

 Open access • Journal Article • DOI:10.1145/2818361

QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming

— [Source link](#) 

Stefano Petrangeli, Jeroen Famaey, Maxim Claeys, Steven Latre ...+1 more authors

Institutions: Ghent University, University of Antwerp

Published on: 20 Oct 2015 - ACM Transactions on Multimedia Computing, Communications, and Applications (ACM)

Topics: Heuristics, Quality of experience, De facto standard and Shared resource

Related papers:

- [A Survey on Quality of Experience of HTTP Adaptive Streaming](#)
- [Towards network-wide QoE fairness using openflow-assisted adaptive video streaming](#)
- [A buffer-based approach to rate adaptation: evidence from a large video streaming service](#)
- [Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale](#)
- [What happens when HTTP adaptive streaming players compete for bandwidth](#)

Share this paper:    

View more about this paper here: <https://typeset.io/papers/qoe-driven-rate-adaptation-heuristic-for-fair-adaptive-video-139igdhevz>

QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming

STEFANO PETRANGELI, Department of Information Technology (INTEC), Ghent University–iMinds
 JEROEN FAMAHEY, Department of Mathematics and Computer Science, University of Antwerp–iMinds
 MAXIM CLAEYS, Department of Information Technology (INTEC), Ghent University–iMinds
 STEVEN LATRÉ, Department of Mathematics and Computer Science, University of Antwerp–iMinds
 FILIP DE TURCK, Department of Information Technology (INTEC), Ghent University–iMinds

HTTP Adaptive Streaming (HAS) is quickly becoming the de facto standard for video streaming services. In HAS, each video is temporally segmented and stored in different quality levels. Rate adaptation heuristics, deployed at the video player, allow the most appropriate level to be dynamically requested, based on the current network conditions. It has been shown that today's heuristics underperform when multiple clients consume video at the same time, due to fairness issues among clients. Concretely, this means that different clients negatively influence each other as they compete for shared network resources. In this article, we propose a novel rate adaptation algorithm called FINEAS (Fair In-Network Enhanced Adaptive Streaming), capable of increasing clients' Quality of Experience (QoE) and achieving fairness in a multiclient setting. A key element of this approach is an in-network system of coordination proxies in charge of facilitating fair resource sharing among clients. The strength of this approach is threefold. First, fairness is achieved without explicit communication among clients and thus no significant overhead is introduced into the network. Second, the system of coordination proxies is transparent to the clients, that is, the clients do not need to be aware of its presence. Third, the HAS principle is maintained, as the in-network components only provide the clients with new information and suggestions, while the rate adaptation decision remains the sole responsibility of the clients themselves. We evaluate this novel approach through simulations, under highly variable bandwidth conditions and in several multiclient scenarios. We show how the proposed approach can improve fairness up to 80% compared to state-of-the-art HAS heuristics in a scenario with three networks, each containing 30 clients streaming video at the same time.

Categories and Subject Descriptors: C.2.3 [Computer-Communication Networks]: Network Operations—Network management; C.2.5 [Computer-Communication Networks]: Local and Wide-Area Networks—Internet

General Terms: Algorithms, Design, Management, Performance

Additional Key Words and Phrases: Experimental evaluation, fairness, HTTP Adaptive Streaming, Microsoft Smooth Streaming, mobile, quality of experience, rate adaptation

The research was performed partially within the iMinds V-FORCE project (under IWT grant agreement no. 130655). This work was partly funded by FLAMINGO, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme. Maxim Claeys is funded by a grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

Authors' addresses: S. Petrangeli, M. Claeys, and F. De Turck, Gaston Crommenlaan 8, box 201, B-9050 Gent, Belgium; emails: {stefano.petrangeli, maxim.claeys, filip.deturck}@intec.ugent.be; J. Famaey and S. Latré, Middelheimlaan 1, B-2020 Antwerp, Belgium; emails: {jeroen.famaey, steven.latre}@uantwerpen.be. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2015 ACM 1551-6857/2015/10-ART28 \$15.00

DOI: <http://dx.doi.org/10.1145/2818361>

ACM Reference Format:

Stefano Petrangeli, Jeroen Famaey, Maxim Claeys, Steven Latré, and Filip De Turck. 2015. QoE-driven rate adaptation heuristic for fair adaptive video streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* 12, 2, Article 28 (October 2015), 24 pages. DOI: <http://dx.doi.org/10.1145/2818361>

1. INTRODUCTION

Nowadays, video streaming applications are responsible for the largest portion of the traffic exchanged over the Internet. Particularly, HTTP Adaptive Streaming (HAS) protocols have become very popular due to their flexibility, and can therefore be considered as the de facto standard for video streaming services. Microsoft's Smooth Streaming, Apple's HTTP Live Streaming, Adobe's HTTP Dynamic Streaming, and MPEG Dynamic Adaptive Streaming over HTTP (DASH) are examples of available HAS technologies. In an HAS architecture, video content is stored on a server as segments of fixed duration at different quality levels. Each client can request the segment at the most appropriate quality level on the basis of the local perceived bandwidth. In this way, video playback dynamically changes according to the available resources, resulting in a smoother video streaming experience. The main disadvantage of current HAS solutions is that the heuristics used by clients to select the appropriate quality level underperform in a multiclient scenario [Akhshabi et al. 2012a; Li et al. 2014; Petrangeli et al. 2014]. In a real scenario, multiple clients simultaneously request content from the HAS server. Often, clients have to share a single medium, and issues concerning fairness among them appear, meaning that the presence of a client has a negative impact on the performance of others. As reported by Akhshabi et al. [2012a], fairness issues are not due to TCP dynamics but mainly arise from the rate adaptation algorithms, as they decide on the actual rate to download. When multiple clients stream a video at the same time, a wrong bandwidth estimation can occur, due to the temporal overlap of the activity-inactivity periods of different clients. This wrong estimation subsequently affects the bit rate selection and thus the clients' Quality of Experience (QoE). This problem is aggravated by the uncoordinated nature of current HAS heuristics. This entails that they are not aware of the presence of other clients nor can they adapt their behaviour to deal with it.

In this article, we investigate the aforementioned problems arising in a multiclient setting. Particularly, we present a fair HAS client able to achieve smooth video playback while coordinating with other clients in order to improve the fairness of the entire system. This goal is reached with the aid of an in-network-based system of *coordination proxies*, in charge of collecting measurements on the network conditions. This information is then used by the clients to refine their quality decision process and develop a fair behavior.

The main contributions of this article are threefold. First, we present a new HAS heuristic called FINEAS (Fair In-Network Enhanced Adaptive Streaming) able to select the best quality depending on network conditions in order to provide a smooth video streaming and improve fairness. Particularly, our heuristic is able to increase the average requested quality level compared to current HAS heuristics and avoid video freezes, while guaranteeing similar QoE to all the clients streaming video, that is, fairness. Second, we design an in-network-based system to help clients coordinate their behavior, which does not require explicit client-to-client communication or a centralized decision process. Consequently, the quality level selection can still be performed locally and independently by each client, without any modification to the general HAS principle. Third, detailed simulation results are presented to characterize the gain of the proposed framework compared to state-of-the-art HAS heuristics.

The remainder of this article is structured as follows. Section 2 reports related work on HAS and multiclient algorithms. Section 3 details the proposed multiclient HAS framework, from both an architectural and algorithmic point of view. In Section 4, we evaluate our solution through simulation and show its effectiveness compared to current HAS heuristics. Section 5 presents the main conclusions. The Online Appendix presents an analytical formulation of the general fairness problem.

2. RELATED WORK

Akhshabi et al. [2012b] present an analysis of the performance and drawbacks of some commercially available HAS heuristics, such as Microsoft's Smooth Streaming, Netflix, and Adobe players. It is shown that current heuristics perform quality selection sub-optimally. Particularly, they fail to adapt to rapid bandwidth changes, leading to drops in the client play-out buffer and unnecessary quality switches. They also analyze the performance of two competing HAS clients sharing the same bottleneck and point out that they are not able to develop a fair behavior. Similar considerations are reported by Müller et al. [2012a] when testing different HAS implementations using real bandwidth traces collected on a mobile network. They also point out that the Microsoft's Smooth Streaming client is able to achieve the highest average bit rate as well as a low number of quality switches.

Many HAS rate adaptation heuristics have been proposed to alleviate the problems highlighted in the previous paragraph. Zhou et al. [2012] present a control theory-based HAS client where the buffer filling level of the client is controlled. A similar approach is also studied by Tian and Liu [2012]. Adzic et al. [2011] propose to add additional information into the video segments to enhance the quality decision algorithm. The client can then decide to switch up or down depending on the effect on bit rate and the intrinsic quality of the next segment to download. Jarnikov and Ozcelebi [2010] study an algorithm based on a Markov Decision Process (MDP), which requires offline training. Xiang et al. [2012] use a similar approach for Scalable Video Coding streaming and adopt a Markov model to describe the variations of wireless channel conditions. Also, the work presented by Claeys et al. [2014] is based on an MDP and Reinforcement Learning theory. In this case, the solution to the MDP is computed online by means of the Q-Learning algorithm, without any a priori knowledge. All these algorithms share a common drawback. Although effective in a single-client scenario, they are not explicitly designed to deal with the presence of multiple clients streaming video simultaneously over a shared network medium. This means that they can fail in these circumstances, as shown by Akhshabi et al. [2012a]. They report that unfairness is mainly caused by the quality adaptation algorithms themselves, since they are not designed to explicitly cope with a multiclient scenario. It is shown that a relevant cause of unfairness is the temporal overlap of the activity-inactivity periods of different clients, since this can lead to wrong bandwidth estimations.

Based on this consideration, most of the algorithms designed to improve fairness in HAS focus on the modification of the time interval at which clients request a new segment. Villa et al. [2012] implement a modified version of the Microsoft's Smooth Streaming (MSS) client, randomizing the quality selection decision interval. The results show that this technique can improve fairness, but the randomization characteristics have to be selected carefully. A similar approach is also used by Jiang et al. [2014]. In this case, the next segment download is randomized taking into account the buffer status in order to avoid freezes. Moreover, a stateful bit rate selection is introduced to allow clients with a low quality to increase it more aggressively. The authors theoretically prove that this selection allows convergence of the clients' bit rates. In the work by Li et al. [2014], 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. The segment fetch time is used by Liu et al. [2012] to decide which quality level

to request. They try to improve fairness by allowing recently joined clients to behave more aggressively. De Cicco et al. [2013] propose the ELASTIC client, which avoids activity-inactivity periods by downloading segments in order to maintain the buffer level close to a set-point. The ELASTIC client is mainly designed to obtain fairness from the network point of view but not to maximize QoE, since quality switches are not included in the rate adaptation heuristic. This results in a high number of switches, even in a fixed bandwidth scenario, and consequently low QoE. The main problem of the presented algorithms is that they are purely client based, that is, no coordination is envisaged among the clients. Although this aspect simplifies the design and implementation of the algorithms, it does not allow solving the fairness problem completely. The lack of coordination mechanisms entails that the fairness problem has to be solved at design time and that the solution has to be encoded in the client's heuristic. Since fairness is inherently an online problem, which depends on the number of clients, network topology, and bandwidth conditions, the obtained solution is suboptimal. As an example, Müller et al. [2012b] pointed out that the network infrastructure, as a system of caching proxies, has a negative impact on the bandwidth estimation process of the clients, and thus also on their performance in terms of QoE and fairness. This type of information, which is not known at design time and thus not available to purely client-based algorithms, affects the effectiveness of the aforementioned solutions.

A first approach to tackle this problem would be a centralized solution, where the video server decides on the quality level to return. This solution is investigated by De Cicco et al. [2011] and Kuschnig et al. [2010]. De Cicco et al. [2011] propose a control theory-based algorithm to control the length of the sender buffer, placed at the server, in order to select the most appropriate quality level. In the work presented by Kuschnig et al. [2010], the server estimates the bandwidth of the TCP connection considering the number of bytes transmitted in a time unit. This estimation is performed on a per-GOP (Group of Pictures) basis. Even if a centralized approach would facilitate the computation of an optimal global fair policy, it is not scalable if the number of clients to serve increases. Moreover, a centralized approach alters the classical HAS principle.

In order to solve the scalability issue, we adopt in this article an in-network approach. In in-network algorithms, intermediary nodes are placed in the network to collect information regarding the available bandwidth and influence the behavior of HAS clients. An example of in-network adaptation is given by Bouten et al. [2012]. A system of network proxies periodically solves an optimization problem to determine the maximum quality the clients can download, based on the current network status and an objective function. Houdaille and Gouache [2012] introduce a bandwidth manager inside the home gateway to manage the flows belonging to the different clients. Based on clients' characteristics and network conditions, the bandwidth manager determines the target bit rate for each client in order to fairly share the available bandwidth. The bandwidth manager then applies traffic shaping techniques to limit the bandwidth of each client and drive it to request the target bit rate. A similar approach is investigated by Georgopoulos et al. [2013]. A centralized OpenFlow controller is used to allocate the bandwidth for each streaming device in order to obtain fairness from a QoE point of view. The authors present a model to correlate video bit rate with video quality, which is used by the controller in the bandwidth allocation process. Also, Ma and Bartos [2011] implement a bandwidth manager to determine the quality levels to assign to the clients, based on the available bandwidth and their subscription level. Mok et al. [2012] introduce a measurement proxy between the clients and the video streaming server, which estimates the available bandwidth and decides the highest possible quality level the clients can support. Based on this information and their buffer status, the clients decide the most appropriate quality level to request. El Essaili et al. [2013] propose a QoE optimizer for wireless networks that computes the optimal rate for the streaming clients, based on the wireless channel conditions. This value is then used

by a QoE proxy, in charge of intercepting and rewriting clients' requests to match the requested quality level with the optimal rate. 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. Even if different clients will select quality differently, at steady state all clients will converge to the quality level enforced by the network. Moreover, these solutions present robustness issues in case of fault or malfunctioning of the in-network elements.

Based on these considerations, we developed a new client-based rate adaptation heuristic to optimize the QoE delivered to the clients, together with an in-network coordination mechanism to improve fairness, which can operate in face of multiple bottleneck links. This approach presents two advantages. First, the in-network computation can be kept very simple and consequently not be computationally demanding, since the quality decision algorithm still runs at the client. Second, it is more robust in case of fault or malfunctioning of the network equipment, as the client can continue to operate (at a suboptimal level) without the in-network system. The concept of an in-network system to help clients obtain fairness was first introduced in our previous work [Petrangeli et al. 2014]. The difference with respect to the work presented in this article is twofold. First, the in-network system presented by Petrangeli et al. [2014] is composed of a single centralized node, which collects and aggregates statistics on clients' QoE performance. Even though the presented approach showed promising results, further experiments have shown unsatisfactory performance in complex scenarios, mainly due to (1) the centralized nature of the solution, which is not able to handle the presence of multiple bottlenecks, and (2) the aggregation of QoE statistics, which is not trivial if clients are located in different subnetworks. In this work instead, we propose a distributed in-network system, able to cope with the presence of multiple bottlenecks, computing an estimate of the clients' fair bandwidth share on bottleneck links. This information is provided to the clients, which use it in their heuristics to achieve fairness. Second, we design a completely new rate adaptation heuristic, called FINEAS, to maximize clients' QoE and exploit the information provided by the in-network system to achieve fairness.

3. PROPOSED FAIR HAS FRAMEWORK

The problem we propose to investigate in this article is twofold. First, clients have to obtain the highest possible video quality. Second, they have to show similar performance if they share bottleneck links, that is, fairness. Based on this consideration, all the clients sharing the same bottlenecks should act fairly, even if they belong to different networks. An analytical formulation of the general fairness problem is presented in the Online Appendix. In order to maximize the QoE delivered to the clients and achieve fairness, we present the FINEAS heuristic. The FINEAS heuristic runs at the clients and performs the quality level selection based on three inputs: the local perceived bandwidth, the video player buffer status, and the so-called *fairness signal*. The fairness signal is an additional measure introduced to achieve fairness, obtained when the client downloads a segment. The fairness signal is computed by a system of network nodes, called *coordination proxies*, and represents an estimate of the fair bandwidth share of all the clients streaming video. In the remainder of this section, we provide an architectural overview of the proposed framework (Section 3.1) and detail the implementation of the FINEAS heuristic (Section 3.2) and of the fairness signal computation (Section 3.3).

3.1. Architectural Overview

As introduced previously, the system of coordination proxies is in charge of helping clients achieve fairness, by computing an estimate of the fair bandwidth share for

all the clients streaming video, even if they belong to different networks. In order to maintain scalability, the computation of the fairness signal is performed periodically and in a hierarchical way by the coordination proxies. A generic coordination proxy P receives an estimate of the fairness signal from its parent node and computes a new estimate of the fairness signal for each of its child proxies. This estimate is computed by monitoring the available bandwidth for HAS traffic on the links connecting proxy P to its child nodes. A root proxy triggers the computation of the fairness signal. At initialization, the coordination proxies start monitoring the bandwidth for HAS traffic. After T_{fair} seconds, given T_{fair} is the fairness signal computation period, the root proxy forwards its first estimate of the signal and triggers its computation for the entire system. In order to limit overhead, the calculated fairness signal can be added as an HTTP header field and returned to the clients when delivering the next segment to play. Particularly, the clients translate the fairness signal into a *reference* quality level, representing the theoretical quality level to request in order to obtain perfect fairness among the clients. This reference gives an indication of the best quality level to achieve fairness, rather than determining the actual quality to be requested. The reason for this behavior is twofold. First, directly requesting the reference quality level would be optimal from the fairness point of view but not from the QoE point of view, because of the frequent switches that would occur. Second, directly requesting the reference quality level would alter the classical HAS principle, as the decision on the quality level to download would no longer be carried out by the clients.

The main advantage of this hybrid approach is threefold. First, no communication is needed among clients and consequently no significant overhead is introduced. Moreover, no client-to-proxy communication is required. The proxies are *transparent* to the clients, as the clients only need to know how to access the fairness signal but not how it is created. Second, the computation and delivery of the fairness signal do not negatively affect the behavior of existing clients. Third, the approach is robust toward proxy failure, as the clients can also operate without the fairness signal.

As far as the coordination proxies' positioning is concerned, the proxies should be located at the main aggregation points of the network in order to monitor the links where a bottleneck can occur. 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 coordination proxy can automatically become active). Since network operators have full control of their delivery infrastructure, they can easily identify which are the most sensible paths in their networks where a bottleneck could occur. This way, they can perform an initial placement of the coordination proxies on network nodes. Note that this assumption does not impact the flexibility of our solution, since in a real scenario the network architecture is given and does not significantly change over time. Furthermore, as coordination proxy functionalities can be implemented via software, proxies can be flexibly relocated in case network conditions consistently change over time. Moreover, coordination proxies can be placed liberally on network nodes, without negatively impacting the fairness signal computation even if a bottleneck does not occur. In this case, a proxy only receives the fairness signal from its parent node and forwards it to its child proxies, without performing any operation on it. In other words, if a bottleneck does not occur, the considered proxy becomes transparent with respect to the computation of the fairness signal. As an example of coordination proxies' placement, Figure 1 depicts a schematic representation of a communication network architecture based on the Evolved Packet Core (EPC), as described by Release 11 of the 3rd Generation Partnership Project (3GPP) [3GPP 2013]. The EPC represents a unified core network for the convergence of both 3GPP (as 2G, 3G, and 4G) and non-3GPP (as WLAN or DSL) access networks, but it is worth noting that our approach can be applied to other network topologies as well. Depending

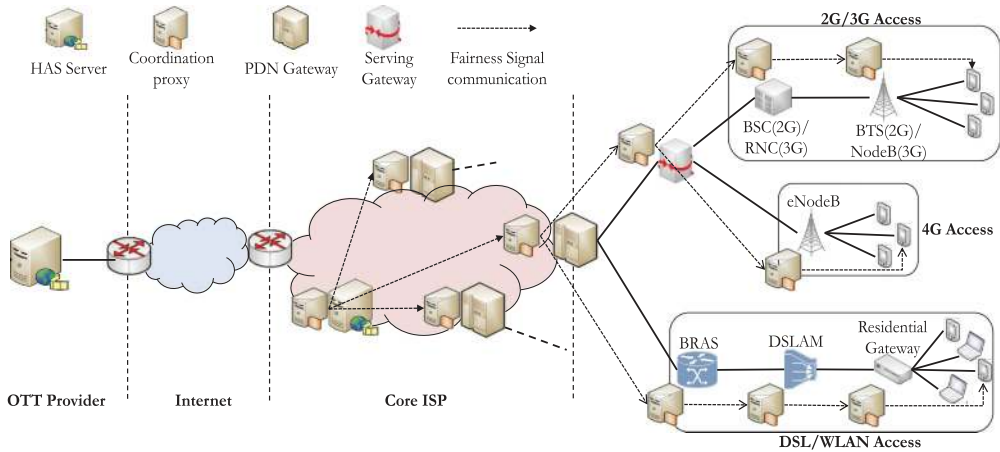


Fig. 1. Schematic representation of a communication network. The possible locations of the coordination proxies are shown. If the streaming service is offered by an Over-the-Top (OTT) provider, the proxy located at the HAS server in the core ISP network has to be moved to the router connecting the core ISP network with the Internet.

on whether the streaming service is offered by the network operator or an Over-the-Top (OTT) provider, the HAS server can be located inside the core ISP network or in a data center connected to the core ISP network through the Internet, respectively. In the latter case, a coordination proxy is located at the edge between the Internet and the core ISP network. In 2G and 3G access networks, a base station and a base station controller manage the radio interface. In the 4G case, these functionalities are carried out by the same node. A DSLAM concentrates the traffic coming from WLAN and DSL access networks and forwards it to the Broadband Remote Access Server (BRAS). A Packet Data Network (PDN) Gateway aggregates the traffic belonging to 3GPP and non-3GPP access networks and directs it into the core ISP network. The aforementioned nodes can provide coordination proxy functionalities, since they aggregate traffic at different network levels and can detect the presence of congested links. It is worth noting that the proposed system can be extended to take into account the presence of caches. In this case, the proxies have to compute a separate fairness signal for each of the considered delivery nodes (e.g., a generic HAS server or a cache). Consequently, our approach can be combined with any existing HAS-optimized caching scheme.

An important aspect of our framework is how the bandwidth estimation process is carried out by the proxies. In this article, we use an implementation based on sFlow, which performs bandwidth estimation through an operation of packet sampling [Schmidt et al. 2013]. Proxies monitor the bandwidth for two groups of flows: HAS traffic and non-HAS traffic (i.e., the traffic generated by non-HAS clients is grouped into the same flow). When a packet is sampled, the packet header is extracted and analyzed in order to match it to a specific flow. A packet belongs to the HAS flow if the IP source address matches that of the HAS server and the TCP source port is equal to 80 (the standard port for HTTP connections). Otherwise, the packet is considered part of the non-HAS traffic flow.

3.2. Client-Side Rate Adaptation

The FINEAS heuristic is executed by the client when a new segment has been downloaded, and before the request of a new one. The goal of the client is to select the most appropriate quality level in order to maximize QoE and achieve fairness. A detailed

ALGORITHM 1: FINEAS client heuristic. The symbol * indicates the parameters belonging to the heuristic.

Data: *videoBitRate*, vector containing the bit rate of each quality level of the video
maxAvailableQuality, highest available quality level of the video
segmentDuration, duration of the video segments (in seconds)
bufferSize, size of the video player buffer, expressing the maximum amount of video that can be stored (in seconds)
*qualityWindow**, time window stating how many quality level samples should be kept in memory (in seconds)
*bufferMin**, panic threshold of the buffer level (in seconds)
*bufferPercentage**, target percentage of the buffer size
 α^* , global utility function weight
Result: *nextQL*, the next quality level to request

```

1  bandwidth ← getLocalPerceivedBandwidth();
2  bufferFillingLevel ← getCurrentBufferFillingLevel();
3  bufferTarget = bufferSize × bufferPercentage;
4  qualityHistoryVector ← updateQualityHistoryVector(currentTime, qualityWindow);
5  averageQuality ← computeAverageQuality(qualityHistoryVector);
6  fairnessSignal ← getFairnessSignal();

7  fairnessQuality = 1;
8  for ql ← 1 to maxAvailableQuality - 1 do
9    if videoBitRate(ql) ≤ fairnessSignal < videoBitRate(ql + 1) then
10     | fairnessQuality =  $\frac{\text{fairnessSignal} + \text{ql} \times \text{videoBitRate}(\text{ql} + 1) - (\text{ql} + 1) \times \text{videoBitRate}(\text{ql})}{\text{videoBitRate}(\text{ql} + 1) - \text{videoBitRate}(\text{ql})}$ ;
11    end
12  end
13  if fairnessSignal ≥ videoBitRate(maxAvailableQuality) then
14    | fairnessQuality = maxAvailableQuality;
15  end
16  if bufferFillingLevel ≤ bufferMin then
17    | nextQL = 1;
18  else
19    maxDownloadableQuality = maxAvailableQuality;
20    for ql ← 1 to maxDownloadableQuality do
21      | estimatedDownloadTime = (videoBitRate(ql) × segmentDuration) / bandwidth;
22      | estimatedBufferLevel =
23      |   bufferFillingLevel - estimatedDownloadTime + segmentDuration;
24      |   if estimatedBufferLevel ≤ bufferMin then
25      |     | maxDownloadableQuality = ql - 1;
26      |     | break;
27      |   end
28    end
29    globalUtilityMax = -1000;
30    for ql ← 1 to maxDownloadableQuality do
31      | estimatedDownloadTime = (videoBitRate(ql) × segmentDuration) / bandwidth;
32      | estimatedBufferLevel =
33      |   bufferFillingLevel - estimatedDownloadTime + segmentDuration;
34      |   qoeUtility = - | ql - maxDownloadableQuality | - | ql - averageQuality |
35      |   - | estimatedBufferLevel - bufferTarget |;
36      |   fairnessUtility = - | ql - fairnessQuality |;
37      |   globalUtility = (1 - α) × fairnessUtility + α × qoeUtility;
38      |   if globalUtility ≥ globalUtilityMax then
39      |     | globalUtilityMax = globalUtility;
40      |     | nextQL = ql;
41    end
42  end
43  return nextQL;

```

description of the heuristic is provided in Algorithm 1. We indicate all the parameters belonging to our heuristic with the symbol *. In Section 4, an extensive evaluation of the impact of these parameters on the algorithm's performance is reported. The FINEAS heuristic can be subdivided into four main parts, which we describe in the following.

Input collection (lines 1–15). First, the client obtains the local perceived bandwidth, the video player buffer filling level *bufferFillingLevel*, and the buffer target *bufferTarget* (lines 1–3). The perceived bandwidth is the bandwidth measured during the download of a segment and is computed as the ratio between the segment size and the segment download time. *bufferFillingLevel* is the amount of video currently stored in the video player buffer, expressed in seconds. It depends on the number of segments currently stored in the buffer and their duration. *bufferTarget* represents the target for the buffer filling level, expressed in seconds. The client requests the next segment in order to maintain the resulting buffer filling level close to *bufferTarget*. This value is set by the parameter *bufferPercentage*, which is expressed as a percentage of the total buffer size. As an example, given *bufferSize* and *bufferPercentage* are equal to 10 seconds and 80%, respectively, the desired buffer target would be equal to $10 \times 0.8 = 8$ seconds. Next, the client updates the vector *qualityHistoryVector*, which contains the quality levels requested in the last *qualityWindow* seconds, and computes the average requested quality over this time window (lines 4–5). The fairness signal is then extracted from the HTTP header of the downloaded segment (line 6). The fairness signal represents the fair bandwidth share per client as computed by the coordination proxies. A client should request the quality level whose bit rate is closest to the fairness signal in order to fairly share resources with the other clients. Consequently, a reference quality level *fairnessQuality* is obtained starting from the *fairnessSignal* and the bit rates of the video (lines 7–15). *fairnessQuality* is then used as the target quality level to request in order to obtain perfect fairness among clients. Note that this mapping produces a real value in the interval $[1, \text{maxAvailableQuality}]$, since in general there is not an exact match between the fairness signal and the bit rates of the video. The client identifies the nearest lower (*ql*) and nearest higher (*ql + 1*) bit rates compared to *fairnessSignal* (line 9). *fairnessQuality* is computed on line 10. This operation corresponds to finding the fitting line passing through the points $a = (ql, \text{videoBitRate}(ql))$ and $b = (ql + 1, \text{videoBitRate}(ql + 1))$, and using the obtained equation to find the coordinate of point $c = (\text{fairnessQuality}, \text{fairnessSignal})$, where the unknown variable is *fairnessQuality*. Particularly, by substituting points *a* and *b* into the equation $y = mx + k$ and solving it with respect to *m* and *k*, the fitting line can be found. *m* and *k* are equal to $\text{videoBitRate}(ql + 1) - \text{videoBitRate}(ql)$ and $(ql + 1) \times \text{videoBitRate}(ql) - ql \times \text{videoBitRate}(ql + 1)$, respectively. By substituting the coordinate of point *c* into the aforementioned equation and solving it with respect to *fairnessQuality*, the formula expressed on line 10 is obtained. If *fairnessSignal* is greater than the highest available bit rate, *fairnessQuality* is assigned to the highest available quality level (lines 13–14).

Buffer status control (lines 16–17). If the buffer level *bufferFillingLevel* is below the panic threshold *bufferMin*, the lowest quality level is directly requested. This way, the client tries to react in a timely manner to a situation where the risk of video freezes is high.

Highest downloadable quality level identification (lines 19–27). Otherwise, the heuristic identifies the highest quality level *maxDownloadableQuality* that can actually be downloaded. For each of the available quality levels, the client computes an estimate of the download time (line 21). We refer here to an estimate as the computation is performed using the local perceived bandwidth, which could change in the future. The download time is used to compute an estimate of the buffer level when the download of the next segment to request will be completed, called *estimatedBufferLevel* (line 22).

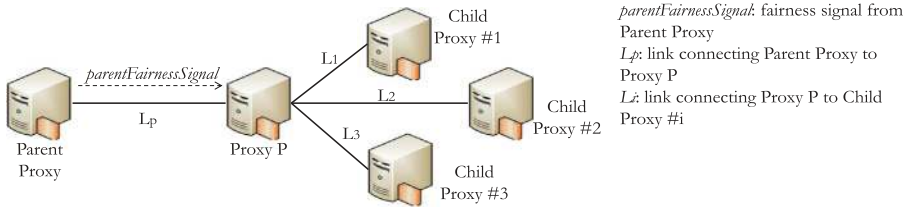


Fig. 2. Schematic representation of the coordination proxy architecture.

The client identifies the first quality level ql such that *estimatedBufferLevel* is lower than the panic threshold *bufferMin* (line 23). This means that if ql or a higher quality level is requested, the buffer level is likely to be below the panic threshold at the end of the segment download. In order to avoid a possible freeze or the request of the lowest quality level at the next step, the highest quality level that can be downloaded (i.e., *maxDownloadableQuality*) is then set to $ql-1$ (line 24).

Final quality level selection (lines 28–39). The client finally requests the quality level maximizing a global utility, given by the weighted sum of a QoE term and a fairness term. The QoE term (line 32) is an estimation of the actual QoE as experienced by the user, consisting of (1) segment quality, (2) quality switching, and (3) freezes. Segment quality is maximized by requesting the highest downloadable quality level (i.e., $-|ql-maxDownloadableQuality|$), switching is minimized by reducing the deviation from the average requested quality within the quality window (i.e., $-|ql-averageQuality|$), and freezes are minimized by keeping the buffer level as close to the buffer target level as possible (i.e., $-|estimatedBufferLevel-bufferTarget|$). The fairness term is maximized by requesting the quality level closest to the *fairnessQuality* (line 33). This way, the client is driven to request the best quality level to fairly share bandwidth with the other clients. For each quality from 1 to *maxDownloadableQuality*, the global utility is computed as the linear combination of the QoE and fairness terms (line 34). The selected quality level is the quality that maximizes this global utility (lines 35–38). A parameter α weighs the importance of the QoE and fairness components on the global utility. For $\alpha \rightarrow 1$, the heuristic tends to only maximize the QoE of the client, without taking into account fairness. For $\alpha \rightarrow 0$, only fairness is optimized. The influence of this parameter is investigated in Section 4.3.

The computational complexity of our heuristic can be estimated as $O(n)$, with n the number of quality levels, given the complexity of the loops on lines 8, 20, and 29 is $O(n)$. The algorithm has been implemented 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 to evaluate scalability. Even with 10,000 available quality levels, the quality selection can still be performed in less than 6 milliseconds.

3.3. Fairness Signal Computation

The hierarchical system of coordination proxies has the fundamental role to help the clients in achieving fairness by periodically computing the fairness signal. This value represents the theoretical fair bandwidth allocation of the clients belonging to a certain network and is computed taking into account all the clients streaming video (even those belonging to other networks). Algorithm 2 describes the operations performed by a generic coordination proxy P to compute its estimate of the fairness signal.

As depicted in Figure 2, proxy P is connected to a parent proxy and to a set of child proxies, each controlling a certain number of clients as indicated in the vector *controlledClients*. From the parent node, P receives the fairness signal *parentFairnessSignal* (line 1). This value represents the fair bandwidth share for each client controlled by

ALGORITHM 2: In-network computation performed at the coordination proxy P

Data: $childProxies$, vector containing the child coordination proxies of proxy P
controlledClients, vector containing the number of HAS clients controlled by each proxy in $childProxies$
Result: $fairnessSignal$, vector containing the fairness signals to forward to the proxies in $childProxies$

```

1  $parentFairnessSignal \leftarrow getParentFairnessSignal();$ 
2  $unusedBandwidth = 0, entitledClients = 0;$ 
3 for  $i$  in  $childProxies$  do
4    $estimatedBandwidth \leftarrow getEstimatedBandwidthForHASTrafficTowardsChildProxy(i);$ 
5    $maxFairnessSignal(i) = estimatedBandwidth/controlledClients(i);$ 
6   if  $maxFairnessSignal(i) \leq parentFairnessSignal$  then
7      $unusedBandwidth += (parentFairnessSignal - maxFairnessSignal(i)) \times$ 
        $controlledClients(i);$ 
8   else
9      $entitledClients += controlledClients(i);$ 
10  end
11 end
12  $childProxies \leftarrow sortIncreasing(childProxies, maxFairnessSignal);$  ##sort proxies in
    $childProxies$  in increasing order of associated  $maxFairnessSignal$ ##
13 for  $i$  in  $childProxies$  do
14   if  $maxFairnessSignal(i) \leq parentFairnessSignal$  then
15      $fairnessSignal(i) = maxFairnessSignal(i);$ 
16   else
17      $bandwidthPerClient = unusedBandwidth/entitledClients;$ 
18      $fairnessSignal(i) =$ 
19      $min(parentFairnessSignal + bandwidthPerClient, maxFairnessSignal(i));$ 
20      $unusedBandwidth -= (fairnessSignal(i) - parentFairnessSignal) \times$ 
        $controlledClients(i);$ 
21      $entitledClients -= controlledClients(i);$ 
22   end
23 return  $fairnessSignal;$ 

```

P , as computed by the parent node. Proxy P aims to fairly redistribute the bandwidth represented by $parentFairnessSignal$ to its child proxies. In case P is the root of the system, $parentFairnessSignal$ has to be assigned to infinite, as we assume there are no bottlenecks before the root proxy.

For each child proxy i , P obtains the estimated bandwidth for HAS traffic on the link connecting P to proxy i (link L_i in Figure 2). Next, it computes the fair bandwidth share on this link, that is, the ratio between the estimated bandwidth on L_i and the number of clients controlled by proxy i (line 5). This value, indicated as $maxFairnessSignal(i)$, represents the maximum fairness signal achievable for proxy i . Proxy P then checks whether L_i or L_p , the link connecting the parent proxy to P , is the actual bottleneck for proxy i . Particularly, if the fair bandwidth per client on link L_i (i.e., $maxFairnessSignal(i)$) is smaller than that on link L_p (i.e., $parentFairnessSignal$), the bottleneck is represented by link L_i . Otherwise, if $maxFairnessSignal(i) > parentFairnessSignal$, L_p acts as a bottleneck for proxy i . When L_i represents the bottleneck (line 6), the clients controlled by proxy i are not able to use all the bandwidth indicated by $parentFairnessSignal$. Particularly, the unused bandwidth is equal to the difference between $parentFairnessSignal$ and $maxFairnessSignal(i)$, multiplied by the number of clients controlled by proxy i (line 7). The leftover bandwidth $unusedBandwidth$ can be

redistributed to the clients whose bottleneck is represented by link L_p . When L_p is the bottleneck, the number of clients entitled to accept part of the exceeding bandwidth is incremented by the number of clients controlled by proxy i (line 9).

The final fairness signal for each child proxy is computed on lines 13–22. Here, proxy P redistributes the exceeding bandwidth to all the entitled clients. If L_i is the bottleneck for child proxy i (i.e., no extra bandwidth can be assigned), the final fairness signal is assigned to $maxFairnessSignal(i)$ (lines 14–15). Otherwise, the surplus bandwidth per client $bandwidthPerClient$ is computed as the ratio between the unused bandwidth and the number of entitled clients (line 17). Next, the final fairness signal for proxy i is assigned as the sum between the parent fairness signal and the surplus bandwidth per client (line 18). The min operation performed on line 18 avoids that $fairnessSignal(i)$ exceeds $maxFairnessSignal(i)$. The amount of unused bandwidth is then decremented by the amount assigned at the previous step (line 19). Also, the amount of entitled clients is decremented by the number of clients controlled by proxy i (line 20).

As an example, we consider the scenario in Figure 2, where proxy P is connected to three child proxies. We assume each child proxy controls 10 HAS clients (i.e., $controlledClients(i) = 10$ for $i = 1,2,3$), and the estimated bandwidth for HAS traffic on links L_1 , L_2 , and L_3 is 10, 20, and 35Mbps, respectively. $parentFairnessSignal$ is 2Mbps. In this case, the variable $maxFairnessSignal(i)$ for each proxy is equal to 1, 2, and 3.5Mbps, respectively. This entails that L_1 represents the main bottleneck for proxy #1, as $maxFairnessSignal(1) \leq parentFairnessSignal$. The unused bandwidth is computed as on line 7 and is equal to $unusedBandwidth = 10$ Mbps. This is not the case for proxy #3, as $maxFairnessSignal(3)$ is greater than $parentFairnessSignal$. Consequently, the number of entitled clients is incremented by 10 clients. The fairness signal for proxies #1 and #2 is assigned directly to 1 and 2Mbps, respectively, as both $maxFairnessSignal(1)$ and $maxFairnessSignal(2)$ are smaller or equal to $parentFairnessSignal$. On the contrary, proxy P can assign the exceeding bandwidth to the clients controlled by proxy #3. The variable $bandwidthPerClient$ is equal to 1Mbps and the final fairness signal is equal to $parentFairnessSignal + bandwidthPerClient = 3$ Mbps, as shown on line 18.

The rationale behind the sort operation performed on line 12 is to reduce the computational complexity of the heuristic. The complexity of the two *for* loops on lines 3 and 13 is $O(n)$, while that of the sort operation is $O(n \times \log(n))$, with n the length of vector $childProxies$. Consequently, the total computational complexity is $O(n \times \log(n))$, which is known to scale well even for large values of n . If the vector $childProxies$ is not sorted, the computation on lines 13–22 has to be performed using two nested *for* loops, for a total computational complexity of $O(n^2)$. In order to evaluate complexity, Algorithm 2 has been implemented on a Dell Latitude E5530 running Ubuntu 12.04 LTS 64-bit, Intel Core i5-3320M CPU @ 2.60GHz processor and 8GB of memory. We incremented the number of child coordination proxies from two to 10,000 and measured the time needed to compute the fairness signal. As expected, the computation time increases logarithmically with the number of coordination proxies. Even for 10,000 child proxies, the fairness signal computation is carried out in less than 0.032 seconds on a general-purpose device. Moreover, it is worth noting that the fairness signal computation is decoupled from the actual segment delivery to the clients. When a proxy receives a segment to forward, it just has to embed its current estimate of the fairness signal into an HTTP header. The actual update of the fairness signal can be considered as an independent and parallel process occurring periodically within the system of coordination proxies.

The fairness signal computation period, that is, the communication interval among the proxies, is an important parameter of our algorithm. A small period entails that the fairness signal closely follows bandwidth variations, at the price of an increased computational and communication overhead. On the contrary, a high period reduces

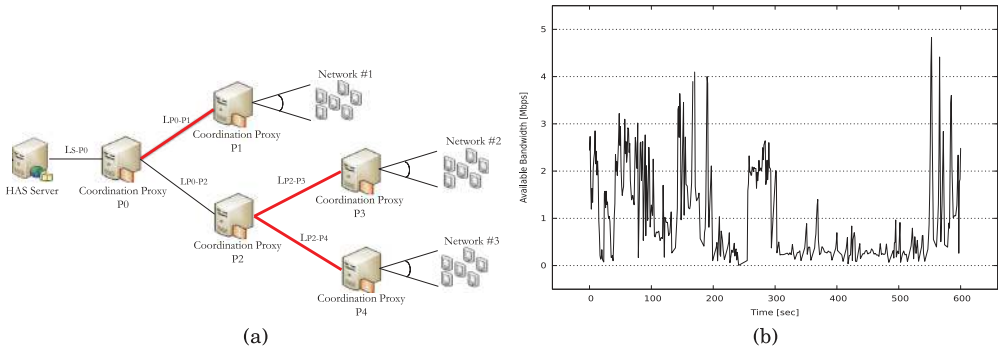


Fig. 3. Simulated topology (a) and an extract of the used bandwidth pattern (b).

overhead but negatively affects the significance of the fairness signal. The impact of this parameter on the performance of our solution is evaluated in Section 4.

4. PERFORMANCE EVALUATION

4.1. Experimental Setup

An NS-3-based simulation framework has been used to evaluate our multiclient framework [Bouten et al. 2012]. The video streamed is *Big Buck Bunny*, composed by 299 segments, each 2 seconds long and encoded at seven different quality levels: 300, 427, 608, 806, 1,233, 1,636, 2,436kbps (the lowest quality level is indicated with 1). The Network Simulation Cradle¹ has been enabled for all the results in order to provide a realistic implementation of the TCP protocol. Unless otherwise stated, the buffer size for each client is equal to five segments, or 10 seconds.

The simulated network topology is shown in Figure 3(a), where the position of the coordination proxies is reported. The root proxy is collocated with the HAS Server (not shown in Figure 3(a)). In order to give an extensive evaluation of the FINEAS heuristic, we simulate 50 episodes of the video trace and average the results over the 50 runs. The capacity on links L_{S-P0} and L_{P0-P2} is kept fixed for all the experiments. A cross-traffic generator introduces UDP traffic on links L_{P0-P1} , L_{P2-P3} , and L_{P2-P4} (highlighted links in Figure 3(a)) in order to vary the available bandwidth for HAS traffic. A different cross-traffic pattern is used for each episode, which varies each second and is scaled with respect to the number of clients. An episode is defined as a single simulation run. As far as the cross-traffic pattern is concerned, we use an open-source dataset collected on a real 3G/HSDPA network [Riiser et al. 2012]. The available bandwidth for one client fluctuates between 202bps and 6,335kbps, with an average of 2,087kbps and a standard deviation of 1,314kbps. An example of the bandwidth pattern is shown in Figure 3(b). Regarding the bandwidth estimation at the coordination proxies, the sFlow packet sampling rate has been set to one packet sampled every 1,000 for all the experiments. As far as the fairness signal is concerned, it is added as an HTTP header field by the proxies and returned to the clients when delivering the next segment.

In order to provide an extensive benchmark of the FINEAS algorithm, we compare our results to those obtained using four other HAS clients. Particularly, we choose a proprietary HAS client, the MSS² client, the Q-Learning-based client described by Claeys et al. [2014], and the client developed by Miller et al. [2012]. We also studied the performance of the FESTIVE algorithm, one of the first algorithms developed to

¹<http://research.wand.net.nz/software/nsc.php>.

²<https://slxextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming>.

Table I. Overview of Evaluated Parameter Configuration

<i>Parameter</i>	<i>Evaluated values</i>	<i>Parameter</i>	<i>Evaluated values</i>
qualityWindow [sec]	10, 20, 30, 50, 70	bufferMin [sec]	2, 4, 6
bufferPercentage	0.4, 0.6, 0.8, 1.0	α	0, 0.2, 0.4, 0.6, 0.8, 1.0
T_{fair} [sec]	2, 4, 6		

explicitly deal with a multiclient scenario [Jiang et al. 2014]. As far as the performance evaluation is concerned, fairness is computed as the standard deviation of clients' QoE. Unless otherwise stated, all the simulated clients start streaming the video at the same time.

4.2. QoE Model

In this section, we define the QoE model used to evaluate the proposed framework. We use a metric in the same range of the Mean Opinion Score (MOS), which can be computed as in Equation (1) [Claeys et al. 2014; De Vriendt et al. 2013]:

$$QoE_j^i = 5.67 \times \frac{\bar{q}_j^i}{q_{max_j^i}} - 6.72 \times \frac{\hat{q}_j^i}{q_{max_j^i}} + 0.17 - 4.95 \times F_j^i. \quad (1)$$

The QoE experienced by client j in network i is the linear combination of the average quality level requested \bar{q}_j^i , its standard deviation \hat{q}_j^i (both normalized with respect to the highest available quality level $q_{max_j^i}$), and F_j^i , which models the influence of freezes and is computed as follows:

$$F_j^i = \frac{7}{8} \times \max\left(\frac{\ln(\phi_j^i)}{6} + 1, 0\right) + \frac{1}{8} \times \left(\frac{\min(\psi_j^i, 15)}{15}\right). \quad (2)$$

ϕ_j^i and ψ_j^i are the freeze frequency and the average freeze duration, respectively. All the coefficients reported in Equations (1) and (2) have been fixed according to the works by De Vriendt et al. [2013] and Claeys et al. [2014].

4.3. Parameter Analysis

In this section, we investigate the impact of the parameters characterizing our HAS framework on clients' performance and select the configuration to use in the remainder of the article. As explained in Section 3.2 and Algorithm 1, the FINEAS heuristic parameters are *qualityWindow*, a time window stating how many quality levels have to be kept in memory; *bufferMin*, the panic buffer threshold; *bufferPercentage*, to tune the target buffer level; and α , the global utility function weight. As far as the in-network coordination is concerned (see Section 3.3), the only parameter to set is the fair signal computation period, named T_{fair} . In order to properly tune our HAS framework and select the best configuration, an elaborate evaluation of the parameter space has been performed. The evaluated values are reported in Table I for a total of 648 feasible configurations (note that the target buffer level needs always to be higher than the panic buffer threshold). We evaluate our solution with the setting reported in Section 4.1, each network containing 10 clients streaming video at the same time. The capacity of links L_{S-P0} and L_{P0-P2} is set to 60Mbps and 40Mbps, respectively, while the bandwidth on links L_{P0-P1} , L_{P2-P3} , and L_{P2-P4} is variable and has an average of 20Mbps and a standard deviation of 13Mbps over the 50 simulated episodes. This simulation setup allows us to have a clear understanding of the impact of the different parameters on the performance of our solution. Depending on the available bandwidth on links L_{P0-P1} , L_{P2-P3} , and L_{P2-P4} , the actual bottlenecks for the three networks dynamically change. This way, we can explore a wide range of network configurations, where the three

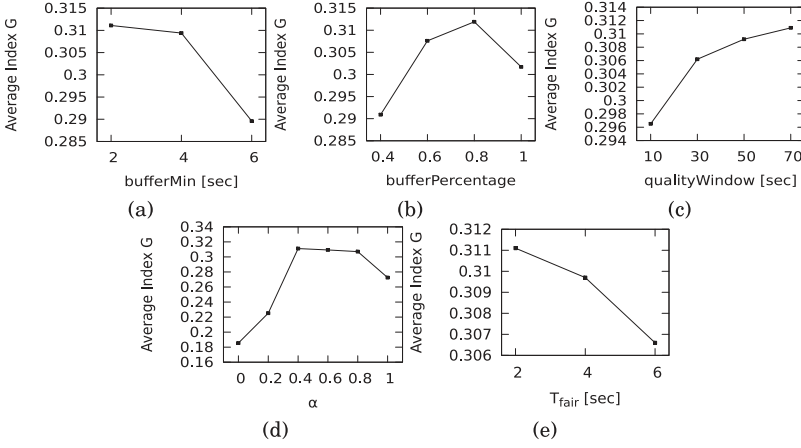


Fig. 4. Analysis of the parameter influence.

networks mutually influence each other (due to the bandwidth limitations on links L_{S-P0} and L_{P0-P2}) or not. Other topologies have been investigated, but the results do not significantly change and are omitted due to space constraints.

The performance evaluation is conducted on the basis of the achieved QoE and fairness. Particularly, we express fairness as the standard deviation of clients' QoE. We introduce a metric G_k to evaluate the overall performance of the analyzed configurations, defined as in Equation (3):

$$G_k = \frac{1}{N} \sum_{i=1}^N J_k^i - \sqrt{\frac{1}{N} \sum_{i=1}^N \left(J_k^i - \frac{1}{N} \sum_{i=1}^N J_k^i \right)^2} \quad \text{with} \quad J_k^i = \overline{QoE}_k^i - \widehat{QoE}_k^i. \quad (3)$$

N represents the number of networks containing HAS clients streaming video. J_k^i represents the performance of network i during the k th episode and is the linear combination of \overline{QoE}_k^i , the average QoE computed over the whole group of clients belonging to network i for episode k , and \widehat{QoE}_k^i , its standard deviation. This way, J_k^i encodes the fairness objective for network i , that is, maximizing QoE while improving fairness. G_k is used to evaluate the overall performance of the analyzed configuration and is given by the linear combination of the average J_k^i over all networks and of the negative of its standard deviation. Our aim is to select a configuration that allows one to maximize G_k in order to achieve the highest possible performance for every network while keeping the deviation among them as low as possible. For every configuration, the index G has been computed as the average of G_k over the 50 simulated episodes. For each evaluated value of each parameter, the average G of the five best configurations containing the evaluated value has been calculated. Using more than five values does not significantly affect the outcome of this analysis. The results are shown in Figure 4. Small values of the panic threshold $bufferMin$ lead to better performance, as they avoid often requesting the lowest quality level. The increased risk for video freezes caused by a low panic threshold is counterbalanced by using higher values of the buffer percentage $bufferPercentage$, which led to higher buffer targets. A tradeoff is present in this case: a high percentage reduces the risk of freezes but also the possibility to request a high quality level. The best performance is reached for a value of 0.8. As far as the $qualityWindow$ is concerned, it appears clearly that the more quality levels stored, the better. This

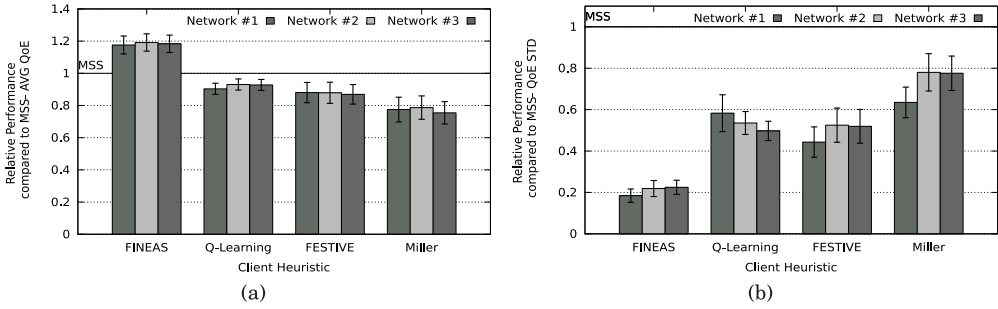


Fig. 5. Comparison between the different clients, from a QoE perspective, for a variable bandwidth scenario. Each network contains 30 clients streaming video. The graphs report the relative performance of the considered clients in terms of (a) average QoE and (b) its standard deviation compared to MSS. The standard deviation of clients' QoE is used as fairness metric.

entails that the client has a more comprehensive vision of the actions taken in the past and can choose the next quality level accordingly. The parameter α states to what extent the clients follow the indication given by the fairness signal. The best value is obtained at 0.4, meaning that the two opposite objectives of maximizing clients' QoE and achieving fairness are well balanced. Strictly following the fairness signal (i.e., values of α close to zero) leads to perfect fairness but bad performance in terms of QoE, because of the frequent switches that would occur. This result also highlights that it is more convenient to incorporate the fairness signal into the adaptation heuristic rather than using it directly to enforce the quality selection process. As expected, a small value of the fairness signal computation period T_{fair} leads to the best performance, as the in-network computation can quickly follow the bandwidth variations. The introduced communication overhead can be kept minimal by adding an HTTP header to transport the fairness signal.

Based on this analysis, the configuration with *qualityWindow* equal to 70 seconds, *bufferMin* to 2 seconds, *bufferPercentage* to 0.8, α to 0.4, and T_{fair} to 2 seconds has finally been chosen. It is worth noting that this configuration is also the best overall, thus showing the validity of the parameter analysis.

4.4. Variable Bandwidth Scenario

After the parameter analysis, we evaluate our solution with an increased number of clients. A fixed-line scenario has also been evaluated, but the results are omitted due to space limits. In a fixed-line scenario, all the simulated heuristics are able to guarantee a high QoE, but only the FINEAS heuristic is able to guarantee fairness.

The same simulation setting as in Section 4.1 is used, with 30 clients per network streaming a video at the same time. In this case, links L_{S-P0} and L_{P0-P2} are fixed to 180Mbps and 120Mbps, respectively, while the bandwidth on links L_{P0-P1} , L_{P2-P3} , and L_{P2-P4} is variable and presents an average of 60Mbps and a standard deviation of 40Mbps over the 50 simulated episodes. In this scenario, the three networks can mutually influence their performance, as the links L_{S-P0} and L_{P0-P2} act as common bottlenecks. It is fundamental to stress that the variable bandwidth pattern has been collected on a real 3G/HSDPA network, as described in Section 4.1. We consider the MSS as reference client and compute, for each simulated episode and for each network, the ratio between the average QoE of the analyzed client and that of the MSS. Figure 5(a) reports the average value over the 50 episodes of this ratio, together with the confidence intervals at 95%. Figure 5(b) reports the average value over the 50 episodes of the ratio between the QoE standard deviation of the MSS algorithm and that of

Table II. Performance Summary in the Variable Bandwidth Scenario, in Terms of Quality Components. The Average Value over the 50 Episodes Is Reported, Together with the Confidence Interval at 95%.

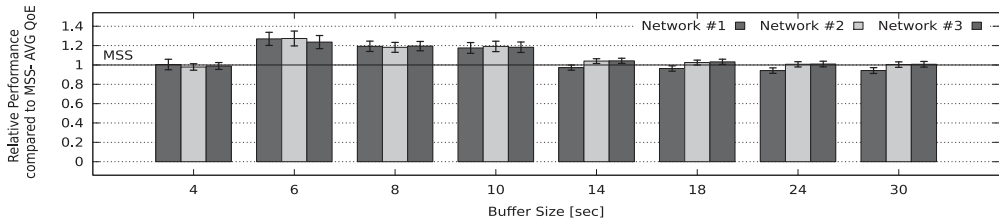
		<i>MSS</i>	<i>FINEAS</i>	<i>Q-Learning</i>	<i>FESTIVE</i>	<i>Miller</i>
Network 1	QoE	2.96±.18	3.41±.15	2.61±.07	2.64±.26	2.35±.32
	\bar{q}_j^i	4.51±.19	5.19±.13	3.69±.03	5.60±.14	5.70±.15
	\hat{q}_j^i	0.91±.08	0.98±.06	0.55±.03	1.23±.04	1.51±.09
	Bitrate [Mbps]	1.01±.08	1.30±.05	0.74±.01	1.47±.05	1.51±.06
	Freeze [sec]	0.08±.02	0.07±.02	0.07±.04	4.21±1.16	3.19±.59
	Freeze number	0.07±.01	0.05±.02	0.06±.03	3.42±.84	3.40±.49
Network 2	QoE	2.76±.15	3.25±.16	2.52±.08	2.44±.23	2.18±.24
	\bar{q}_j^i	4.21±.16	5.16±.15	3.68±.04	5.14±.11	5.24±.12
	\hat{q}_j^i	0.83±.07	1.12±.06	0.62±.04	1.14±.03	1.38±.06
	Bitrate [Mbps]	0.89±.06	1.29±.06	0.73±.01	1.28±.04	1.32±.05
	Freeze [sec]	0.08±.02	0.09±.03	0.10±.03	3.31±.87	2.66±.48
	Freeze number	0.08±.01	0.07±.02	0.11±.02	2.95±.71	2.99±.39
Network 3	QoE	2.78±.15	3.25±.16	2.52±.07	2.44±.22	2.11±.24
	\bar{q}_j^i	4.21±.16	5.16±.15	3.63±.04	5.14±.11	5.23±.12
	\hat{q}_j^i	0.81±.07	1.12±.06	0.57±.03	1.14±.03	1.40±.05
	Bitrate [Mbps]	0.89±.06	1.29±.06	0.73±.01	1.28±.04	1.31±.05
	Freeze [sec]	0.08±.03	0.09±.03	0.11±.03	3.35±.89	2.86±.51
	Freeze number	0.07±.01	0.07±.02	0.11±.02	2.91±.69	3.15±.39

the analyzed client, together with the confidence intervals at 95%. The QoE standard deviation is used as a fairness metric. Our solution is able to increase the average QoE by almost 20% for each of the three networks and to improve fairness with almost 80% when compared to MSS. Also, the FESTIVE and Miller clients improve fairness but consistently reduce the average QoE. This entails that the final QoE at the clients with these two heuristics is lower than that obtained using the MSS heuristic. These results show the suboptimality of these two heuristics in case of a variable bandwidth, caused by frequent quality switches and video freezes. As far as the Q-Learning client is concerned, it improves fairness by about 50% with respect to MSS, but with a loss of 8% to 10% in terms of average QoE. This negative behavior is mainly due to the mutual influence among the learning processes of the clients and the uncoordinated nature of Q-Learning [Claus and Boutilier 1998]. When a client selects a certain quality level, it uses a portion of the shared bandwidth. This decision has an impact on the performance of the other clients and thus also on their learning process. Since the clients do not share any information, this leads to a suboptimal quality adaptation policy.

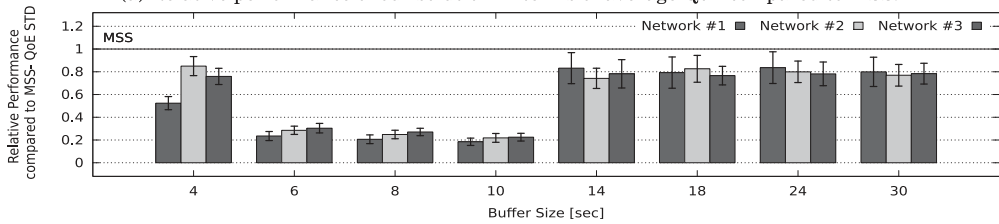
Table II summarizes the results for this scenario. We compute, for each episode and for each network, the average of clients' QoE, \bar{q}_j^i , \hat{q}_j^i , requested bitrate, freeze time, and freeze number. \bar{q}_j^i and \hat{q}_j^i represent, respectively, the average quality level requested by client j in network i and the standard deviation from this average (see Equation (1)). We then average these values over the 50 simulated episodes. The largest gain of the FINEAS heuristic is an increased average quality level (15% higher than MSS) and requested bit rate, without affecting the freeze duration. The FESTIVE and Miller clients reach the highest quality level, but the freeze duration, number of freezes, and quality switches increase. This means that the rate adaptation process is not performed optimally and, consequently, does not lead to the highest possible QoE. The main problem of the Q-Learning algorithm is the low average requested quality level, which strongly affects the final QoE. This behavior is a consequence of the mutual influence among the clients. When a client requests a high quality segment, it could experience a freeze due to the congestion caused by the other clients. This way, the clients develop a more conservative policy, where high quality levels are avoided to

Table III. Statistical Significance of the Average QoE and the Average Standard Deviation of the QoE, Using a Two-Tail Paired T-Testing with Significance Level 0.05

	<i>MSS</i>	<i>FINEAS</i>	<i>Q-Learning</i>	<i>FESTIVE</i>	<i>Miller</i>
Average QoE	2.78±.50a	3.24±.55b	2.50±.38c	2.46±.58c	2.17±.60d
Average QoE Standard Deviation	0.69±.16a	0.13±.04b	0.32±.05c	0.31±.08c	0.46±.09d



(a) Relative performance of our solution in terms of average QoE compared to MSS.



(b) Relative performance of our solution in terms of QoE standard deviation compared to MSS.

Fig. 6. Evaluation of the FINEAS heuristic, from a QoE perspective, for a variable bandwidth scenario with different buffer sizes. Each network contains 30 clients streaming video. The x-axis reports the buffer size, in seconds.

limit video freezes. This problem is further intensified by the nonstationarity of the simulated environment.

Table III reports the statistical significance of the results. For each algorithm and for each episode, we compute the average QoE and the average standard deviation over the three networks. Next, we perform a two-tail paired t-test for each pair of heuristics. Results are reported as the mean (and its standard deviation) of the average QoE and the average QoE standard deviation, over the 50 simulated episodes. Taking two heuristics, they are statistically different if they are associated to different letters. For example, the results for the MSS and FINEAS heuristics are statistically different. Only the Q-Learning and the FESTIVE clients do not show any significant difference in performance.

4.5. Influence of the Buffer Size

In this section, we investigate clients' performance using different buffer sizes. Based on the results of the previous section, we decided to evaluate our solution and the MSS one, which outperforms the Q-Learning, the FESTIVE, and the Miller client in terms of QoE (see Figure 5(a)). The same simulation setting of the previous section has been used, with different buffer sizes from 4 to 30 seconds. In the 4 seconds case, the panic threshold of the FINEAS heuristic has been disabled (i.e., *bufferMin* set to zero). The results of this investigation are depicted in Figure 6. As expected, when the buffer size increases, the differences between our solution and MSS decrease. Particularly, the MSS heuristic presents a big performance gain in terms of average QoE and its standard deviation when the buffer is larger than 14 seconds. Consequently, the gain brought by the FINEAS algorithm decreases. Nevertheless, the FINEAS heuristic is still able to outperform MSS by about 20% in terms of fairness. A large buffer simplifies

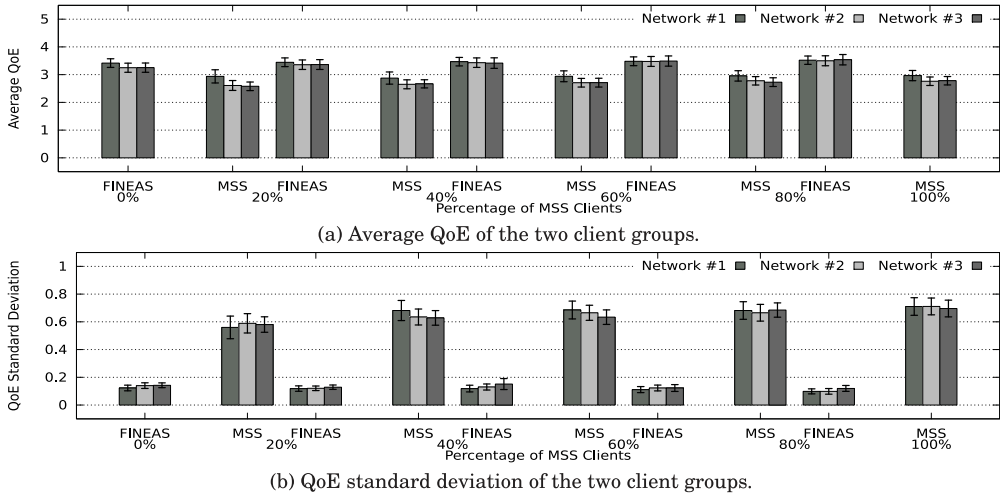


Fig. 7. Evaluation of the heterogeneous scenario, from a QoE perspective. Each network contains 30 clients streaming video. The x-axis reports the percentage of MSS clients on the total.

the rate adaptation process, since it reduces the risk of freezes and thus the dependency on the current available bandwidth. When the buffer is sufficiently filled, a client can request the best quality level to maximize its QoE even though the current available bandwidth would not allow it. Despite that, a large buffer should be avoided because it causes a long delay when streaming live contents and a big memory occupation on the device. Also, when the buffer is only a few seconds (4 seconds case in Figure 6), the performance of the FINEAS client is very close to that of MSS. Unlike the previous case, if the buffer size is very small, the client has to follow almost precisely the available bandwidth in order to minimize the risk of freezes. Consequently, the optimization possibilities are reduced. Nevertheless, our solution always outperforms MSS from the fairness point of view, while achieving a similar or higher average QoE.

4.6. Client Heterogeneity

So far, we have investigated a homogeneous scenario, where all the clients are equipped with the same heuristic. In a real scenario, different client types can stream video at the same time. We analyze here the interaction between two different groups of clients, one equipped with the FINEAS heuristic and the other equipped with the MSS one. The fairness signal is computed considering the totality of clients streaming video but is analyzed by our clients only. The simulation settings are those reported in Section 4.4. We increase the size of the MSS group from 20% to 80% of the total for a number of MSS clients ranging from 18 to 72; the assignment of the MSS clients to each network is made randomly at the beginning of each simulated episode. In order to evaluate the results, we compute for each simulated episode, for both client groups, the average QoE and its standard deviation. We then average these values over the 50 simulated episodes. Figure 7 reports the obtained results, together with the confidence intervals at 95%. The FINEAS group is always able to outperform the MSS one, with a gain in the average QoE between 20% and 30% for all the networks in all scenarios. Interestingly, the performance of the two groups remains constant even if the number of MSS clients changes. Thanks to the FINEAS heuristic and the in-network fairness signal computation, our clients are always able to select the best quality level to maximize their own QoE and optimize fairness. This optimization is carried out considering all the HAS clients, since the fairness signal is computed

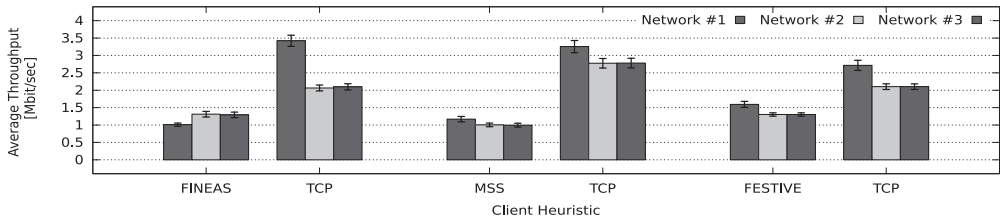


Fig. 8. Evaluation of a scenario with concurrent TCP clients. TCP clients are 25% of the total. The x-axis reports the different heuristics, the y-axis the average throughput for each network over the 50 simulated episodes.

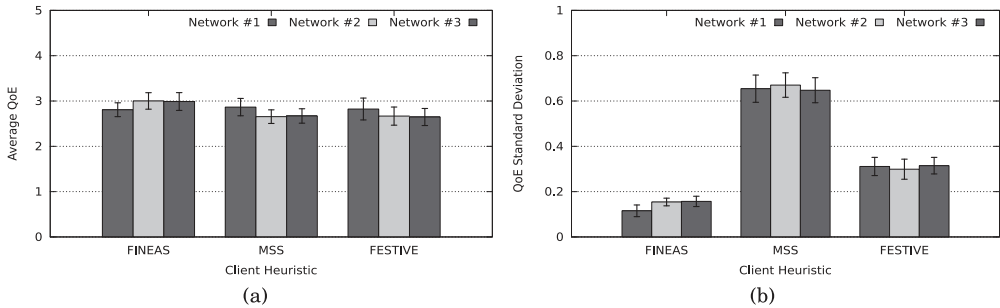


Fig. 9. Evaluation of a scenario with concurrent TCP clients, from a QoE perspective. TCP clients are 25% of the total. The x-axis reports the different heuristics, the y-axis the average QoE (a) and its standard deviation (b) for the HAS clients subgroup.

considering all the clients streaming video, and not only the ones equipped with our solution. This entails that our clients do not behave greedily with respect to the MSS group. The difference in performance is thus mainly due to the better rate adaptation of the FINEAS clients.

In a second set of experiments, we investigate the interaction between different HAS clients (FINEAS, FESTIVE, and MSS) and generic TCP clients. The simulation settings are those reported in Section 4.4. The number of TCP clients is equal to 23 (i.e., 25% of the total); the assignment of the TCP clients to each network is made randomly at the beginning of each simulated episode. Each TCP client establishes a connection with the HAS server, where a TCP sender is installed. Each client downloads the video content from the server in one continuous stream at the fixed rate of 5Mbps. Consequently, the TCP start-up behavior is negligible compared to the total video duration and does not affect the outcome of this analysis. During each experiment, the same HAS heuristic is used. As far as the fairness signal computation is concerned, it is performed considering the HAS clients only, since we are not in control of the traffic generated by other non-HAS applications (browsing, file downloading, etc.). In order to evaluate the results, we compute for each simulated episode, for both the HAS and TCP groups, the average TCP throughput. We then average this value over the 50 simulated episodes. We also compute the average QoE and its standard deviation over the 50 episodes for the HAS group. Figures 8 and 9 report the obtained results, with the confidence intervals at 95%. Based on the simulated topology and the bandwidth pattern (see Section 4.4), each client should obtain, on average, a bandwidth of 2Mbps. From Figure 8, it appears clearly that the TCP group behaves greedily with respect to the HAS group, independently from the adopted heuristic. TCP clients in Network #1, which are the closest to the HAS server and thus experience the lowest RTT, obtain the highest throughput. If we consider the results from a QoE point of view, it is possible to

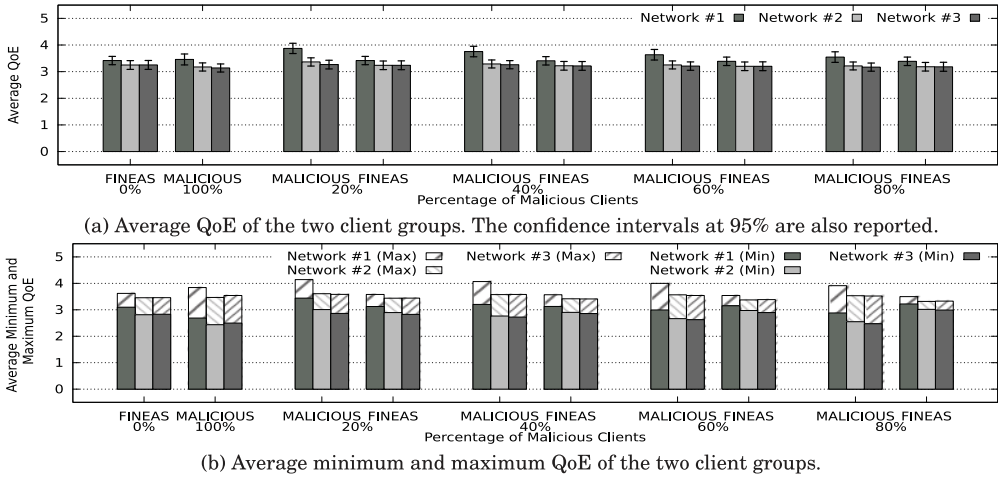


Fig. 10. Evaluation of the malicious scenario, from a QoE perspective. Each network contains 30 clients streaming video. The x-axis reports the percentage of malicious clients on the total.

draw two conclusions. First, the FINEAS heuristic is still able to reach the best results both from the QoE and the fairness perspective, as one can see in Figures 9(a) and 9(b), respectively. Particularly, the average QoE for Network #1 is the same as that of the MSS and the FESTIVE clients, while the QoE standard deviation is considerably lower in the FINEAS case. The gain achieved in terms of average QoE for Network #2 and #3 is about 13%. Second, exploited network resources being equal, the FINEAS client results in a better overall perceived video quality, that is, it is more efficient. As observed in Figure 8, the average TCP throughput reached by the FINEAS and FESTIVE clients for Networks #2 and #3 is almost equal and higher than that reached by MSS. Nevertheless, the QoE achieved by the FINEAS clients is considerably higher than that achieved by the two other heuristics, entailing a better utilization of network resources. The same consideration applies to Network #1. Even though the average throughput reached by the FINEAS clients is lower than MSS and FESTIVE, the average QoE is the same.

4.7. Malicious Clients and Proxy Failure

In this section, we analyze the robustness of our solution in the presence of malicious clients and in case of a coordination proxy failure.

In the first set of experiments, we simulate the presence of malicious clients equipped with a modified FINEAS heuristic that ignores the fairness signal. The only objective of a malicious client is to maximize its own QoE, neglecting the presence of other clients and fairness. The simulation settings are those reported in Section 4.4. The fairness signal computation is performed considering the totality of clients streaming video but is analyzed by the normal FINEAS clients only. We increase the size of the malicious group from 0% to 100% of the total for a number of malicious clients ranging from 0 to 90; the assignment of the malicious clients to each network is made randomly at the beginning of each simulated episode. In order to evaluate the results, we compute for each simulated episode, for both client groups, the average QoE, its standard deviation, and the minimum and the maximum QoE. We then average these values over the 50 simulated episodes. The results are reported in Figure 10. The first two histograms report the results for the two limit cases where the clients are all equipped with the FINEAS heuristic or are all malicious. As observed in Figure 10(a), the two groups of

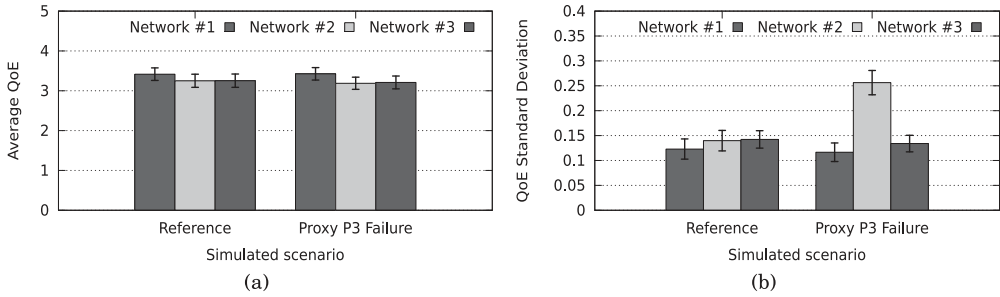


Fig. 11. Performance evaluation, from a QoE perspective, of a scenario where Coordination Proxy P3 (see Figure 3(a)) is defective. Each network contains 30 clients streaming video. The graphs report the average QoE (a) and its standard deviation (b).

clients obtain very similar average QoE. The FINEAS group obtains a QoE standard deviation half of that obtained by the malicious group (not shown in the article due to space limitations), as the former follows the fairness signal. More interestingly, Figure 10(b) highlights that a malicious behavior is actually detrimental for some clients belonging to the malicious group. This occurs when a malicious client would obtain a better QoE by following the fairness signal instead of ignoring it. Figure 10(b) shows the average minimum and maximum QoE obtained by the two groups for each subnetwork. The average minimum QoE for the FINEAS group is always higher than or equal to the average minimum QoE of the malicious group; that is, some malicious clients obtain a QoE lower than that they could obtain behaving fairly. Moreover, this behavior becomes more evident as the number of malicious clients increases. The only exception is represented by Network #1 in the scenario with 20% malicious clients. Nevertheless, the FINEAS group outperforms the malicious one from the fairness point of view while achieving a similar QoE.

In the second set of experiments, coordination proxy P3 is lacking, so that clients in Network #2 do not receive the fairness signal anymore (see Figure 3(a)). This scenario represents a situation where proxy P3 is not deployed or cannot forward the fairness signal because of a fault. Figure 11 shows the results, obtained averaging the QoE and its standard deviation over the 50 simulated episodes. As expected, the failure of proxy P3 mainly affects the performance of Network #2 from the fairness point of view: the QoE standard deviation is almost doubled compared to the reference scenario presented in Section 4.4, while the average QoE is the same. This means that some clients experience a lower QoE compared to the reference scenario, as shown also in the malicious clients case. Interestingly, Networks #1 and #3 are unaffected by the failure of proxy P3, both from the QoE and the fairness points of view. This behavior can be explained considering two aspects. First, both networks are still provided with the fairness signal. Second, the average bandwidth consumed by Network #2 does not differ significantly from that consumed in the reference scenario. The proxy failure affects Network #2 from the fairness point of view only, while the average requested quality level, equal to 5.42, does not significantly change compared to the reference scenario (see Table II). This entails that the bandwidth available to Networks #1 and #3 in case of proxy failure and in the reference scenario is similar and no differences in performance between the two scenarios are noticeable.

5. CONCLUSIONS

In this article, we presented FINEAS, a HAS heuristic able to dynamically adapt its behavior depending on network conditions in order to obtain a high QoE. Moreover, this client is able to select the best quality level in order to achieve fairness from the QoE

point of view. This was necessary as state-of-the-art rate adaptation heuristics introduce nonnegligible differences in obtained quality among clients. Fairness is achieved by means of a system of intermediate nodes, called coordination proxies, in charge of collecting information on the overall network conditions. This information is then provided to the FINEAS clients, which use it to refine their quality decision process. Numerical simulations using NS-3 have validated the effectiveness of the proposed approach. Particularly, we have compared our solution with the Microsoft's Smooth Streaming client, a Q-Learning based client [Claeys et al. 2014], a single-client heuristic [Miller et al. 2012], and the FESTIVE algorithm [Jiang et al. 2014]. In the evaluated bandwidth scenarios, we were able to show that our multiclient HAS framework resulted in a better video quality and in a remarkable improvement of fairness, up to 20% and 80%, respectively, when compared to the reference algorithms.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

REFERENCES

- 3GPP. 2013. 3GPP system - fixed broadband access network interworking (3GPP TS 23.139 version 11.3.0 Release 11). 1–90. <http://www.3gpp.org/DynaReport/23139.htm>
- V. Adzic, H. Kalva, and B. Furht. 2011. Optimized adaptive HTTP streaming for mobile devices. *Proceedings of SPIE, Applications of Digital Image Processing*.
- S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. 2012a. What happens when HTTP adaptive streaming players compete for bandwidth? In *22nd International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'12)*. ACM, 9–14.
- S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis. 2012b. An experimental evaluation of rate-adaptive video players over HTTP. *Signal Processing: Image Communication* 27, 4, 271–287.
- N. Bouten, J. Famaey, S. Latré, R. Huyssegems, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. 2012. QoE optimization through in-network quality adaptation for HTTP Adaptive Streaming. In *2012 International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualization Management (SVM'12)*. 336–342.
- M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck. 2014. Design and optimization of a (FA)Q-learning-based HTTP adaptive streaming client. *Connection Science* 26, 1, 27–45.
- C. Claus and C. Boutilier. 1998. The dynamics of reinforcement learning in cooperative multiagent systems. In *15th National / 10th Conference on Artificial Intelligence / Innovative Applications of Artificial Intelligence (AAAI'98 / IAAI'98)*. American Association for Artificial Intelligence, 746–752.
- L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. 2013. ELASTIC: A client-side controller for dynamic adaptive streaming over HTTP (DASH). In *2013 International Packet Video Workshop (PV'13)*. 1–8.
- L. De Cicco, S. Mascolo, and V. Palmisano. 2011. Feedback control for adaptive live video streaming. In *2nd Annual ACM Conference on Multimedia Systems (MMSys'11)*. ACM, 145–156.
- J. De Vriendt, D. De Vleeschauwer, and D. Robinson. 2013. Model for estimating QoE of video delivered using HTTP adaptive streaming. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM'13)*. 1288–1293.
- A. El Essaili, D. Schroeder, D. Staehle, M. Shehada, W. Kellerer, and E. G. Steinbach. 2013. Quality-of-experience driven adaptive HTTP media delivery. In *2013 IEEE International Conference on Communications (ICC'13)*. 2480–2485.
- P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. 2013. Towards network-wide QoE fairness using openflow-assisted adaptive video streaming. In *2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking (FhMN'13)*. ACM, New York, NY, 15–20.
- R. Houdaille and S. Gouache. 2012. Shaping HTTP adaptive streams for a better user experience. In *3rd Annual ACM Conference on Multimedia Systems (MMSys'12)*. ACM, 1–9.
- D. Jarnikov and T. Ozelebi. 2010. Client intelligence for adaptive streaming solutions. In *2010 IEEE International Conference on Multimedia and Expo (ICME'10)*. 1499–1504.
- J. Jiang, V. Sekar, and H. Zhang. 2014. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking* 22, 1 (Feb. 2014), 326–340.

- R. Kuschnig, I. Kofler, and H. Hellwagner. 2010. An evaluation of TCP-based rate-control algorithms for adaptive internet streaming of H.264/SVC. In *1st Annual ACM SIGMM Conference on Multimedia Systems (MMSys'10)*. ACM, 157–168.
- Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. 2014. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications*, 719–733.
- C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj. 2012. Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network. *Image Communication* 27, 4, 288–311.
- K. J. Ma and R. Bartos. 2011. HTTP live streaming bandwidth management using intelligent segment selection. In *2011 IEEE Global Telecommunications Conference (GLOBECOM'11)*. 1–5.
- K. Miller, E. Quacchio, G. Gennari, and A. Wolisz. 2012. Adaptation algorithm for adaptive streaming over HTTP. In *2012 International Packet Video Workshop (PV'12)*. 173–178.
- R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. 2012. QDASH: A QoE-aware DASH System. In *3rd Annual ACM Conference on Multimedia Systems (MMSys'12)*. ACM, 11–22.
- C. Müller, S. Lederer, and C. Timmerer. 2012a. An evaluation of dynamic adaptive streaming over HTTP in vehicular environments. In *4th Workshop on Mobile Video (MoVid'12)*. ACM, 37–42.
- C. Müller, S. Lederer, and C. Timmerer. 2012b. A proxy effect analysis and fair adaptation algorithm for multiple competing Dynamic Adaptive Streaming over HTTP clients. In *2012 IEEE Visual Communications and Image Processing Conference (VCIP'12)*. 1–6.
- S. Petrangeli, M. Claeys, S. Latré, J. Famaey, and F. De Turck. 2014. A multi-agent Q-Learning-based framework for achieving fairness in HTTP Adaptive Streaming. In *2014 IEEE Network Operations and Management Symposium (NOMS'14)*. 1–9.
- H. Riiser, T. Endestad, P. Vigmstad, C. Griwodz, and P. Halvorsen. 2012. Video streaming using a location-based bandwidth-lookup service for bitrate planning. *ACM Transactions on Multimedia Computing, Communications and Applications* 8, 3, Article 24 (Aug. 2012), 19 pages.
- R. de O. Schmidt, R. Sadre, A. Sperotto, and A. Pras. 2013. Lightweight link dimensioning using sFlow sampling. In *2013 International Conference on Network and Service Management (CNSM'13)*. 152–155.
- G. Tian and Y. Liu. 2012. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT'12)*. ACM, 109–120.
- B. Villa, P. Heegaard, and A. Instefjord. 2012. Improving fairness for adaptive HTTP video streaming. In *Information and Communication Technologies*, Robert Szabo and Attila Vidacs (Eds.). Lecture Notes in Computer Science, Vol. 7479. Springer, Berlin, 183–193.
- S. Xiang, L. Cai, and J. Pan. 2012. Adaptive scalable video streaming in wireless networks. In *3rd Annual ACM Conference on Multimedia Systems (MMSys'12)*. ACM, 167–172.
- C. Zhou, X. Zhang, L. Huo, and Z. Guo. 2012. A control-theoretic approach to rate adaptation for dynamic HTTP streaming. In *2012 IEEE Visual Communications and Image Processing Conference (VCIP'12)*. 1–6.

Received April 2014; revised April 2015; accepted May 2015