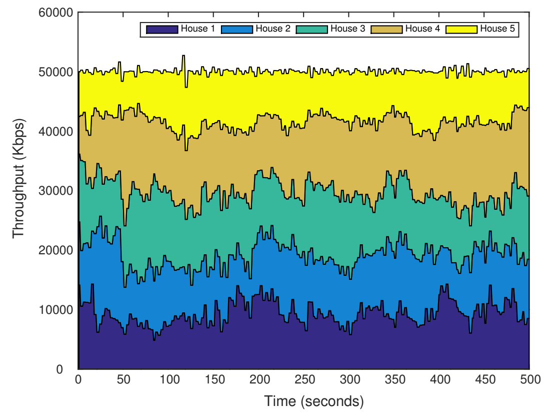

Fig. 5. UFair across households

in the case of multiple households. In these circumstances, the adaptive streams in each house are affected by hosts within the same house, as well as the behaviour of hosts in other houses.

Each house contains multiple hosts requesting MPEG-DASH video content, and one or two hosts generating background traffic. The video clients utilise *Scootplayer*[4]; a highly configurable MPEG-DASH compliant player with support for accurate logging. These clients connect to the foreground server and stream the multi-bitrate MPEG-DASH version of *Big Buck Bunny*.

B. Experiments

In order to evaluate the effectiveness of the UFair QoE model, and to assess the capability of SDQ to achieve the required functionality, we conducted two representative experiments: one where the QoE model was used for network resource allocation and the other with no QoE model applied. They both used the network topology (including the link speed) shown in Figure 3. The experiment containing the QoE model monitored the network statistics of each client, as well as each household. This data is then analysed by the QoE model to determine the most optimal resource allocation strategy. Recommendations given by the QoE model are then applied through SDQ's traffic enforcement functions, including restrictions per flow and per household. For the baseline experiment, network statistics are still monitored but no additional traffic management is applied.

C. Results

This section compares the results of the two experiments (one with QoE model active and the other without) ran with the assistance of the SDQ framework.

Figures 4 and 5 depict the allocation of network resources on the aggregation link between all households. Without a global network view, TCP-based resource allocation causes fluctuations in the network (Figure 4), which ultimately deteriorates the user experience through an increased switching of streams.

[4]http://github.com/broadbent/scootplayer

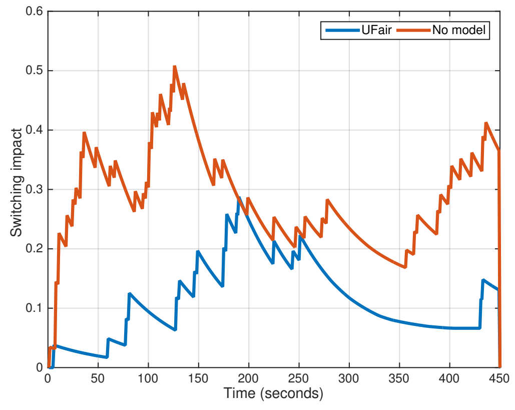

Fig. 6. Switch impact

The QoE model uses switching impact (impact in QoE when switching to a different quality stream) as one of the QoE indicators when recommending resource allocation decisions between user clients of a household [17]. The QoE model, utilising the full functionality of SDQ, incorporates this metric, along with context information such as link capacity at each household, and user/application level requirements (e.g. playback resolution). This leads to an allocation of network resources that is more intelligent and stable. Importantly, it allows UFair, together with SDQ, to reduce the unnecessary switching events across all relevant media flows.

To emphasise this stability, Figure 6 shows the resultant switching impact of a single client in the experiments. Together, these results demonstrate the effectiveness of QoE enforcement function delivered using OpenFlow's *metering* feature.

Through this experimentation, SDQ has shown to be an effective framework to support QoE experiments relating to adaptive video content. The whole toolset, including streamlining the orchestration of the QoE model, virtualisation infrastructure, and physical OpenFlow equipment, allows researchers to focus on human factor modelling and helps to verify the feasibility of any QoE model.

VI. Conclusions and Future Work

Often, researchers use simulations to test QoE models due to the ease, agility and low cost that they offer when compared to real world testing. However, simulations often fail to recognise some of the additional effects that are present in actual networks and the technologies that use them. This leads to models behaving differently in reality, lessening the contribution of experimental findings. This paper introduces SDQ, a framework that uses SDN to facilitate rapid experimentation of QoE models in such a realistic environment. SDQ aids QoE researchers by reducing the barrier to entry for SDN-assisted QoE experiments. An example use case, using the UFair model, demonstrates how a QoE model previously tested solely in simulation can be evaluated in the real world.

There are a number of opportunities to further this work. This includes making the specification and realisation of experimental environments even simpler. For example, this could be done through the implementation of a drag & drop interface so that non-expert users can create topologies quickly and easily. From a technical perspective, there is still effort necessary to implement a full feature-set of OpenFlow capabilities on a broader range of both software and hardware-based forwarding devices. Driven specifically by the need for *metering*, this work also includes further supplementing the OpenFlow specification by enabling the defining of queueing and drop parameters and methods.

A large body of work in the QoE domain focuses on the use of wireless technology. This in itself may have untold side-effects on the results, furthering the disparity between simulation and reality [20]. To further the work presented in this paper, future efforts could explore supplementing the existing hardware provision by including a wide range of devices. This includes those that support different transmission technologies, such as WiFi, LiFi and cellular technologies.

Acknowledgements

The authors are grateful to the UK Engineering and Physical Sciences Research Council (EPSRC) for funding the TOUCAN project (EP/L020009/1), which supported some of the work presented in this paper, and are grateful to Lancaster University for funding Lyndon Fawcett's PhD studentship in association with TOUCAN.

Furthermore, we would like to thank Jamie Bird for his technical contribution towards the framework.

References

[1] Mininet network emulator. *http://mininet.org/*.

[2] Ryu OpenFlow controller. *http://osrg.github.io/ryu/*.

[3] Cisco Visual Networking Index: Forecast and Methodology, 2014-2019 White Paper, 2015.

[4] P. Agyapong, M. Iwamura, D. Staehle, W. Kiess, and A. Benjebbour. Design considerations for a 5G network architecture. *Communications Magazine, IEEE*, 52(11):65–75, 2014.

[5] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. What Happens When HTTP Adaptive Streaming Players Compete for Bandwidth. *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'12)*, page 6, 2012.

[6] J. Esteban, S. A. Benno, A. Beck, Y. Guo, V. Hilt, and I. Rimac. Interactions between http adaptive streaming and tcp. In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, pages 21–26. ACM, 2012.

[7] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking (TON)*, 9(4):392–403, 2001.

[8] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming. In *ACM SIGCOMM 2013 Workshop on Future Human-centric Multimedia Networking (FhMN)*, 2013.

[9] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, Rimid, and Unstable: Picking a Video Streaming Rate is Hard. In *Proc. ACM IMC*, pages 225–238, 2012.

[10] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198. ACM, 2014.

[11] 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*, pages 87–92. IEEE, 2013.

[12] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *Proc. ACM CoNEXT*, pages 97–108, 2012.

[13] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, 2014.

[14] E. Liotou, G. Tseliou, K. Samdanis, D. Tsolkas, F. Adelantado, and C. Verikoukis. An SDN QoE-Service for dynamically enhancing the performance of OTT applications. In *Quality of Multimedia Experience (QoMEX), 2015 Seventh International Workshop on*. IEEE, 2015.

[15] A. Mansy, B. Ver Steeg, and M. Ammar. SABRE: A Client based Technique for Mitigating the Buffer Bloat Effect of Adaptive Video Flows. In *Proc. 3rd annual ACM Conference on Multimedia Systems*, MMSys '12. ACM, 2012.

[16] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network Function Virtualization: State-of-the-art and Research Challenges. (c):1–28, 2015.

[17] M. Mu, M. Broadbent, N. Hart, A. Farshad, N. Race, D. Hutchison, and Q. Ni. A Scalable User Fairness Model for Adaptive Video Streaming over Future Networks. *IEEE Journal on Selected Areas in Communications*, 2016.

[18] M. Mu, S. Simpson, A. Farshad, Q. Ni, and N. Race. User-level Fairness Delivered: Network Resource Allocation for Adaptive Video Streaming. *2015 IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2015.

[19] H. Nam, K.-H. Kim, J. Y. Kim, and H. Schulzrinne. Towards QoE-aware video streaming using SDN. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 1317–1322. IEEE, 2014.

[20] C. Newport, D. Kotz, Y. Yuan, R. S. Gray, J. Liu, and C. Elliott. Experimental evaluation of wireless simulation assumptions. *Simulation*, 83(9):643–661, 2007.

[21] M. Seufert, T. Hosfeld, and C. Sieber. Impact of intermediate layer on quality of experience of HTTP adaptive streaming. *2015 11th International Conference on Network and Service Management (CNSM)*, 17(1):256–260, 2015.

[22] G. Tian and Y. Liu. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 109–120. ACM, 2012.

[23] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie. A Survey on Software-Defined Networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2015.