Network-based Dynamic Prioritization of HTTP Adaptive Streams to Avoid Video Freezes

Stefano Petrangeli*, Tim Wauters*, Rafael Huysegems†, Tom Bostoen†, Filip De Turck*
*Department of Information Technology (INTEC), Ghent University- iMinds
Gaston Crommenlaan 8 (Bus 201), 9050 Ghent, Belgium, email: stefano.petrangeli@intec.ugent.be
†Alcatel Lucent Bell Labs, Copernicuslaan 50, B-2018 Antwerpen, Belgium

***Abstract*—HTTP Adaptive Streaming (HAS) is becoming the de-facto standard for video streaming services over the Internet. In HAS, each video is segmented and stored in different qualities. Rate adaptation heuristics, deployed at the client, allow the most appropriate quality level to be dynamically requested, based on the current network conditions. Current heuristics under-perform when sudden bandwidth drops occur, therefore leading to freezes in the video play-out, the main factor influencing users' Quality of Experience (QoE). In this article, we propose an Openflow-based framework capable of increasing clients' QoE by reducing video freezes. An Openflow-controller is in charge of introducing prioritized delivery of HAS segments, based on feedback collected from both the network nodes and the clients. To reduce the side-effects introduced by prioritization on the bandwidth estimation of the clients, we introduce a novel mechanism to inform the clients about the prioritization status of the downloaded segments without introducing overhead into the network. This information is then used to correct the estimated bandwidth in case of prioritized delivery. By evaluating this novel approach through emulation, under varying network conditions and in several multi-client scenarios, we show how the proposed approach can reduce freezes up to 75% compared to state-of-the-art heuristics.**

Keywords*—HTTP Adaptive Streaming, OpenFlow, Prioritization, Quality of Experience, Video Freezes

I. Introduction

Nowadays, video streaming applications are responsible for the largest portion of the Internet traffic. Particularly, HTTP Adaptive Streaming (HAS) protocols have become very popular and can therefore be considered as the de facto standard for video streaming services over the Internet. Microsoft's Smooth Streaming, Apple's HTTP Live Streaming, Adobe's HTTP Dynamic Streaming and MPEG Dynamic Adaptive Streaming over HTTP (DASH) are examples of available HAS technologies. In a HAS architecture, video content is stored on a server as segments of fixed duration at different quality levels. Each client can request the segment at the most appropriate quality level on the basis of the local perceived bandwidth. In this way, video playback dynamically changes according to the available resources. Such dynamic adaptation results in a smooth video streaming experience. Nevertheless, several inefficiencies have still to be solved in order to further improve users' Quality of Experience (QoE). As reported by Akshabi et al. and Riiser et al., current rate adaptation heuristics perform quality selection sub-optimally, especially when a sudden bandwidth drop occurs [1], [2]. This leads to unnecessary quality switches and video play-out interruptions, which negatively affect the final QoE of the users. Similar conclusions are drawn by the Conviva report on HAS [3]. The report reveals that almost 27% of the analysed HAS sessions exhibit at least one video freeze. This problem is mainly due to the unmanaged nature of current HAS technologies. This entails that clients are not aware of the real network conditions nor are they assisted in improving the delivered QoE. In this paper, we investigate the aforementioned problems arising in HAS under volatile bandwidth conditions. Particularly, we present an OpenFlow-based controller in charge of collecting information from both the network nodes and the HAS clients. Based on current network conditions and clients' status, the controller can decide to prioritize the delivery of particular HAS segments in order to avoid video freezes at the clients. The OpenFlow standard allows to separate the data forwarding plane of the packets from their control plane. This separation provides a flexible way to introduce innovative management solutions in multimedia delivery networks.

The main contributions of this paper are three-fold. First, we present an OpenFlow-based controller that helps clients avoiding video freezes under scarce bandwidth conditions. This controller has the fundamental characteristic to be capable of collecting feedback from both the network nodes and the HAS clients to take the best decision on the segment to prioritize. Moreover, this feedback is collected without introducing extra signalling into the network. Second, a communication mechanism is proposed between the controller and the clients to inform the clients about the prioritization status of the downloaded segment. Prioritized segments are marked with a specific DSCP (Differentiated Services Code Point) field, which is extracted by the clients during the download of a segment to understand whether the segment was prioritized or not. This information is then used to correct the bandwidth estimation process in case of a prioritized delivery. Third, detailed experimental results are presented to characterize the gain of our OpenFlow-based framework compared to state-of-the-art HAS heuristics.

The remainder of this article is structured as follows. Section II reports related work on HAS optimization and OpenFlow. Next, Section III details the proposed OpenFlow-based framework both from an architectural and algorithmic point of view. In Section IV, we evaluate our solution through emulation and show its benefits compared to current HAS heuristics. Section V presents the main conclusions.

II. Related Work

Akshabi et al. present an analysis of the performance and drawbacks of some commercially available HAS heuristics, such as Microsoft Smooth Streaming, Netflix and Adobe players [1]. They show that current rate adaptation heuristics perform quality selection sub-optimally. Particularly, these heuristics fail to adapt to rapid bandwidth changes. As a result, interruptions in the video play-out and unnecessary quality switches occur. Similar conclusions are drawn by Müller et al. based on tests of different HAS implementations using real bandwidth traces collected on a mobile network [4]. They also point out that the Microsoft Smooth Streaming client is able to achieve the highest average bit rate as well as a low number of quality switches.

Many adaptation heuristics have been proposed to alleviate the problems highlighted in the previous paragraph [5]. Tian et al. present a control theory-based HAS client where the buffer filling level of the client is controlled [6]. Adzic et al. propose to add additional information into the manifest file about the objective quality of the video segments to enhance the rate adaptation algorithm [7]. The work presented by Claeys et al. is based on the Markov Decision Process (MPD) [8]. The solution to the MDP is computed online by means of the Q-Learning algorithm. Even though purely client-based heuristics simplify the design and implementation of the algorithms, such heuristics fail in case of sudden bandwidth drops. This failure leads to video freezes and consequently low QoE.

In order to solve this issue, we adopt in this paper an in-network approach, where intermediary nodes are placed in the network to collect information regarding the available bandwidth and to improve the behaviour of the clients. The use of an OpenFlow controller to optimize the behaviour of HAS clients has been studied by Egilmez et al. [9]. They propose to dynamically re-route HAS traffic to avoid congested links. As traffic re-routing is only possible in the core ISP (Internet Service Provider) network while congested links mainly arise in the access network, this approach is not able to fully optimize the behaviour of HAS clients. Several other works apply traffic-shaping techniques to limit the bandwidth assigned to each client and to drive them to request the target bit rate [10], [11]. Georgopoulos et al. design a centralized OpenFlow controller to allocate the bandwidth for each streaming device in order to obtain fairness from a QoE point of view [12]. Essaili et al. propose a QoE optimizer for wireless networks that computes the optimal rate for the streaming clients, based on the wireless channel conditions [13]. This value is subsequently used by a QoE proxy in charge of intercepting and rewriting clients' requests to match the requested quality level with the optimal rate. In our previous work [14], an intermediate node collects QoE statistics on the clients' behaviour and returns this information to them. This measurement is used by the clients to obtain fairness from a QoE point of view. The idea of prioritization in HAS was first introduced by Bouten et al. [15]. In their work, the decision on which segment to prioritize is made independently by each client. Based on its buffer level, a client asks to prioritize the next segment to download when requesting this segment to the HAS server. Nodes equipped with diff-serv capabilities are in charge of enforcing prioritization in the network. Although this approach does not require any intelligence in the network, it exhibits two main drawbacks. First, the prioritization decision performed by the clients is not network-aware, i.e., clients do not know if sufficient resources are available to guarantee the prioritization of the requested segments. This drawback could easily lead to a congestion of the prioritized queue in case of sudden bandwidth drops. Second, malicious clients could repeatedly request a prioritized delivery, as the clients'

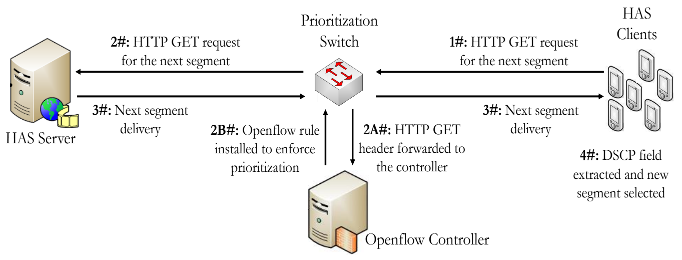
Prioritization Switch
HAS Clients
HAS Server
Openflow Controller
2#: HTTP GET request for the next segment
1#: HTTP GET request for the next segment
3#: Next segment delivery
3#: Next segment delivery
2B#: Openflow rule installed to enforce prioritization
2A#: HTTP GET header forwarded to the controller
4#: DSCP field extracted and new segment selected

Fig. 1: Logical sequence diagram of the proposed solution.

behaviour is not monitored. In our work instead, the proposed OpenFlow controller obtains feedback from both the network nodes and the HAS clients. Therefore, it can decide which are the most appropriate segments to prioritize based on the network conditions and clients' status.

III. Proposed OpenFlow-based Framework

In this section, we detail the implementation of the OpenFlow-based framework introduced in the previous sections. The main component of this framework is an OpenFlow controller deciding which segments should be prioritized in order to avoid interruptions in the video play-out of the clients. As the controller can collect feedback from both the network nodes and the clients, it has a comprehensive view of the network and clients' conditions, and can consequently take the best decision to maximize clients' QoE. Another important aspect of our solution is that clients are aware of the prioritization status of the downloaded segments. This information is used by the clients to refine their quality selection process and to timely react to bandwidth drops in the network. If clients are not aware of the prioritization status of the downloaded segment, two problems arise in their quality decision process. First, the bandwidth perceived by the clients in case of prioritization does not match the real bandwidth. Second, a prioritized segment entails that the decision of the client is not optimal or that a sudden bandwidth drop has occurred. Consequently, the prioritization status is used by the clients as an additional feedback on the quality of their rate adaptation process or on the network conditions. In the remainder of this section, we provide an architectural description of the proposed framework (Section III-A) and detail the controller heuristic to enforce prioritization (Section III-B).

A. Architectural Description

As introduced previously, the OpenFlow controller helps the clients avoiding freezes in case of scarce network resources, e.g. bandwidth drops, by introducing prioritization in the delivery of the video segments. Prioritization is enforced in the network by using an OpenFlow-enabled switch, the so-called *prioritization switch*, which is equipped with a best-effort and a prioritized queue. Based on controller decisions, the prioritization switch enqueues clients' segments in one of these queues. Prioritization switches should be positioned on the links where a bottleneck is most likely to occur. Potential bottlenecks can be identified by analysing the underlying network or at runtime by monitoring links conditions. For example, if the traffic exceeds a certain percentage of the link capacity, a prioritization switch can automatically become active. Depending on the number of bottlenecks, two types of network scenarios can be distinguished. When all clients share a common bottleneck, the best option is to use a single prioritization switch located before the bottleneck. In a second and more complex scenario, a multitude of bottlenecks may simultaneously be present. In this case, we need a system of prioritization switches, which exchange information with one or more controllers to decide which segment to prioritize. In this paper, we focus on the first network scenario while we propose to investigate the more complex one in future work.

An illustrative sequence diagram of the proposed framework is shown in Figure 1. The OpenFlow controller intervenes each time a client requests a new segment from the HAS server and decides whether the analysed segment should be prioritized or not. To perform this decision, the controller obtains relevant measurements from both the prioritization switch and the HAS client requesting the new segment. Network measurements are obtained by using the OpenFlow protocol, which provides well-defined APIs to collect data from the OpenFlow switches. More specifically, the controller periodically polls the prioritization switch to compute the average throughput of the best-effort and prioritized queue (not shown in Figure 1). Given the complexity of implementing a direct communication channel between the clients and the controller, client related measurements are transmitted by introducing an additional field in the header of the HTTP GET message sent by the client when requesting a new segment. The prioritization switch is configured to forward the HTTP GET header to the controller via an OpenFlow rule. The information signalled by the clients is the current buffer filling level and the size and duration of the requested segment. Based on this feedback, the controller decides whether the analysed segment should be prioritized or not. The actual logic implemented by the controller is presented in Section III-B. Next, the controller installs a new OpenFlow rule on the prioritization switch to guarantee a proper delivery of the analysed segment, i.e., best-effort or prioritized delivery. As introduced previously, an important element of our solution consists of the prioritization-awareness of the HAS clients. Similarly to what is presented by Araujo et al., this communication is carried out by using in-network signalling instead of a direct communication channel between the controller and the clients [16]. More specifically, the prioritization switch can be configured to mark prioritized packets with a specific DSCP field. This field is extracted by the clients during the download of a segment to understand whether the segment was prioritized or not. As stated previously, this information is highly relevant for the quality decision process of the clients. When a client downloads a prioritized segment, the *prioritization mode* is triggered. In this mode, prioritized segments are ignored in the calculation of the estimated bandwidth because the bandwidth perceived in case of prioritization does not match the real network conditions. In addition, the client directly requests the next segment at the lowest quality. In this way, the client tries to minimize the risk of video freezes, which is high as the prioritization indicates. It is worth noting that the prioritization mode is independent from the actual rate adaptation heuristic implemented by the client.

An important aspect of our framework is how client related measurements are collected by the controller. In this paper, we used an explicit feedback from the client. Another alternative would be to use session-reconstruction to infer the buffer filling level of the clients, as shown by Huysegems et al. [17]. In this case, the network can decide autonomously which segment to prioritize without any feedback from the clients.

Algorithm 1 : Prioritization algorithm. The symbol * indicates the parameters belonging to the heuristic.

```
Require: prioBan, bandwidth guaranteed to the prioritized channel
    maxConsecutivePrio*, maximum number of consecutive prioritizations allowed
    safetyMarginDT*, safety margin in the estimation of the segment download time (always $\geq 0$)
    $\alpha$*, smoothing factor of the exponential average for the queues' throughput estimation
Ensure: prioStatus, prioritization decision. FALSE means no prioritization

1: $buffer = GetClientBufferFromHTTP()$
2: $segmentSize = GetSegmentSizeFromHTTP()$
3: $segmentDuration = GetSegmentDurationFromHTTP()$
4: $numConsecutivePrio = GetNumConsecutivePrioritizations()$
5: $thrBe = GetThroughputBeQueue(\alpha)$
6: $thrPr = GetThroughputPrioQueue(\alpha)$

7: $prioStatus = FALSE$
8: if $numConsecutivePrio < maxConsecutivePrio$ then
9:     $totClBe = GetNumClientsBestEffortQueue()$
10:    $eDTBe = (1 + safetyMarginDT) \times (segmentSize/(thrBe/(totClBe + 1)))$
11:    $eThrPr = thrPr + segmentSize/segmentDuration$
12:    if $eDTBe \leq buffer$ or $eThrPr > prioBan$ then
13:        $prioStatus = FALSE$
14:    else
15:        $totClPr = GetNumClientsPrioritizationQueue()$
16:        $eDTPr = (1 + safetyMarginDT) \times (segmentSize/(\min(thrBe + thrPr, prioBan)/(totClPr + 1))$
17:        if $eDTPr \leq buffer$ then
18:            $prioStatus = TRUE$
19:        else
20:            $prioStatus = FALSE$
21:        end if
22:    end if
23: end if
```

B. OpenFlow Controller Logic

The OpenFlow controller helps clients avoiding play-out interruptions, by enforcing prioritization into the network. The controller logic is based on two types of inputs: the throughput of the best-effort and prioritized queue from the prioritization switch and the buffer filling level, the size and duration of the requested segment from the HAS client. The decision on which segment to prioritize is carried out computing an estimate of the segment download time in the best-effort and prioritized queue. If a best-effort delivery does not guarantee a timely download of the segment, i.e., if the download time is larger than the client buffer filling level, the segment is prioritized. Algorithm 1 details the operations performed by the controller.

As described in Section III-A, Algorithm 1 is executed every time a client requests a new segment to download. First, the controller extracts the client's buffer filling level and the size and duration of the requested segment from the HTTP GET message header (line 1-3). The controller also keeps track of the number of consecutive prioritizations obtained for the analysed client (line 4). Then, it obtains the throughput of the best-effort and prioritized queue (line 5 and 6), which is computed using exponential smoothing, with the smoothing factor equal to α. If the number of consecutive prioritizations is greater than *maxConsecutivePrio* (line 8), the segment is enqueued in the best-effort queue. This way, the controller tries to fairly share the prioritized channel among all the clients. Otherwise, an estimation of the download time in the best-effort queue *eDTBe* is computed (line 10). *eDTBe* is given by the ratio between the requested segment size *segmentSize* and

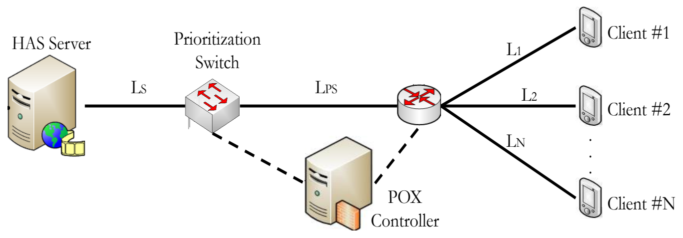

Fig. 2: Emulated topology on Mininet.

the bandwidth per-client available in the best-effort queue. We assume here the available bandwidth is shared fairly among the clients. This last value is obtained by subdividing the best-effort queue throughput *thrBe* by *totClBe*+1, the number of segments currently waiting in the best-effort queue plus one. Because *eDTBe* represents only an estimate of the segment download time, we introduce a safety margin *safetyMarginDT*. The algorithm proceeds as in the following. If *eDTBe* is smaller than the buffer filling level of the client (line 12), the segment is not prioritized because the risk of a video freeze is negligible. We assume here that the buffer decrease during the interval between the client's request and the HTTP header analysis is negligible compared to the total buffer level. The controller also evaluates whether prioritizing the segment could congest the prioritized queue. Particularly, the current throughput of the prioritized queue plus the throughput of the analysed segment (line 11) has to be lower than the guaranteed throughput *prioBan* (second condition on line 12). If this condition is not met, the segment is not prioritized. This way, the controller tries to avoid congesting the prioritized queue and thus negatively impacting the delivery of the other segments. Otherwise, the controller computes an estimate of the download time in the prioritization queue *eDTPr* (line 16) similarly to what is done for the best-effort queue. In this case, the total available bandwidth is the minimum between the guaranteed bandwidth *prioBan* and the total throughput observed on the prioritization switch interface, i.e., $thrBe + thrPr$. Finally, the segment is entitled for prioritization only if *eDTPr* is lower than the buffer level (line 17), i.e., only if a prioritized delivery is actually effective in avoiding a video freeze.

As far as possible scalability issues are concerned, it is worth noting that the computation complexity of the controller heuristic is *O(n)* with *n* the number of controlled clients. Thus, the controller heuristic scales well even for a large number of clients. Moreover, as explained in Section III-A, a prioritization switch should be located before the main network bottleneck, which is typically a link in the access network. Consequently, only few thousand clients would need to be managed by the controller at the same time in the worst case scenario.

IV. Performance Evaluation

A. Experimental Setup

The proposed OpenFlow framework is implemented on the Mininet Network Emulator[1]. The HTTP server, where the video content is stored, is an Apache Server. The video streamed is *Big Buck Bunny*, composed by 299 segments, each 2 seconds long and encoded at 7 different quality levels: 300, 427, 608, 806, 1233, 1636, 2436 kbps. The HAS clients are implemented on top of the libdash library [18], the official reference software of the ISO/IEC MPEG-DASH standard. The libcurl library[2] is used to modify the HTTP GET header and signal the buffer filling level and the size and duration of the requested segment to the controller. Libpcap[3] is used to extract the DSCP field from the received packets and thus enable prioritization-awareness. The buffer size for each client is equal to 5 segments or 10 seconds. The controller is implemented using POX[4], an extendible Python-based controller. Open vSwitch 1.9.3[5] is used to realize the OpenFlow switches. The prioritization switch is equipped with a best-effort and a prioritized queue. A strict-priority policy was used for the experiments. The prioritized queue can transmit at a guaranteed rate of $0.25 \times N$ Mbps, with N the number of clients. The switch polling time of the controller to obtain the new throughput of the best-effort and prioritized queues is set to 0.5 seconds.

The emulated network topology is shown in Fig. 2, where the position of the prioritization switch is illustrated. The prioritized queue is installed on the interface towards link L_{PS}. In order to provide an extensive evaluation of the proposed framework, we emulate 30 episodes of the video trace and average the results over the 30 runs. A different variable bandwidth pattern for each episode is used on link L_{PS}, which varies each second and is scaled with respect to the number of clients. As far as the bandwidth pattern is concerned, an open-source dataset collected on a real 3G/HSDPA network is used [19]. The available bandwidth for one client has an average of 2087 Kbps and a standard deviation of 1314 Kbps. The bandwidth on links L_S and L_i, with i from 1 to N, is kept constant and equal to $3 \times N$ Mbps and 5 Mbps, respectively.

The rate adaptation heuristic embedded in the HAS clients is the QoE-RAHAS heuristic [20], with parameters *qualityWindow*, *bufferMin* and *bufferPercentage* set to 70 seconds, 4 seconds and 100%, respectively. In order to provide an extensive benchmark of the proposed framework, we compare our results to those obtained using a popular proprietary HAS client, the Microsoft ISS Smooth Streaming (MSS) client [21] with parameters *panic threshold*, *lower threshold*, *upper threshold* and *improved timeout* equal to 25%, 40%, 80% and 4 seconds, respectively. All the aforementioned coefficients were set according to our previous work [22].

The QoE model used to evaluate the proposed framework is a metric in the same range of the Mean Opinion Score (MOS), which was proposed by De Vriendt et al. [23] and further improved by Claeys et al. [8]. The QoE experienced by a client is a function of the average requested quality level, its standard deviation, the number of freezes and the average freeze duration.

B. Controller Parameters Analysis

In this section we investigate the impact of the controller parameters on the performance of our solution and select the configuration to use in the remainder of the paper. As explained in Section III-B and Algorithm 1, the controller heuristic parameters are: (i) *maxConsecutivePrio*, the maximum number of consecutive prioritizations allowed for a given client, (ii)

[1]http://mininet.org

[2]http://curl.haxx.se/libcurl

[3]http://www.tcpdump.org

[4]https://OpenFlow.stanford.edu/display/ONL/POX+Wiki

[5]http://openvswitch.org

TABLE I: Overview of evaluated parameter configuration

Parameter	*Evaluated values*
maxConsecutivePrio	1, 2, 3, 6, ∞
safetyMarginDT	0, 0.05, 0.1, 0.2
α	0.25, 0.5, 0.75, 1

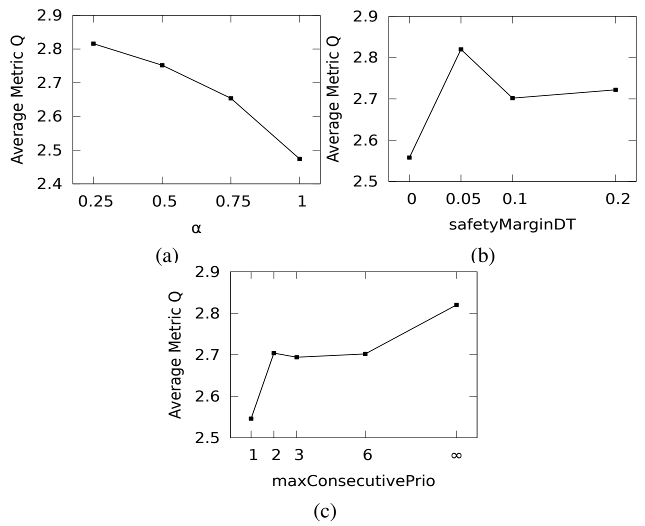

Fig. 3: Analysis of the parameter influence.

safetyMarginDT, the safety margin for the estimation of the segment download time and (iii) α, the smoothing factor of the exponential average used to compute the throughput of the best-effort and prioritized queue. In order to properly tune our framework and select the best configuration, an elaborate evaluation of the parameter space was performed. The evaluated values are reported in Table I, for a total of 80 configurations. We evaluate our solution with the setup described in Section IV-A. The network contains 30 clients streaming video at the same time. The capacity of link L_S is set to 90 Mbps, while the bandwidth on link L_{PS} is variable and has an average of about 63 Mbps and a standard deviation of about 40 Mbps over the 30 emulated episodes. The best configuration is selected on the basis of the obtained QoE. We introduce a metric to evaluate the overall performance of the analysed configurations, defined as $\overline{QoE}_k$. $\overline{QoE}_k$ is the average QoE computed over the whole group of clients for the *k-th* episode. For every configuration, the average *Q* of the performance metric $\overline{QoE}_k$ over the 30 episodes is computed. Then, for each evaluated value of each parameter, the average performance metric *Q* over the five best configurations containing the evaluated value is calculated. Using more than 5 values does not significantly affect the outcome of this analysis. The results are shown in Figure 3. Small values of the smoothing factor α lead to the best performance. This means that the throughput estimation of the best-effort and prioritized queues is computed also considering the past estimations. Taking the past into account helps filtering small and temporary bandwidth drops that can be absorbed by the client's buffer without the need of prioritization but does not affect the timely identification of significant drops for the correct estimation of the segment download time. As far as the parameter *safetyMarginDT* is concerned, a 5% increase of the estimated segment download time results in the best average metric *Q*. As the controller can compute only an estimate of the segment download time, a safety margin equal to zero entails that the controller can fail identifying segments that need a prioritized delivery. Also when the safety margin is large, the performance of our solution drops because the controller overestimates the risk for video freezes. Consequently, prioritization is enforced in a sub-optimal way. It is interesting to notice that limiting the number of consecutive prioritizations does not have a positive impact on performance. When a segment is prioritized, the client is forced to request the next segment at the lowest quality. This reduces the risk for further buffer-depletion and typically allows to raise the buffer level. As a result, the next segment does not need a prioritized delivery. Consequently, an explicit, enforced limitation of the number of consecutive prioritizations is not required.

Based on this analysis, the configuration with *maxConsecutivePrio* not limited, *safetyMarginDT* equal to 0.05 and α to 0.25 was finally chosen. It is worth noting that this same configuration is also the best overall.

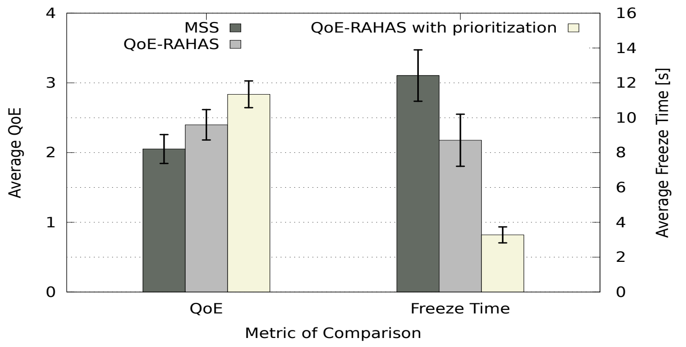

Fig. 4: Heuristics comparison, from a QoE and freeze time perspective, for 30 clients streaming video. The graph reports the performance of the heuristics in terms of average QoE (primary y-axis) and average freeze time (secondary y-axis).

TABLE II: Performance summary in terms of quality components. The average value over the 30 episodes is reported, together with the 95% confidence interval. Quality levels are expressed from 1 (low) to 7 (high).

	MSS	*QoE-RAHAS*	*QoE-RAHAS$_{PRIO}$*
QoE	2.05±.20	2.40±.21	2.83±.19
Average quality level	5.95±.09	5.93±.09	5.91±.08
Quality standard deviation	1.66±.07	1.71±.06	1.68±.06
Freeze time [s]	12.41±1.4	8.70±1.4	3.27±.45
Freeze number	7.13±0.86	5.71±1.02	1.59±0.28

C. Rate Adaptation Heuristic Comparison

In this section, we compare the performance of our solution with that obtained by using the MSS and a classical implementation of the QoE-RAHAS heuristic. The same settings as in the previous section are used with 30 clients streaming video at the same time. For each episode and for each heuristic, we compute the average QoE and the average freeze time for the entire group of clients. Figure 4 reports the average value over the 30 runs of these metrics together with the confidence intervals at 95%. Our OpenFlow-based solution is able to reduce the freeze duration by 75% and 60% when compared to the MSS and the QoE-RAHAS heuristics, respectively. This improvement has a direct effect on the delivered QoE, which increases by 18% compared to MSS and by 15% compared to QoE-RAHAS. It is worth noting that also a classical implementation of the QoE-RAHAS heuristic is able to outperform the MSS one both in terms of average freeze time and QoE. The introduction of network-based prioritization allows to further improve the heuristic's performance. As it is possible to see from Table II, the main difference among the heuristics lies in the freeze time. Due to the high variability of the used bandwidth pattern, clients can experience a considerable amount of video freezes. The introduction of prioritization allows to remarkably improve this metric and, consequently, the final QoE delivered to the users. The average requested quality slightly decreases in the QoE-RAHAS$_{\text{PRIO}}$ case because clients are forced to request the lowest quality level in case of a prioritized delivery, as explained in Section III-A. Another interesting metric to analyse is the number of prioritized segments enforced by the controller. We first average this value for each run over the entire group of clients. Then, the average of this value over the 30 runs is computed. The obtained value is equal to 4.78. This entails that, on average, less than 2% of the video is delivered in a prioritized way. Nevertheless, as reported above, this is sufficient to considerably reduce freezes and increase QoE. This result is mainly due to two aspects. First, as the controller has a comprehensive view of the network and clients' conditions, it is able to prioritize the segments that most likely will result in a video freeze. Second, when a segment is prioritized, the client enters the *prioritization mode* and directly requests the lowest quality level. This approach helps reducing congestion occurring during bandwidth drops or scarce bandwidth conditions and consequently decreasing the risk of freezes also for the other clients.

V. Conclusions

In this paper, we presented an OpenFlow-based framework to improve the Quality of Experience of HAS clients and reduce interruptions in the play-out of the video clients. This was necessary as state-of-the-art rate adaptation heuristics suffer from non-negligible video play-out freezes in case of sudden bandwidth drops or scarce network resources. This objective is achieved by introducing prioritization in the delivery of HAS segments. An OpenFlow controller collects information on the overall network conditions and the HAS clients' status and decides whether a particular segment has to be prioritized or not in order to avoid a freeze at the client. Clients are also aware of the prioritization status of the downloaded segments in order to react properly to prioritization. Extensive emulation experiments using Mininet have validated the effectiveness of the proposed approach. Particularly, we have compared our OpenFlow-based framework with the proprietary Microsoft ISS Smooth Streaming client and a classical implementation of the QoE-RAHAS heuristic [20]. In the evaluated bandwidth scenarios, we were able to show that our OpenFlow framework has resulted in a better video quality and in a reduction of video freeze time up to 18% and 75% respectively, when compared to the benchmark algorithms.

Future research includes the investigation of session-reconstruction methods to obtain the buffer filling level of the clients at the controller side. In this case, the network can decide autonomously which segment to prioritize without any feedback from the clients. We propose to study more complex network scenarios involving multiple bottleneck links, by leveraging a distributed system of OpenFlow controllers communicating with each other, and to evaluate the performance of the OpenFlow controllers in terms of scalability properties and hardware requirements. Moreover, we plan to implement other state-of-the-art rate adaptation heuristics to obtain additional insight into the advantages of the proposed framework and to investigate how our solution can benefit from the Server- and Network-assisted DASH (SAND), a recent standardized solution proposed by MPEG for network-assisted adaptive streaming.

Acknowledgement

The research was performed partially within the iMinds V-FORCE project (under IWT grant agreement no. 130655). This work was partly funded by FLAMINGO, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

References

[1] S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptive video players over http," *Image Commun.*, vol. 27, no. 4, pp. 271–287, Apr. 2012.

[2] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz, "A comparison of quality scheduling in commercial adaptive http streaming solutions on a 3g network," in *Proceedings of the 4th Workshop on Mobile Video*. NY, USA: ACM, 2012, pp. 25–30.

[3] CONVIVA, "2014 viewer experience report." [Online]. Available: http://www.conviva.com/reports/2014_Viewer_Experience_Report.pdf

[4] C. Müller, S. Lederer, and C. Timmerer, "An evaluation of dynamic adaptive streaming over http in vehicular environments," in *Proceedings of the 4th Workshop on Mobile Video*, ser. MoVid '12. ACM, 2012.

[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," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, 2014.

[6] 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*, ser. CoNEXT '12. ACM, 2012, pp. 109–120.

[7] V. Adzic, H. Kalva, and B. Furht, "Optimized adaptive http streaming for mobile devices," in *Proceedings of SPIE, Applications of Digital Image Processing*, 2011.

[8] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck, "Design and optimization of a (fa)q-learning-based http adaptive streaming client," *Connection Science*, vol. 26, 2014.

[9] H. Egilmez, S. Civanlar, and A. Tekalp, "An optimization framework for qos-enabled adaptive video streaming over openflow networks," *Multimedia, IEEE Transactions on*, vol. 15, April 2013.

[10] R. Houdaille and S. Gouache, "Shaping http adaptive streams for a better user experience," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. ACM, 2012, pp. 1–9.

[11] 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*, ser. MMSys '12. ACM, 2012, pp. 11–22.

[12] 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*, ser. FhMN '13. New York, NY, USA: ACM, 2013, pp. 15–20.

[13] A. El Essaili, D. Schroeder, D. Staehle, M. Shehada, W. Kellerer, and E. G. Steinbach, "Quality-of-experience driven adaptive http media delivery," in *ICC*, 2013, pp. 2480–2485.

[14] S. Petrangeli, M. Claeys, S. Latré, J. Famaey, and F. De Turck, "A multi-agent q-learning-based framework for achieving fairness in http adaptive streaming," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–9.

[15] N. Bouten, M. Claeys, S. Latre, J. Famaey, W. Van Leekwijck, and F. De Turck, "Deadline-based approach for improving delivery of svc-based http adaptive streaming content," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–7.

[16] J. Araujo, R. Landa, R. Clegg, and G. Pavlou, "Software-defined network support for transport resilience," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–8.

[17] R. Huysegems, B. De Vleeschauwer, K. De Schepper, C. Hawinkel, T. Wu, K. Laevens, and W. Van Leekwijck, "Session reconstruction for http adaptive streaming: Laying the foundation for network-based qoe monitoring," in *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*, 2012, pp. 1–9.

[18] C. Mueller, S. Lederer, J. Poecher, and C. Timmerer, "Libdash - an open source software library for the mpeg-dash standard," in *Multimedia and Expo Workshops (ICMEW), 2013 IEEE International Conference on*, July 2013, pp. 1–2.

[19] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Video streaming using a location-based bandwidth-lookup service for bitrate planning," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 8, no. 3, pp. 24:1–24:19, Aug. 2012.

[20] S. Petrangeli, J. Famaey, M. Claeys, and F. De Turck, "A qoe-driven rate adaptation heuristic for enhanced adaptive video streaming," 2014. [Online]. Available: http://users.ugent.be/~spetrang/QoE-RAHAS.pdf

[21] A. Zambelli, "The microsoft ISS smooth streaming (MSS) client," https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming.

[22] S. Petrangeli, N. Bouten, E. Dejonghe, J. Famaey, P. Leroux, and F. De Turck, "Design and evaluation of a dash-compliant second screen video player for live events in mobile scenarios," in *International Symposium on Integrated Network Management (IM), 2015 IEEE/IFIP*, May 2015.

[23] J. De Vriendt, D. De Vleeschauwer, and D. Robinson, "Model for estimating qoe of video delivered using http adaptive streaming," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, May 2013, pp. 1288–1293.