EdgeDASH: Exploiting Network-Assisted Adaptive Video Streaming for Edge Caching

Suzan Bayhan*, Setareh Maghsudi†, and Anatolij Zubow‡
*University of Twente, The Netherlands, e-mail: s.bayhan@utwente.nl
†Universität Tübingen, Germany, e-mail: setareh.maghsudi@uni-tuebingen.de
‡Technische Universität Berlin, Germany, e-mail: anatolij.zubow@tu-berlin.de

***Abstract*—With increasing demand for video streaming applications, exploiting cached content at the network edge becomes paramount to prevent congestion in the link between the wireless access network and the content providers. However, it is often challenging to exploit the caches in current client-driven video streaming solutions due to two key reasons. First, even those clients interested in the same content might request different quality levels as a video content is encoded into multiple qualities to match a wide range of network conditions and device capabilities. Second, the clients, who select the quality of the next chunk to request, are unaware of the cached content at the network edge. Hence, it becomes imperative to develop network-side solutions to exploit caching, in particular for the scenarios in which multiple video clients compete for some bottleneck capacity. In this paper, we propose EdgeDASH which is a network-side control logic running at a WiFi AP to facilitate the use of cached video content. In particular, an AP can assign a client station to a video quality different than its request, in case the alternative quality provides a better utility. This includes, for example, a function of bits delivered from the cache, video bit rate, and the buffer stalls. We formulate the quality assignment problem as an optimization problem and develop several heuristics with polynomial complexity. Our simulations show that EdgeDASH facilitates significant cache hits and decreases the buffer stalls only by changing the client's request by one quality level, however with some increase in session instability. From our analysis, we conclude that EdgeDASH benefits are more visible especially when the clients with identical interests compete for a bottleneck link's capacity, over the baseline where the clients determine the quality adaptation.**

***Index Terms*—Adaptive video streaming, caching, edge, MPEG SAND, network assistance, resource allocation, WiFi, WLANs.**

I. Introduction

The ever-increasing demand for connectivity and the emergence of high-bandwidth applications have pushed network operators to seek for solutions that increase the network capacity while using the resource efficiently. The *edge networking* paradigm is foreseen as a solution to the aforementioned challenge, which aims at delivering content or computation from the proximity of the users, thereby decreasing the traffic in the network core. Edge caching, in particular, suggests storing the content at the periphery of the network so that the network traffic and service latency becomes lower. Moreover, the operator's cost decreases due to lower backhaul or inter-ISP traffic [1]. These benefits are highly desirable especially for video streaming content, which is bandwidth-hungry and might jeopardize user's satisfaction under high latency. Moreover, for networks with a limited backhaul as depicted in Fig.1, it is crucial to exploit the cached content to decrease the congestion in the link. Although edge caching of the video content is essential for realizing the aforementioned benefits, there are several challenges due to the nature of video delivery. In what follows, we briefly describe some challenges.

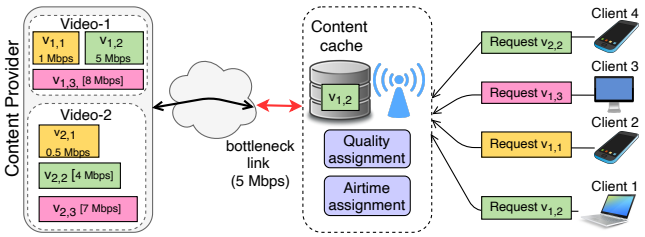


Fig. 1. The WiFi AP can overwrite a request from its DASH clients to avoid congestion in its bottleneck link and to leverage its cache. Here, the AP delivers $v_{1,2}$ encoded at 5 Mbps from its cache by overwriting the requests of Client-2 and Client-3. By doing so, the AP can download the requested content $v_{2,2}$ by Client-4 with bitrate 4 Mbps from the content provider without experiencing congestion in its bottleneck link with capacity of 5 Mbps.

First, current video streaming schemes (i.e., HTTP-based adaptive streaming [2]), rely on multi-level quality representation at the server-side as depicted in Fig. 1 and quality selection at the client-side. Although moving the control over the rate adaptation and content request process to the client offers high scalability, the network or content providers (CP) would suffer due to lack of control over the delivery process. This might result in a lower caching opportunity for the network provider since the network treats different qualities of the same content as different content. Second, clients are unaware of the cached content and thereby cannot favor such cached content in pure client-driven video streaming solutions. Consequently, edge cache hits for video content are limited.

In this paper, we propose to exploit the network assistance, introduced by MPEG server and network-assisted Dynamic Adaptive Streaming over HTTP (SAND) [3], for realizing the potential of edge caching even if the users request different qualities. While network assistance can be implemented in all stages of the content delivery, we focus on the radio access network (RAN), which is a WiFi network. Our scheme, referred to as EdgeDASH, is easy to deploy at the WiFi Access Points (AP) and can work with any client player. EdgeDASH is transparent to the SAND-compliant video clients.

Network assistance at a WiFi AP offers many benefits, including more informed decisions facilitated by the bandwidth

feedback from the WiFi AP and better downlink (DL) resource allocation at the WiFi AP considering the clients' diversity and their statistics, e.g., buffer occupancy. Note that network assistance can take many forms, from airtime or quality assignment to transcoding [4]. But, some network assistance functions are possible only for HTTP traffic as encrypted traffic is opaque to the intermediate network nodes [3].[1] Our focus here is on exploiting the cached content for video clients which compete for a bottleneck link's capacity. In this setting, as an AP needs to access to the content information, we assume that network traffic is not encrypted. Although prior studies [6]–[9] have established the benefits of network-side solutions, to the best of our knowledge, only a few proposals such as [1] consider cached content delivery in a wireless RAN. *Hence, our goal is to develop a quality allocation scheme at a WiFi AP to enable delivery from the cache while considering the clients' performance.* To this end, our contributions are as follows.

- We devise a resource allocation scheme in which the WiFi AP might overwrite the client decisions (i) to favor the consumption of the content from the edge cache and (ii) to decrease the burden on the capacity-limited bottleneck link. In contrast to earlier works, e.g., [10], which transfers the quality selection decision to the network, our approach keeps the client still in the quality selection process. This design choice is motivated by the fact that a client might prefer a certain rate due to various concerns, e.g., a client with limited remaining battery or mobile data budget might prefer streaming the video at the lowest rate. Moreover, due to their rich content catalogue, some users prefer video services such as YouTube to stream music [11]. In such cases, the clients might use third party applications, e.g., FireTube, to deactivate the video, or select the lowest quality due to the above-mentioned concerns. If the network-side DASH solution ignores the client's decision totally, it leads to unsatisfactory user experience. To remedy this, our solution defines a tolerance parameter which restricts the WiFi AP's quality assignment policy to a limited set of bitrates in the neighborhood of the quality selected by the client.
- Moreover, our proposal suggests that an AP allocates its DL airtime to its clients considering the assigned video qualities and the clients' statistics, e.g., buffer level, so that the clients do not experience buffer stalls or low video bitrates. Such network control is particularly useful when the core network has a bottleneck link and there are multiple video clients.
- Finally, we provide an analysis of our proposals and discuss practical aspects such as implementation using SAND [3].

Notation: Throughout the paper, we use u_i and v_j to denote client *i* and video j, respectively. We consistently use index i and index j to indicate users and videos, respectively. Moreover, k and m correspondingly represent the index of a video chunk and quality level. The request of client i is characterized by the following features: video j, chunk k, and quality m. Therefore, we denote the request as $r_i \equiv v_{j,k,m}$. The content that is delivered by the AP is then $\widehat{r}_i \equiv v_{j,k,\widehat{m}}$. Note that m and $\widehat{m}$ are not necessarily equal, meaning that the delivered content may have a different quality level than the requested one. We will denote all requests by $\lambda = [r_1, \cdots, r_N]$ and the content that will be delivered for these requests by $\widehat{\lambda} = [\widehat{r}_1, \cdots, \widehat{r}_N]$, where N is the number of clients.

[1] In case of encrypted traffic, network provider and CP shall cooperate to implement the network assistance by signaling certain information [5].

II. Background on Adaptive Video Streaming

To account for the diversity of end-user equipment and network conditions, a video server supporting Dynamic Adaptive Streaming over HTTP (DASH) divides the video into small chunks of identical length, e.g., 2-10 seconds, and encodes each chunk into multiple qualities, e.g., bitrates. The number of qualities and the required bitrates are a design decision of the CP. Chunk specification, as well as the location of each chunk, is stored in the media presentation description (MPD) file and transmitted to the client at the beginning of a video session. Any communication between the client and the server is performed using the HTTP protocol, which is another merit of DASH: CPs can use ordinary HTTP servers, and video content can flow through network equipment without being filtered at the HTTP-friendly firewalls.

After receiving the MPD manifest file, the client knows the properties of the video content, e.g., the number of quality levels and average bitrate for each quality level. Based on this knowledge and some other network state information, adaptive bitrate selection (ABR) algorithm [2] at the client decides on which video quality to select for the next chunk. Usually, chunks are downloaded one by one to avoid any waste of resources if the user decides to quit the session. The downloaded chunks are stored in the client's playout buffer until their playout time. After some certain number of chunks are downloaded to the buffer (e.g., a few tens of seconds), the video starts to play out. This duration between the user's request and the first playout is referred to as *startup latency*. Throughout the streaming session, there might be times where the buffer is empty resulting in video *stalls*. Studies show that video stalls and long startup latency decrease the user's satisfaction level drastically [5], [12]. *Stalling ratio* measures the fraction of time a user stays in stall state during the video session. Note that initial playout policy has a significant impact on the stalling ratio, e.g., if the playout starts without a certain media in the playout buffer, the stalls are highly likely under network congestion. On the other hand, if the playout waits till many chunks are buffered to avoid stalls, the initial latency might be very high exceeding a user's patience.

While there is no single metric capturing user's quality-of-experience (QoE) commonly accepted in the literature, key factors affecting user's QoE are as follows: (i) stalling ratio, (ii) startup latency, (iii) quality switches resulting in *instability* [13], and (iv) visual video quality measured in terms of average video bitrate [12]. Hence, an ABR scheme aims at maximizing the average bitrate while minimizing the stalling ratio and keeping the startup latency and the session instability at a tolerated level to the human perception. It is a challenge to maintain the balance between these conflicting goals, especially in a multi-user setting where users compete for shared resources.

Network-assistance, introduced recently by SAND [14], aims to alleviate possible performance problems by enabling

protocol messages to be exchanged among network components, e.g., wireless AP or CDN edge servers. However, SAND does not specify how to efficiently use such messages, leaving space for many opportunities. In particular, SAND defines four message types: (i) status messages, (ii) metrics messages such as buffer occupancy, (iii) packets enhancing reception, and (iv) packets enhancing delivery. If a network entity is capable of processing these messages or a subset of them, then it is called a DASH-aware network element (DANE). Through these messages, a client and DANE can communicate for having better decisions on the next chunk to request or the next chunk to deliver [14], [16].

III. Related Work

We categorize the related work into two groups, namely *network-assisted DASH* and *caching for video streaming*.

Network-assisted DASH: So far, several papers demonstrate the benefits of using DANEs for video streaming. For example, in [17], DANE allocates bandwidth equally among the clients and recommends a bitrate to the clients for the next chunk based on the allocated bandwidth. The client follows the recommendation only if its estimation is higher than the recommended value and the buffer exceeds a certain threshold level. Motivated by the shortcomings of purely client-driven rate adaptation approaches, [7] proposes to use an SDN controller to enable centralized control over rate adaptation of multiple DASH clients. In [7], the SDN controller collects some information from the clients, e.g., device capabilities, buffer occupancy, and the like, to maximize QoE of each client as well as to optimize fairness and resource utilization. Similar to [7], [8] focuses on the architecture of network-assistance and develop some schemes using SDN. *Despite sharing identical motivation with [7], our solution leverages SAND and retains the client-driven design of DASH. We provide also an algorithmic solution to run at a WiFi AP.*

Regarding WiFi-based network assistance, [8] provides a comprehensive analysis of DANE assistance for rate selection and queuing on a WiFi network. [18] designs a stall-aware video streaming system that uses DANE messages when available. One of the early works on this subject is [9], where the WiFi AP applies traffic shaping to decrease the frequency of quality switching. Authors experimentally show the advantage of two video clients' benefit from traffic shaping. [19] proposes to allocate AP resources using a weighted fair queuing approach and overwriting the client's decisions when necessary, i.e, the client adaptation logic is not altered. The closest work to ours is SEBRA [20], in which a WiFi AP selects the video bitrates and the channel airtime for each video client, upon the receipt of a chunk request. SEBRA assumes a high-capacity AP to ISP link as opposed to our model with a bottleneck link. With increasing number of wireless devices and video traffic, we believe that it becomes imperative to consider bottleneck links between access network and the CP. Also, SEBRA solves chunk selection problem at every incoming request, whereas our proposal works only periodically, thereby attaining higher scalability. *In addition, our solution differs from [9] and [20] which focus only on radio access resource allocation, in that we exploit edge caching in a more generic setting along with bandwidth allocation to mitigate the performance impairments due to the bottleneck links.*

Caching for video streaming: In [21], the authors explore the effect of caching on DASH rate adaptation algorithm. They then develop a solution to mitigate the rate fluctuations that arise due to the client's overestimation of the available bandwidth when the requested chunk is cached and served directly from the cache server [21]. In [22], the authors suggest placing the video contents on the edge servers strategically such that the initial latency remains below the tolerated latency. Moreover, the clients consult the cellular base station only if unable to find the requested quality at the edge servers. *Our work differs from [22] in many ways. First, we allow for delivering an approximate quality of the requested chunk if serving the latter increases the cache hits without drastically decreasing the user's satisfaction level. Second, in contrast to the earlier works that consider the cellular networks, our setting is a single WiFi cell that operates asynchronously. We discretize the continuous time of WiFi into resource allocation intervals and quality selection intervals to mark the points of action by the WiFi AP and the DASH clients, respectively.*

A very similar study to ours is [1], which suggests maintaining some desired trade-off between the visual quality of the video and the cache hits at the ISP network. They design a coordinated bitrate selection strategy at the DASH clients such that clients will favor already-cached chunks at a slight loss of video quality to increase cache hits. While the solution of [1] is for an ISP network, our solution is hosted on the WiFi radio access network which is more practical than placing the network assistance functionality deep in the core network.

Lastly, by the emergence of end-to-end traffic encryption, both network assistance and caching schemes that rely on content-related knowledge become incompatible. However, there are some works such as [23] and [24] designing mechanisms to enable network control, e.g., caching, even for encrypted traffic. For example, [23] designs a caching scheme where the CPs can leverage the benefits of caching without revealing their content to the cache provider. [24] derives QoE of an encrypted streaming session using supervised learning.

IV. System Model

We consider a single WiFi AP and multiple WiFi stations with active video streaming sessions as in Fig. 1. The link between the WiFi AP and the serving video server is the bottleneck link with a fixed capacity of Γ_{bh} Mbps. In what follows, we describe other elements of our setting.

Video content: Let $\mathcal{V} = \{v_1, \cdots, v_V\}$ denote the set of V videos. Each video content v_j is divided into multiple chunks. Each chunk $v_{j,k}$ is then encoded into several qualities denoted by $\mathcal{Q}_j = \{0, 1, \cdots, Q_j - 1\}$ with $|\mathcal{Q}_j| = Q_j$. The qualities are uniform across all chunks; consequently, we do not include k in the quality description. We denote the bitrate of quality m by $q_{j,m}$ bps. The video provider determines the duration of each video chunk, typically between 2 to 10 seconds, which may differ across different contents. We denote the chunk duration of v_j by τ_j seconds. Note that the encoding process is

inherently variable, therefore the chunks might have different sizes. As a result, the actual size of the k^{th} chunk, denoted by $s_{j,k,m}$, might deviate from the average chunk size which is calculated as $s_{j,m} = q_{j,m} \times \tau_j$ bits [25]. Since the MPD manifest includes only the bitrates $q_{j,m}$ to keep the file size small so that the video client can download it without a long delay, the AP knows only $s_{j,m}$, not $s_{j,k,m}$.

DASH users: Let $\mathcal{N} = \{u_1, \cdots, u_N\}$ be the set of N clients. Moreover, C_i indicates the physical layer capacity of the link connecting each client u_i to the AP. Each video client has a playout buffer of B_{max} seconds. Moreover, B_i indicates the buffered video duration (in seconds) at the client. We do not assume any particular client rate adaptation algorithm. As described in Section I, we denote the requested content of client i by r_i. If required, we identify the requested content with its features, namely video j, chunk k, and quality m, as $r_i \equiv v_{j,k,m}$. In case it is not necessary to specify the quality level, we omit the last index and use $r_i \equiv v_{j,k}$ to simplify the notation. In addition to the video clients, there could also be clients with background traffic. However, an AP can slice its resources for video and other less-QoS sensitive traffic. Hence, we only consider the video traffic.

Finally, the AP has a storage capacity of S bits for caching. The cache admission policy is as follows: the AP admits all the contents while it applies the least-recently-used (LRU) replacement policy for managing its cache space. We denote the cache status of the WiFi AP by $\mathcal{S} = [x_{j,k,m}]$, where $x_{j,k,m}$ returns 1 if chunk k of video j with quality m is stored in the cache. As the cache capacity is limited to S bits, the inequality $\sum_j \sum_k \sum_m x_{j,k,m} s_{j,k,m} \leqslant S$ must hold at any time. Please recall that an AP cannot observe the content of the incoming requests for encrypted traffic. Hence, we assume an unencrypted setting. However, for encrypted traffic, proxy-server based approaches as proposed in [23] can be adopted to realize our proposal. Briefly, the AP hosts a proxy per CP, which runs our proposal taking the backhaul bandwidth and DL airtime allocated by the AP as its resource constraints.

TABLE I
KEY NOTATIONS.

Notation	Description
u_i, $\mathcal{N}$, N	DASH client i, set of clients, number of clients
v_j, Q_j, $\mathcal{V}$ and V	Video j, number of quality levels of video j, set and number of videos
$v_{j,k,m}$	Video j, chunk k, quality level m
$s_{j,k,m}$	Chunk size in bits for $v_{j,k,m}$
$s_{j,m}$	Average chunk size in bits for v_j, quality level m
τ_j	Chunk duration of v_j in seconds
T_{ap}	Resource allocation interval of the AP
B_{max}, B_i	Buffer capacity (seconds), buffer occupancy of u_i
S	Capacity of cache in bits
$x_{j,k,m}$	Equals 1 if v_j's chunk k and quality m is in cache
θ_i	Airtime allocated to u_i
ϕ_i	Decision variable to serve u_i from cache
C_i	Physical layer link rate of u_i
Γ_{bh}	Bottleneck link capacity (Mbps)
$q_{j,m}$	Bitrate of quality level m for v_j
r_i, $\widehat{r}_i$	Request of u_i and delivered request of u_i ($v_{j,k,\widehat{m}}$)
λ, $\widehat{\lambda}$	Set of all requests, and set of all delivered requests
$\mathcal{N}^0$	Set of clients waiting for service but have already been assigned a quality level for their request.
$\mathcal{N}^1 = \mathcal{N} \setminus \mathcal{N}^0$	Set of clients waiting to be assigned a quality level for their request.
μ_c	Weight of cache delivery as compared to delivery from the backhaul
$\Delta\mathcal{E}$	Tolerated quality difference

V. EDGEDASH: RESOURCE ALLOCATION AT THE WIFI AP TO ENABLE EDGE CACHING FOR VIDEO STREAMING

Here, we introduce our solution, namely EdgeDASH, which runs on a WiFi AP for DASH- and cache-aware resource allocation. While aiming at increasing the number of cache hits, EdgeDASH considers two aspects: bandwidth efficiency and QoS of the users.

A. Description of EdgeDASH WiFi AP

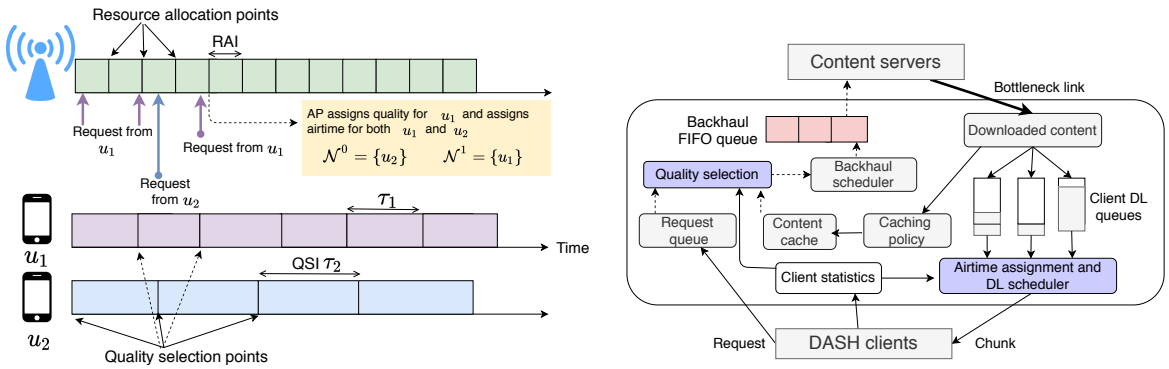
Let us first explain the time scale of actions at the client's player and the WiFi AP. Fig. 2a illustrates the time points at which a client and the WiFi AP take actions. Each client decides on the next chunk's bitrate with a period approximately equal to the chunk duration of the demanded video. We refer to this period *Quality Selection Interval (QSI)* [15] which depends on chunk scheduling at the client player (e.g., periodic requests, immediate requests after completion of each chunk, or randomized chunk scheduling [13]). As observed in [21], in steady state, QSI equals to the chunk duration denoted by τ.[2] Since the clients watch different videos, the chunk duration τ varies across clients. In case of shorter chunk duration, a client can react quickly to changes in the channel or network dynamics, e.g., it selects a different quality matching the client's observed link capacity.

As Fig. 2a illustrates, the AP decides on DL resource allocation periodically, which is referred to as *resource allocation interval* (RAI). In addition to resource allocation, WiFi AP solves the video quality selection problem at the beginning of each RAI. After collecting the client's chunk requests, the AP might overwrite the client's decision for utilizing its cache resources better. Given that QSI might differ across clients, the WiFi AP can select the shortest chunk duration as its RAI, e.g., $T_{ap} = \min_{\forall v_j \in \mathcal{V}}(\tau_j)$, where T_{ap} denotes the RAI length. However, since the QSI is in the order of seconds whereas WiFi works in a finer time scale for scheduling its medium access, RAI can be set as a few milliseconds.

Fig. 2a shows that the requests arrive at the AP asynchronously. This happens due to different chunk scheduling algorithms at clients and different chunk duration of consumed contents. Consequently, at the beginning of a RAI, the AP needs to allocate its resources considering the new requests and those clients who have already been assigned a quality level at a previous RAI. Let $\mathcal{N}^0$ denote the set of clients who have already been assigned quality levels. Moreover, we gather the rest of users in $\mathcal{N}^1$. While assigning video qualities, the AP should ensure that the allocated resources are sufficient to deliver the selected quality of the video for each user.

Fig. 2b illustrates the key functional blocks at an

[2][21] reports that in the steady state, time between two consequent chunk requests equals to chunk duration, i.e., τ seconds. Initially, the client requests video segments until the buffer becomes full, e.g., 10 seconds. Since the buffer cannot accommodate any new chunk, the client consumes one chunk before requesting the next chunk resulting in a QSI duration equal to chunk duration.



(a) Quality selection intervals and AP resource allocation interval.

(b) Functional blocks in an EdgeDASH WiFi AP.

Fig. 2. (a) Resource allocation and quality selection time intervals and (b) AP functional blocks.

EdgeDASH WiFi AP, as briefly described next. *Request queue* stores the clients' chunk requests. The *quality selection* module checks the request queue as well as its content cache to decide which requests to send toward the content servers and which ones to satisfy from its cache in case of any match. As the AP has a backhaul connectivity with a capacity of Γ_{bh} (Mbps), it takes this capacity limit into account in addition to the client statistics collected from the clients. In this step, the AP may decide to serve the user's request by a cached chunk of a different quality, if the client's streaming quality does not become deteriorated by this difference significantly. *Backhaul scheduler* takes the output of the quality selection module to download requested chunks. The chunks are downloaded in a FIFO manner and in case the selected qualities exceed the capacity, there will be a certain queuing latency at the backhaul link. After fully downloading the chunks, the AP moves them to the DL queue of each client. The *DL scheduler* then allocates the DL radio resources, i.e., its airtime, and delivers the chunks in a round-robin manner.

B. Problem Formulation

Decision variables: Recall that the request of u_i for the content $v_{j,k,m}$ is denoted by r_i. For each user in $\mathcal{N}^1$, the WiFi AP will decide on the following three parameters: quality level ($\widehat{m}_i$), airtime (θ_i), and delivery from cache or not (ϕ_i). For users in $\mathcal{N}^0$, the AP assigns only airtime. We explain these decision variables as follows:

- $\widehat{m}_i$: It takes a value from the set of available quality levels of the video that u_i has requested, i.e., $\widehat{m}_i \in \mathcal{Q}_j$. The corresponding bitrate is $q_{j,\widehat{m}_i}$. Since the quality level might be modified by the AP, we will represent the assigned content as $\widehat{r}_i = v_{j,k,\widehat{m}_i}$.
- θ_i: The WiFi AP has to determine the share of DL airtime allocated to each user, denoted by $\theta_i \in [0, 1]$.
- ϕ_i: The client's request will be satisfied from the cache (i.e., $\phi_i = 1$) or from the backhaul (i.e., $\phi_i = 0$).

Objective: Our proposal aims at delivering a large number of bits from the cache while maintaining a high streaming quality. To this end, the AP maximizes the number of delivered bits (hence the visual quality) prioritizing the cache delivery over the backhaul delivery while considering the buffer level as video stall is known to be a significant factor in decreasing the user satisfaction. Hence, the AP should favor video qualities that are encoded at higher bitrates but can still ensure that the client's buffer level is above a certain threshold ($B_{\min}$).

Let $\hat{B}_i$ denote the expected buffer level (in seconds) of a client i when the current chunk with quality $\widehat{m}_i$ is delivered to the client. In addition, we introduce μ_c as a tuneable parameter that reflects the desirability of cache delivery over the backhaul delivery. In Section VII, we describe the procedure of AP to calculate $\hat{B}_i$ given the assigned quality $\widehat{m}_i$ and assigned airtime θ_i.[3] We first define the utility $\mathcal{U}_i$ of user u_i as follows:

$$\mathcal{U}_i = \begin{cases} \mathcal{U}_1 & \text{if } \hat{B}_i \geqslant B_{\min} \\ \mathcal{U}_2, & \text{if } B_{\min} > \hat{B}_i > 0 \\ \mathcal{U}_3, & \text{otherwise.} \end{cases} \tag{1}$$

where $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3$ are defined as follows:

$$\mathcal{U}_1 = \log(q_{j,\widehat{m}_i})(\mu_c\phi_i + (1-\phi_i)) + \log(\min(\hat{B}_i, B_{\max})) \tag{2}$$

$$\mathcal{U}_2 = \log(\hat{B}_i)(\mu_c\phi_i + (1-\phi_i)) \tag{3}$$

$$\mathcal{U}_3 = \hat{B}_i. \tag{4}$$

The rationale behind our choice of utility function is the following: when a candidate bitrate ensures a buffer level above $B_{\min}$, then the associated utility is the logarithmic function of the video bitrate considering the weight of the cache delivery and the estimated buffer level. We prefer logarithm function to reflect the diminishing returns with increasing bitrate in terms of user's perceptual quality [6]. When a candidate bitrate does not cause buffer stalls but cannot satisfy the target buffer level, then the corresponding utility is the logarithm of the bitrate multiplied by the cache weight. Finally, if the candidate bitrate is expected to result in negative buffer levels, we define the utility as the estimated buffer level. Note that a negative buffer level indicates a stall with its magnitude reflecting the expected stall duration. As such, by incorporating the buffer level in the utility function, we aim at avoiding inappropriate bitrates that might result in buffer stalls. We use the expected buffer stall duration as the utility to differentiate among the quality levels that fail to sustain a smooth playout. As a

[3]For the simplicity of the notation, we do not include the assigned bitrate and airtime in denoting the estimated buffer level which depends on these two factors, i.e., $\hat{B}_i(q_{j,\widehat{m}}, \theta_i)$.

result of this choice, if all quality levels are also expected to lead to buffer stalls, we select the bitrate that results in the shortest estimated buffer stall duration. Under a request delivery decision $\widehat{\lambda} = [\widehat{r}_i], \forall u_i \in \mathcal{N}$, and airtime allocation decision θ, we formalize the objective as:

$$\underset{\widehat{m}\in\mathcal{Q},\theta\in[0,1]}{\text{maximize}} \sum_{\forall i\in\mathcal{N}^1} \mathcal{U}_i. \tag{5}$$

Note that (5) concerns with the quality assignment for users in $\mathcal{N}^1$. The performance of the users in $\mathcal{N}^0$ will be handled by appropriate constraints, which we present next.

Constraints:

- If u_i is assigned a chunk with the quality level $\widehat{m}_i$ that exists in the cache ($x_{j,k,\widehat{m}_i} = 1$), then it is served from the cache and the content is not downloaded again. In this case, the following constraints guarantee that ϕ_i is 1.

$$\sum_{m\in\mathcal{Q}_j} x_{j,k,m}\mathbb{1}_{(\widehat{m}_i=m)} \leqslant \phi_i Q_j, \forall u_i \in \mathcal{N}^1, \text{and } r_i \equiv v_{j,k,m}.$$

$$\sum_{m\in\mathcal{Q}_j} x_{j,k,m} \geqslant \phi_i, \forall u_i \in \mathcal{N}^1, \tag{6}$$

  where $\mathbb{1}_{g(\cdot)}$ is the indicator function that yields 1 if the Boolean statement $g(\cdot)$ is true and 0 otherwise. Thus, the indicator function in (6) is 1 if the assigned quality is m.
- The difference in the requested and delivered quality levels is smaller than or equal to $\Delta\mathcal{E} = \{0, 1, 2, \cdots\}$. We refer to $\Delta\mathcal{E}$ as the *tolerated quality difference*. We assume that the system designer selects tolerated quality difference for each client independently. Alternatively, a client ABR might also be modified to signal its tolerance level to the WiFi AP. Obviously, $\Delta\mathcal{E} = 0$ implies that only requested quality and no alternative is acceptable. Formally,

$$\widehat{m}_i - m \leqslant |\Delta\mathcal{E}|, \forall i \in \mathcal{N}^1 \text{ and } r_i \equiv v_{j,k,m}. \tag{7}$$

- As the backhaul capacity is limited to Γ_{bh} Mbps, we have:

$$\sum_{\forall i\in\mathcal{N}^1} (1-\phi_i) q_{j,\widehat{m}_i} \leqslant \Gamma_{bh}, \tag{8}$$

  in which we assume that downloading of the previously requested content for users in $\mathcal{N}^0$ is already completed. In case it is not, the AP considers the remaining bandwidth for these new requests.
- The DL airtime of the WiFi AP is limited to 1, which can be formalized as:

$$\sum_{\forall i\in\mathcal{N}} \theta_i \leqslant 1. \tag{9}$$

  For taking the uplink traffic into account, one can further restrict the allocated time, implying that $\sum_{\forall i\in\mathcal{N}} \theta_i \leqslant \theta$ where $0 < \theta < 1$ [20].
- While utilities defined in (1) favor qualities that would avoid buffer stalls, the AP can additionally assert for each client a lower bound on the expected buffer level. Formally,

$$\hat{B}_i \geqslant \Phi, \forall i \in \mathcal{N}, \tag{10}$$

  where lower values of Φ (e.g., zero) increase the chances of finding a feasible solution.

In this setting, the challenge is to solve the following optimization problem:

$$\max_{\widehat{m},\theta,\phi} \sum_{\forall i\in\mathcal{N}^1} \mathcal{U}_i \tag{11}$$

$$\text{s.t. } (6), (7), (8), (9), (10), \tag{12}$$

$$\widehat{m}_i \in \mathcal{Q}_j, \forall i \in \mathcal{N}^1 \text{ and } r_i \equiv v_{j,k,m} \tag{13}$$

$$\theta_i \in [0, 1], \forall i \in \mathcal{N} \tag{14}$$

$$\phi_i \in \{0, 1\}, \forall i \in \mathcal{N}^1. \tag{15}$$

Due to the existence of decision variables belonging to a discrete set, the problem in (11)-(15) is computationally hard. In the next section, we address this challenge by first modeling our problem as a multiple choice knapsack problem (MCKP), which is NP-hard [26]. Afterward, we propose a heuristic solution based on an approximation algorithm for solving MCKP [27]. In our approach, we first assume equal airtimes for each client and concentrate on the quality assignment. Afterward, we assign airtimes given the quality levels.

VI. Video Quality Assignment as a Multiple Choice Knapsack Problem

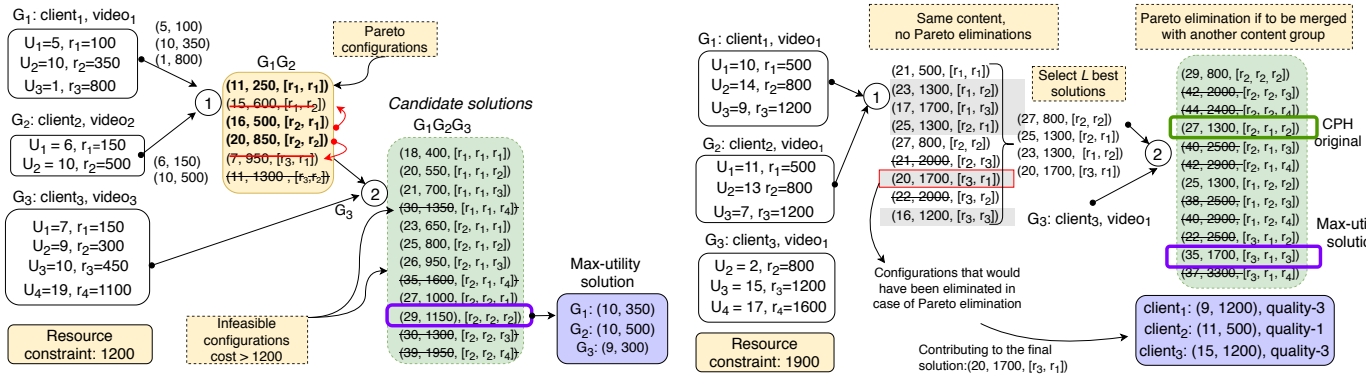
To simplify the NP-hard problem formulated in Section V, we divide it into two problems, namely (i) quality selection and (ii) airtime assignment. In brief, the solution is as follows: In the first step, we adapt an existing solution for 0-1 MCKP, namely *Compositional Pareto-algebraic Heuristic* (CPH) [27], to our problem assuming equal airtimes for all clients. In the second step, we propose an airtime assignment approach that aims at minimizing the buffer stalls.

A. Compositional Pareto-algebraic Heuristic (CPH)

In the following, we provide a brief overview of *compositional Pareto-algebraic Heuristic* (CPH) [27] and introduce a procedure to adapt it to solve the formulated video quality assignment problem.

CPH is designed to solve multidimensional MCKP. Fig. 3a shows a toy example for a single-dimensional MCKP in which there are three groups (corresponding to the clients), each with multiple items (corresponding to the quality levels). For each item in each group, there is an associated utility and resource consumption value. Moreover, the knapsack has a maximum capacity (corresponding to the capacity of the bottleneck link, e.g., minimum of DL and backhaul capacity). The objective of CPH is to select an item from each group such that the total utility is maximized without violating the resource constraint in each dimension. Note that although CPH is designed for multidimensional MCKP problems, it does not entail extra complexity or overhead when applied to solve the single-dimensional problems. Rather than considering all of the groups at once, CPH takes two groups and merges them simply by applying the Cartesian product.[4] Afterward, in this set, CPH eliminates the configurations that are *dominated* or that are infeasible due to a violation of resource constraints.

[4]As shown in Fig. 3a, Cartesian product of G_1 and G_2 results in G_1G_2.



(a) CPH in original form. (b) CPH adapted for the quality selection problem.

Fig. 3. (a) A schematic overview of CPH, as explained in Section VI: To reduce the size of each partial configuration, at each step, CPH removes configurations that are Pareto-dominated or violate the resource constraints. (b) CPH adapted to our quality selection problem: The clients that request the same content are merged without Pareto elimination. This variant of CPH keeps the shaded configurations in Step 1 due to the following reason. These configurations might later yield the maximum utility when another client requests the same content. In this example, the maximum utility consists of a partial configuration (20, 1700) that would be eliminated by CPH resulting in a lower utility, i.e., 27 instead of 35.

For instance, in Fig. 3a, configuration (20, 850, $[r_2, r_2]$)[5] dominates (7, 950, $[r_3, r_1]$) as its utility is higher while its cost is lower. At each step, CPH only keeps the Pareto-optimal configurations, which significantly reduces the complexity. CPH continues merging the set of Pareto-optimal points with one of the remaining sets (see Step 2 in Fig. 3a). Optionally, to control its runtime and space complexity, at each intermediate step, CPH retains only L configurations with the highest utility out of all Pareto-optimal points; where L is a parameter to be tuned based on the desired runtime and acceptable complexity. After merging all sets, the configuration with the highest utility is selected. In Fig. 3a, the best decision is to assign quality level 2 to all clients which achieves a utility equal to 29 with resource consumption value of 1150. We do not consider any optional steps, but, various approaches to optimize the merging step, e.g., the order of sets to be merged, are introduced in [27]. Example in Fig. 3a corresponds to our quality assignment problem when the clients request different contents and the AP's cache does not contain any of the candidate chunks.

In its original setting, CPH is not applicable to our formulated problem where the AP has a cache and there might be clients requesting the same content. Hence, we propose the following procedure summarized in **Alg. 1** to adapt CPH to our setting. First, we consider the cached content and assign their resource consumption to zero (Line 5). As described before, CPH merges the groups by applying Cartesian product to the selected groups (Line 11 and Line 12) where utility and cost of the two members are added. Nonetheless, in our problem, it is necessary to check if the two settings correspond to the same content (Line 13 where $v_{ps,a}$ denotes video id(s) of the configuration a in the partial solution set G_{ps}), so that we do not add the cost twice (Line 14). This implies that the AP downloads each video item from the backhaul only once. In fact, this step breaks the main requirement of CPH that the utility and resource consumption of two configurations are additive. In case a content is requested by multiple clients, the MCKP abstraction might result in low utility. If different contents are requested, we simply add the costs and utilities (Line 16).

Consider the example in Fig. 3b in which all clients request the same content. Notice that the utilities for the same quality level might differ from one client to another as the clients might have different buffer levels or channel link capacities. We modify CPH as follows: if a content is requested by multiple clients, we first merge these groups requesting the same content without Pareto elimination. Skipping the Pareto elimination step is crucial, as a Pareto-dominated configuration in a single-user setting (e.g., gray-shaded configurations in Fig. 3b) might yield the highest utility in a multi-user setting with a feasible cost. In Fig. 3b, the final solution achieving the maximum utility stems from one of the configurations that would have been eliminated if Pareto elimination had been applied. As Fig. 3b shows, CPH without any modifications would result in a lower utility: 27 as compared to 35. After all groups associated with this content are merged one by one, we reduce the set to Pareto-optimal points, since from now on we do not have the risk of eliminating potentially good solutions. While this approach works for small settings, the complexity increases significantly for scenarios such as when many clients request the same contents. Therefore, at each intermediate step, only L configurations with the highest utility can be kept for maintaining a higher scalability. However, tuning L is not straightforward. Hence, we apply Pareto minimization (Line 19). For cases where with no feasible configuration, the AP does not overwrite the client requests (Line 21).

For a video with Q quality levels, the worst-case complexity of CPH is $\mathcal{O}(N \max(Q \log Q, L^4))$ [27]. Note that the actual complexity is usually much lower, e.g., there are $2\Delta\mathcal{E} + 1$ quality levels in each configuration set rather than Q.

B. Airtime assignment for minimizing buffer stalls

After assigning the quality levels, i.e, $\widehat{\lambda} = [\widehat{r}_i]$, we proceed with airtime assignment. To decrease the probability of buffer

[5] A configuration is identified as a triple. For example, in (20, 850, $[r_2, r_2]$), 20 represents the total utility, 850 is the total cost, and $[r_2, r_2]$ shows the determined quality level for each client.

Algorithm 1 CPH-based quality assignment (CPH)

Input: Requests to be assigned a quality ($\lambda = [r_i]$), client-AP link capacity considering equal airtime allocation, AP DL client queues ($\mathcal{D}_i$), client buffer level (B_i), available backhaul capacity (Γ_{bh}), cache status ($\mathcal{S} = [x_{j,k,m}]$)
Output: $\widehat{\lambda}$: video chunks to deliver for each u_i.
Find set of tolerated quality levels ($\mathcal{M}_i$) for each client request r_i using tolerated quality difference $\Delta\mathcal{E}$.
Calculate utility $\mathcal{U}_{i,m}$ for each client and quality $m \in \mathcal{M}_i$.
Calculate the cost of delivering each quality as: $\omega_{j,m} = q_{j,m}$ if $x_{j,k,m} = 0$; and 0, otherwise.
Form a group per u_i using utility and costs $G_i = \{< \mathcal{U}_{i,m}, \omega_{i,m}, v_{j,k,m} >\}$ where $m \in \mathcal{M}_i$.
Set partial solution: $G_{ps} = G_i$ and $\mathcal{G} = \bigcup_{\forall l \in \lambda} G_l \setminus G_i$.
while $\mathcal{G} \neq \emptyset$ **do**
  Select a group G_i from $\mathcal{G}$ for merging with G_{ps}.
  Initialize partial solution $G'_{ps} = \emptyset$.
  for $G_{ps,a} \in G_{ps}$ **do**
    for $G_{i,b} \in G_i$ **do**
      if $v_{i,b} \in v_{ps,a}$ or $v_{i,b} == v_{ps,a}$ **then**
        $G_{ps,c} =< \mathcal{U}_{ps,a} + \mathcal{U}_{i,b}, \omega_{ps,a}, [v_{ps,a}, v_{i,b}] >$
      else
        $G_{ps,c} =< \mathcal{U}_{ps,a} + \mathcal{U}_{i,b}, \omega_{ps,a} + \omega_{i,b}, [v_{ps,a}, v_{i,b}] >$
      $G'_{ps} = G'_{ps} \bigcup G_{ps,c}$
  $\mathcal{G} = \mathcal{G} \setminus G_i$ and $G_{ps} = G'_{ps}$
  Get Pareto-optimal points: $G_{ps} = \text{Pareto-min}(G_{ps})$.
if $G_{ps} = \emptyset$ **then**
  return $\widehat{\lambda} = [r_i]$.
else
  Get the configuration with the maximum utility from G_{ps} and retrieve the corresponding quality levels $\widehat{r}_i$.
  return $\widehat{\lambda} = [\widehat{r}_i]$.

stalls, the AP aims at sustaining a minimum buffer level (in seconds) for its clients, e.g., $B_{\min} > 0$. To this end, the AP calculates the required airtime to reach the aforementioned target level based on the current value B_i. However, if there is not sufficient content in the AP's downlink queue for some specific client, the allocated airtime is wasted. Hence, the AP shall consider the queue size (in bits) for each client ($\mathcal{D}_i$). Formally, the AP calculates the required airtime for each specific client u_i as follows:

$$\theta_i = \min(\mathcal{D}_i, (B_{\min} - B_i)\bar{b}_i)/(C_i T_{ap}), \tag{16}$$

where $\bar{b}_i$ is the average bit rate of the chunks that are in the DL queue for client u_i. If the required airtime is positive, the AP moves u_i to the list of clients that might experience buffer stalls, referred to as the set of *risky* clients. In case the sum of all the required airtime by such clients exceeds 1, then the AP allocates each client some airtime which is proportional to its actual need normalized by the total required airtime of the risky clients. The clients who have already sufficient media in their buffer (e.g., two chunks are already in the buffer) are not served in this interval. If the total required airtime for risky clients is less than 1, first each risky client receives its required airtime. Next, the AP divides the remaining airtime equally among the remaining clients not in the risky set.

VII. Quality assignment for avoiding buffer-stalls (BUFF)

As discussed in Section VI, in some cases (e.g., when multiple clients request the same content), CPH might have inferior performance compared to some heuristic that does not use the MCKP abstraction. To address this issue, we propose BUFF, whose objective is to assign a high video rate while avoiding buffer stalls. As the cached chunks might be prioritized, BUFF also uses a weighted sum as its objective: $\log(q_{j,\widehat{m}_i})(\mu_c\phi_i + (1-\phi_i))$.

Let $\hat{B}_i$ denote the buffer level of client u_i when the currently requested chunk, e.g., chunk i, is downloaded to u_i. $\hat{B}_i$ depends on three factors, namely, (i) current state of the buffer, (ii) from where the chunk is delivered (i.e., backhaul or the cache), and (iii) the state of the current downlink queue of u_i at the AP. Based on the combination of the aforementioned factors, a number of scenarios might occur. Below, we explain these cases which are illustrated in Fig. 4.

- **Case I: Backhaul, empty client DL queue-** In this case, u_i requests some chunk i that is not stored in the cache. Consequently, it has to be downloaded from the backhaul link that might also have other chunk requests waiting in the backhaul queue. Let T_b denote the latency due to backhaul download, which depends on the quality of the chunk as well as the queue size at the backhaul FIFO queue. Moreover, the AP does not have any other chunks to deliver to u_i, meaning that the DL queue of u_i is empty. The downloaded chunk is then transmitted in the DL according to the airtime allocated to u_i. Let T_{dl} indicate the required time to complete the transmission. Thus, in total, the delivery of the chunk to u_i takes $T_b + T_{dl}$ time units. Meanwhile, the client buffer depletes, resulting in $\hat{B}_i = B_i - (T_b + T_{dl})$. As mentioned earlier, $\hat{B}_i$ can take negative values, reflecting the buffer stall period.
- **Case II: Backhaul, non-empty client DL queue-** This case is similar to Case I, except for the fact that the client's DL queue is not empty. As a result, the AP transmits the existing chunks to the client while simultaneously downloading the chunk from the backhaul. Hence, on one hand, the buffer level decreases due to the playout, and on the other hand, it increases by downloading the queued chunks. Therefore, we calculate the number of chunks that can be transmitted during T_b. Let $\mathcal{D}_i$ and $\tau_{\mathcal{D}}$ indicate the queue size in bits and the corresponding number of chunks, respectively. Then, the AP requires $\mathcal{D}_i/(C_i\theta_i)$ seconds to transmit all bits in its DL queue. If T_b is shorter than this duration, the newly-downloaded chunk has to wait until all bits are delivered. Otherwise, the chunk does not experience any queuing delay in the downlink. Consequently, the estimated buffer yields $\hat{B}_i = B_i - \max(\mathcal{D}_i/(C_i\theta_i), T_b) - T_{dl} + \tau_{\mathcal{D}}$.
- **Case III: Cache, empty client DL queue-** In this case, it is clear that $\hat{B}_i = B_i - T_{dl}$.
- **Case IV: Cache, non-empty client DL queue-** Here the AP first transmits the existing chunks in the DL queue. Subsequently, we calculate $\hat{B}_i$ as $\hat{B}_i = B_i - \mathcal{D}_i/(C_i\theta_i) - T_{dl} + \tau_{\mathcal{D}}$.

Algorithm 2 BUFF

1: **Input**: Requests to be assigned a quality ($\lambda = [r_i]$), client-AP link effective capacity, AP DL client queues ($\mathcal{D}_i$), client buffer level (B_i) , available backhaul capacity (Γ_{bh}), cache status ($\mathcal{S} = [x_{j,k,m}]$)
2: **Output**: $\widehat{\lambda}$: video chunks to deliver for each u_i.
3: Initialize quality assignment list as $\widehat{r} = []$.
4: Find the candidate qualities for each client request using tolerated quality difference $\Delta\mathcal{E}$.
5: Calculate utility of each client and the quality pair $\mathcal{U}_{i,m}$. Add it to the set of utilities $\mathcal{U}$.
6: Calculate the delivery costs $\omega_{i,m}$.
7: **while** $\lambda! = \emptyset$ and $\Gamma_{bh} > 0$ **do**
8: Get the best setting: $(i^*, m^*) = \arg\max_{i,m} \mathcal{U}$.
9: Assign quality m^* to the request of u_{i^*}: $\widehat{r}_{i^*} = m^*$ and the corresponding chunk is content v_{j^*,k^*,m^*}.
10: Remove r_{i^*} from unassigned requests, i.e., $\lambda = \lambda \setminus r_{i^*}$, and all configurations of u_i, i.e., $\mathcal{U} = \mathcal{U} \setminus \mathcal{U}_{i^*,}$.
11: Set the cost of all requests for the chunk v_{j^*,k^*,m^*} to zero, i.e., ω_{l,m^*}=0, where $r_l = v_{j^*,k^*}$.
12: Decrease the available backhaul as $\Gamma_{bh} = \Gamma_{bh} - \omega_{m^*}$.
13: Remove infeasible settings with $\omega_{l,m} > \Gamma_{bh}$ from $\mathcal{U}$.
14: Append assigned quality $\widehat{r}_{i^*}$ to $\widehat{\lambda}$.
15: **return** $\widehat{\lambda}$

In the rest of this section, we describe our proposed algorithm *BUFF* whose procedure is briefly summarized in **Alg. 2**. The AP first finds all of the quality levels that are in the tolerated range of the client (Line 4). Moreover, the AP analyzes the expected buffer level assuming identical airtime allocation. Since the goal is to avoid any buffer stall, the AP omits the quality levels that cannot fulfill this goal. However, if the quality level is the minimum level that can be assigned to the client, the AP keeps it as the only viable option. It then calculates the utility of each quality level (Line 5). Afterwards, it greedily assigns the quality levels by picking the highest utility among all of the client-quality pairs (Line 8). After the AP assigns some client u_i a quality level, the cost of the remaining clients demanding the same chunk becomes zero since the AP does not download a content multiple times (Line 11). The AP decreases the available backhaul capacity considering the bitrate assigned in this step (Line 12) and removes the quality levels whose bitrate exceed the available remaining backhaul capacity (Line 13). BUFF terminates either when at least one of the following conditions holds: (i) all clients are assigned a quality level; (ii) the backhaul capacity is exhausted. Finally, BUFF uses the same airtime assignment approach in Sec.VI-B.

The computational complexity of BUFF is calculated as follows. For each client, BUFF calculates the utility for all tolerated quality levels resulting in $\min(2\Delta\mathcal{E}+1, Q)N$ operations. Afterward, it finds the maximum utility at each iteration. The iterations continue until either all clients are assigned a quality level or the backhaul is exhausted. Assuming the first case occurs earlier, then the complexity yields $\mathcal{O}(\min(2\Delta\mathcal{E}+1, Q)N^2)$.

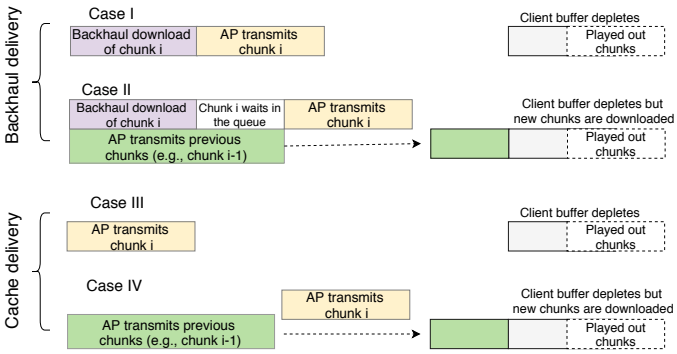


Fig. 4. Illustration of the buffer dynamics while the current requested video is delivered. **Case I:** During the download, the AP waits as it has no bits to transmit to the client. Hence, the client's buffer will deplete. **Case II:** Client buffer both depletes with the rate of playout and increases with the rate of downloaded video from the AP's queue. Time needed to download the current chunk is shorter than the time needed to transmit all bits in the queue. As a result, the currently downloaded chunk experiences queuing latency in the AP's queue before it is delivered to the client. **Case III:** The content will be delivered directly from the cache. **Case IV:** Since the AP has other chunks to deliver to this client, the content fetched from the cache waits till the previous chunks are delivered to this client.

VIII. Performance Evaluation

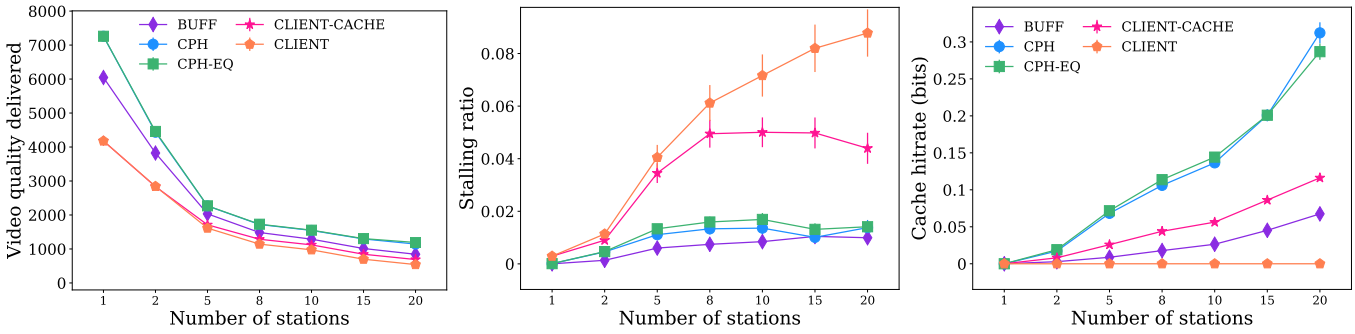
To evaluate our proposals, we conduct simulations using our system-level WiFi simulator developed in Python.[6]

A. Evaluation setting

The WiFi AP operates on a channel of 40 MHz and we model the AP-client link as a Keenan-Motley channel [28]. At the client-side, the adaptation algorithm is the rate-based adaptive (RBA) algorithm [29]. Briefly, the client selects the highest bitrate smaller than the estimated rate. As rate estimation approach, we use harmonic-rate estimation algorithm [30] which computes the harmonic mean of the previously downloaded five chunks. For the initial chunks, there is no data to estimate the rate; therefore, the client picks the lowest bitrate until it fills out its buffer. In essence, most of the DASH clients follow this approach to ensure low startup latency. A client can generate back to back requests for the chunks to fill its buffer quicker in the initial phase, i.e., before watching the first chunk of the video. After the client fills the buffer and playout starts, the client can have at most three requests on the fly. Moreover, when the buffer is full, the client does not generate any new requests until the buffer has some space for the new chunks.

As our video content catalogue, we used both a real video trace [25] and synthetic video set. The data set in [25] has 23 video clips encoded using H.264 encoder, each with either 16 mins or 10 mins length. We follow the recommendation of some earlier work[7] to use chunk sizes of 2-4 seconds that finely addresses the trade-off between overheads and throughput efficiency. The average video bitrates range from 232 Kbps to 4273 Kbps. Each video file has the actual chunk size information. We also generate synthetic traces with higher bitrates, e.g., from 100 Kbps to 15 Mbps, and with 19 quality levels. As the trends are similar, we report results from

[6]Source code of the simulator can be provided on request.
[7]Please see more at https://bitmovin.com/mpeg-dash-hls-segment-length/.



(a) Video quality in Kbps. (b) Stalling ratio. (c) Cache bit hit ratio.

Fig. 5. Impact of the number of DASH clients on the video quality, the cache bit hit ratio, and the stalling ratio.

the synthetic trace. We assign each client a video randomly from the video catalog considering Zipf content popularity model with exponent 1.2. We assume the existence of a high-capacity cache to keep the impact of the cache admission and replacement policy minimal. The cache is initially empty and gets filled over time with requested contents that are downloaded from the CP. Note that our aim here is to analyze the performance of our proposals when the content is expected to be in the cache. Hence, we choose a setting, e.g., Zipf exponent 1.2, small content catalogue and a large cache, which ensures that the AP finds some requested content in the cache and might decide to deliver such cached content.[8]

Unless otherwise stated, we use the following parameters: $\Delta\mathcal{E} = 2$, $B_{\max} = 15$ seconds and $B_{min} = 4$ seconds, $\mu_c = 1.3$, $\Gamma_{bh} = 20$ Mbps, $N = 10$, $V = 10$, $T_{ap} = 0.5$ seconds, and RAI $= T_{ap}$. Clients are uniformly distributed in the coverage area of the AP, which we model as a circle with a radius of 70 meters.

We evaluate the following schemes:

- **CPH-EQ:** CPH with equal airtime allocation,
- **CPH**: This approach is CPH with an airtime allocation that considers the buffer occupancy of each client,
- **BUFF**: BUFF with airtime allocation identical to CPH,
- **CLIENT:** The AP acts as a repeater without checking if the requested content is already cached or not. Moreover, it allocates the airtime equally among its clients, and
- **CLIENT-CACHE:** Similar to CLIENT, this scheme does not assign a quality level. However, different from CLIENT, it first checks the cache and delivers the requested content from the cache upon availability.

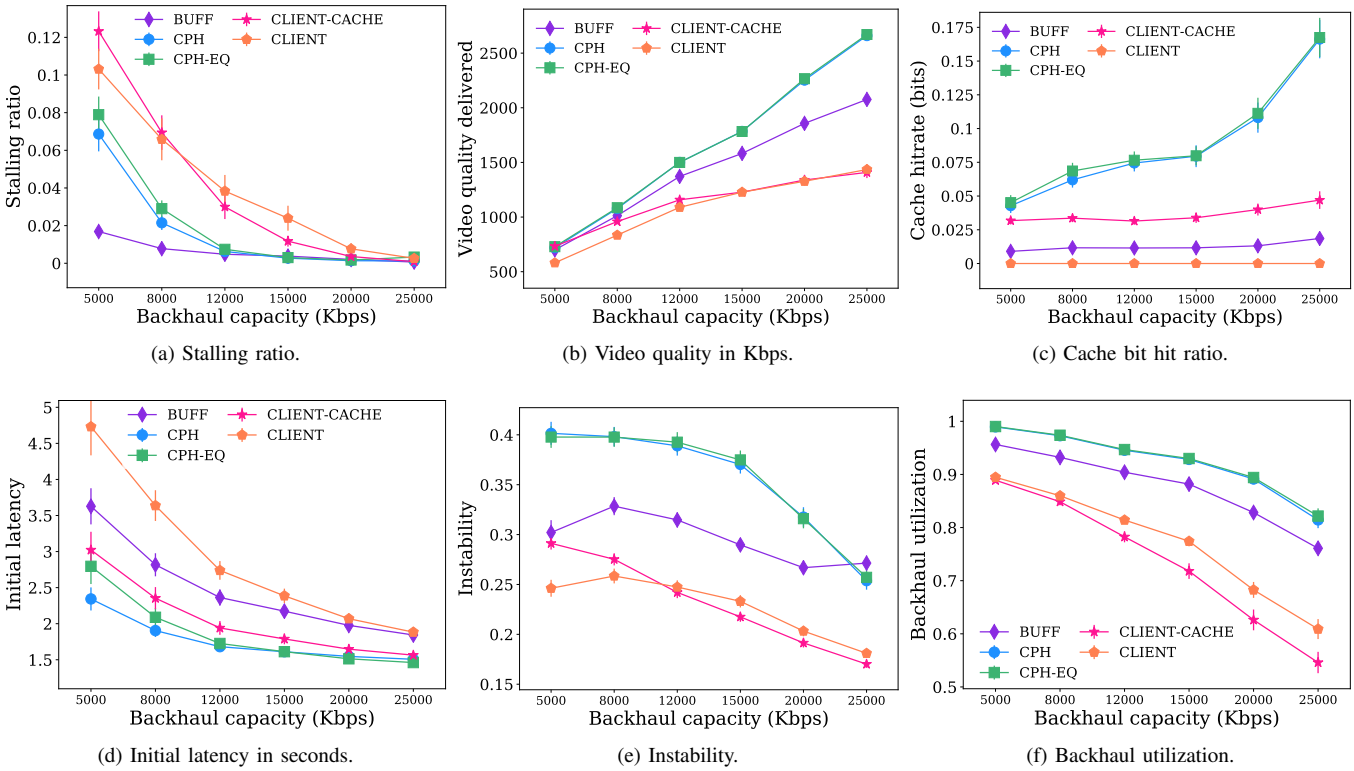
Among the aforementioned schemes, CLIENT is the baseline, as it corresponds to the usual operation of client-driven DASH. When CPH variants and BUFF cannot find a feasible solution, the AP does not change the requests and delivers the requested qualities. We run each scenario for 200 times and report the average of the statistics along with 95% confidence intervals.

B. Performance analysis

Impact of the number of clients: Fig. 5 shows the performance of each scheme as a function of the number of DASH clients (N). In Fig. 5a, we observe that all schemes maintain a lower video bitrate with larger N while CPH variants sustain the highest video bitrates without resulting in many stalls (Fig. 5b). For example, for a single user, CPH provides 3 Mbps higher bitrate corresponding to 74% improvement over CLIENT, whereas the improvement is 112% when $N = 20$ enabled by 0.3 Mbps higher bitrate. Similarly, BUFF provides 45% and 56% improvement for $N = 1$ and $N = 20$, respectively. With increasing N, the stall ratio increases for all schemes. However, the growth shows a higher rate for the variant of CLIENT such that the video session might become very unpleasant. As a comprehensive example, consider the following scenario. With only a few clients, streaming is smooth without interruptions for all schemes, as shown by Fig. 5b; nevertheless, for $N = 5$, CLIENT results in stalls more frequently, specifically around 3-4% of the session. With larger values of N, stalls might occur even more often, as frequent as 6-8% of the session, whereas it remains around 1% for our proposals.

With respect to the video bitrate performance, the schemes can be sorted as follows: CPH or CPH-EQ, BUFF, CLIENT-CACHE, and CLIENT. There is almost no quality difference between CPH and CPH-EQ while we observe a slightly higher cache bit hit ratio in Fig.5c achieved by CPH-EQ in some settings. This higher cache bit hit rate could be due to the similar link capacity observations of the clients, which result in requesting the same video qualities. Moreover, we believe that CPH-EQ can maintain the same performance as that of CPH because of the high capacity of the AP-client links. As opposed to the backhaul link whose utilization is around 90%, the client-AP capacity is sufficient to serve all of the clients without resulting in long queues at the AP. In this scenario, the AP-client perceived link capacity is around 5-38 Mbps per client. Note that the perceived capacity depends on the activities of all of the clients since there might be some time intervals where the clients are in the OFF-state in the well known ON-OFF cycle of the DASH [2]. Under high AP-client link capacity, the airtime allocation affects the performance only marginally, as confirmed by almost no performance difference between CPH and CPH-EQ for low N. However, for larger N, as Fig. 5b shows, CPH maintains

[8]Our simulations of a scenario with Zipf exponent 0.7, Γ_{bh}=50 Mbps, V=100 show that trends are largely in agreement with the ones reported here.



(a) Stalling ratio. (b) Video quality in Kbps. (c) Cache bit hit ratio.

(d) Initial latency in seconds. (e) Instability. (f) Backhaul utilization.

Fig. 6. Impact of increasing backhaul capacity under $N = 10$, synthetic video data, 10 videos.

slightly lower stalling ratio than CPH-EQ. Moreover, although BUFF achieves the lowest buffer stalls, it comes at the expense of lower video quality compared to CPH variants and lower cache hits compared to CLIENT-CACHE.

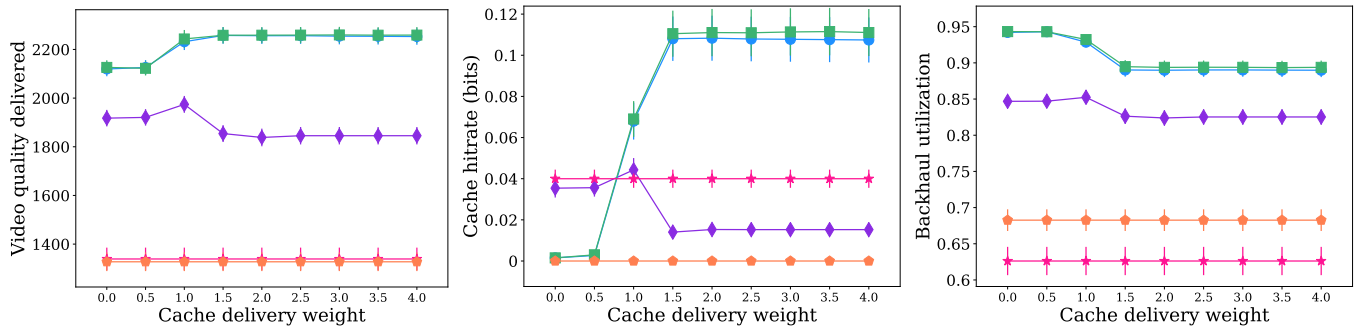
Despite enabling cache delivery in CLIENT-CACHE, this capability without quality assignment does not suffice to improve client performance as we observe high stalling ratio in Fig. 5b. With increasing N, CLIENT-CACHE starts to find content in the cache and therefore we observe a slight decrease in the stalling ratio in Fig. 5b. In summary, for a large number of DASH clients, CPH achieves a significantly higher cache hit ratio as seen in Fig.5c while simultaneously providing the highest video bitrate and keeping the buffer stalls very close to that of the BUFF.

Please note that our schemes may suffer from the same problems as the client-driven approaches. The problems arise since the bitrate associated with a quality level is only an average value, which might differ from the actual chunk size. For example, the actual chunk size might be much larger than the one calculated using the denoted bitrate which then requires a longer time to download from the backhaul and to transmit to the client. Also, our proposals might suffer from the sub-optimal decisions at the client's quality selection scheme as the AP considers the requested rate and deviate from it only within the limits of the tolerated difference. Another option is to completely overwrite the client's requests which, however, conflicts with the client-driven nature of DASH.

Impact of the backhaul capacity: To analyze the impact of the backhaul capacity Γ_{bh}, we fix the number of clients to 10. As Fig. 6 shows, for all backhaul capacity settings, CPH and CPH-EQ outperform the variants of CLIENT and BUFF in most of the performance metrics. For example, when backhaul capacity is sufficient ($\Gamma_{bh} > 20$ Mbps), the stalling ratio is zero for all schemes in Fig. 6a. Nonetheless, the video bitrates are lower for CLIENT variants and BUFF (Fig. 6b).

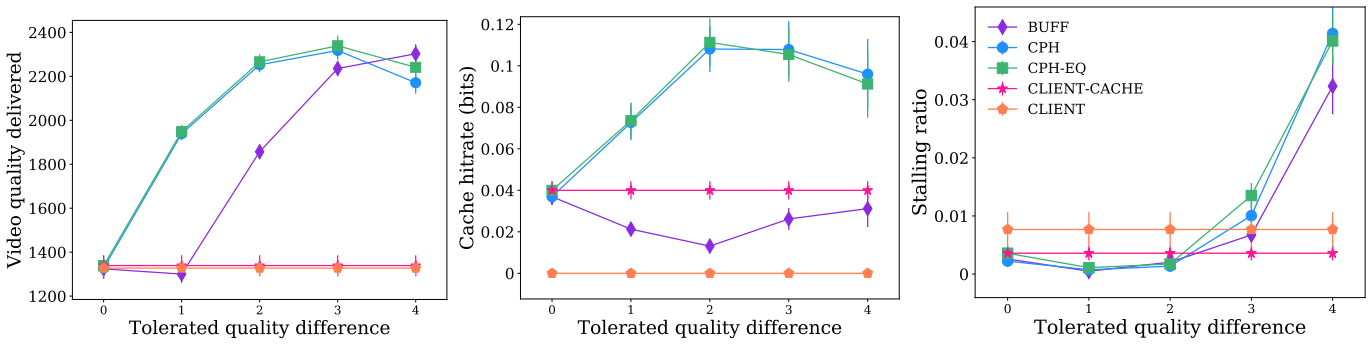
Comparing BUFF and CPH, we can argue that BUFF is a better choice when $\Gamma_{bh} = 5$ Mbps as it sustains a lower stalling ratio compared to CPH and CPH-EQ. In all other cases, CPH and CPH-EQ should be the choice not only for leveraging the cached content (Fig. 6c) but also for a shorter initial latency (Fig. 6d). However, we observe in Fig.6e that this performance improvement comes at the expense of higher instability [13] due to more frequent quality changes to fully utilize the existing cached copies and the available backhaul capacity. This effect is more visible when the backhaul is a bottleneck, e.g., $\Gamma_{bh} = 5$ Mbps. With increasing backhaul capacity, the instability of the CPH variants approach to that of CLIENT variants as the AP does not need to overwrite the client requests as often. However, we still observe a higher instability for CPH variants which motivates the need for including instability as a constraint in the decision logic of CPH. Finally, we report backhaul utilization ratio in Fig. 6f which shows that the CPH variants and BUFF could utilize the backhaul capacity better while CLIENT variants leave the capacity underutilized presumably due to an inaccurate estimation of the link capacity.

Impact of the cache delivery weight: Fig. 7 shows the impact of increasing cache delivery weight μ_c when Γ_{bh}=20 Mbps. CPH variants benefit from increasing μ_c from 1 to 1.5 slightly in terms of higher video quality and significantly regarding



(a) Video quality in Kbps. (b) Cache bit hit ratio. (c) Backhaul utilization.

Fig. 7. Impact of cache weight μ_c, synthetic video files.



(a) Video quality in Kbps. (b) Cache bit hit ratio. (c) Stalling ratio.

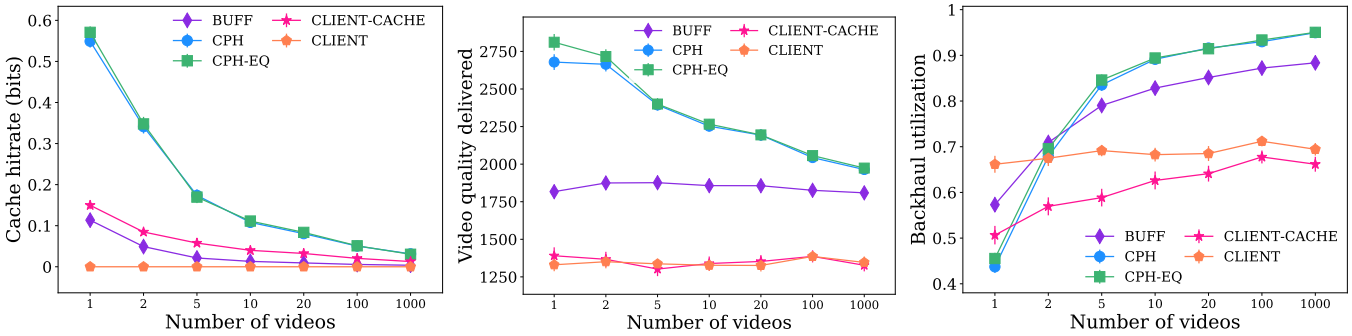
Fig. 8. Impact of tolerated quality difference, 10 synthetic traces, 20 Mbps backhaul capacity.

cache hits as shown in Fig. 7a and Fig. 7b. The performance becomes stable after $\mu_c > 1.5$. Similarly, Fig.7c shows that increasing μ_c decreases the backhaul utilization first but stabilizes afterward. In this setting with 20 Mbps backhaul capacity, almost all schemes sustain a smooth streaming session without a buffer stall. The best setting for μ_c is 1.5 for this scenario as the cache hit ratio increases to approximately 12% for CPH variants from 7% when $\mu_c = 1$.

Impact of the tolerated quality difference: Fig. 8 shows the impact of the tolerated quality difference ($\Delta\mathcal{E}$). Note that $\Delta\mathcal{E} = 0$ corresponds to the case where AP does not overwrite the client's decision. In other words, the AP serves the client only with the requested quality, which is naturally fetched from the cache upon availability. For $\Delta\mathcal{E} = 0$, our proposed schemes offer benefits compared to CLIENT in terms of cache hits (Fig.8b) in case of a relatively high backhaul capacity as in this case. However, CLIENT-CACHE suffices to realize the mentioned performance improvement when $\Delta\mathcal{E} = 0$ as observed in Fig.8b. When $\Delta\mathcal{E}$ is 1 or 2, CPH variants offers an improvement in video quality in Fig.8a and in cache hits in Fig. 8b. Although stall ratio is already low around 1% for all schemes (Fig.8c), CPH variants can nevertheless provide some benefits with respect to buffer stalls. Another observation is a visible increase in stall ratio for network-assisted schemes when $\Delta\mathcal{E} = 4$ in Fig.8c. Hence, $\Delta\mathcal{E}$ should be set to 1 or 2 to ensure that CPH variants can still achieve an improvement in cache hits and video quality. Meanwhile, increasing $\Delta\mathcal{E}$ for BUFF leads to higher video quality but lower cache hits.

Impact of number of videos: Fig. 9 shows the change of performance with increasing number of video contents. If the traffic is only for one video, our schemes, which exploit the cache, provide a significant improvement in the cache hits (Fig. 9a). More precisely, when $V = 1$, almost 57% of the bits are served from the cache under CPH and CPH-EQ, compared to 15% under CLIENT-CACHE and 11% under BUFF. Delivering from the cache results in the client's rate adaptation algorithm to request a higher quality video. As a result, compared to BUFF and CLIENT variants, CPH and CPH-EQ offer better quality, which can be observed in Fig. 9b. With more diverse contents or diverse interests of the users (e.g., a lower Zipf exponent such as 0.7), it becomes less likely for the AP to find the content in the edge cache. Consequently, the benefits offered by our proposals are expected to diminish. For example, cache hit ratio decreases significantly when there are 1000 contents, as observed in Fig. 9a. For a larger video catalog in the order of thousands (e.g., the movie catalog of NetFlix[9]), we expect that the video providers or network providers implement popularity-based caching or pre-caching to increase the chances of cache hits and to exploit the cached content for higher user satisfaction. Despite the diminishing benefits, our proposed schemes can still offer some performance improvements over CLIENT in terms of video bitrates as shown in Fig. 9b and better utilization of the backhaul link as shown in Fig. 9c. In this setting, all schemes maintain almost a smooth playout without buffer stalls. However, for lower backhaul capacities, e.g., $\Gamma_{bh} = 8$

[9]https://www.statista.com/statistics/563381/netflix-available-movies-by-country-in-europe/



(a) Cache bit hit ratio. (b) Video quality in Kbps. (c) Backhaul utilization.

Fig. 9. Impact of number of videos, synthetic video data, $\Gamma_{bh} = 20$ Mbps.

Mbps, we also observe improvement in stall ratio. This performance improvement is due to the AP's resource management approach that takes the clients' states, e.g., buffer levels, into account. Comparing BUFF and CPH variants, we observe the same trend as in the earlier scenarios: BUFF suffers from lower cache hits and video quality, but at the same time offers few buffer stalls.

C. Discussion on a Practical System

Now, we discuss briefly the implementation issues of EdgeDASH. An AP can calculate the utilities in (5) using the following parameters: i) available bitrates of the requested video, ii) current client buffer level, iii) expected buffer level of a client after the delivery of the current chunk, and iv) minimum target buffer level. The AP and clients can use the following SAND messages encapsulated in HTTP header to convey this information: *AcceptedAlternatives*, *DeliveredAlternative*, and *ClientQoS* [3]. Using AcceptedAlternatives, the client can notify the AP about the other quality levels it will accept. Using DeliveredAlternative, the AP can notify the client if it delivers an alternative rather than the requested quality. Using ClientQoS, the client can inform the AP about its buffer level. Using DeliveryBoostRequest, the client can request the network (DANE) to assist it and ask for buffer boost. Finally, DANE can respond with a DeliveryBoostResponse as a response to DeliveryBoostRequest.

A clear limitation of EdgeDASH is that it is applicable only to HTTP traffic in which an AP can extract the contents of a video request message for the operation of EdgeDASH. Given that an increasing fraction of Internet traffic is encrypted, it is important to design solutions that can work for encrypted content, as appears in [23] and [24] as examples. However, we leave this aspect to a future work since network assistance for encrypted content requires completely a new approach.

Another possible direction is the design of cache management policies; a content-aware cache admission and replacement scheme can exploit the popularity of chunks and also use the information from clients, e.g., AnticipatedRequests message type defined in SAND [3] which lets a DASH client to inform a DANE about the chunks the client might request next. Since the CPs such as YouTube collect user statistics, they can predict the popularity of each chunk and quality level. While such statistics are not publicly available, CPs would also benefit from sharing this information with network-assistance elements. Therefore, popularity values can be fed from the CP to the network providers.

IX. Conclusions & Future Work

Since video streaming is a dominant traffic accounting for a big share of network load, it is paramount to provide solutions to improve the performance of video traffic as well as to leverage some state-of-the-art approaches for decreasing the burden on the network. With this goal in mind, we propose network-side quality and resource assignment solutions running at a WiFi AP. Our proposal takes advantage of the cached contents to both decrease the congestion in a bottleneck link and to improve video streaming performance, e.g., by offering higher video bitrate or lower buffer stalls. Our simulations show the following: by a moderate adaptation of the clients' quality requests (e.g., offering a few quality levels higher or lower than the requested), a WiFi AP can improve cache hits while decreasing buffer stalls and increasing video bitrates. Moreover, an AP can allocate its downlink airtime considering the statistics of video clients, e.g., buffer levels. As future work, we plan to evaluate our schemes on a prototype using the state-of-the-art client driven DASH players to understand better the behavior of EdgeDASH in interaction with TCP and changes in the wireless channel.

References

[1] A. Mehrabi, M. Siekkinen, and A. Ylä-Jääski, "Cache-aware QoE-traffic optimization in mobile edge assisted adaptive video streaming," IEEE Transactions on Networking, 2018.

[2] A. Bentaleb et al., "A survey on bitrate adaptation schemes for streaming media over HTTP," IEEE Communications Surveys & Tutorials, vol. 21, no. 1, pp. 562–585, 2018.

[3] DASH Industry Forum, "Guidelines for implementation: DASH-IF interoperability points, version 4.2," April 09 2018.

[4] Y. Jin, Y. Wen, and C. Westphal, "Optimal transcoding and caching for adaptive streaming in media cloud: An analytical approach," IEEE Transactions on Circuits and Systems for Video Technology, vol. 25, no. 12, pp. 1914–1925, 2015.

[5] S. Petrangeli, J. V. D. Hooft, T. Wauters, and F. D. Turck, "Quality of experience-centric management of adaptive video streaming services: Status and challenges," ACM Trans. Multimedia Comput. Commun. Appl., vol. 14, no. 2s, pp. 31:1–31:29, May 2018.

[6] V. Nathan et al., "End-to-end Transport for Video QoE Fairness," ser. ACM SIGCOMM'19. ACM, 2019, pp. 408–423.

[7] A. Bentaleb, A. C. Begen, and R. Zimmermann, "SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking," in ACM on Multimedia Conference. ACM, 2016, pp. 1296–1305.

[8] J. W. Kleinrouweler, S. Cabrero, and P. Cesar, "Delivering stable high-quality video: An sdn architecture with DASH assisting network elements," in Proceedings of the 7th International Conference on Multimedia Systems. ACM, 2016, p. 4.

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

[10] A. Mehrabi, M. Siekkinen, and A. Ylä-Jääski, "Edge computing assisted adaptive mobile video streaming," IEEE Transactions on Mobile Computing, vol. 18, no. 4, pp. 787–800, 2018.

[11] L. Liikkanen and A. Salovaara, " Music on YouTube: user engagement with traditional, user-appropriated and derivative videos," Computers in Human Behavior, vol. 50, pp. 108–124, 2015.

[12] Y. Chen, K. Wu, and Q. Zhang, "From QoS to QoE: A tutorial on video quality assessment," IEEE Communications Surveys & Tutorials, vol. 17, no. 2, pp. 1126–1165, 2015.

[13] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE," IEEE/ACM Transactions on Networking (TON), vol. 22, no. 1, pp. 326–340, 2014.

[14] E. Thomas et al., "Applications and deployments of server and network assisted DASH (SAND)," 2016.

[15] A. Galanopoulos and A. Argyriou, "Adaptive video streaming over LTE unlicensed," arXiv preprint arXiv:1608.00239, 2016.

[16] "Universal mobile telecommunications system (umts); lte; study on server and network-assisted dynamic adaptive streaming over http (dash) (sand) for 3gpp multimedia services (3gpp tr 26.957 version 14.1.0 release 14)," Tech. Rep., 2017.

[17] J. W. Kleinrouweler, B. Meixner, and P. Cesar, "Improving video quality in crowded networks using a DANE," in 27th Workshop on Network and Operating Systems Support for Digital Audio and Video, ser. NOSSDAV'17, 2017, pp. 73–78.

[18] A. H. Zahran, J. J. Quinlan, K. K. Ramakrishnan, and C. J. Sreenan, "SAP: Stall-Aware Pacing for Improved DASH Video Experience in Cellular Networks," in Proceedings of the 8th ACM on Multimedia Systems Conference, ser. MMSys'17, 2017, pp. 13–26.

[19] H. Ma, J. Hao, and R. Zimmermann, "Access point centric scheduling for dash streaming in multirate 802.11 wireless network," in 2014 IEEE International Conference on Multimedia and Expo (ICME), July 2014, pp. 1–6.

[20] E. Khorov, A. Krasilov, M. Liubogoshchev, and S. Tang, "SEBRA:SAND-enabled bitrate and resource allocation algorithm for network-assisted video streaming," in 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). IEEE, 2017, pp. 1–8.

[21] D. H. Lee, C. Dovrolis, and A. C. Begen, "Caching in HTTP adaptive streaming: Friend or foe?" in Proceedings of Network and Operating System Support on Digital Audio and Video Workshop. ACM, 2014, p. 31.

[22] C. Li et al., "Qoe-driven mobile edge caching placement for adaptive video streaming," IEEE Transactions on Multimedia, vol. 20, no. 4, pp. 965–984, April 2018.

[23] A. Araldo, G. Dán, and D. Rossi, "Caching encrypted content via stochastic cache partitioning," IEEE/ACM Transactions on Networking, vol. 26, no. 1, pp. 548–561, 2018.

[24] C. Gutterman et al., "Requet: real-time qoe detection for encrypted youtube traffic," in Proceedings of the 10th ACM Multimedia Systems Conference. ACM, 2019, pp. 48–59.

[25] J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Datasets for AVC (H. 264) and HEVC (H. 265) evaluation of dynamic adaptive streaming over HTTP (DASH)," in Proceedings of the 7th International Conference on Multimedia Systems. ACM, 2016, p. 51.

[26] D. Pisinger, "A minimal algorithm for the multiple-choice knapsack problem," European Journal of Operational Research, vol. 83, no. 2, pp. 394–410, 1995.

[27] H. Shojaei, T. Basten, M. Geilen, and A. Davoodi, "A fast and scalable multidimensional multiple-choice knapsack heuristic," ACM Trans. Des. Autom. Electron. Syst., vol. 18, no. 4, pp. 51:1–51:32, Oct. 2013.

[28] C. Tornevik, J.-E. Berg, F. Lotse, and M. Madfors, "Propagation models, cell planning and channel allocation for indoor applications of cellular systems," in IEEE 43rd Vehicular Technology Conference. IEEE, 1993, pp. 867–870.

[29] T. Mangla et al., "Video through a crystal ball: Effect of bandwidth prediction quality on adaptive streaming in mobile environments," in Proceedings of the 8th International Workshop on Mobile Video. ACM, 2016, p. 1.

[30] Y. Sun et al., "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in Proceedings of the 2016 ACM SIGCOMM Conference. ACM, 2016, pp. 272–285.

Suzan Bayhan is an Assistant Professor at the University of Twente and an Adjunct Professor (Docent) at the University of Helsinki. Suzan earned her Ph.D. in Computer Engineering in 2012 from Bogazici University and worked at the University of Helsinki, Aalto University, and TU Berlin between 2012-2019. Her current research interests include spectrum sharing and coexistence of wireless networks, WiFi and LTE radio resource management, and edge computing.

Setareh Maghsudi is an assistant professor at the University of Tübingen. During 2017-2020, she was an assistant professor at the Technical University of Berlin, where she also received her Ph.D. degree (summa cum laude) in 2015. During 2015-2017, she held post-doctoral positions at the University of Manitoba-Canada, and Yale University-USA. She has received several competitive fellowships, including from the German Research Foundation (DFG) and Japan Society for the Promotion of Science (JSPS). Her main research interests lie in the intersection of network analysis and optimization, game theory, machine learning, and data science.

Anatolij Zubow received the M.Sc. degree in computer science and the Ph.D. degree from the Humboldt University of Berlin in 2004 and 2009, respectively. He has been an Interim (Gast-) Professor of electrical engineering and computer science with the Chair for Telecommunication Networks, Technische Universität Berlin, since October 2018. His research interests are in architectures and protocols of wireless communication networks as well as in protocol engineering with impact on performance and QoS aspects. He is currently focusing mainly on coexistence of heterogeneous wireless technologies in unlicensed spectrum, high-performance IEEE 802.11 networks, software-defined wireless networking, and ultra-reliable low latency communication.