A Machine Learning-Based Framework for Preventing Video Freezes in HTTP Adaptive Streaming

Stefano Petrangeli[a,*], Tingyao Wu[b], Tim Wauters[a], Rafael Huysegems[b], Tom Bostoen[b], Filip De Turck[a]

[a]*Department of Information Technology (INTEC), Ghent University – imec, Ghent, Belgium*
[b]*Nokia Bell Labs, Antwerp, Belgium*

Abstract

HTTP Adaptive Streaming (HAS) represents the dominant technology to deliver videos over the Internet, due to its ability to adapt the video quality to the available bandwidth. Despite that, HAS clients can still suffer from freezes in the video playout, the main factor influencing users' Quality of Experience (QoE). To reduce video freezes, we propose a network-based framework, where a network controller prioritizes the delivery of particular video segments to prevent freezes at the clients. This framework is based on OpenFlow, a widely adopted protocol to implement the Software-Defined Networking principle. The main element of the controller is a Machine Learning (ML) engine based on the Random Undersampling Boosting algorithm and fuzzy logic, which can detect when a client is close to a freeze and drive the network prioritization to avoid it. This decision is based on measurements collected from the network nodes only, without any knowledge on the streamed video or on the client's characteristics.

In this paper, we detail the design of the proposed ML-based framework and compare its performance with other benchmarking HAS solutions, under various video streaming scenarios. Particularly, we show through extensive experimentation that the proposed approach can reduce video freezes and freeze time with about 65% and 45% respectively, when compared to benchmarking algorithms. These results represent a major improvement for the QoE of the users watching multimedia content online.

Keywords: HTTP Adaptive Streaming, Quality of Experience, Video Freezes, Machine Learning, RUSBoost, Software-Defined Networks

1. Introduction

Video streaming currently represents one of the most important applications running over the Internet. Traditional video delivery techniques using the RTP/RSTP protocol suite and progressive download have now been replaced by HTTP Adaptive Streaming (HAS) protocols, which can be considered as the de-facto standard for video streaming services nowadays. In HAS, the video is stored on a server and is encoded at different quality levels. Each version of the same video is also temporally segmented. The HAS clients are equipped with a heuristic to dynamically select the most appropriate quality level to download, based on information as the locally perceived bandwidth and the video player buffer filling level. Despite the capability of HAS solutions to adapt to varying network conditions, current solutions are still suffering from interruptions of the video playout, also called video freezes [1]. The 2015 Conviva report shows that almost 30% of the analyzed HAS sessions are affected by at least one freeze [2]. Moreover, this problem becomes more prominent during live streaming sessions, where the video player buffer has to be reduced as much as possible in order to minimize the end-to-end delay. This inefficiency is extremely detrimental for the Quality of Experience (QoE) of the users and consequently for video streaming providers.

In order to reduce the occurrence of video freezes, we propose in this paper a network-based approach, where intermediate network elements are designed to support the delivery of the video. The proposed network framework is built upon the Software-Defined Networking (SDN) principle, a recently proposed innovative network architecture [3]. In SDN, the control plane of the packets is managed by a network controller, while the data forwarding plane is the responsibility of the SDN switches. This decoupling allows to exploit network functionalities in a flexible and real-time manner, as the SDN controller can be easily programmed to control low-level network resources. In this paper, we present an SDN controller in charge of prioritizing the delivery of particular video segments in order to avoid video freezes at the clients. The main element of the controller is a Machine Learning (ML) engine, based on the Random Undersampling Boosting (RUSBoost) algorithm [4] and fuzzy logic, which can identify when a client is close to a video freeze and decide whether the client's segments

*Correspondence to: Stefano Petrangeli, Department of Information Technology (INTEC), Ghent University – imec, Technologiepark-Zwijnaarde 15, 9052 Ghent, Belgium

Email address: stefano.petrangeli@ugent.be (Stefano Petrangeli) should be prioritized or not. The RUSBoost algorithm is particularly suited for classification problems affected by class imbalance, as in the freeze prediction case. In fact, only a small percentage of a video is usually affected by freezes, since in normal conditions the HAS principle is able to achieve a continuous playout.

The main contributions of this paper are three-fold. First, we present a ML-based framework to help clients avoiding video freezes caused by network congestion. Particularly, we assume that congestion occurs in the edge or aggregation network, where alternative routing is unavailable, and handle it using a prioritized queue. An SDN controller is equipped with a ML engine, which allows the controller to predict when an HAS client will experience a video freeze and decide whether the currently downloaded segment should be prioritized or not. This decision is based on measurement data collected from the network nodes only, with no extra signalling required from the clients. Second, we design a freeze predictor based on the RUSBoost classification algorithm, which is embedded in the network controller. The freeze predictor is designed to detect conditions possibly leading to a freeze at the clients. The freeze predictor does not require any a priori knowledge on the characteristics of the video, as the segment duration or the bit-rates of the different quality levels, nor on the client's configuration, in terms of maximum buffer size and start-up buffering time. Although this information can enhance the performance of the controller's logic as shown in our previous work [5], it is difficult to obtain in a real environment. By properly training the freeze predictor, it is still possible to reduce the number of video freezes experienced by the clients, without making any assumption on the streamed videos and the clients themselves. Third, detailed experimental results are presented to show the performance of the proposed approach under different congestion levels, network topologies and streamed videos. We also evaluate the performance of several classification algorithms for the freeze prediction task, and show that the RUSBoost algorithm is able to achieve the highest accuracy.

The remainder of this article is structured as follows. Section 2 reports related work on HAS optimization. Next, Section 3 details the proposed machine learning-based framework both from an architectural and algorithmic point of view. In Section 4, we evaluate our solution through emulation and show its benefits compared to current HAS heuristics. Section 5 presents the main conclusions.

2. Related Work

Many rate adaptation heuristics have been proposed to optimize the QoE of HAS clients [6]. Yin et al. present a control-theory-based HAS client based on the model predictive control approach [7]. In the work by Li et al., the download of a chunk is scheduled to obtain a continuous average data rate sent over the network and to maintain the buffer level close to a given threshold [8]. This approach also helps reducing bit-rate oscillations when multiple HAS clients compete for the same bandwidth. The same problem is also investigated by Jiang et al [9]. They study different design aspects that may lead to fairness improvements, including the quality selection interval, a stateful quality level selection and a bandwidth estimation using an harmonic mean instead of a normal one. Sun et al. improve the throughout prediction of HAS clients by developing a prediction model based on past video sessions, which is built offline in a node located in the streaming provider network [10]. This model is then used online in the rate adaptation heuristic of the video clients, which remain the sole responsible of the actual quality adaptation. Even though purely client-based heuristics simplify the design and implementation of the algorithms, such heuristics can fail in case of sudden and unexpected bandwidth drops. This failure leads to video freezes and consequently low QoE. This issue is further worsened in live video streaming scenarios, where the playout buffer has to be reduced in order to minimize the camera-to-display delay.

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 status of the clients. Consequently, the network has a comprehensive view of the clients' conditions and can help them achieving a high QoE. As an example of such a principle, Ivesic et al. show the benefit of an application-aware QoE-driven connection admission control for general multimedia services in LTE networks [11]. Ganjam et al. present a hierarchical centralized control system to optimize the delivery of HAS streams [12]. The root controller periodically creates a general model of the streaming system, which is then used by the children controllers (on a coarser time-scale) to take decisions on the best QoE strategy for the video clients. A similar approach is also employed by Mukerjee et al. to optimize the performance of live video streams [13]. The use of an SDN controller to optimize the behavior of HAS clients has been studied by Egilmez et al. [14]. They propose to dynamically re-route HAS traffic to avoid congested links. Uzakgider et al. investigate the performance of an SDN-controller equipped with a Q-Learning algorithm to re-route traffic for layered adaptive streaming [15]. As traffic re-routing is only possible in the core Internet service provider network while congested links mainly arise in the edge network [16], these approaches are not able to fully optimize the behavior of HAS clients. Several other papers apply traffic shaping techniques to limit the bandwidth assigned to each client and to drive them to request a target bit-rate. Georgopoulos et al. propose a centralized SDN controller allocating bandwidth for each streaming device, in order to obtain fairness from a QoE point of view [17]. The authors present a model to correlate video bit-rate with video quality, which is used by the controller in the bandwidth allocation process. The use of an SDN controller to obtain video quality fairness among different HAS clients is also investigated by Cofano et al. [18]. The principal disadvantage of the aforementioned algorithms is the active role of the in-network elements in the quality decision process. This aspect entails an alteration of the classical HAS principle, as the network de-facto decides which quality level the clients can download. Moreover, these approaches are not designed to foresee the occurrence of video freezes and avoid them. In our solution instead, the quality level decision is completely left to the clients. The SDN controller supports the delivery of particular segments to avoid a freeze but does not have any active role in the quality decision process of the clients.

In this paper, we propose a ML-based framework to foresee the occurrence of video freezes for HAS clients and reduce them using network-based prioritization. This approach presents two advantages. First, the in-network calculation can be kept very simple and consequently not computationally demanding, since the quality decision algorithm still runs at the client. Second, the controller is transparent for both the HAS clients and the HAS server, as the controller only supports the delivery of the segments to avoid a freeze but does not interfere in the quality decision process of the clients. Moreover, it is more robust in case of fault or malfunctioning of the network equipment, as the clients can continue to operate (at a sub-optimal level) without the in-network system. A network-based prioritization framework was already presented in our previous work [5], where we analyzed the prioritization system in terms of dimensioning of the prioritization queue, scalability issues and its performance compared with several purely client-based solutions. Even though the presented approach showed excellent results, it is affected by two main drawbacks. First, the controller's logic needs to know the characteristics of the streamed video, in terms of segment duration and bit-rates of the different quality levels. Second, it requires an estimation of the buffer filling level of the clients' video player. This in turn entails that the controller has to know the initial buffering time of the client (i.e., the amount of video buffered before the playout can start). As no extra signalling is foreseen between the clients and the controller, this information can only be obtained by intercepting the manifest file requested by a client before starting the video. This aspect limits the general applicability of the proposed framework in a real environment. The main contribution of this paper is therefore a complete re-design of the controller's logic using a ML-based approach. A freeze prediction algorithm located at the controller is able to detect beforehand when a client is going to freeze and drive the network prioritization. The freeze prediction algorithm is trained to correlate information as the network bandwidth and the timing of consecutive segment requests issued by a client to the occurrence of a freeze. Consequently, the prediction does not require any a priori assumption on the video characteristics or the buffer behavior of the clients.

A ML-based approach to off-line detect the occurrence of video freezes in HAS has also been proposed by Wu at al. [19]. A decision tree is trained to identify whether a completed video session is affected by video freezes. In this work instead, the identification is performed on-line and on a segment basis, rather than off-line and on the basis of a completed video session. Our machine learning framework is designed to foresee future freezes and avoid them, in contrast to identifying freezes that have already occurred as investigated by Wu et al. [19].

3. Machine Learning-Based Framework

In this section, we detail the implementation of the proposed ML-based framework introduced in the previous sections. The main component of this framework is an SDN controller, deciding which segments should be prioritized in order to avoid interruptions in the video playout of the clients. This decision is driven by a ML module, the so-called *freeze predictor*, which is trained off-line and used on-line to drive the network prioritization. Each time a client requests a new segment, the predictor identifies whether the client could experience a freeze. The proposed network framework is implemented using OpenFlow, which currently represents one of the most common SDN protocols.

Another important aspect of our solution is that the clients are aware of the prioritization status of the downloaded segments. This information is used by the clients to adjust their quality selection process. If the clients are not aware of the prioritization status of the downloaded segment, a problem could arise in their quality decision process, as the bandwidth perceived by the clients in case of prioritization does not match the real available bandwidth. Moreover, a prioritized segment entails that the decision of the client was 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 3.1) and detail the ML-based controller heuristic to enforce prioritization (Section 3.2).

3.1. Architectural Description

As previously introduced, 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 the decisions of the controller, the prioritization switch enqueues the clients' segments in one of these queues. As far as the prioritization switch positioning is concerned, the switch should be located before the main bottleneck

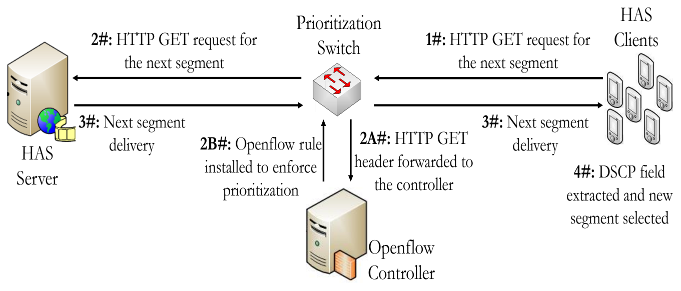

Prioritization Switch
HAS Clients
2#: HTTP GET request for the next segment
1#: HTTP GET request for the next segment
HAS Server
3#: Next segment delivery
2B#: Openflow rule installed to enforce prioritization
2A#: HTTP GET header forwarded to the controller
3#: Next segment delivery
4#: DSCP field extracted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

Figure 1: The OpenFlow controller intercepts clients' requests and decides whether the requested segment should be prioritized or not.

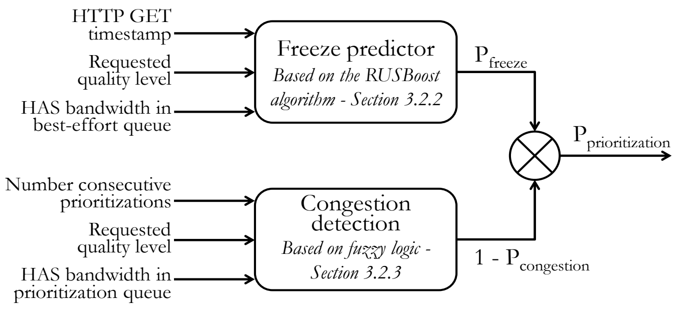

HTTP GET timestamp
Requested quality level
HAS bandwidth in best-effort queue
Freeze predictor
Based on the RUSBoost algorithm - Section 3.2.2
P_freeze
P_prioritization
Number consecutive prioritizations
Requested quality level
HAS bandwidth in prioritization queue
Congestion detection
Based on fuzzy logic - Section 3.2.3
1 - P_congestion

Figure 2: The controller's logic is based on a freeze predictor, to identify when a client is close to a freeze, and a congestion detection module, to check whether prioritizing the segment would congest the prioritization queue.

of the network, which is typically a link in the edge or aggregation network [16]. Potential bottlenecks can be identified by analyzing the underlying network architecture or at runtime by monitoring link conditions (e.g., if the traffic exceeds a certain percentage of the link capacity, a prioritization switch can automatically become active).

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 analyzed segment should be prioritized or not. Particularly, the prioritization switch forwards to the controller, via an OpenFlow rule, the IP packets flowing from the client to the server. The controller can analyze these packets and identify the HTTP GET requests issued by the client. When a GET is received, the controller decides whether the delivery of the analyzed segment should be prioritized or not.

This decision is made by analyzing the flow of intercepted GET requests issued by a client and network measurements collected from the prioritization switch. 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 available bandwidth for the HAS clients in the best-effort queue (not shown in Figure 1). The freeze predictor module uses these inputs to detect whether the client is going to experience a freeze during the download of the requested segment. If prioritization is needed, a fuzzy module checks if enough resources are available in the prioritization queue to prioritize the segment and effectively prevent a freeze. The complete ML-based logic implemented by the controller, as well as the network bandwidth estimation algorithm, is presented in Section 3.2.

Next, the controller installs a new OpenFlow rule on the prioritization switch to guarantee a proper delivery of the analyzed segment, i.e., best-effort or prioritized delivery. This way, the controller only supports the delivery of particular video segments, rather than determining the quality to be requested by the clients. This approach is robust toward controller and switch failures, as the clients can still operate even if prioritization cannot be enforced into the network. It is worth noting that the prioritization switch does not require any specialized feature to be used in the proposed framework. Particularly, the only required features for the switch are: (i) support the use of queues and (ii) support the OpenFlow protocol. Consequently, any general purpose OpenFlow-enabled switch can easily implement the prioritization switch functionalities.

As introduced previously, an important element of our approach consists of the prioritization-awareness of the HAS clients. This communication is carried out by using in-network signaling instead of a direct communication channel between the controller and the clients [5]. More specifically, the prioritization switch can be configured to mark prioritized packets with a specific Differentiated Services Code Point (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. 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. The only modification required at the client side is the extraction of the DSCP field from the downloaded segments and the prioritization mode, while no changes are required in the rate adaptation heuristic itself. This aspect also entails that the proposed framework and the HAS clients can keep on operating even if the prioritization mode cannot be executed or is not implemented.

3.2. OpenFlow Controller

The prioritization logic of the OpenFlow controller is executed each time a client requests a new segment to download, and is composed of two modules, the *freeze* *predictor* and the *congestion detection* modules. Figure 2 gives a high level overview of the interaction between these two modules in the prioritization decision of the controller. The freeze predictor module is designed to understand whether the client could experience a freeze during the download of the requested segment. The prediction is performed by analyzing the flow of GET requests issued by the client, the requested quality level and the available bandwidth for HAS traffic in the best-effort queue. The predictor is trained off-line using the RUSBoost classification algorithm. The congestion detection module is used to avoid congesting the prioritization queue and to allow a fair share among the different clients. This detection is performed by a fuzzy engine based on inputs as the bandwidth in the prioritization queue, the requested quality level and the number of consecutive prioritizations experienced by a client. The freeze prediction and the congestion detection modules return the probability for the current segment to freeze and the probability for the current segment to congest the prioritization queue, respectively. By combining these two factors, the controller obtains the actual probability of prioritizing the segment. A client is prioritized with higher probability when the risk of a freeze is high, as indicated by the freeze predictor, and prioritizing the segment does not congest the prioritized queue nor is unfair to the other clients, as indicated by the congestion detection module.

Section 3.2.1 details the operations performed to estimate the available bandwidth for HAS traffic in the best-effort queue. Sections 3.2.2 and 3.2.3 detail the freeze predictor and the congestion detection modules, respectively.

3.2.1. HAS Bandwidth Estimation

As previously introduced, the controller periodically polls the prioritization switch to estimate the available bandwidth for HAS traffic in the best-effort queue. We indicate with T_{poll} the polling period. The communication between the controller and the switch is carried out using the OpenFlow protocol. OpenFlow does not provide a direct API to collect bandwidth information for a specific type of traffic (e.g., HAS traffic). Consequently, we adopt a two-steps approach to compute the available bandwidth for HAS traffic in the best-effort queue.

The controller first obtains from the prioritization switch the total downstream bandwidth for HAS traffic during the last T_{poll} seconds. The controller sends an OpenFlow aggregate flow statistic request message for all the flows whose IP source address matches that of the HAS Server, which we assume is known to the controller. A flow belongs to the HAS group if the IP source address matches that of the HAS server. It is worth noting that this measurement represents the total downstream bandwidth for HAS traffic flowing through the prioritization switch, i.e., both in the prioritized and best-effort queue. The controller also obtains the downstream bandwidth for HAS traffic in the prioritization queue during the last T_{poll} seconds. This measurement can be obtained using an OpenFlow queue statistic request message to the switch.

Next, the bandwidth for the HAS clients in the best-effort queue is computed, as the difference between the total HAS downstream bandwidth and the prioritized bandwidth. The final throughput is calculated as the exponential average of the current throughput and past samples. Based on our previous work [5], we set the polling time T_{poll} and the weight of the exponential average to 1 second and 0.25, respectively.

3.2.2. Freeze Predictor Module

The freeze predictor is the core element of the proposed machine learning framework. When a segment is requested by a client, the freeze predictor has to decide whether the download of the segment is going to be affected by a freeze or not. This problem can be modelled as a classification problem with two classes: freeze and non-freeze. The classification is based on measurements obtained by the controller without any a priori assumption on the video and clients' configuration, in terms of video bit-rates, segment duration, maximum buffer size and start-up buffering time. This aspect clearly complicates the classification task but allows a more general applicability of the proposed approach in a real environment. As the HAS clients do not explicitly communicate with the controller, the controller would need to intercept the video manifest to access the aforementioned information. Conversely, we only use data obtained by the OpenFlow protocol (e.g., the network bandwidth) or by analyzing the flow of HTTP GET requests issued by the HAS clients (e.g., the timing of the GET requests). In light of the above, the inputs for the freeze predictor are:

- ban_{HAS}, the bandwidth for HAS traffic in the best-effort queue when the segment is requested;
- Δban_{HAS}, the difference between two consecutive samples of the HAS bandwidth;
- ΔGET, the inter-arrival time between two consecutive GET requests issued by a client;
- $\overline{\Delta GET}$, the average GET inter-arrival time of a client;
- q, the requested segment quality level, which is expressed as an integer ranging from 0 (the lowest level) to q_{max} (the maximum quality level, different from video to video and not available to the controller);
- Δq, the difference between the qualities of two consecutive segment requests.

ban_{HAS} is measured using the OpenFlow protocol, as explained in Section 3.2.1. The risk of freezes is directly related to the network bandwidth: when congestion is high, more freezes are likely to occur. Δban_{HAS} indicates whether network conditions are improving or not. Intuitively, the risk of freezes decreases when network conditions improve, i.e., when Δban_{HAS} is greater than zero.

ΔGET is an important input for the freeze predictor, as it accounts for the buffer dynamic of the client. In steady-state conditions, a client issues a GET with a period equal to the segment duration. In this condition, no freezes can occur at the client. When the available bandwidth drops, the inter-arrival time ΔGET starts to increase, as the client has to wait for the complete download of a segment before requesting a new one. Consequently, the buffer starts to decrease and a possible freeze can occur. ΔGET allows to detect this condition and anticipate the freeze. The average GET inter-arrival time $\overline{\Delta GET}$ provides a rough estimation of the segment duration of the video, in steady-state conditions. This estimate allows to discriminate between videos with different segment durations and better exploit the information provided by ΔGET. As an example, a 4 seconds inter-arrival time is normal for a 4-seconds segment video, while it indicates that the buffer is depleting in a 1-second segment video. $\overline{\Delta GET}$ allows to discriminate between the two cases, as the average inter-arrival time is going to be different for different segment durations.

The last two inputs, q and Δq, account for the behavior of the client. A client usually requests a low quality when the bandwidth is scarce or the buffer is close to depletion. Conversely, a high quality is more susceptible to freezes if the bandwidth suddenly drops. This measurement is therefore highly valuable for the predictor. Δq shows if the client's conditions are improving. A client increases the quality only when the perceived bandwidth and/or the buffer allow it. The requested quality level can be extracted by analyzing the URL of the HTTP GET request. Different qualities must be associated with different URLs and this dissimilarity can be used by the controller to extract the requested quality.

By using these six inputs, the predictor is able to foresee the occurrence of a video freeze at the client. When a new segment is requested and a freeze is foreseen, the controller can enforce prioritization in order to avoid it. We divided this problem in two different phases: an off-line step where the predictor is built and an on-line step where the predictor is used to drive the network prioritization. The off-line phase is carried out collecting a large amount of HAS clients' logs in a controlled environment. Each entry of the training set is associated with a segment request made by a client, and is composed of the six aforementioned inputs and a label indicating whether the download of the requested segment resulted in a freeze or not. A machine learning algorithm can then be used to build the freeze predictor.

Several challenges complicate the freeze prediction task. First, no assumptions are made on the clients' configuration and on the streamed videos. Second, the training set is imbalanced as most of the segment requests are not associated with a video freeze. In fact, the adaptive streaming principle is able to accommodate bandwidth variations, and network prioritization should only be used in emergency situations. Class imbalance is a well-known issue in the machine learning field, which can negatively affect the performance of classical classification algorithms [4]. In imbalanced training sets, the occurrence of the so-called negative class is much higher than the occurrence of the positive class, which represents the class of interest for the considered problem. In our case for example, most of the segments are not associated with a freeze, while we are interested in detecting segments affected by freezes (the positive class), which are the minority. Several algorithms have been proposed in literature to address this problem, with the RUSBoost algorithm emerging for its simplicity, accuracy and low computational complexity [4].

The RUSBoost algorithm combines two techniques: random undersampling and boosting. In random undersampling, most of the examples belonging to the negative class are removed from the training set, in order to obtain a desired balanced class distribution (e.g., 50%). This approach reduces the training time (as the new training set is much smaller than the original one), at the cost of a loss of information, as many negative class entries are removed. This drawback is counterbalanced using boosting. In boosting, a set of classifiers is iteratively built. At each iteration, the new classifier focuses on training examples that were misclassified at the previous iteration. After training, each classifier in the set, also called ensemble, participates in a vote to classify new examples. In the RUSBoost case, at each iteration, the new classifier is trained with a different undersampled subset of the original training set. This way, the loss of information due to undersampling is mitigated because data excluded in a given iteration can be included in other ones.

In light of the above, the RUSBoost algorithm represents a viable choice for the freeze predictor task. As shown in Figure 2, the outcome of the freeze predictor is P_{Freeze}, the probability for the current requested segment to freeze. The classifiers in the ensemble are associated with a weight, indicating how good or bad a given classifier is. When classifying a new request, each classifier contributes to the final decision according to its weight. All the weights of the classifiers indicating that the current segment will not freeze are summed in the $W_{NoFreeze}$ variable. Conversely, all the weights of the classifiers indicating a freeze are summed in the W_{Freeze} variable. The output probability P_{Freeze} is simply computed as $W_{Freeze}/(W_{Freeze}+W_{NoFreeze})$.

In Section 4.2, the off-line training phase of the freeze predictor using the RUSBoost algorithm is analyzed in detail and compared with other state-of-the-art classification algorithms.

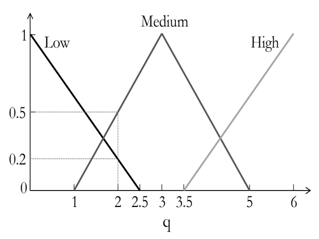
3.2.3. Congestion Detection Module

The second module completing the ML-based controller logic is the congestion detection module. This module is designed to understand whether prioritizing a particular segment would congest the prioritization queue. Prioritizing a segment is only useful when a freeze is probable and enough resources are available in the prioritization queue to effectively prevent the freeze. If too many clients are prioritized at the same time, the performance of the whole framework would drop. Moreover, this module ensures that the prioritization queue is fairly shared among all HAS clients, by limiting the maximum amount of consecutive prioritizations a client can benefit from. For each segment requested by a client, three inputs are used by the congestion detection module: (i) q, the segment's quality level, (ii) N_{Prio}, the number of consecutive prioritizations enjoyed by the client and (iii) $clientBan_{Prio}$, the bandwidth per client in the prioritization queue, computed as the ratio between the bandwidth in the prioritization queue and the number of prioritized clients. It is worth mentioning that $clientBan_{Prio}$ is only an estimate, as bandwidth is not always shared fairly among clients.

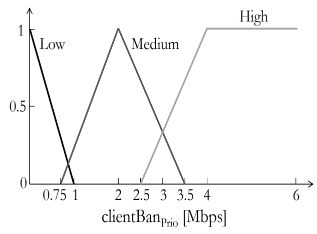
q and $clientBan_{Prio}$ allow to assess the risk of prioritizing the requested segment. Even though the bit-rates of the video are not known to the controller, a higher quality segment will always take more resources to be transported than a lower quality segment. Consequently, the controller has to find a trade-off between prioritizing few high quality segments or many low quality segments. $clientBan_{Prio}$ is the fundamental input to understand the conditions of the prioritization queue. When many clients are prioritized at the same time, the bandwidth per client decreases and, consequently, also the probability of a successful prioritization. N_{Prio} is used to limit an unfair usage of the prioritization system. The probability of prioritization diminishes as the number of consecutive prioritizations increases, as to allow all clients to benefit from the system.

As explained in the previous paragraph, the correlation and the influence of the different inputs are rather simple. Despite that, an immediate translation into mathematical formulation is not straightforward. For this reason, we decided to implement the congestion detection module using fuzzy logic. Fuzzy logic allows to take complex decisions based on uncertain inputs, mimicking the reasoning of the human logic. It is based on fuzzy sets, which provide a rough modelling of the main characteristics of the analyzed problem, and fuzzy rules, which combine the knowledge modelled by the sets and make a final decision. These rules are expressed using human concepts rather than strict measurements. The fuzzy sets and the fuzzy rules are designed considering common sense knowledge of the analyzed domain. The fuzzy-based congestion detection module is composed of three input sets (one for each input), one output set for the output variable $P_{congestion}$ and six fuzzy rules to correlate the sets, which are described in the following.

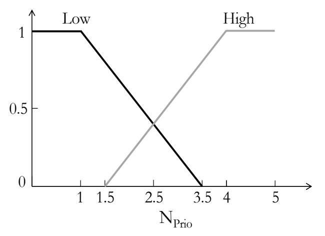
The fuzzy set associated to the quality q is composed of three membership functions: low, medium and high (Figure 3a). At any given point of the input range, the value of q is associated to one or more of these membership functions, with a certain degree. As an example, a value of q equal to 2 is considered 0% high, 50% medium and 20% low. On the contrary, a value of q equal to 0 (i.e., the lowest one) is considered 100% low.



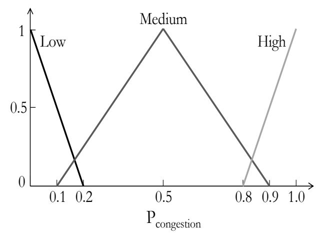
(a) Input q is divided into three symmetric input sets: low, medium and high.



(b) $clientBan_{Prio}$ is considered low when below 1 Mbps and high when above 2.5 Mbps.



(c) Only two membership functions compose the fuzzy set for N_{Prio}: low and high.



(d) $P_{congestion}$ is considered high when higher than 0.8 and low when below 0.2.

Figure 3: The fuzzy-based congestion detection module is composed of four fuzzy sets: three for each input (3a, 3b, 3c) and one for the output (3d).

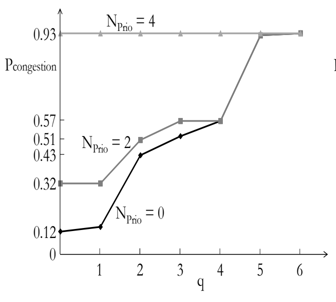
Also $clientBan_{Prio}$ is composed of a low, medium and high membership function (Figure 3b). When the bandwidth per client drops below 1 Mbps, it starts to be considered low. Intuitively, the lower qualities of a video are encoded at low bit-rates, usually below the 1 Mbps threshold. Consequently, if we want to assure a successful prioritization for at least the lower qualities of a video, a consistent amount of bandwidth should always be available in the prioritization queue. A bandwidth higher than 2.5 Mbps starts to be considered high, since even the higher quality segments (usually encoded at high bit-rates) can be prioritized without congesting the prioritized queue. Although fixed in this paper, these values can be easily modified by the service provider in order to reflect the specific conditions of the video streaming service, without altering the general design of the proposed fuzzy system.

Only two membership functions compose the fuzzy set for N_{Prio} (Figure 3c). In this case, a specific value of N_{Prio} is either considered low or high. A fine-grained representation of this variable is not needed in this case: only when a large number of consecutive prioritizations occur (e.g., higher than 3) the congestion detection module should avoid prioritizing the client.

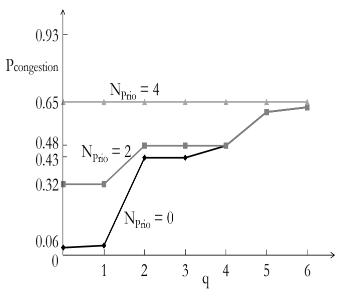
The output set for $P_{congestion}$ is also divided in low, medium and high (Figure 3d). When $P_{congestion}$ is higher than 80%, the probability of congesting the queue is considered high. Conversely, it is low when below 20% and

Table 1: The six fuzzy rules of the congestion detection module.

1. If $clientBan_{Prio}$ is HIGH then $P_{congestion}$ is LOW
2. If $clientBan_{Prio}$ is LOW then $P_{congestion}$ is HIGH
3. If N_{Prio} is HIGH then $P_{congestion}$ is HIGH
4. If q is HIGH and $clientBan_{Prio}$ is NOT HIGH then $P_{congestion}$ is HIGH
5. If q is LOW and $clientBan_{Prio}$ is NOT LOW then $P_{congestion}$ is LOW
6. If q is MEDIUM and $clientBan_{Prio}$ is MEDIUM and N_{Prio} is LOW then $P_{congestion}$ is MEDIUM



(a) $clientBan_{Prio} = 1$ Mbps



(b) $clientBan_{Prio} = 3$ Mbps

Figure 4: An example of the fuzzy decision plane obtained when $clientBan_{Prio}$ is fixed to 1 Mbps (4a) and 3 Mbps (4b). $P_{congestion}$ increases with the requested quality and the number of consecutive prioritizations, and decreases as the bandwidth per client increases.

considered medium otherwise.

The core of the fuzzy engine is represented by the fuzzy rules, which correlate the input and output sets and allow to make a decision (also called inference). The fuzzy rules for the congestion detection module are presented in Table 1. The first rule states that when the bandwidth is very high the probability of congesting the queue is considered low, independently from q and N_{Prio}. The opposite conclusion is drawn by the second rule. The third rule is designed to ensure a fair share of the prioritized system among all clients, by limiting the number of consecutive prioritizations for a client. The other rules take into account the requested quality level. As an example, a medium quality level will cause a medium congestion when the prioritization queue is in decent conditions and the client has not been prioritized too many times (rule six in Table 1).

Despite the simplicity of the proposed fuzzy model, a complex behavior can be obtained during the decision phase. As an example, Figures 4a and 4b show the decision plane of the fuzzy engine when fixing N_{Prio} to 0, 2 and 4 and $clientBan_{Prio}$ to 1 Mbps and 3 Mbps, respectively. The x-axis reports the requested quality level, while the y-axis the value of $P_{congestion}$. As expected, an increase of the bandwidth per client (i.e., less clients prioritized at the same time) results in a smaller probability of congestion. The highest value when $clientBan_{Prio}$ is 1 Mbps is 0.93 (Figure 4a), while this value drops to 0.65 when $clientBan_{Prio}$ is 3 Mbps (Figure 4b). $P_{congestion}$ follows an increasing monotonic trend with respect to the requested quality. In this case, the shape of the curves changes depending on the number of consecutive prioritizations and convergence is reached for the highest quality levels. Independently from N_{Prio}, a high quality level is always associated to a high risk of congesting the queue. Interestingly, when N_{Prio} is very high (i.e., higher than 4), $P_{congestion}$ only depends on $clientBan_{Prio}$. This behavior is mainly caused by the third rule of the fuzzy engine (see Table 1), which tries to limit as much as possible a large number of consecutive prioritizations. Consequently, the segment would be considered for prioritization only when sufficient bandwidth is available in the prioritization queue.

The performance of the congestion module in combination with the freeze predictor are thoroughly analyzed in Section 4.

Table 2: Characteristics of the HAS videos. The nominal average bit-rate for the 2-seconds segment version is reported, together with the standard deviation (between brackets). All values are expressed in kbps.

Big Buck Bunny	*Of Forest and Man*	*Tears of Steel*	*Elephant's Dream*	*Red Bull Playstreets*
334(±66)	366(±75)	254(±105)	382(±123)	300(±41)
522(±75)	553(±101)	507(±217)	795(±176)	896(±134)
791(±101)	824(±168)	811(±346)	1494(±526)	1180(±226)
1244(±257)	1519(±376)	1516(±680)	2444(±728)	1993(±291)
1546(±417)	2529(±839)	2427(±1057)	3431(±1481)	2995(±376)
2494(±531)	3798(±1631)	3020(±1349)	4228(±2390)	3991(±514)
3078(±867)	–	4028(±1746)	–	–

4. Performance Evaluation

4.1. Experimental Setup

The proposed OpenFlow framework has been implemented on the Mininet Network Emulator[1]. The HTTP server, where the video content is stored, is a Jetty Server[2]. We use five different HAS videos to evaluate the proposed framework under different streaming scenarios, selected from an MPEG-DASH compliant public repository[3]. The main characteristics of the videos are summarized in Table 2. The videos, which belong to different genres (animation, documentary, sports and movie), are encoded in variable bit-rate (VBR) and are capped to the first 10 minutes, if longer. Each video is available in a 1, 2, 4 seconds segment version, for a total of 15 different videos.

The HAS clients are implemented on top of the libdash library, the official software of the ISO/IEC MPEG-DASH standard [20]. The Libpcap library[4] is used to extract

[1]`http://mininet.org`
[2]`http://eclipse.org/jetty`
[3]http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/
[4]`http://www.tcpdump.org`

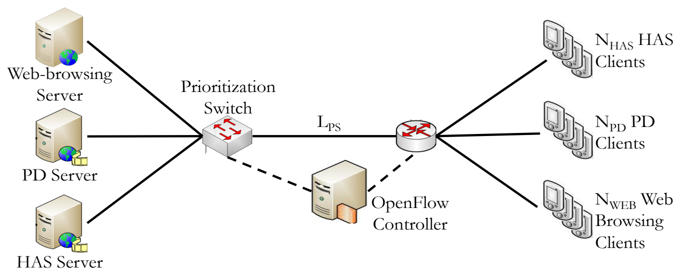


Figure 5: The emulated topology on Mininet is composed of several HAS, progressive download and web browsing clients, connected to the respective servers via the bottleneck link L_{PS}.

Table 3: Characteristics of the cross-traffic applications [22].

PD video streaming clients	**Web browsing clients**
Inter-arrival video request	*Inter-arrival page request*
Pareto distribution with mean 350 seconds and standard deviation to mean ratio of 2	Pareto distribution with mean 8.5 seconds and standard deviation to mean ratio of 1.5
Video size	*Page size*
Pareto distribution with mean 38 MBytes and standard deviation to mean ratio of 1.5	Based on web statistics [22]. Average 0.7 MBytes and standard deviation 1.5 MBytes.

the DSCP field from the received packets and thus enable prioritization-awareness. The rate adaptation heuristic used by the HAS clients is the FINEAS algorithm[5] [21]. The controller is implemented using POX[6], a Python-based controller, while the OpenFlow switches are realized via Open vSwitch[7]. The prioritization switch is equipped with a best-effort and a prioritized queue. Based on our previous work [5], a strict-priority policy is used for the experiments, i.e., the prioritized queue can transmit at a guaranteed rate, equal to 15% of the total channel capacity.

The emulated network topology is shown in Figure 5, where the position of the prioritization switch is illustrated. The prioritized queue is installed on the interface towards link L_{PS}. N_{HAS} HAS clients stream the video sequence at the same time from the same HAS Server. In order to provide an extensive evaluation of the proposed framework, we emulate 10 episodes of the video trace and average the results over the 10 runs. The capacity on link L_{PS} is kept fixed during each episode while the capacity on all the other links is over-provisioned. During our evaluation, we tested different values of the fixed capacity of link L_{PS} in order to investigate the performance of the proposed solution under different levels of network congestion.

Cross-traffic for HAS clients is introduced by two other types of application: N_{WEB} web browsing clients and N_{PD} PD video streaming clients. This applications mix allows to evaluate the performance of the proposed framework under realistic network conditions and to analyze the impact of prioritization both on HAS and background traffic. The implementation of the cross-traffic applications used in this paper is obtained by following the indications presented by Akhtar et al. [22]. PD clients can stream from a single PD server, shown in Figure 5. Web browsing clients can download a web page from seven different web servers, which has been shown to be the average number of web hosts used to download a web page [22] (for simplicity, only one host is shown in Figure 5). In order to download a web page, a web browsing client opens a single TCP connection to each of the available web servers. Table 3 summarizes the most important characteristics of the cross-traffic applications. The inter-arrival video request time and the size of the video for PD clients are both based on a Pareto distribution, as well as the inter-arrival page request time for the web browsing clients. The web page size is based on web statistics published by Google [22], and presents an average of 0.7 MBytes and a standard deviation of 1.5 MBytes. In our experiments, the number of HAS clients, PD clients and web browsing clients is fixed to 30, 25 and 15, respectively. The percentage of video streaming traffic has been set according to the Sandvine report on Internet QoE, which shows that video streaming applications currently dominate Internet traffic [23].

In order to provide an extensive benchmark of the proposed framework, we compare our results with those obtained using three other HAS solutions. First, the FINEAS heuristic described by Petrangeli et al. without in-network computation [21], which provides the baseline for the proposed ML-based framework. Second, the Microsoft ISS Smooth Streaming (MSS) client, a popular proprietary HAS client[8]. In our previous work [5], we showed already that these two purely client-based solutions were able to achieve a low number of video freezes as well as a good video quality when compared to other heuristics. Finally, we also analyze the performance of the network-based prioritization system proposed in our previous work [5]. In this case, the controller has a perfect knowledge of the status of the clients, in terms of video player buffer filling level, requested segment bit-rates and video segment duration and can therefore fully optimize the behavior of the HAS clients. Conversely, our ML-based solution does not rely on any of this information. Comparing the performance of this solution with that of the proposed ML-based framework allows to assess the trade-off between the amount of knowledge possessed by the controller and the effectiveness of the prioritization system. The heuristic embedded in the HAS clients for both network-based solutions is the FINEAS algorithm.

4.2. Training the Freeze Predictor Off-line

As explained in Section 3.2.2, a fundamental component of the proposed prioritization framework is the freeze

[5]In this work, the in-network computation proposed by Petrangeli et al. has not been implemented.

[6]`https://github.com/noxrepo/pox`

[7]`http://openvswitch.org`

[8]`https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming`

Table 4: Characteristics of the emulated streaming scenarios.

	Video player buffer size [s]	*Segment duration [s]*	*L_{PS} [Mbps]*
Scenario 1	6	1	65
Scenario 2	10	2	60
Scenario 3	12	4	55

predictor. Based on inputs as the available bandwidth for HAS traffic, the inter-arrival time of consecutive GET requests and the requested quality, the predictor should be able to detect conditions possibly leading to a video freeze and drive the network prioritization to avoid it. The freeze predictor is based on the RUSBoost algorithm, due to its accuracy in imbalanced classification problems.

In order to train the freeze predictor, we collected a large number of HAS logs using the experimental setup reported in Figure 5. The network emulation setup allows us to have full control of the experiments and to collect logs about the clients' behavior after the streaming session. As an example, we can identify off-line when a client has experienced a video freeze. Consequently, we can create a training set of labeled data, which can be used to build a predictor. Each data point of the training set is associated with a segment request made by a client, and is composed of the six inputs of the predictor (see Section 3.2.2) and a label indicating whether the download of the requested segment resulted in a freeze or not. A machine learning algorithm can then be used to build a freeze predictor. The collected logs should be representative of the possible different video streaming scenarios, in terms of clients, network and videos configuration. For this reason, we experimented with three different streaming scenarios, reported in Table 4. We varied the buffer size of the clients, the segment duration of the video and the capacity of link L_{PS} (see Figure 5). Each scenario has been tested with all the videos reported in Table 2, for a total of 15 different experiments.

In total, 4500 clients' logs have been collected, resulting in almost 1.5 million individual segment requests. Only 2.5% of the segment requests are affected by a video freeze, which confirms the imbalanced nature of the analyzed classification problem. We divided the logs into a training set, to train the predictor, and a validation set, to test its performance. Particularly, 85% of the logs collected with the *Big Buck Bunny*, *Of Forest and Man* and *Tears of Steel* videos are used to train the predictor. The remaining 15% is used as validation set. Moreover, the logs collected with the *Elephant's Dream* and *Red Bull Playstreet* videos are used as an additional validation set. This choice allows to assess to what extent the predictor can adapt to untrained videos, which is a fundamental requirement for the real deployment of the proposed approach.

We compare the performance of the RUSBoost algorithm in the freeze predictor task using four other popular classifiers: Random Forest, AdaBoost, GradientBoost and 1-Nearest Neighbourhood (1-NN) [24]. The Random Forest classifier builds an ensemble of decision trees, each contributing towards the final classification. In a random forest, N different decision trees are trained on a different sub-sample of the given dataset. Each training set is a random sample with replacement of the original dataset, and has therefore the same size as the original one. When used for classification, each tree in the random forest participates in a vote, where the class selected by the majority of the trees in the random forest is finally chosen. This process allows to increase the classification accuracy and control over-fitting, a common problem affecting standard decision trees. The AdaBoost algorithm applies the same boosting technique as the RUSBoost, without the random undersampling filtering on the training dataset. In boosting, each sample of the training set is associated with a weight. At the beginning of the training process, all the weights have the same value. At each iteration a decision tree is classified and the weights are updated. Particularly, the weights of samples that are misclassified are increased, while the weights of samples that correctly classified are decreased. This way, at each iteration, the new decision tree is forced to focus on those training samples that were misclassified by the previous decision trees. The GradientBoost classifier is a generalization of boosting methods. The number of decision trees is set to 50 for all the algorithms previously introduced. Differently from the other methods, the 1-NN classifier does not create a generalized model of the problem (e.g., the ensemble of decision trees created by the boosting algorithms). A new data point is classified with the same label as the label of the closest data point in the training set. This classifier does not require to fit a model and is typically "memory-based" (i.e., the whole training set has to be stored to classify unknown samples). Given a new point to classify, the k closest neighbors in the training set are identified, and the new point is classified using a majority vote among the k neighbors. Despite its simplicity, this method has proven to be very effective in many real classification problems. As an example, Jian et al. report how a K-NN classifier can be used to optimize the behavior of 5G networks, in terms of handover selection or energy savings [25]. The training time for the different classifiers is in the order of tens of minutes on a Dell Latitude E5530 running Ubuntu 12.04 LTS 64-bit, Intel Core i5-3320M CPU @ 2.60GHz processor and 8 GB of memory. It is worth stressing that the training phase can be carried out off-line, without any real-time constraint. Only the trained model is actually used on-line in the OpenFlow controller to actively avoid the occurrence of video freezes.

The results of the ML algorithms comparison are presented in Figure 6. The y-axis reports the percentage of correctly predicted freezes, also called true positives. Each bar of the graph represents the percentage of true positives over the entire validation set, composed of logs from the trained videos and untrained videos. The RUSBoost classifier is able to consistently outperform the other classifiers. The freeze prediction accuracy is 99% for trained

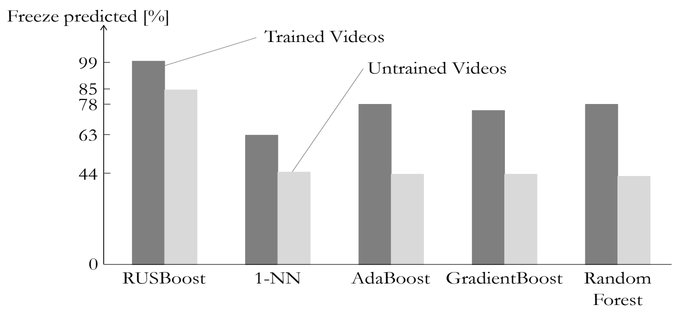


Figure 6: The RUSBoost algorithm is able to outperform all the other ML solutions, both for trained and untrained videos. The y-axis reports the percentage of correctly predicted freezes.

videos (*Big Buck Bunny*, *Of Forest and Man*, *Tears of Steel*) and 85% for untrained videos (*Elephant's Dream* and *Red Bull Playstreet*). Methods based on an ensemble of decision trees (AdaBoost, GradientBoost and Random Forest) are also able to reach good performance on trained videos, with almost 80% of the freezes correctly predicted. Creating an ensemble of weak classifiers helps in reducing the negative effects due to the unbalanced training set. Despite that, their performance consistently drops when exposed to untrained videos. The 1-NN classifier obtains the lowest performance[9]. As this classifier does not build a model of the analyzed problem, it is not able to infer the general conditions possibly leading to a freeze.

Another important parameter of the freeze predictor is the percentage of non-freezes correctly predicted, i.e., the percentage of video segments not leading to a video freeze that are correctly classified as such. This metric is referred to as true negative. As for the true positives, also the true negatives should be maximized. All the classification algorithms result in a high true negatives accuracy. The RUSBoost classifier presents an accuracy of 95%, while all the other classifiers have an accuracy higher than 98%. The 3%-4% loss of the RUSBoost classifier is largely counterbalanced by the gain in terms of true positives, which reaches 40% for untrained videos, as shown in Figure 6. These results therefore confirm the suitability of the RUSBoost algorithm for the freeze predictor task.

Table 5 reports the freeze prediction accuracy of the different classifiers, for each video. The total predicted freeze time is also reported, between brackets. This value corresponds to the total amount of freeze time associated to the correctly predicted freezes. The RUSBoost algorithm is able to outperform the other classifiers, independently of the video. As expected, higher accuracies are obtained for trained videos. The lowest performance is reached in the 2-seconds *Red Bull Playstreet* video, where only 37% of the total freeze time can be anticipated. Nevertheless, the performance of the RUSBoost algorithm is

Table 5: Summary of the off-line freeze prediction task. The percentage of correctly predicted freezes is reported, together with the corresponding percentage of predicted freeze time (between brackets).

		RUSBoost	*AdaBoost*	*Random Forest*	*1-NN*	*Gradient Boost*
Big Buck Bunny	1s	90(99)	78(85)	79(86)	64(73)	76(85)
	2s	100(100)	78(80)	78(83)	68(69)	62(68)
	4s	97(98)	49(56)	48(60)	45(70)	52(67)
Of Forest and Man	1s	100(100)	81(86)	83(87)	66(73)	79(85)
	2s	100(100)	85(86)	88(90)	68(69)	82(85)
	4s	96(99)	70(91)	60(86)	45(70)	60(86)
Tears of Steel	1s	100(100)	81(86)	83(89)	76(81)	84(87)
	2s	100(100)	73(74)	71(73)	53(50)	69(73)
	4s	98(98)	62(79)	56(79)	42(59)	51(70)
Elephant's Dream	1s	94(93)	54(62)	59(69)	51(58)	60(70)
	2s	74(69)	26(27)	21(22)	35(33)	26(29)
	4s	63(64)	17(18)	8(9)	25(24)	8(10)
Red Bull Playstreet	1s	93(81)	64(60)	70(65)	61(55)	76(69)
	2s	72(37)	32(20)	13(10)	35(18)	10(6)
	4s	92(96)	36(50)	39(59)	33(45)	37(54)

still acceptable, even for untrained videos. The AdaBoost classifier is the second best choice in terms of accuracy. As previously mentioned, the RUSBoost algorithm adds the random undersampling technique on top of the AdaBoost classifier. The 1-NN algorithms is able to achieve constant performance, both for trained and untrained videos. As it appears from Table 5, the freeze predictor is designed to anticipate possible video freezes, but it does not have any notion on the possible duration of these freezes. As an example, predicting 63% of the freezes in the 4-seconds *Elephant's Dream* video leads to 64% of the total freeze time predicted. This value drops to 37% in the 2-seconds *Red Bull Playstreet* video, even though the percentage of correctly predicted freezes is 72%.

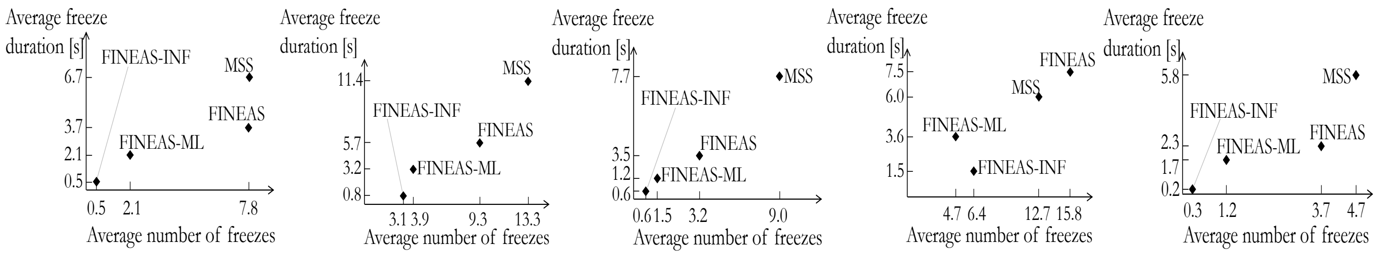
In light of the above results, the RUSBoost algorithm has been chosen to implement the freeze predictor functionalities for the rest of the paper.

4.3. On-line Freeze Reduction

In this section, we plan to investigate the performance of the proposed ML-based framework in terms of on-line freeze reduction. The freeze predictor trained in Section 4.2 is now plugged into the network controller together with the fuzzy-based congestion detection module, in order to foresee the occurrence of video freezes during an ongoing streaming session and try to actively avoid them (see Section 3). We compare the performance of our solution, referred to as FINEAS-ML, with that of the MSS and FINEAS clients and the network-based solution proposed in our previous work [5], called FINEAS-INF. As explained in Section 4.1, this solution has access to very detailed information about the video characteristics and the client's buffer filling status. The experiments have been repeated for each video and for each HAS solution, using the same streaming scenarios as in Table 4, for a total of 60 experiments. Each experiment has been repeated 10 times.

A high level summary of the obtained results is presented in Figure 7. For each experiment and for each iteration, we first compute the average number of freezes and

[9]Several values of the k parameter have been tested, with no significant differences in the obtained results.



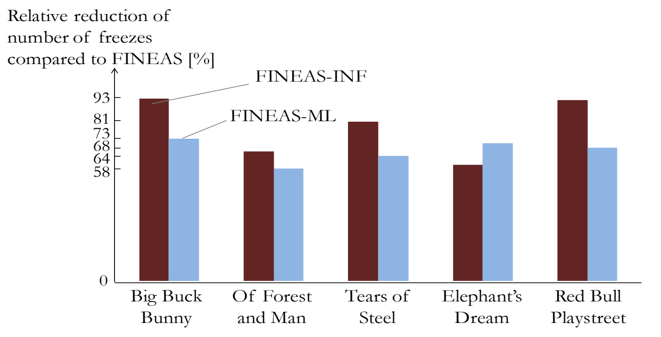
(a) Big Buck Bunny (b) Of Forest and Man (c) Tears of Steel (d) Elephant's Dream (e) Red Bull Playstreet

Figure 7: The purely client-based MSS and FINEAS heuristics present the worst performance, independently of the video. The proposed ML-based solution can consistently reduce the amount of video freezes, in all scenarios. The best performance is reached by the FINEAS-INF solution, which can use detailed information about the video and the clients' characteristics. The x-axis and y-axis report the average number of freezes and the average freeze duration, respectively.

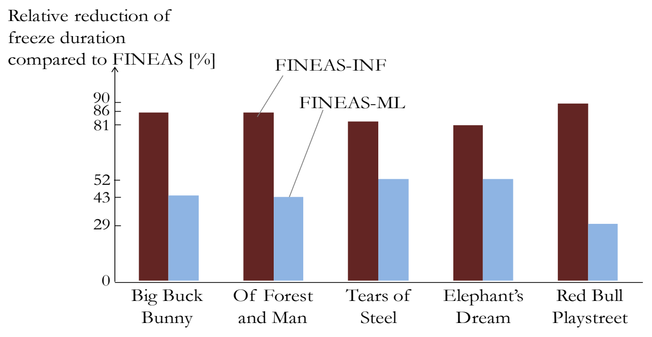
the average freeze duration over the entire group of HAS clients. We then average these results per video. The x-axis reports the average number of video freezes obtained with the MSS, FINEAS, FINEAS-ML and FINEAS-INF solutions. The y-axis reports the average freeze duration, in seconds. As expected, the MSS and FINEAS algorithms present the worst performance, independently from the streamed video. Both algorithms are purely client-based and can fail fully optimizing the QoE of the video streaming session. Apart from the *Elephant's Dream* video (see Figure 7d), the FINEAS client is always able to outperform the MSS client. The number of freezes strongly depends on the video characteristics. As reported in Table 2, all videos have different bit-rates, ranging between 254 and 4228 kbps and are encoded in VBR. In order to maintain the visual quality constant for each quality level, the bits saved during the encoding of simple scenes are reused to encode more complex scenes (e.g., with a lot of movement). Consequently, the bit-rate of a specific quality level is not constant but varies depending on the video content itself. VBR encoding complicates the rate adaptation of the clients, as the segment size is not constant and it is not known in advance. As such, the videos *Of Forest and Man* and *Elephant's Dream*, which are characterized by a high variability (see Table 2), result in the highest amount of video freezes (Figures 7b and 7d). Particularly, the FINEAS heuristic results in the worst performance for the *Elephant's Dream* video, which presents the highest bit-rate and variability for the lowest quality. The FINEAS heuristic has a more aggressive behavior than the MSS one and is therefore more susceptible to freezes for this specific video. Despite the high standard deviation of the bit-rates of the *Tears of Steel* video (see Table 2), results for the FINEAS heuristic are in line with those of the *Big Buck Bunny* and *Red Bull Playstreet* videos. The nominal bit-rates of the *Tears of Steel* video are the lowest or second lowest among the entire set, which simplifies the adaptation of the clients. The FINEAS-ML and FINEAS-INF solutions are able to outperform purely client-based solutions, both in terms of number of freezes and freeze duration. It is worth noting that the heuristic embedded in the HAS clients for both network-based solutions is the FINEAS algorithm, which represents the baseline for the performance analysis. Our proposed ML-based approach is able to consistently reduce the amount of freezes for all videos. This result also entails a strong reduction of the average freeze duration. The FINEAS-INF solution is able to reach the best performance overall and to almost completely eliminate video freezes. As previously explained, in this case the network controller has very detailed information about the characteristics of the videos (nominal bit-rates and segment duration) and the status of the client's buffer and it can take the best decision in terms of segment prioritization. These results show a clear trade-off between the performance of the prioritization system and the accessibility of the inputs for the network controller. Unlike the FINEAS-INF approach, our ML-based solution does not require any special assumption on the clients' behavior, nor on the video characteristics, but it is still able to considerably reduce the amount of video freezes.

Figure 8 quantifies the gain brought by the proposed ML-based solution and the FINEAS-INF one, when compared to the FINEAS heuristic. Figures 8a and 8b report the relative reduction in terms of number of freezes and freeze duration, respectively, for the five analyzed videos. Despite not having access to refined information on the client's status and video characteristics, our proposed ML-based solution shows similar performance to the FINEAS-INF one, in terms of freeze reduction (Figure 8a). Particularly, our ML-based solution is able to reduce video freezes with about 65% compared to FINEAS, which is about 15% less than FINEAS-INF.

It is worth noting that worse performance is obtained in terms of freeze reduction compared to the results presented in Section 4.2. The reasons for this behavior are two-fold. First, network prioritization is inherently an on-line process, while the freeze identification presented in Section 4.2 was performed off-line, on completed video sessions. Due to the limited resources of the prioritization queue, whose bandwidth is fixed to 15% of the total capac-



(a) The ML-based solution is able to reduce the amount of video freezes with about 65% when compared to the FINEAS client, which is 15% less than what achieved by the FINEAS-INF solution.



(b) The proposed network-based prioritization can reduce freeze time with about 45%, even without having any information on the client's and video characteristics. The FINEAS-INF solution, which has access to this information, can reduce freeze time with about 85%.

Figure 8: Relative comparison between the FINEAS-INF and ML-based solutions compared to the FINEAS heuristic, in terms of number of freezes (8a) and freeze duration (8b).

ity of link L_{PS} (see Figure 5), not all the segments possibly leading to a freeze can actually be prioritized. In Section 3.2.3, we presented the fuzzy-based congestion detection module, whose role is to understand whether prioritizing a segment would congest the prioritization queue. When network congestion is high, many segments are identified by the freeze predictor module as in risk of a video freeze, but not all of them can be prioritized due to the decision of the congestion detection module. Moreover, as the network controller of our ML-based approach does not have any information on the video characteristics and clients' status, it results in a more conservative prioritization behavior. On average, 10% of the video is prioritized in the FINEAS-ML case, while this number rises to 15% in the FINEAS-INF case. Second, bandwidth drops leading to video freezes can occur after a prioritization decision has been taken by the network controller. While the FINEAS-INF solution knows the buffer filling level of the clients and can employ a safety prioritization in this case, the same is not true for our ML-based solution, which only has limited information. This aspect also explains the better results obtained by the FINEAS-INF solution in terms of freeze time reduction, shown in Figure 8b. Our ML-based approach and FINEAS-INF solution can reduce freeze time with about 45% and 85% compared to FINEAS, respectively. As reported in Section 4.2, the freeze predictor based on the RUSBoost algorithm is designed to reduce the amount of video freezes. This condition however does not always result in a correspondent freeze time reduction.

Table 6 reports the complete results, for each video and for each different version in terms of segment duration. We report the average requested quality, number of freezes and freeze duration. We also perform an analysis to assess whether the differences between the different heuristics are statistical significant or not (t-test, $p \leq 0.05$). Each heuristic is assigned a letter, from *a* for FINEAS to *d* for MSS. Results associated with one of these letters are not statistically different from the results obtained by the heuristic associated with the correspondent letter. As an example, if a result of the FINEAS heuristic is associated with the letter *d*, then it is not statistically different from the result obtained by the MSS client, and viceversa. Overall, the results of the FINEAS-INF and FINEAS-ML approaches show strong statistical significance when compared to the FINEAS and MSS heuristics. The FINEAS-INF solution is able to almost completely eliminate video freezes, in all video streaming scenarios. Only in the 1-second segment version of the *Of Forest and Man* and *Elephant's Dream* videos, this approach shows sub-optimal performance. In the FINEAS-INF approach, one of the inputs of the network controller are the nominal bit-rates of the streamed video. As reported in Table 2 though, these two videos present a high bit-rate variability. Moreover, encoding the video with 1-second segments results in a slightly higher nominal bit-rate (about 5%) compared to the 2-seconds version, due to the cost of more frequent I-frames at the beginning of the segments. These disturbances complicate the task of the controller.

Thanks to the RUSBoost and fuzzy algorithms, our ML-based approach is able to reduce video freezes when compared to the baseline FINEAS heuristic, without needing any explicit information on the video or client's configuration. Our approach presents worse performance compared to the FINEAS heuristic only in two cases (4-seconds *Big Buck Bunny* and 2-seconds *Red Bull Playstreet*), but these differences are not statistically significant. Similar conclusions can also be drawn for the MSS heuristic. These results again confirm the suitability of the proposed approach for the on-line freeze reduction task. Another important aspect to analyze is the requested video quality. In order to maximize users' QoE, a heuristic should be able to not only avoid freezes, but also to request the highest possible video quality. As reported in Table 6, only minor differences can be noted among the different HAS solu-

Table 6: Summary of the obtained results in terms of average quality (q, expressed as an integer between 0 and the highest quality level, which varies between 5 or 6 depending on the video), number of freezes (NF) and freeze time (FT, in seconds). Differences between cells associated with the same uppercase letter are not statistically significant (t-test, p≤0.05).

		FINEAS(a)			*FINEAS-ML(b)*			*FINEAS-INF(c)*			*MSS(d)*		
		q	NF	FT[s]	q	NF	FT[s]	q	NF	FT[s]	q	NF	FT[s]
Big Buck Bunny	1s	2.98	17.4	7.33	2.95	4.31	3.69^{d}	2.97	0.59	0.13	3.1	9.54	3.02^{b}
	2s	2.84	4.83^{d}	2.52^{d}	2.68	0.68	0.91	3.02	0.02	0.01	2.79	6.02^{a}	3.42^{a}
	4s	2.18	1.16^{b}	1.26^{b}	2.08	1.31^{a}	1.71^{a}	2.05	0.84	1.41	2.14	7.93	13.7
Of Forest and Man	1s	2.72	16.93^{d}	7.43^{d}	2.47	6.40	3.69	2.67	8.62	0.97	2.70	15.80^{a}	6.49^{a}
	2s	2.63	8.32^{d}	4.66^{d}	2.36	2.54	2.29	2.24	0.04	0.01	2.60	7.18^{a}	5.18^{a}
	4s	2.06	2.77^{b}	4.99^{b}	1.85	2.88^{a}	3.76^{a}	1.84	0.74	1.39	1.94	16.83	22.47
Tears of Steel	1s	2.90	5.04	2.80^{d}	2.78	1.62	1.31	2.74	0.96	0.22	2.64	8.96	3.25^{a}
	2s	2.67	2.76	1.63	2.61	0.29	0.39	2.69	0.03	0.01	2.60	5.50	4.77
	4s	2.03	2.66	5.21	1.96	1.81	2.94	1.90	0.90	1.56	2.01	8.78	19.11
Elephant's Dream	1s	2.07	27.60^{d}	10.39^{d}	1.94	6.69	3.92	2.04	16.91	3.31	2.05	24.69^{a}	8.52^{a}
	2s	1.89	9.51	4.92	1.79	3.25	2.83^{d}	1.88	0.68	0.24	1.68	5.73	3.32^{b}
	4s	1.59	10.27^{d}	7.24^{d}	1.51	4.03	4.11	1.52	1.65	1.05	1.69	7.81^{a}	6.22^{a}
Red Bull Playstreet	1s	2.27	8.80	3.45	2.07	1.58	1.53^{d}	2.15	0.31	0.05	2.16	4.57	1.35^{b}
	2s	2.18	0.76^{b}	0.83^{b}	2.05	1.05^{a}	$2.08^{a,d}$	2.03	0.08	0.01	1.85	2.43	1.85^{b}
	4s	1.64	1.69	2.75	1.52	0.91	1.38	1.53	0.40	0.59	1.68	7.07	14.36

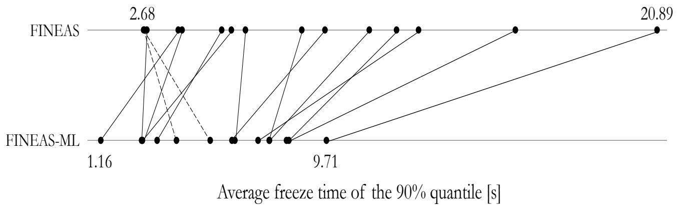


Figure 9: Our ML prioritization framework has a consistent impact on the freeze distribution, by reducing freeze time for all the clients. The figure reports the average freeze time of the 90% quantiles, for all the possible network and video configurations.

tions. The average quality slightly decreases (with about 5%) in the FINEAS-INF and ML-based cases when compared to the FINEAS heuristic. This result is mainly due to the prioritization mode, where a client is forced, by design, to request the lowest quality level in case the most recently downloaded segment has been prioritized. It is also worth mentioning that the quality tends to decrease when longer segments are used. In this situation, the clients have less fine-grained decision points to adjust the quality to the available bandwidth, consequently resulting in a more conservative behavior. This condition is especially true when the maximum buffer size of the clients is limited to a few seconds in order to reduce the end-to-end delay in live video streaming scenarios, as in our experiments.

Another important aspect to consider is the distribution of video freezes among the different clients. Ideally, the freeze reduction reported in the previous paragraphs should be equally shared among the different clients, in order to provide fairness to the entire system. To show this behavior, we computed the 90% quantile of the average client freeze time, for all the different network and video configurations as reported in Table 6. The 90% quantile expresses the maximum freeze time experienced by 90% of the clients. By using the proposed ML prioritization framework, we should be able to reduce not only the average freeze time (as shown in Figure 8) but also its 90% quantile, which would entail that all the clients benefit from the prioritization system. The results of this analysis are reported in Figure 9, where the 90% quantile of all the 15 different streaming configurations are reported, for both FINEAS and FINEAS-ML. Using our prioritization framework results in a consistent reduction of the 90% quantile of the average freeze time, when compared to the purely client-based FINEAS heuristic. As explained before, this reduction entails that all the HAS clients benefit from prioritization. In only two configurations the 90% quantiles increase, which correspond to the streaming scenarios where the 4-seconds *Big Buck Bunny* and 2-seconds *Red Bull Playstreet* videos are used (see dotted lines in Figure 9). As reported in Table 6, in these two cases the average freeze time is higher when prioritization is used (even though this difference is not statistically significant).

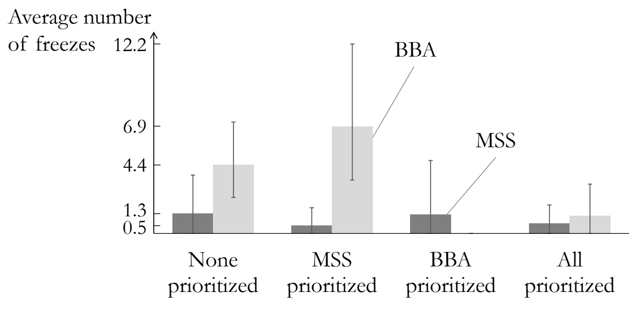
4.4. Heterogeneous Clients

In this section, performance results are shown of the prioritization framework when different rate adaptation heuristics are used instead of the FINEAS algorithm. As reported in Section 4.2, the freeze predictor based on the RUSBoost algorithm has been trained using logs generated by FINEAS clients only. Despite that, the prioritization system should provide good performance even when different clients are equipped with different heuristics, a common situation in real deployments. The same topology as in Figure 5 is used for the experiments, with capacity on link L_{PS} fixed to 65 Mbps. The clients stream the 1-second version of the *Big Buck Bunny* video.

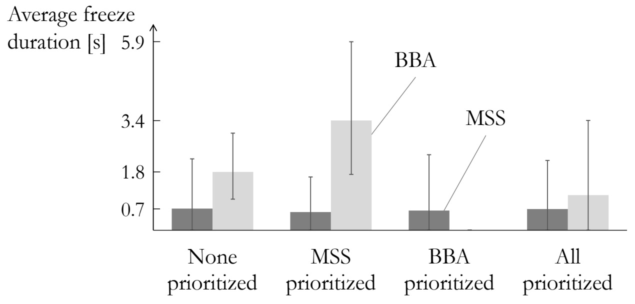
In a first set of experiments, all HAS clients use the BBA algorithm by Huang et al. [1]. The BBA client is an example of a purely buffer-based algorithm, where the next quality is selected based on the buffer level only. Particularly, Huang et al. showed on a real field test that the BBA client is able to outperform the standard Netflix

Table 7: Summary of the results when the BBA algorithm is used as adaptation heuristic, in terms of average quality (q, expressed as an integer between 0 and 6), number of freezes (NF) and freeze time (FT, in seconds). The 10% and 90% quantiles are reported between brackets.

	q	NF	FT[s]
BBA	3.2(2.4-3.9)	4.6(1.5-9.0)	1.9(0.5-3.8)
BBA-ML	2.8(2.4-3.2)	2.0(0.3-3.6)	1.8(0.1-3.4)



(a) Average number of freezes.



(b) Average freeze duration.

Figure 10: Even in a heterogeneous scenario, our system can always reduce the number of freezes for the prioritized clients. When only one group is prioritized, the influence on the non-prioritized group depends on the underlying heuristic.

player. This choice allows to assess the performance of the prioritization system when the clients are equipped with a completely different heuristic compared to the one used for training. The main results are reported in Table 7, for a buffer size fixed to 20 seconds. The proposed ML-based framework can consistently reduce the number of freezes by 57%. It is worth stressing that this result is achieved even though the freeze predictor module has been trained with a different adaptation heuristic. Even though freezes can be limited, the average freeze time is similar for both the BBA and BBA-ML solutions. The prioritization system is effective in avoiding short freezes, while long freezes are more difficult to predict because of the different behavior of the purely buffer-based BBA heuristic compared to the FINEAS heuristic used for training. As expected, the video quality is slightly reduced when prioritization is enforced, since the clients are forced to request the lower quality in case of prioritization.

In a second set of experiments, we investigate a heterogeneous scenario, where 50% of the clients are equipped with the BBA heuristic and 50% with the MSS one. We analyze the performance of the system both when all the clients are prioritized and when only one group can benefit from prioritization. The same experimental setup as for the previous experiment has been used. The correspondent results are shown in Figure 10, in terms of average number of freezes and freeze duration. For each emulated iteration, we also computed the 10% and 90% quantiles, which quantify the maximum number of freezes and freeze time experienced by 10% and 90% of the HAS clients, respectively. Each point of the graphs is therefore associated with the average 10% and 90% quantiles over the 10 iterations. When none of the clients is prioritized, the BBA heuristic exhibits worse performance than the MSS one, both in terms of number of freezes ($\times 3.4$) and freeze duration ($\times 2.6$). This behavior is due to the purely buffer-based nature of the BBA client, which does not take into account the available bandwidth and is therefore more susceptible to freezes unless a very large buffer is used. When only the MSS group is prioritized, freezes can be reduced by 62% (see Figure 10a) for MSS clients, while freeze duration is similar to the non-prioritized case (see Figure 10b). Interestingly, prioritizing the MSS group has a negative impact on the performance of the BBA group, as both the number of freezes and freeze time increase when compared to the non-prioritized case. When prioritization is enforced by the controller, the bandwidth available for the clients in the best-effort queue can drop. As BBA clients do not consider this metric in their adaptation, they are affected by more video freezes. When only the BBA group is prioritized, freezes can be completely eliminated, for all BBA clients. This result does not negatively affect the performance of the MSS clients, which consider both the buffer level and the available bandwidth in their quality adaptation, and can therefore react quickly to changing bandwidth conditions. When all the clients can benefit from prioritization, freezes are reduced for both groups. Even though the ML-based prioritization system is trained in a scenario where all the clients are equipped with the FINEAS algorithm, the learned client model can be effectively re-used to optimize the behavior of other adaptation heuristics, even in heterogeneous scenarios.

4.5. Multiple Bottlenecks Network Topology

So far, we have investigated a single bottleneck scenario, where congestion can occur on one link only (i.e., link L_{PS} in Figure 5). In real networks however, bottlenecks can arise on different links simultaneously and at different network levels. The goal of this section is therefore to investigate the performance of the proposed ML-based solution in a multi-bottleneck network scenario. In this scenario, each possible bottleneck should be associated with an independent network controller, equipped with the same set of algorithms as described in Section 3.2. It is

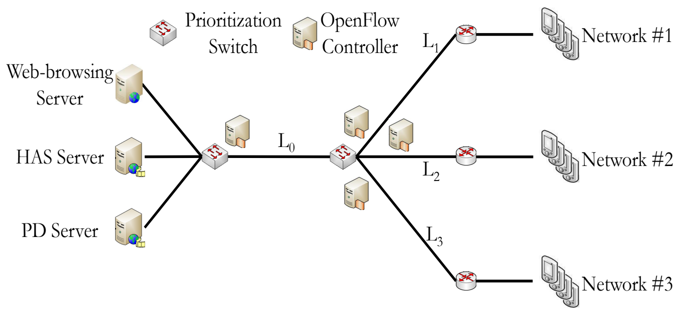
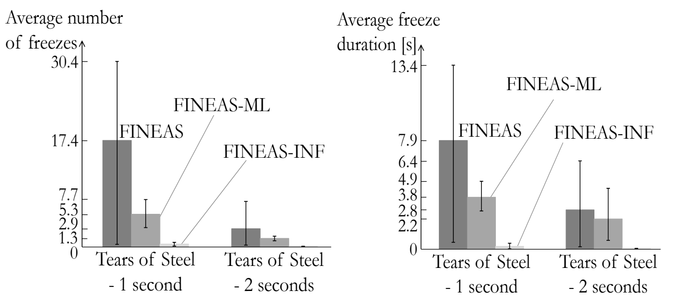


Figure 11: Three networks compose the topology emulated on Mininet. Links L_0, L_1, L_2 and L_3 are the bottlenecks and are equipped with a prioritization queue. Each link is associated with an independent OpenFlow controller.

important to stress that no communication should be envisioned among the independent controllers, as this approach would require extra-signalling inside the network. Despite that, the controllers should still be able to achieve a global coordinated behavior in order to provide end-to-end prioritization. This distributed approach presents two advantages with respect to a centralized one, where one single controller is responsible for all the network bottlenecks. First, it is inherently more scalable. In our previous work [5], we showed that the proposed prioritization system can control up to 5000 HAS clients simultaneously. The controller could therefore be overloaded when a centralized solution is used. Second, a distributed approach is easier to deploy, as no modifications to the design presented in Section 3.2 are required.

In light of the above, the multi-bottleneck topology shown in Figure 11 has been emulated on Mininet. The HAS, PD and web browsing clients are divided into three networks. Each network is composed of 10 HAS clients equipped with the FINEAS heuristic, 8 PD clients and 5 web-browsing clients, for a total of 69 clients. Links L_0, L_1, L_2 and L_3 represent the possible bottlenecks for the HAS clients. Each link is equipped with a prioritization queue, with bandwidth equal to 15% of the channel capacity, and is controlled by an independent network controller. Even though the controllers for links L_1, L_2 and L_3 can be located together, we decided to logically split them in order to show the performance of the system in a fully distributed scenario. Two different video streaming scenarios have been evaluated. In the first one, the HAS clients are equipped with a 6 seconds buffer and stream the 1-second segment version of the *Tears of Steel* video. Capacity on links L_0 and $L_{1\text{-}3}$ is fixed to 75 Mbps and 30 Mbps, respectively. In a second scenario, the 2-seconds segment *Tears of Steel* video is streamed, and the clients use a 10 seconds buffer. L_0 has a capacity of 70 Mbps, while links $L_{1\text{-}3}$ have a capacity of 28 Mbps. Each experiment has been repeated 10 times. Depending on the cross-traffic on links L_1, L_2 and L_3, the actual bottlenecks for the three networks dynamically change. This way, we can explore a wide range of network configurations, where the three



(a) Average number of freezes (b) Average freeze duration

Figure 12: The proposed ML-based approach can reduce the amount of freezes and the freeze time with respect to the FINEAS heuristic, even in a multi-bottleneck scenario with independent controllers. The FINEAS-INF solution achieves the best results overall.

networks could possibly influence each other.

Figure 12 reports the obtained results for the FINEAS, FINEAS-ML and FINEAS-INF approaches, in terms of average number of freezes and average freeze duration. Each point of the graphs is also associated with the average 10% and 90% quantiles over the 10 iterations. The client-based FINEAS heuristic results in the highest amount of freezes and freeze time. The presence of congestion at multiple network levels makes it harder for the client to avoid freezes, which are up to 3 times higher than in the single bottleneck case. Moreover, some of the clients experience very high freeze time (up to 13.4 seconds), as the quantiles indicate. Network-based prioritization can consistently improve the performance of the system, both for the proposed FINEAS-ML approach and the FINEAS-INF one. Even though independent, the controllers are able to take coordinated actions on which client to prioritize. This behavior is due to the type of inputs used by the controllers to decide on prioritization. Particularly, each controller can obtain *local* information about the status of the controlled link and *global* information about the HAS clients. Information about the clients is composed of the quality level of the requested segment and the time between consecutive HTTP GET requests, as described in Section 3.2.2. The status of the controlled link is a local information that is only available to the specific controller. Conversely, the requested quality and the GET inter-arrival time are measurements that can be obtained by all controllers, independently of their position. Consequently, the controllers are fed with some inputs that are global and shared with all the other controllers. This specific condition facilitates coordination among the otherwise independent controllers. As expected, the FINEAS-INF solution is able to reach the best performance, both in terms of video freezes and freeze duration. Our ML-based solution can reduce video freeze with 70% in the 1-second segment *Tears of Steel* video and 55% in the 2-seconds segment version (Figure 12a), when compared to FINEAS.

These results correspond to a 50% and 13% freeze time reduction (Figure 12b). The low dispersion indicated by the 90% quantiles also shows that all the clients obtain similar performance and confirms the effectiveness of the proposed prioritization system. It is worth stressing that these results are obtained in a completely distributed scenario, where the controllers do not communicate among eachother, and without using any information on the streamed video or on the clients' status. Despite that, the proposed ML-based approach can still provide good performance, even in a multi-bottleneck network scenario.

5. Conclusions

We presented in this paper a novel network-based framework to prevent the occurrence of video freezes for HAS clients. The main element of this framework, implemented using the OpenFlow protocol, is a network controller. This controller can prioritize the delivery of video segments likely leading to a freeze using a dedicate queue. Prioritization is driven by a machine learning engine, based on the RUSBoost algorithm and fuzzy logic. The RUSBoost algorithm is used to detect whether a client is close to a freeze, while fuzzy logic allows to understand whether the conditions of the prioritization queue are good enough to successfully prioritize the segment. No knowledge on the video, in terms of bit-rates and segment duration, is required nor on the client's configuration, in terms of initial buffering time. This aspect simplifies the practical applicability of the proposed framework in a real deployment. Results obtained through emulation showed that our ML-based approach can consistently reduce video freezes with about 65% and freeze time with 45%, when compared to the benchmarking heuristics FINEAS and MSS. Moreover, the proposed approach has also been evaluated in a multi-bottleneck network scenario, where we showed that a system of distributed independent controllers is still able to reduce the amount of video freezes with about 60%. These results represent an important step toward the optimization of the final QoE of the users watching online videos.

Acknowledgement

The research was performed partially within the strategic research project SMILE-IT funded by VLAIO and the iMinds PRO-FLOW project (under VLAIO grant agreement no. 150223). 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] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, M. Watson, A buffer-based approach to rate adaptation: Evidence from a large video streaming service, in: Proceedings of the 2014 ACM Conference on SIGCOMM, ACM, New York, NY, USA, 2014, pp. 187–198.

[2] CONVIVA, 2015 viewer experience report, http://www.conviva.com/conviva-viewer-experience-report/vxr-2015/.

[3] R. Masoudi, A. Ghaffari, Software defined networks: A survey, Journal of Network and Computer Applications 67 (2016) 1 – 25.

[4] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, A. Napolitano, Rusboost: A hybrid approach to alleviating class imbalance, Trans. Sys. Man Cyber. Part A 40 (1) (2010) 185–197.

[5] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, F. De Turck, Software-defined network-based prioritization to avoid video freezes in http adaptive streaming, International Journal of Network Management 26 (4) (2016) 248–268.

[6] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, P. Tran-Gia, A survey on quality of experience of http adaptive streaming, IEEE Communications Surveys Tutorials 99, 2014.

[7] X. Yin, A. Jindal, V. Sekar, B. Sinopoli, A control-theoretic approach for dynamic adaptive video streaming over http, SIGCOMM Comput. Commun. Rev. 45 (4) (2015) 325–338.

[8] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, D. Oran, Probe and adapt: Rate adaptation for http video streaming at scale, IEEE Journal on Selected Areas in Communications 32 (4) (2014) 719–733. doi:10.1109/JSAC.2014.140405.

[9] J. Jiang, V. Sekar, H. Zhang, Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive, Networking, IEEE/ACM Transactions on 22 (1) (2014) 326–340.

[10] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, B. Sinopoli, Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction, in: Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, ACM, New York, NY, USA, 2016, pp. 272–285.

[11] K. Ivesic, L. Skorin-Kapov, M. Matijasevic, Cross-layer qoe-driven admission control and resource allocation for adaptive multimedia services in lte, Journal of Network and Computer Applications 46 (2014) 336 – 351.

[12] A. Ganjam, J. Jiang, X. Liu, V. Sekar, F. Siddiqi, I. Stoica, J. Zhan, H. Zhang, C3: Internet-scale control plane for video quality optimization, in: Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15, USENIX Association, Berkeley, CA, USA, 2015, pp. 131–144.

[13] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, H. Zhang, Practical, real-time centralized control for cdn-based live video delivery, in: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15, ACM, New York, NY, USA, 2015, pp. 311–324.

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

[15] T. Uzakgider, C. Cetinkaya, M. Sayit, Learning-based approach for layered adaptive video streaming over sdn, Computer Networks 92, Part 2 (2015) 357 – 368.

[16] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, A. Pescapè, Measuring home broadband performance, Communications of the ACM 55 (11) (2012) 100–109.

[17] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, 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, FhMN '13, ACM, New York, NY, USA, 2013, pp. 15–20.

[18] G. Cofano, L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, S. Mascolo, Design and experimental evaluation of network-assisted strategies for http adaptive streaming, in: Proceedings of the 7th International Conference on Multimedia Systems, MMSys '16, ACM, New York, NY, USA, 2016, pp. 3:1–3:12.

[19] T. Wu, R. Huysegems, T. Bostoen, Scalable network-based video-freeze detection for http adaptive streaming, in: 2015 IEEE 23rd International Symposium on Quality of Service

(IWQoS), 2015, pp. 95–104.

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

[21] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, F. De Turck, Qoe-driven rate adaptation heuristic for fair adaptive video streaming, ACM Trans. Multimedia Comput. Commun. Appl. 12 (2) (2015) 28:1–28:24.

[22] S. Akhtar, A. Francini, D. Robinson, R. Sharpe, Interaction of aqm schemes and adaptive streaming with internet traffic on access networks, in: Global Communications Conference (GLOBECOM), 2014 IEEE, 2014, pp. 1145–1151.

[23] Sandvine, Exposing the technical and commercial factors underlying internet quality of experience, `https://www.sandvine.com/trends/global-internet-phenomena/`.

[24] T. J. Hastie, R. J. Tibshirani, J. H. Friedman, The elements of statistical learning : data mining, inference, and prediction, Springer series in statistics, Springer, New York, 2009.

[25] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. C. Chen, L. Hanzo, Machine learning paradigms for next-generation wireless networks, IEEE Wireless Communications PP (99) (2016) 2–9.