SDNDASH: Improving QoE of HTTP Adaptive Streaming Using Software Defined Networking

Abdelhak Bentaleb
School of Computing, National University of Singapore
Singapore
bentaleb@comp.nus.edu.sg

Ali C. Begen
Ozyegin University & Networked Media, Inc.
Istanbul, Turkey
ali.begen@ozyegin.edu.tr

Roger Zimmermann
National University of Singapore & FX Palo Alto Laboratory
Singapore & USA
rogerz@comp.nus.edu.sg

ABSTRACT

HTTP adaptive streaming (HAS) is being adopted with increasing frequency and becoming the de-facto standard for video streaming. However, the client-driven, on-off adaptation behavior of HAS results in uneven bandwidth competition and this is exacerbated when a large number of clients share the same bottleneck network link and compete for the available bandwidth. With HAS each client independently strives to maximize its individual share of the available bandwidth, which leads to bandwidth competition and a decrease in end-user quality of experience (QoE). The competition causes scalability issues, which are quality instability, unfair bandwidth sharing and network resource underutilization. We propose a new software defined networking (SDN) based dynamic resource allocation and management architecture for HAS systems, which aims to alleviate these scalability issues and improve the per-client QoE. Our architecture manages and allocates the network resources dynamically for each client based on its expected QoE. Experimental results show that the proposed architecture significantly enhances scalability by improving per-client QoE by at least 30% and supporting up to 80% more clients with the same QoE compared to the conventional schemes.

CCS Concepts

•**Information systems** → **Multimedia streaming;**
•**Networks** → **Network resources allocation;**

Keywords

QoE; DASH; Bitrate adaptation logic; Scalability issues; SDN; OpenFlow; Convex optimization; MPC.

1. INTRODUCTION

In recent years, a new paradigm of Internet video streaming called HTTP adaptive streaming (HAS) has emerged and quickly become the main method for streaming. Cisco is estimating that by 2019 video traffic will account for approximately 80% of the total transmitted Internet data volume [8]. As a consequence, network operators are looking for novel ways to better utilize their network capacities. While several non-interoperable commercial HAS systems are available from various vendors, the Motion Picture Experts Group (MPEG) in conjunction with the 3^{rd} Generation Partnership Project (3GPP) developed a standardized solution termed *Dynamic Adaptive Streaming over HTTP* (DASH) [22] in 2012. A number of solutions have emerged based on this standard, where the source video is segmented into short duration chunks of 2–10 seconds, each of which is encoded at several different bitrate levels and resolutions. Then the generated chunks are stored on a server from which the clients fetch them. During playback each client uses its bitrate adaptation logic to dynamically fetch the chunk encoded at the optimal bitrate level based on metrics such as the average throughput and the buffer occupancy.

In DASH, the bitrate adaptation logic is integrated into the client and unspecified in the standard. However, a completely client-side adaptation may be sub-optimal in large systems and recently Seufert *et al.* [20] showed that utilizing some centralized management can enhance the overall video quality, while improving the quality of experience (QoE) among competing DASH clients in a shared network environment. Hence, for large-scale deployments it is important to design a robust DASH scheme that provides scalability when a multitude of clients compete for the available bandwidth, while at the same time ensuring high QoE and increased system performance. A robust DASH scheme should strive for the following main properties: *Quality stability*, *fairness* and *high utilization*.

In a shared network environment, a purely client-driven scheme has at least two drawbacks. First, inaccurate available bandwidth estimations may result in frequent quality and bitrate oscillations. This leads to sub-optimal bitrate decisions that degrade clients' QoE [2]. Second, service differentiations (*e.g.*, gold, silver and bronze clients) and management policies cannot be assured on the delivered quality [11]. An earlier measurement study [1] has indicated that the above listed properties are violated by most of the widely used commercial video streaming solutions when a large set of clients compete for the available bandwidth. The issue arises when purely client-driven bitrate adaption logic algorithms are working independently without any coordination. Recently, a new architecture for communication networks termed *Software Defined Networks*

MM '16, October 15-19, 2016, Amsterdam, Netherlands

DOI: http://dx.doi.org/10.1145/2964284.2964332

(SDN) [21] has emerged, with the goal to manage networks more flexibly and programmatically. The main purpose of SDN is to decouple the packet forwarding process from the network control, which is centralized in a programmable controller. SDN promotes migration of network functions from dedicated hardware to virtual machines running on shared commodity hardware. This is also referred to as *Network Function Virtualization* (NFV).

To address the fundamental scalability limitations we propose an SDN-based dynamic resource allocation and management architecture for DASH systems that aims to maximize per-client QoE, whilst taking into account different clients' needs (*e.g.*, buffer sizes, display resolutions and quality requirements, etc.) and network requirements (*e.g.*, available bandwidth, latency, etc.). The main contribution of our architecture is that it is able to scale to a large number of DASH clients while maximizing the per-client QoE in a fair manner, and improving the per-client network resource allocation, provisioning and monitoring. To achieve this, our architecture leverages SDN to include an external application that communicates with the controller to select for each client the optimal quality and bitrate level for the next chunks to maximize per-client QoE. To the best of our knowledge, no current work exists that proposed such a complete end-to-end architecture to address DASH scalability in the context of a shared network. Experimental results confirm that our architecture outperforms existing state-of-the-art bitrate adaptation algorithms.

The rest of this paper is organized as follows. Section 2 surveys recent related bitrate adaptation algorithms. Section 3 describes our proposed SDN-based dynamic resource allocation and management architecture for DASH systems and Section 4 presents our architecture components. A performance evaluation and comparison is presented in Section 5. Finally, Section 6 concludes the paper and outlines future directions.

2. RELATED WORK

Several HAS bitrate adaptation logic algorithms (also called ABR algorithms) and heuristics have been published in the literature. We present a set of the most recent algorithms that address some of the scalability issues.

Huang *et al.* [7] proposed a class of buffer-based rate selection algorithms, termed BBA, that are based only on the current playback buffer occupancy as bitrate selection heuristic and not on any bandwidth estimation. BBA aims to achieve two goals. First, it tries to maximize the average video quality by selecting the highest bitrate level that the network can support. Second, it avoids unnecessary re-buffering events caused by buffer starvation. However, this scheme exhibits many limitations such as decreasing the QoE in the case of long-term bandwidth variations and slow bandwidth fluctuation detection.

Jiang *et al.* [9] proposed a general bitrate adaptation framework called FESTIVE that consists of a set of methods that strive to achieve a trade-off between video stability, fairness and efficiency. A stateful bitrate selection heuristic is used to compensate for the biased interaction between the bitrate and the estimated bandwidth, as well as to trade-off between stability and efficiency using a delayed update approach. However, the method may manifest instability when the number of clients increases, likely due to a bandwidth overestimation effect. Furthermore, it is not very responsive to bandwidth fluctuations, which may lead to a large buffer undershoot. It has difficulty to ensure bandwidth fairness between competing DASH clients efficiently in a distributed shared network environment.

Georgopoulos *et al.* [6] proposed an SDN-assisted QoE Fairness Framework, called QFF, which seeks to optimize the QoE by ensuring video quality fairness among multiple heterogeneous competing DASH clients. To this end, QFF takes into account two main constraints, namely the devices' resolutions and current network requirements. The proposed framework components connect to the SDN controller and leverage OpenFlow [15] capabilities to satisfy QoE optimization and fairness. However, the QoE model (utility function) considers only the device resolution and currently available bandwidth and not the current buffer occupancy. Thus, the client may be subject to buffer starvation, reducing end-user QoE. Further, the proposed framework cannot support a large number of clients, as it generates a lot of overhead that can negatively affect the performance of the whole system.

The authors of QDASH [17] designed a QoE-aware system as a proxy between the clients and the streaming server. The aim is to avoid video quality oscillations by ensuring a gradual change in bitrate levels by introducing an intermediate level into the bitrate switching process (in between quality changes) instead of suddenly switching up/down to the target bitrate level. However, QDASH generates significant network overhead especially with a large number of clients.

Thomas *et al.* [23] were motivated by the fact that client-driven rate adaptation provides less control for the network and service providers, *e.g.*, precluding service differentiation. Thus, they proposed the Server and Network-assisted DASH (SAND) architecture. SAND introduces a centralized control component that offers asynchronous network-to-client and network-to-network communication. It receives QoE metrics from the clients and returns network feedback measurements, which are used by clients heuristically in their adaptation logics.

3. SDNDASH ARCHITECTURE

We propose a novel SDN-based dynamic resource allocation and management architecture for both DASH systems and DASH-like approaches including Apple HLS. It leverages an SDN conceptual design [17] and has the capabilities to dynamically monitor and allocate network resources and to maximize QoE for each client, while avoiding scalability issues among a large number of competing clients in a shared network environment. Our design supports various client-driven bitrate adaptation logic algorithms by optimally limiting the possible set of selectable bitrate levels and qualities for each client and offering them as recommendations. Subsequently the client uses its corresponding recommendation as an upper bound together with its buffer level during the bitrate selection process. This strategy enables the clients to react quickly to sudden bandwidth variations and maximize their QoE. Furthermore, our architecture supports heterogeneous clients (*e.g.*, smartphones, tablets, connected TVs, etc.), as it optimizes the per-client QoE while reaching the maximum fairness level in terms of optimized QoE. Furthermore, our method avoids sub-optimal, purely client-driven network resource management and bitrate level decisions, where

the available bandwidth is randomly distributed among the competing clients. Our method is not restricted to the equal sharing of network resources between the clients. Simple network management mechanisms do not take into account any criterion such as device characteristics (heterogeneity), QoE, buffer occupancy, etc., for allocation, thus leading to scalability issues while reducing per-client and overall QoE. For example, a large-screen connected TV client may obtain a lower QoE compared to a a small-screen smartphone client. Such issues lower user satisfaction, in contrast to our novel SDN-based architecture (SDNDASH), which can easily account for such heterogeneous factors.

As illustrated in Figure 1, our architecture consists of **three layers**, namely application, control and network layers (planes), and **six entities** within those layers: DASH server, DASH clients, external SDN-based application, RYU SDN controller [19], internal SDN-based application and forwarding devices. Each layer contains a set of core entities that are described at the high level as follows.

3.1 Application Layer

This layer defines a set of services for end users that ensure an effective network management, programming and communication. It contains three core elements:

A) DASH Server: Represents an HTTP server that stores the media chunks of each video content and the corresponding manifest (MPD) files. We added a simple XML file called *chunk quality file* to list chunks with their perceptual quality, which is measured based on the SSIMPlus index [18] objective metric. In our experiments we performed offline quality measurements for the videos in a DASH dataset [4] to produce the chunk quality file, which is stored at the DASH server together with the MPD. We use SSIMPlus to represent the quality because in practice the correspondence between bitrate and perceptual quality is not only non-linear and but variable over time (for details see Section 4.1 and Begen *et al.*. [12]).

B) DASH Clients: We use the DASH reference video player dash.js as the DASH client. To support our architecture, we integrated three additional classes into the bitrate adaptation algorithm of dash.js, namely:

1. *Communication Interface*: A new client HTTP-based communication interface, which allows dash.js clients [5] to interact with our external SDN-based application.
2. *Distributed QoE Optimizer*: It allows the dash.js bitrate adaptation logic to use the optimal recommendation values (optimal bitrate level and quality) as an upper-bound together with the buffer level in its decisions.
3. *File Logger*: Periodically records state variables of dash.js clients including: buffer size, QoE, chunk RTT delays, bitrate switches, stalls, etc.

C) External SDN-Based Resource Management Application: This is implemented outside of the SDN RYU controller and responsible for QoE-optimization, resource allocation and monitoring. Our external SDN-based application implements **six modules** (1-6) and **three components** (i-iii) as described in Figure 1:

1. *Packet monitoring engine* monitors and inspects packets from different entities of our architecture (*e.g.*, get the MPD file from the DASH server). Thereafter, it parses and classifies the inspected data and offers them to different modules. Our packet monitoring uses a packet sampling approach to differentiate between the cross and DASH traffic, thus dynamically estimating the required throughput for each. Hence, our optimizer can work with only the DASH traffic fraction of the bandwidth to produce the optimal quality assignment for each client.
2. *Management module* is the main manager of the components, collecting the required input data from various modules and then offering them to components. It also receives the output data from different components for plotting and visualization purposes.
3. *Per-client QoE measurement* measures and estimates the QoE of each DASH client. Thereafter, it records the per-client QoE in a specific table called *QoE table*.
4. *MAP tables* defines three kinds of mapping tables including *bitrate level* $\leftrightarrow$ *device resolution, bitrate level* $\leftrightarrow$ *chunk perceptual quality*, and *device name* $\leftrightarrow$ *device capabilities*.
5. *Bandwidth estimator* uses the PANDA algorithm [13] to estimate the available bandwidth accurately. PANDA was proposed to avoid the root cause that triggers bitrate oscillations.
6. *Log and network/server statistics* records all logging files and statistics (*e.g.*, per-client QoE, client information, optimal per-client bitrate level and quality recommendations) periodically. These data are then plotted and visualized.

i. *Specification component* offers three main functionalities. At each step, it first generates a specific data structure that describes the current per-client QoE requirements as a QoE policy, as shown in Figure 2. Second, it maximizes per-client QoE (based on Eq. 10) and outputs the optimal per-client bitrate level and quality for the next chunks to be downloaded. Finally, it manages and stores the per-client optimized QoE, and then passes these outputs to the *communication component* for plotting and visualization.

ii. *Abstraction programming component* is responsible for formulating the per-client network resource requirements (QoS) based on the optimal per-client QoE resulting from the *specification component*. To achieve this goal, it offers two main functionalities. First, it implements an exponential mapping function (Eq. 1) that allows for the correlation between the optimized QoE metric and QoS metric.

$$\begin{cases} QoS = K \times \{(W \times BW)\} \\ QoE = \alpha \times e^{-\beta \times QoS} + \gamma, \end{cases} \tag{1}$$

where BW is the bandwidth, K is a constant access network factor (depending on the type of network, wired or wireless), W is a weight factor, which defines the relative importance degree of QoS metric based on application type (*e.g.*, DASH video streaming bandwidth metric is the most important). α, β are network layer parameters, which are defined based on the QoS service differentiation (gold, sliver and bronze clients). γ is a video codec parameter. Eq. 1 is obtained from the mapping model proposed by Fiedle *et al.* [14]. Second, the *abstraction programming component* generates a specific data structure that describes each per-client QoS requirement as a QoS policy, as shown in Figure 2. These QoS policies are sent to the internal SDN-based application for QoS provisioning.

iii. *Communication component* ensures the interaction between different entities of the application layer. For

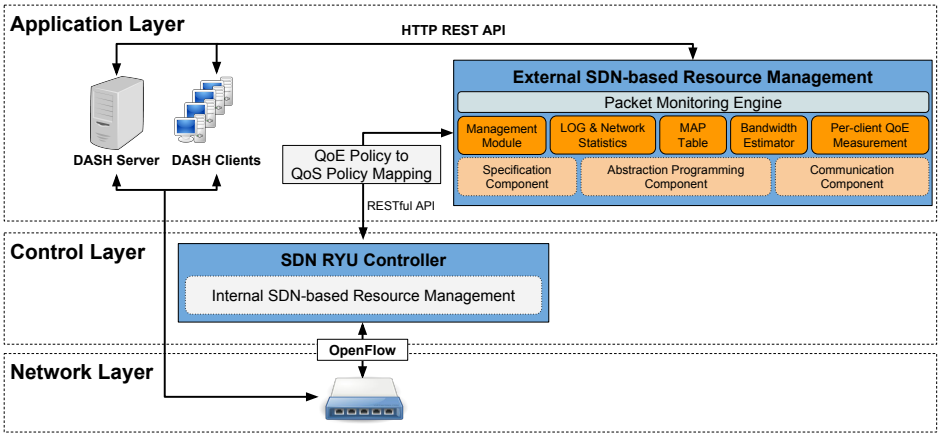


Figure 1: Our proposed SDN-based architecture.

example, it sends the per-client optimal bitrate levels and quality recommendations to the corresponding client. Further, it provides a communication interface between the application layer and control layer (Restful API). For instance, it sends the per-client QoS policy messages to the internal SDN-based application.

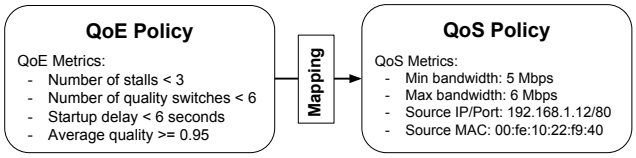


Figure 2: QoE and QoS policies example.

3.2 Control Layer

The control layer represents the central element in the network that has the ability to manage and control network infrastructure and network resources. Further, it defines two communication interfaces to simplify resource abstraction and data exchange between layers (Northbound interface), and also network device programming (Southbound interface). It consists of an SDN controller and an internal SDN-based resource management application.

D) RYU SDN Controller: This represents the control plane of our architecture. It is a central entity that manages flow control to enable intelligent network management. We select and integrate the RYU framework [19] within our architecture as an SDN controller. Therefore, the RYU controller receives per-client QoS policy messages coming from the external SDN-based application for network resource provisioning. The SDN-controller then passes these messages to the internal SDN-based application to configure, allocate, provision, and monitor the network resources for each client accordingly.

E) Internal SDN-Based Resource Management Application: This is implemented inside the SDN RYU controller and responsible for resource allocation, monitoring and QoS provisioning. Upon receiving the QoS policy that contains the per-client bandwidth allocation of each client, it constructs an OpenFlow based messages (*e.g.*, Multipart Request, Flow Mod: OFPFC_ADD)[1] for each QoS policy in order to interact with forwarding devices using OpenFlow. Thus, the internal application dynamically allocates and provisions bandwidth for each DASH client based on per-client QoS policy sent by the external SDN-based application. Further, it monitors per-client resource allocation based on QoS policy timeout until its corresponding timeout becomes zero, where it generates a Flow Mod OpenFlow message (OFPFC_DELETE) that orders forwarding devices to delete the flow from their tables (flow tables and meter tables). Otherwise, it modifies the flow in order to satisfy the new client QoS requirements using a Flow Mod OpenFlow message (OFPFC_MODIFY).

3.3 Network Layer

The network layer represents the infrastructure that is responsible for data forwarding and carrying user traffic.

F) Forwarding Devices: The forwarding devices include switches and routers that support OpenFlow v1.3 and meter table installations such as: CPqD switch[2]. These devices forward data traffic based on the forwarding control policies that are established by the SDN RYU controller, and per-flow rate (QoS provisioning) based on the QoS policy OpenFlow messages coming from the internal SDN-based application. Further, they delete, modify and add flows based on different control policies.

4. ARCHITECTURE DESIGN

To simplify the design of our architecture, we divide the solution into five steps, which are detailed as follows:

4.1 Chunk Quality Measurement

The DASH client in our architecture uses both the chunk bitrate level and perceptual quality during the bitrate adaptation process. This allows us to alleviate many number of quality oscillations, thus, we can ensure consistency in video quality [12]. Hence, the main purposes of this step are summarized as follows: To *(a)* confirm that there is non-linear correlation between chunk perceptual quality and its bitrate level, *(b)* find a relationship (non-linear mapping function) between chunk bitrate levels and qualities, and *(c)* create chunk qualities manifest file.

To achieve the goals above, we perform an objective video quality measurement on two reference source video files with different resolutions. One is animation video called *Big*

[1]https://www.opennetworking.org/openflow

[2]http://cpqd.github.io/ofsoftswitch13/

Buck Bunny and the second is a documentary film called *Of Forests And Men* [4]. We select these two videos because they are widely used by researchers in the field of DASH and are of different genres (animation vs. documentary). This measurement is conducted using an objective video quality assessment model, namely the Structural Similarity Index Plus (SSIMPlus) index [18]. SSIMPlus provides an accurate estimation of the video perceptual quality mimicking the human visual system, video content features including resolution, temporal and spatial complexity, device screen options like size and brightness.

We downloaded the original YUV video files of Big Buck Bunny[3] and Of Forests And Men[4] in various resolutions, while we acquired the processed video files with the same resolutions and various bitrate levels as shown in Table 1 from the DASH dataset [4]. These resolutions fit capabilities of most devices such as smartphones, tablets and connected TVs. In presenting our results, we performed 11 objective tests for each video and computed the averages.

Table 1: Resolutions and bitrates (Kbps).

Resolution	Big Buck Bunny	Of Forests And Men
360p	50, 100, 150, 200, 250, 300, 400, 500	100, 150, 200, 250, 300, 400
480p	600, 700	500, 600, 700
720p	900, 1200, 1500, 2000	800, 1000, 1300, 1500, 1800, 2100, 2500, 3200, 3700
1080p	2100, 2400, 2900, 3300, 3600, 3900	/

The results of our objective quality measurements are shown in Figure 3. The scatter plot shows the mapping of bitrate levels to SSIMplus (top plot) and Big Buck Bunny's per-chunk quality (bottom plot). From the results we see that the correlation between bitrate level and video quality is not linear, although the quality gradually increases as the bitrate increases. Finally, chunk quality is different from one video to another.

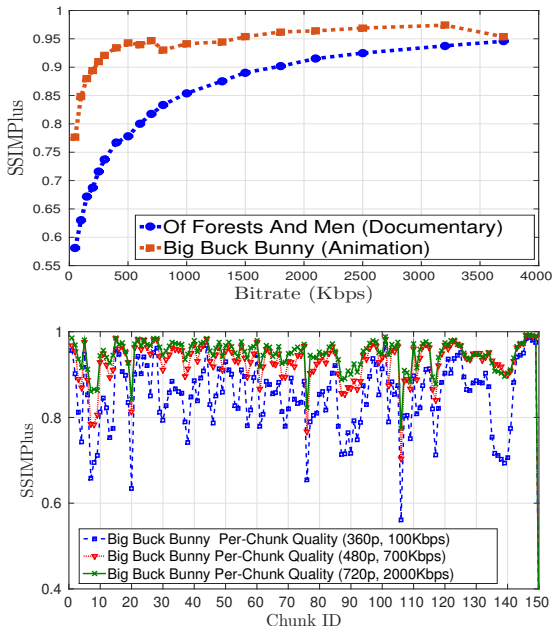


Figure 3: SSIMPlus objective quality measurement.

[3]http://media.xiph.org/video/derf/

[4]http://www.offorestsandmen.org/en/film-en

4.2 QoE Metrics

In this step, we present a set of QoE metrics that are used to calculate the user satisfaction. Inspired by Seufert *et al.* [20], our QoE model utilizes the most influential metrics on end-user satisfaction including: startup delay, number of stalls, average video quality and number of video quality switches.

- *Chunk Quality (Q_{i,l_i})*: Assume a video stream that includes N chunks and each chunk is encoded with L bitrate levels, with l representing a specific bitrate level. At each step i, the quality of chunk i which is encoded at l_i is defined as:

$$Q_{i,l_i} = SSIMPlus_{i,l_i} \tag{2}$$

where $SSIMPlus_{i,l}$ represents the quality value of chunk i at level l. $SSIMPlus_{i,L_{max}}$ denotes the quality of the chunk i with the highest bitrate level L_{max}.

- *Startup Delay (D_s)*: This represents the delay between the request of the first chunk and its rendering onset. An efficient video streaming system must achieve a startup time less than the maximum buffer occupancy ($buff_{max}$).

$$D_s << buff_{max} \tag{3}$$

- *Number of Stalls (S_{nbr})*: Video stalls occur when the client buffer is empty. In other words, if the playback buffer size reaches its minimum threshold $buff_{min}$ and the current chunk i has not been downloaded yet, there will be a stall. The total number of stalls can be calculated as:

$$S_{nbr} = \frac{1}{N}\sum_{i=1}^{N} S_i \quad \text{if } 0 \leq buff_i < buff_{min} \tag{4}$$

where, $buff_i$ is current buffer level in seconds, and S_i is 0 or 1.

- *Average Video Quality (TQ_{avg})*: This represents the total average quality of the downloaded chunks, which can be calculated as:

$$TQ_{avg} = \frac{1}{N}\sum_{i=1..N}^{l=1..L} Q_{i,l} \tag{5}$$

- *Video Quality Switches (SQ_{avg})*: This represents the total quality differences between successively downloaded chunks[5]. It is defined as:

$$SQ_{avg} = \frac{1}{N-1}\sum_{i=1}^{N-1} |Q_{i+1,l_{i+1}} - Q_{i,l_i}| \tag{6}$$

Finally, we define the QoE as a weighted function of the above four metrics as:

$$QoE_i = \alpha TQ_{avg} - \beta SQ_{avg} - \gamma S_t - \delta D_t \tag{7}$$

where $\alpha, \beta, \gamma, \delta$ are non-negative weighting factors. Such factors are selected based on [9] and our own experiments (many tests were performed for value tuning, and we selected trade-off values of 0.25 for each factor).

4.3 Objective Function

At each step, our architecture aims to optimize the global and per-client QoE over all DASH clients during an active

[5]This metric helps SDNDASH maintain a consistent quality.

video playback session, while ensuring a fair bandwidth allocation based on the QoE. For that purpose, we formulate our problem as a linear convex optimization problem. Our convex optimization framework is flexible enough to accommodate a set of dynamic constraints including:

- *Buffer Occupancy Level:* The selected bitrate level and quality must not be affecting the buffer occupancy. For that reason, we define two buffer thresholds $buff_{min}$ and $buff_{max}$. At each step i, the buffer level must be between these two thresholds $buff_{min} < buff_i < buff_{max}$ in order to avoid stalls.
- *Available Bandwidth (BW_e):* The selected bitrate must not exceed the current available bandwidth.
- *Content Type Constraint* (CT): The selected chunk quality must fit the content type.
- *Device Capabilities* (DC): The decision of bitrate level and quality must fit the device capabilities.

A network topology is represented by a directed or undirected graph $G = (V, E)$, depending on the nature of the links, where V represents the set of nodes and E is the set of links between the nodes. The network topology consists of a set of V nodes including: DASH clients $C \subset V$, DASH server $s \in V$, external SDN-based application $a \in V$, SDN controller with internal SDN-based application $b \in V$ and forwarding devices $F \subset V$. These nodes are connected by a set of links E, where each node $v_x \in V$ has at least one link $e_{xy} \in E$ connecting to another node $v_y \in V$, where $x, y = 1, 2, ..., |V|$. A set of DASH clients C share the available bandwidth of capacity BW_{all} of the last mile M (a shared network environment). Periodically at each step i, the SDN-based external application a estimates the available bandwidth BW_e, maximizing the per-client QoE $MAX(QoE_{i,c_n})$ with $c_n \in C$ (representing a client) by finding the optimal bitrate level l_{i,c_n} and quality $Q_{i,l_{i,c_n}}$ for the next chunk to be downloaded, and allocates the suitable amount of bandwidth BW_{alloc_i,c_n} from BW_e that fits $MAX(QoE_{i,c_n})$. Obviously, the total bandwidth allocated to all clients should not exceed the total capacity on the bottleneck link BW_{all}:

$$\sum_{n=1}^{C} BW_{alloc_i,c_n} \leq BW_{all}, \forall i \text{ (during all steps i).} \quad (8)$$

We assume that the internal SDN-based application can provision and update network resources per client periodically at each time slot Δt (from one step till the next step).

The DASH server s, has a set of video contents R. Each video $r \in R$ consists of T seconds of media and is divided into a set of fixed small chunks, and the total number is denoted as N. Each chunk has a duration of τ seconds with $N = \frac{T}{\tau}$ and is associated with a set of B representations (bitrate levels and their qualities), for which every representation $\rho \in P$ has a bitrate $l_\rho \in L$ and quality $q_\rho \in Q$, where L and Q are bitrate level list and perceptual qualities, respectively. At each step i, for each client $c_n \in C$, the set of available chunks with their bitrate level list L and their corresponding perceptual qualities Q_L (SSIMPlus) is presented in manifest files (MPD and quality list) as:

$$\begin{cases} L_{i,c_n} = \{l^1_{i,c_n}, l^2_{i,c_n}, ..., l^k_{i,c_n} ...., l^N_{i,c_n}\} \\ Q_{i,c_n} = \{q^1_{i,c_n}, q^2_{i,c_n}, ..., q^k_{i,c_n} ...., q^N_{i,c_n}\}. \end{cases} \quad (9)$$

Our *QoE Optimizer* solves the objective function by finding for each client $c_n \in C$ the optimal bitrate level $l^k_{i,c_n} \in L$ and its corresponding quality $q^k_{i,c_n} \in Q$ that maximize c_n's QoE (see Eq. 7) based on the available bandwidth, buffer occupancy, video content type and device capabilities constraints. We note that, $q_{i,c_n}(.)$ represents the non-linear mapping function for chunk i in the c_n DASH client. Hence, $q_{i,c_n}(l_{i,c_n})$ denotes the bitrate level with its corresponding quality.

At each step i, and for each client $c_n \in C$ our objective function with its constraints can be expressed formally as:

$$\begin{cases} \text{Find} & q_{i,c_n}(l_{i,c_n}), \text{ where:} \\ \max & QoE_{i,c_n} \\ \text{s.t.} & l_{i,c_n} \leq BW_{i,e}, \forall i, \forall c_n \\ & \sum_{n=1}^{C} BW_{alloc_i,c_n} < BW_{all}, \forall i \\ & buff_{min,c_n} \leq buff_{i,c_n} \leq buff_{max,c_n}, \forall i, \forall c_n \\ & MAP(q_{i,c_n}(l_{i,c_n}), CT_{c_n}), CT \in MAP\ tables, \forall i, \forall c_n \\ & MAP(q_{i,c_n}(l_{i,c_n}), DC_{c_n}), DC \in MAP\ tables, \forall i, \forall c_n. \end{cases} \quad (10)$$

4.4 Solution

The objective function (Eq. 10) can be solved using an online linear convex optimal control, termed online MPC (Model Predictive Control) [3]. The total complexity of our solution is independent of the total number of clients C, since the effect of more frequent updates balances out the effect of shorter horizon within each step. At each step i, and for each DASH client $c_n \in C$, our *QoE Optimizer* uses online MPC (see Algorithm 1) to find the optimal $q_{i,c_n}(l_{i,c_n})$ for the next chunk to be downloaded that can maximize the QoE of c_n. Then, the optimal values are mapped into QoS metric (bandwidth) and sent to our internal SDN-based application for QoS provisioning.

Algorithm 1 Simple optimization problem solution.

```
1: procedure –Online MPC dynamic programming
2:     Input:
3:     N, i, buff_min,c_n, buff_max,c_n, BW_all, BW_i,e, CTc_n, DCc_n
4:     Output:
5:     q_i,c_n(l_i,c_n), ∀c_n
6:     for (each step i = 1,2,....,N) do
7:         for (each client c_n, n = 1,2,....,C) do
8:             for (L_i,c_n and Qi, c_n, ∀c_n) do
9:                 q_i,c_n(l_i,c_n) = Find-MAX(Q_i,c_n(L_i,c_n))
10:                if (l_i,c_n ≤ BW_i,e and Σ_{n=1}^{C} BW_alloc_i,c_n <
11:                BW_all and buff_min,c_n ≤ buff_i,c_n ≤
12:                buff_max,c_n and MAP(q_i,c_n(l_i,c_n), CT_c_n)
13:                and MAP(q_i,c_n(l_i,c_n), DC_c_n), ∀c_n) then
14:                    Return(q_i,c_n(l_i,c_n), ∀c_n, ∀(step i + 1))
15:                    Lookup_store(buff_i,c_n, BW_i,e, CTc_n, DCc_n,
16:                    q_i,c_n(l_i,c_n), c_n, ∀c_n)
17:                else
18:                    Remove(q_i,c_n(l_i,c_n))
19:                    Go to (8)
20:                end if
21:            end for
22:        end for
23:    end for
24: end procedure
```

4.5 dash.js Client Integration

At each step, the online MPC dynamic program (Algorithm 1) computes the optimal bitrate level and its quality for each client. The results are sent to the SDN

controller for QoS provisioning and each corresponding client for bitrate adaptation process. To enable this, we modified the original bitrate adaptation logic algorithm of dash.js [5]. The modifications are denoted by the dashed grey boxes in Figure 4.

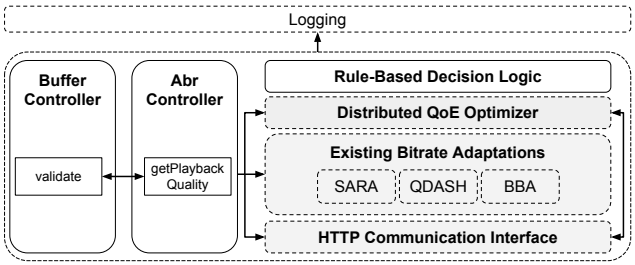


Figure 4: dash.js modifications shown in grey.

5. PERFORMANCE EVALUATION

To evaluate the performance of our architecture and modified client (`SDNdash.js`), we carried out a set of VoD streaming experiments using 50 clients and 100 Mbps total network bandwidth over a complete video streaming system as illustrated in Figure 5. Six test scenarios with different client numbers and bandwidths were also performed. Due to space limit, we show only one test with 50 clients. To simplify the maximization problem, we used the same device capabilities (connected TV) and content type (animation) in our experiments. Our evaluation system is a hybrid combination between a real system and an emulated virtualization system (DASH clients and OpenFlow switches are created using Mininet [16]). We compare the performance of our architecture `SDNdash.js` against three bitrate adaptation logic heuristics found in literature QDASH [17], BBA [7] and SARA [10] (We use these colors for all comparison results), where we implemented them as a class in the bitrate adaptation logic module (see Figure 4) within the newest dash.js (stable master branch v2.0 release) [5]. Further, we compare `SDNdash.js` with conventional bitrate adaptation heuristic of dash.js [5] that combines buffer level with chunk downloading-based available bandwidth and equal bandwidth allocation algorithm during bitrate decision process. These comparisons are conducted using three scalability metrics, namely video stability, fairness and utilization.

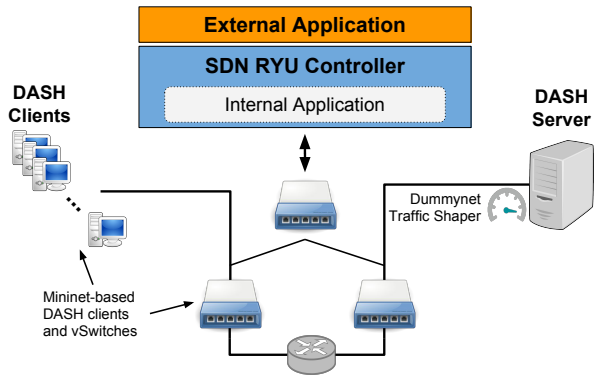


Figure 5: Network topology for the evaluation.

5.1 Network Topology

Figure 5 shows our network topology, which consists of dash.js clients, DASH server, SDN RYU controller with our resource management internal application, the SDN-based external application and three CPqD vSwitches. We also throttle the bandwidth to *100 Mbps* using *Dummynet* in order to emulate a shared network environment scenario.

5.2 Input and Constant Parameters

- **Bandwidth Estimator:** Table 2 shows the parameters used in our bandwidth estimator module, which are set according to the default values in PANDA.

Table 2: Bandwidth estimator parameters.

Parameter	N	κ	ω	β	τ	$buff_{min}$	$buff_{max}$
Value	150 steps	0.14	0.3	0.2	4 seconds	8 seconds	35 seconds

- **QoE, QoS and Optimizer:** All input parameters used by our QoE, QoS and optimizer modules are selected based on suggestions from [20] and [14].
- **MPD File:** We used the Big Buck Bunny video [4], which is $T = 600$ seconds long and consists of $N = 150$ chunks (150 downloading steps), where each chunk has a duration of $\tau = 4$ seconds. This video is encoded using H.264 format at 20 different bitrate levels.
- **Various Bitrate Adaptation Logic Heuristics:** For a fair comparison, we used the same input parameters suggested by the respective authors.
- **Normalized QoE:** Our QoE model outputs a value between 0 and 1. We define a normalized QoE (N-QoE) to represent a user's satisfaction MOS (1 to 5) and report the normalized values in Table 3.

Table 3: Normalized QoE.

QoE	N-QoE	Quality
$0.8 \leq value \leq 1$	between 4 and 5	Excellent
$0.6 \leq value < 0.8$	between 3 and 4	Good
$0.4 \leq value < 0.6$	between 2 and 3	Fair
$0.2 \leq value < 0.4$	between 1 and 2	Poor
$0 \leq value < 0.2$	between 0 and 1	Bad

5.3 Results, Discussion and Analysis

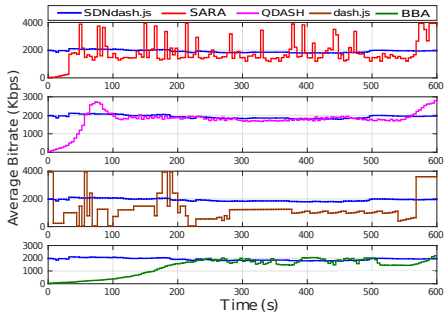
We evaluate the effectiveness of our proposed method by comparing the average results over all clients in video stability, fairness and utilization.

5.3.1 Video Stability

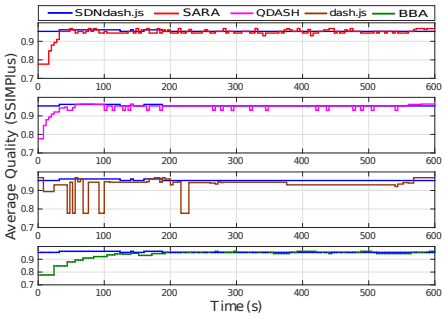
We evaluate the video stability in terms of bitrate level switches, stalls and perceptual quality oscillations. Figures 6a and 6b depict video stability of all competing DASH clients. We can see that our architecture accommodates all its `SDNdash.js` clients by achieving the best video stability compared to other heuristics (see Table 4). This is because `SDNdash.js` clients switch between adjacent bitrate levels smoothly while maintaining a high average bitrate level as much as possible, where the overall average bitrate level of all clients ranges from 1,473 Kbps to 2,087 Kbps (Figure 6a). Moreover, it minimizes the total number of bitrate level switches and stalls as well, and the result is five switches with zero stalls (see Table 4). This is due to the fact that `SDNdash.js` provides an accurate available bandwidth estimation, which eventually leads to a suitable per-client

Table 4: Video stability.

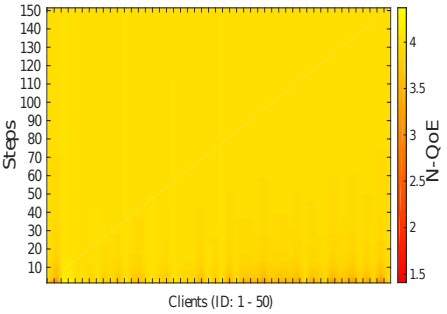
	AVG Bitrate Level (Kbps)	AVG # of Bitrate Switches	AVG # of Stalls	AVG Quality (SSIMPlus)	AVG Quality Variance	AVG # of Oscillations
SDNdash.js	**1,473 to 2,087**	**3%**	**0%**	**0.957 to 0.967**	**0.01**	**7**
QDASH	45 to 2,944	98%	8%	0.92 to 0.967	0.047	52
BBA	45 to 2,089	26%	4%	0.93 to 0.967	0.037	22
SARA	45 to 3,936	46%	6%	0.93 to 0.967	0.037	83
dash.js	45 to 3,936	16%	7%	0.77 to 0.973	0.203	38



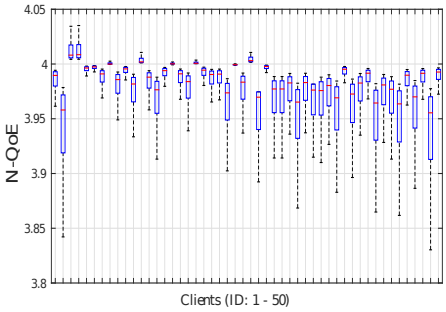
(a) Average bitrates over time.



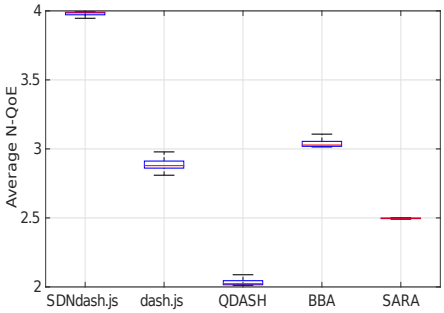
(b) Average qualities over time.



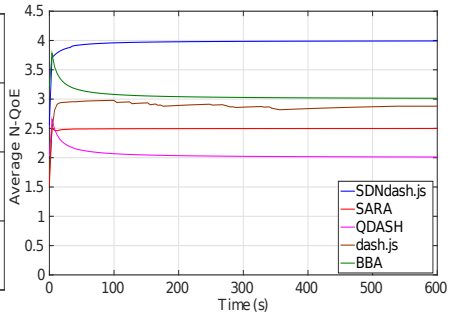
(c) **SDNdash.js** N-QoE fairness.



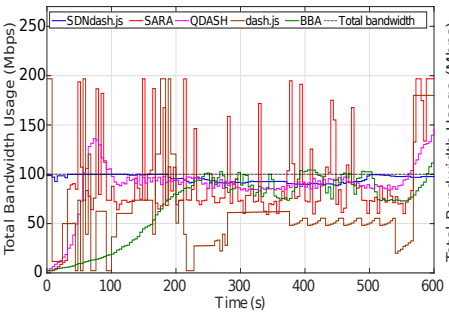
(d) **SDNdash.js** N-QoE maximization.



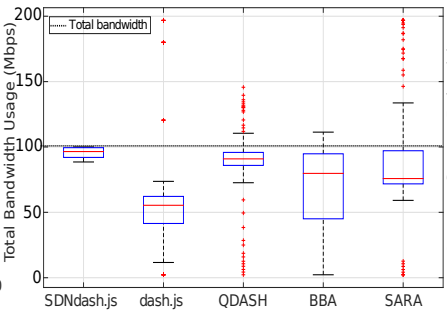
(e) Average N-QoE maximization.



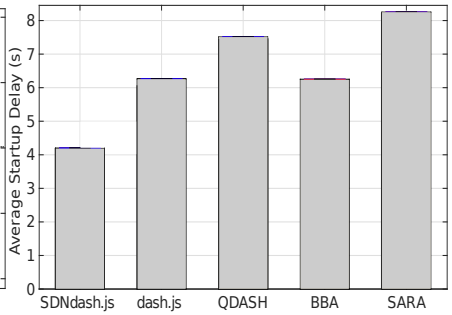
(f) Average N-QoE over time.



(g) Bandwidth utilization.



(h) Bandwidth utilization variation.



(i) Average startup delays.

Figure 6: Experimental results of stability, fairness and utilization for 50 clients and 100 Mbps total system bandwidth.

bitrate level and quality decision and network resource allocation at each step, thus increasing per-client QoE.

Our main results for video stability are summarized in Table 4, where:

- *AVG Bitrate Level* and *AVG Quality* are the average range of the selected bitrate and selected quality across 50 clients, respectively,
- *AVG Number of Bitrate Switches* and *AVG Number of Stalls* are the percentile average of number of bitrate switches and number of stalls across 50 clients, respectively, and
- *AVG Number of Oscillations* is the total average number of quality (SSIMPlus) switches across 50 clients during the video streaming session.

5.3.2 *Fairness and QoE*

This part analyzes the fairness in terms of QoE and end-user satisfaction level (QoE). Figure 6c depicts the heat-map of N-QoE distribution of each **SDNdash.js** client. As can be seen, the QoE is generally distributed fairly between the competing DASH clients (orange color). This confirms that our architecture provides QoE fairness between competing clients and eliminates the unfairness problem compared to purely client-driven heuristics QDASH, BBA, SARA and dash.js as shown in Figure 6e. We anticipated these results because **SDNdash.js** uses the SDN-based application to provide a fairness in terms of QoE to all clients, where they ensure: (1) optimal bitrate level and quality decisions that maximize per-client QoE at each step (2) dynamic

network resource allocation based on optimal decisions that maximize end-user QoE in a fair manner across all clients.

Figure 6e shows a boxplot of N-QoE maximization variation (from low to high) of each `SDNdash.js` client during the DASH video steaming session. The minimum value of whisker (dashed line) represents the first QoE value at the start of the playback, the red line is the median of all QoE values of each client, the maximal value of whisker is the last QoE value at the end of video playback. Hence, Figure 6d illustrates the QoE of each `SDNdash.js` client maximized at every step. For example, the QoE of client 1 increases from 3.87 at the video start to 3.94 at the video end. Furthermore, `SDNdash.js` achieves a higher performance in terms of average per-client QoE compared to QDASH, BBA, SARA and dash.js. This can be seen from Figures 6e and 6f, where QoE in `SDNdash.js` is maximized at each step: it starts from 3 at step 1 (t = 4s) to reach approximately the maximum value of 4 at step 30 (t = 120s). Figure 6i also shows that `SDNdash.js` delivers a lower average startup delay compared to other clients.

5.3.3 Utilization

Figure 6g shows the total bandwidth usage over time. `SDNdash.js` has the best bandwidth utilization without any violation[6]. Our *bandwidth estimator* module can accurately estimate the available bandwidth, and subsequently, the SDN-based external application chooses the optimal bitrate level and quality at each step that can maximize the end-user QoE without violating available bandwidth and/or affecting other clients' QoE. Furthermore, Figure 6h shows the total bandwidth utilization by DASH clients. We can see that `SDNdash.js` achieves around 95% without any violation from the total available bandwidth of *100 Mbps* compared to QDASH: 80% with significant violations, BBA: 77% with minor violations, SARA: 94% with significant violations and dash.js: 50% with minor violations. The bandwidth utilization in QDASH, SARA and dash.js fluctuates significantly with many violations during the video streaming session. This is because of the incorrect bitrate level decisions that are made due to the overestimation of the available bandwidth per client. This eventually leads to congestion as illustrated in Figure 6g, thus, increasing stalls, bitrate switches, etc., which produces a low per-client QoE.

5.4 Summary of Results

Note, in the list items below, the four consecutive numbers represent results for QDASH, BBA, SARA and dash.js, in that order:

- `SDNdash.js` achieves the highest video stability with 40%, 20%, 60% and 30% improvement.
- `SDNdash.js` achieves 87%, 16%, 45%, and 22% less bitrate switches.
- `SDNdash.js` outperforms existing algorithms in terms of high and consistent quality by 42%, 7%, 41% and 24% among all clients.
- `SDNdash.js` improves the QoE of DASH clients by 35%, 25%, 30% and 28% over other heuristics.
- `SDNdash.js` outperforms QDASH, BBA, SARA and dash.js in terms of fairness by 40%, 25%, 30% and 35%, respectively.
- `SDNdash.js` improves utilization by 12%, 13%, 5% and 52%.

[6]Bandwidth violation occurs when the total of the bitrates selected by all clients is greater than the total available bandwidth.

5.5 Interesting Real-World Cases

We present two interesting issues in real-world cases and discuss potential future research directions.

5.5.1 Arbitrary Arrival/Departure Times

A user can join and leave the network at different times. This case represents one of the main real-world cases that we are interested in. For each bitrate adaptation heuristics implemented within dash.js player (`SDNdash.js`, QDASH, BBA, SARA and original dash.js), we performed a new test with five clients sharing a 10 Mbps bottleneck link (We used the same parameters as in the previous experiment). Each client arrives 60 seconds after the previous one (ΔT = 60 s). Table 5 highlights the performance of each client. `SDNdash.js` is more stable in terms of bitrate level and video quality than other algorithms, and achieves a high QoE at each step. Moreover, `SDNdash.js` improves the bandwidth utilization by approximately 30%.

5.5.2 Heterogeneous System

With the diversity of device capabilities nowadays, we expect that there will be heterogeneity in DASH clients: smartphones, tablets and connected TVs. In our experimental scenarios we considered a homogeneous device population. One interesting question is how our architecture behaves with heterogeneous clients. In Section 3, we designed our architecture to handle both homogeneous and heterogeneous systems. So far, we did just preliminary experiments to study the interaction and behavior between multiple heterogeneous clients with different device capabilities and communication technologies (wired, WiFi, LTE). The early results are encouraging, though we consider this topic an interesting problem to solve in a future study.

6. CONCLUSIONS

In HAS, scalability issues arise when a set of multiple clients compete for the available bandwidth in a shared network environment. These problems are growing as the number of clients is increasing in the last mile network, and thus, resulting in an unsatisfactory end-user experience. In this paper, we proposed a new SDN-based resource allocation and management architecture for DASH systems, called SDNDASH. SDNDASH equally applies to other DASH-like approaches including Apple HLS. It leverages the flexibility, capabilities and features of SDN in the dynamic rapid/simple network level infrastructure management. Therefore, SDN enables our architecture to create an efficient and smart adaptive video streaming network. The experimental results demonstrate that our implementation (`SDNdash.js`) provides a good network stability with optimized end-user QoE across all the clients in a shared network environment. As part of our future work, we plan to evaluate our architecture using a large scale SDN-based testbed like FELIX[7], to test various

[7]http://www.ict-felix.eu/

Table 5: BL, QT, N-QoE and total bandwidth of `SDNdash.js` clients.

	Bitrate Level (Kbps)			Quality (SSIMPlus)			N-QoE (Maximum = 5)			Total Bandwidth Usage (Kbps)
	MIN	MAX	MED	MIN	MAX	MED	MIN	MAX	MED	
Client 1	2,089	3,936	2,409	0.964	0.967	0.964	4.5	4.9	4.8	MIN = 80% MAX = 100% MED= 95%
Client 2	2,089	2,944	2,409	0.964	0.965	0.964	3.8	4.7	4.5	
Client 3	1,473	2,409	2,089	0.945	0.964	0.964	3.9	4.85	4.7	
Client 4	1,473	2,944	2,089	0.945	0.967	0.964	3.9	4.65	4.45	
Client 5	1,009	3,639	2,089	0.93	0.967	0.964	3.93	4.69	4.5	

heterogeneous scenarios with multiple SDN controllers and various clients, and to execute trace-driven emulations with both Mininet [16] and ns-3[8] for various network conditions and types including wireless scenarios.

Acknowledgements

This research has been supported by Singapore's Ministry of Education (MOE) Academic Research Fund Tier 1, grant number T1 251RES1415.

7. REFERENCES

[1] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. What Happens when HTTP Adaptive Streaming Players Compete for Bandwidth? In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, pages 9–14, 2012.

[2] S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis. An Experimental Evaluation of Rate-daptive Video Players over HTTP. *Signal Processing: Image Communication*, 27(4):271–287, 2012.

[3] E. F. Camacho and C. B. Alba. *Model Predictive Control*. Springer Science & Business Media, 2013.

[4] DASH External Dataset. http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/.

[5] DASH Industry Forum. http://www.dashif.org/.

[6] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. Towards Network-wide QoE Fairness using Openflow-assisted Adaptive Video Streaming. In *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*, pages 15–20, 2013.

[7] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. Using The Buffer to Avoid Rebuffers: Evidence From a large Video Streaming Service. *arXiv preprint arXiv:1401.2209*, 2014.

[8] C. V. N. Index. The Zettabyte Era–Trends and Analysis. *Cisco white paper*, 2014.

[9] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with Festive. In *Proceedings of the ACM 8th international conference on Emerging networking experiments and technologies*, pages 97–108, 2012.

[10] P. Juluri, V. Tamarapalli, and D. Medhi. SARA: Segment Aware Rate Adaptation Algorithm for Dynamic Adaptive Streaming over HTTP. In *Communication Workshop (ICCW), 2015 IEEE International Conference on*, pages 1765–1770, 2015.

[8]https://www.nsnam.org/

[11] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Helping Hand or Hidden Hurdle: Proxy-assisted Http-based Adaptive Streaming Performance. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*, pages 182–191, 2013.

[12] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran. Streaming Video over HTTP With Consistent Quality. In *Proceedings of the 5th ACM Multimedia Systems Conference*, pages 248–258, 2014.

[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. *Selected Areas in Communications, IEEE Journal on*, 32(4):719–733, 2014.

[14] F. Markus, H. Tobias, and P. Tran-Gia. A Generic Quantitative Relationship between Quality of Experience and Quality of Service. *IEEE Network Magazine*, 2010.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[16] Mininet. http://mininet.org/.

[17] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. QDASH: A QoE-aware DASH System. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 11–22, New York, NY, USA, 2012.

[18] A. Rehman, K. Zeng, and Z. Wang. Display Device-adapted Video Quality-of-experience Assessment. In *IS&T/SPIE Electronic Imaging*, pages 939406–939406. International Society for Optics and Photonics, 2015.

[19] RYU SDN Framework. http://osrg.github.io/ryu/.

[20] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hobfeld, and P. Tran-Gia. A Survey on Quality of Experience of HTTP Adaptive Streaming. *Communications Surveys & Tutorials, IEEE*, 17(1):469–492, 2015.

[21] Software Defined Networking (SDN). https://www.opennetworking.org/.

[22] T. Stockhammer. Dynamic Adaptive Streaming over HTTP–: Standards and Design Principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144, 2011.

[23] E. Thomas, M. van Deventer, T. Stockhammer, A. C. Begen, and J. Famaey. *Enhancing MPEG DASH Performance via Server and Network Assistance*. Stevenhage: IET, 2015.