SABR: Network-Assisted Content Distribution for QoE-Driven ABR Video Streaming

DIVYASHRI BHAT, University of Massachusetts Amherst
AMR RIZK, Technische Universität Darmstadt
MICHAEL ZINK, University of Massachusetts Amherst
RALF STEINMETZ, Technische Universität Darmstadt

State-of-the-art software-defined wide area networks (SD-WANs) provide the foundation for flexible and highly resilient networking. In this work, we design, implement, and evaluate a novel architecture (denoted as SABR) that leverages the benefits of software-defined networking (SDN) to provide network-assisted adaptive bitrate streaming. With clients retaining full control of their streaming algorithms, we clearly show that by this network assistance, both the clients and the content providers benefit significantly in terms of quality of experience (QoE) and content origin offloading. SABR utilizes information on available bandwidths per link and network cache contents to guide video streaming clients with the goal of improving the viewer's QoE. In addition, SABR uses SDN capabilities to dynamically program flows to optimize the utilization of content delivery network caches.

Backed by our study of SDN-assisted streaming, we discuss the change in the requirements for network-to-player APIs that enables flexible video streaming. We illustrate the difficulty of the problem and the impact of SDN-assisted streaming on QoE metrics using various well-established player algorithms. We evaluate SABR together with state-of-the-art dynamic adaptive streaming over HTTP (DASH) quality adaptation algorithms through a series of experiments performed on a real-world, SDN-enabled testbed network with minimal modifications to an existing DASH client. In addition, we compare the performance of different caching strategies in combination with SABR. Our trace-based measurements show the substantial improvement in cache hit rates and QoE metrics in conjunction with SABR indicating a rich design space for jointly optimized SDN-assisted caching architectures for adaptive bitrate video streaming applications.

CCS Concepts: • **Information systems → Multimedia streaming**;

Additional Key Words and Phrases: SDN, OpenFlow, QoE, ABR streaming, DASH, network-assisted streaming, video quality metrics, caching

ACM Reference format:
Divyashri Bhat, Amr Rizk, Michael Zink, and Ralf Steinmetz. 2018. SABR: Network-Assisted Content Distribution for QoE-Driven ABR Video Streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* 14, 2s, Article 32 (April 2018), 25 pages.
https://doi.org/10.1145/3183516

This work was funded in part by NSF grant 1419199, by the DFG as part of the Collaborative Research Center 1053 MAKI (TP B4, C3), and the IREP Program.
Authors' addresses: D. Bhat and M. Zink, Department of Electrical and Computer Engineering, University of Massachusetts Amherst, 151 Holdsworth Way, Amherst, MA 01003, USA; A. Rizk and R. Steinmetz, Technische Universität Darmstadt, Rundeturmstr 10, 64293 Darmstadt, Germany.

https://doi.org/10.1145/3183516

1 INTRODUCTION

The software-defined networking (SDN) paradigm has transformed the way networks are controlled and managed today. For example, the emergence of software-defined wide area network (SD-WAN) technology, such as Google's B4 approach [26], has vastly improved network utilization. The separation of control and data planes allows a much more fine-grain control of network traffic than in the case of traditional networks. For example, B4 utilizes SDN features such as traffic engineering and prioritization to achieve utilizations beyond 95% as compared to 40% to 50% for traditional networking approaches.

Recent approaches apply the software-defined paradigm to resources that include computation and storage in addition to networks, which are coined as software-defined infrastructure (SDI) [44, 46]. In this article, we present an SDI architecture that supports adaptive bitrate (ABR) video streaming. This work augments the traditional operation of content delivery networks (CDNs) by harnessing the capabilities of the SDI infrastructure. This focus on ABR video streaming is driven by the fact that video-on-demand (VoD) is the killer application in today's Internet. According to the latest Sandvine report [3], 71% of the downstream Internet traffic at peak hours in North America is real-time entertainment such as live streaming and VoD. It is forecast that this will increase to 80% by 2020. Such high demand for video content requires approaches not only to efficiently transport the data but also to manage the delivery network from the content providers to the customers. Video streaming at today's scale would be unthinkable without significant infrastructure and services provided by CDNs [8, 35]. In this work, we provide an architecture that uses SDI to efficiently manage CDN networks and improve the quality of experience (QoE). We believe that this system fits nicely into a framework where content delivery is part of the ISP's business model. Existing examples of such systems include approaches of content providers such as Netflix [4] to install servers inside ISP data centers to enhance video QoE.

One characteristic that distinguishes the widespread ABR video streaming systems from non-ABR systems is that for ABR, CDN caches may not contain all quality versions of a video. Thus, providing clients with information such as the presence of qualities of desired segments in particular caches allows the client to make a well-informed decision on segment retrieval. Additionally, providing bottleneck bandwidth information on the paths between the client and the caches that currently host the sought segments not only aids the client's decision but also eliminates the client's need to use less accurate bandwidth estimation methods, such as end-to-end probing or application layer rate estimation. This approach is denoted as network-assisted ABR streaming.

Although the preceding described functionality may be partially provided in traditional, non-SDN networks, as shown, for example, in Ganjam et al. [22], our approach, SDN-assisted ABR (denoted as SABR), requires only minimal modification at the streaming client and reduces the load on the streaming guidance system by providing necessary information to the clients but not participating directly in the video retrieval decision made by the player. This is crucial, as we require the clients to retain full control of their streaming algorithms for scaling and stability reasons. Hence, we design SABR with a graceful interruption property enabling clients to ignore network assistance at any time.

Since SABR keeps track of the content stored in its domain's caches and is able to redirect clients to any of these caches, the traditional approach of autonomous content admission and least recently used (LRU) eviction might not be the best strategy. We study several cache admission and eviction strategies that we tailor to the overall SABR architecture. These strategies take into account that (i) video popularities follow a heavy tail distribution including many "one-hit wonders"[1] [35], (ii) more than 36% of videos are not completely watched according to recent video

[1]A video is deemed a one-hit wonder if it is requested only once.

session traces of a CDN with global footprint [30], and (iii) certain video qualities are more popular than others.

From a performance perspective, the streaming assistance system prevents significant video quality drops described, such as in Adhikari et al. [8], by providing alternative sources for different video qualities. In addition to better QoE, SABR optimizes the caching architecture to improve CDN metrics such as server offloading (i.e., the percentage of client requests serviced by the caching network) and the midgress (i.e., the intracache network traffic).

In this work, we use dynamic adaptive streaming over HTTP (DASH) [47], as it is a popular, open standard for ABR streaming and quite close to other ABR approaches such as Apple's HLS [2], Microsoft's Silverlight [5], and Adobe's HDS [1]. Although we demonstrate the functionality and benefits of our approach for DASH, we believe that it can be easily extended to other ABR streaming approaches. In this article, we make the following contributions:

- ***Architecture.*** We design an SDN-assisted control plane architecture to support and improve ABR video streaming in CDNs.
- ***Formalization.*** We present a formalization of the ABR streaming problem to show the origin of the benefits of our architecture.
- ***Implementation.*** We implement the proposed architecture and analyze its performance. This implementation includes (i) and SDN measurement service and archive that is used to monitor network paths; (ii) SABR, a module that aggregates monitoring information and communicates with the SDN controller and the CDN caches; (iii) a minimally modified DASH client implementing various streaming algorithms, which makes well-informed decisions on segment retrieval based on the communication with SABR; (iv) a modified SDN controller that is used to dynamically install paths to chosen caches; and (v) content placement strategies that demonstrate the impact of CDN caching using SABR.
- ***Analysis.*** We show an extensive analysis of the SDN-assisted streaming system through a series of experiments conducted in the CloudLab testbed [45]. Our results show that SDN monitoring provides better bandwidth estimates than a purely client-based estimation throughout the entire duration of the session. We show that our system significantly improves the QoE (e.g., the overall video quality bitrate at the client), reduces the server load ratio, and provides higher network utilization.
- ***Caching strategies.*** We investigate the impact of various content placement and caching strategies using SABR and analyze their performance with different state-of-the-art ABR streaming algorithms at the clients. Evaluation results from experiments performed in the CloudLab testbed reveal that a time-to-live (TTL)-based, cooperative caching approach results in significantly better QoE compared to the LRU approach for similar average cache sizes.

An earlier version of this article was published at MMSys 2017 [13]. This article extends our earlier publication by the following: (i) extensive analysis of caching algorithms, in which we use different types of caching algorithms such as TTL-based caching, and popularity- and quality-based caching differentiation, and (ii) real-world session abandonment, in which we parameterize the user streaming behavior (i.e., the video session length using real-world statistics).

The rest of this article is structured as follows. Section 2 describes our monitoring and control framework. In Section 3, we provide the details of the REST service application and the aggregation and processing of the network monitoring information. The SDN-assisted streaming client details are given in Section 4. Caching algorithms used in combination with SABR are described in Section 5. We describe the evaluation environment and the extensive analysis in Sections 6 and 7, respectively, before providing a discussion of the results in Section 8. We summarize the related work in Section 9 and provide our conclusion and thoughts on future work in Section 10.

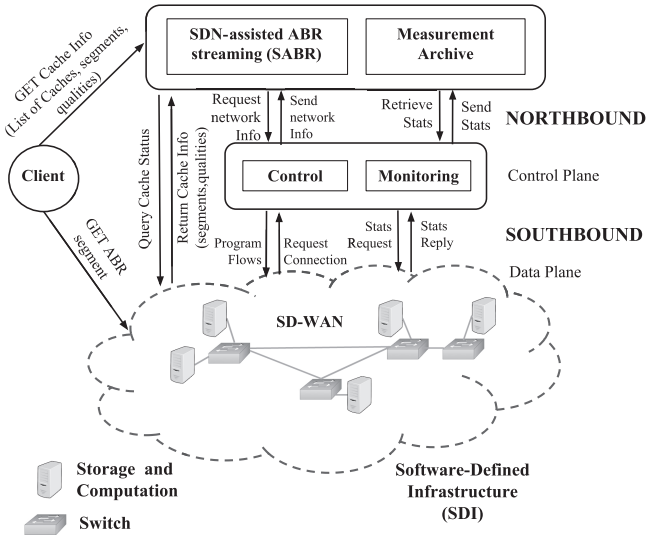


Fig. 1. SABR architecture. Details of the Southbound interface, Northbound interface, and client implementation are given in Sections 2 through 4, respectively.

2 SOUTHBOUND: A MONITORING AND CONTROL FRAMEWORK

In this section, we describe the design of an SDI that supports network-assisted ABR streaming through monitored network information. In particular, we present an OpenFlow (OF) [37] Southbound API that is used to orchestrate an SDN-assisted CDN for ABR streaming. Our implementation is based on the work by Adrichem et al. [50]. We use OF since it is currently the most popular instantiation of SDN.

2.1 Software-Defined Infrastructure

Next, we use the term *SDI* to denote a network of software defined switches that are co-located with storage and compute power. In the case of a CDN, this storage and compute power can be used for caching. SDIs simplify third-party policy implementation and allow network administrators to provision, monitor, and efficiently control virtual networks, computation, and storage. An example of a fully operational network that offers SDI capabilities are the GENI [12] and CloudLab [45] testbeds. We use the latter for evaluation of SABR and describe this in detail in Section 6.1.

Figure 1 shows the design of an SDI infrastructure to provide application services for the most popular ABR streaming instantiation (i.e., DASH). For a detailed description of DASH, we refer the interested reader to Sodagar [47]. Figure 1 depicts the underlying SDI of different autonomous systems and a control plane that is tasked with flow programming and monitoring. Details on the Northbound interface and the client are given in Sections 3 and 4, respectively.

2.2 Monitoring Infrastructure

As depicted in Figure 1, the monitoring module is logically separated within the controller. Its task is to decide on the statistics to be monitored, as well as the corresponding sampling times. This provides the flexibility to deploy tailored monitoring algorithms that extract only the information needed for a given application. We believe that the performance gain of our passive system outweighs the potential benefits of an active one such as that in Mushtaq et al. [40]. Currently, the monitoring system only queries switches that belong to a network path between a client and a

cache/server. On these switches, only the ports that are part of these paths are monitored, which minimizes the monitoring overhead.

Given the link capacity information, the system collects the bytes transferred per port over fixed intervals to determine the available bandwidth along the path, as well as the bottleneck link. This information is used by the Northbound interface (see Section 3) to assist ABR streaming applications.

We logically separate the monitoring infrastructure from the controller infrastructure such that they can be deployed on different machines. However, for optimal performance that avoids unnecessary network latency, we recommend that the monitoring infrastructure be co-located with the controller since decisions on video delivery are made based on real-time traffic. We detail the archival and processing of the monitoring information in the following section.

3 NORTHBOUND: A REST API SERVICE AND ARCHIVE

In this section, we present the Northbound interface that provides information such as available caches, the bottleneck bandwidth to each cache, and an indication of cache content, which can all be retrieved through standard REST APIs, to the client. Figure 1 shows the components of this system where the preceding information is sourced at an SDI as described in Section 2.1. Next, we describe the system components that use the Northbound interface.

3.1 Measurement Archive

We briefly describe the measurement archival and processing module depicted in Figure 1. OF is used to implement a feedback-based measurement and control system for an ABR video CDN. The monitoring module from Section 2.2 is connected to a distributed database system powered by MongoDB [6], which enables redundancy and scalability through fast and easy replication. MongoDB also provides REST APIs for database transactions [7], enabling information insertion using simple HTTP commands.

3.2 SDN-Assisted ABR Streaming: SABR

In the following, we describe details of the SABR module. First, we show how SABR uses the information collected by the monitoring system to select the best cache in the CDN, such as by calculating the available bandwidth from the client to the respective caches. Then, we show how SABR provides the DASH client with monitoring information through a REST API. This information includes available bandwidth estimates and cache occupancy. This could easily be extended to include further switch information, such as queue lengths or flow table update statistics, if necessary. Finally, we use dynamic SDN routing to provide paths between clients and desired caches. Dynamic path computation and selection in an SDN network is a widely investigated optimization problem [9]. Here, we consider the specific problem of how to efficiently manage a CDN with the support of the SDN control plane. The bottleneck bandwidth is deduced from the monitored traffic information and the known link capacities. In addition to accurate traffic monitoring and prediction on the streaming time scale (Section 2.2), the SDN infrastructure allows to control the routes from clients to caches to prevent available bandwidth-driven oscillations between multiple caches.

Since monitoring only gives us an observation of the past, we require predictions of the future bottleneck bandwidths to help the clients determine the quality of the segments that will be retrieved next. For predicting the available bandwidth in the short-term future at each port along different paths between clients and caches, we consider the well-established autoregressive integrated moving average (ARIMA) time series model. In the following, we denote the monitored value of transferred bytes over one port i at time slot $[t, t + \delta_i)$ as instantaneous rate $r_i(t)$. The

available bandwidth is estimated based on the known link capacity and an estimate of the contending traffic.

ARIMA. The archival and processing module takes for every port i the instantaneous rates $r_i(t)$ over a history window W of time slots and provides a forecast of the utilization over the duration of N^* segments using the ARIMA time series forecasting method, which has three components: an autoregressive component of order a, a differencing component with parameter d, and a moving average component of order v. In a nutshell, the differencing component removes trends by differencing d times, the autoregressive component contributes to a linear regression over the last a observed values, and the moving average components may be understood as a linear regression over the last v noise terms. We decide on the ARIMA(a,d,v) parameterization using the Akaike information criterion [16], which is affine to parsimonious models that are more likely[2] to have produced the observations. Our implementation makes use of the ARIMA routines within the `R forecast` [43] library.

3.3 Caching

SABR's characteristic to provide alternative sources for video segments (instead of the client always being directed to the closest cache) opens new avenues for content placement and cache eviction strategies. With respect to content placement, we differentiate between cooperative and noncooperative caching. Since SABR can redirect a client's request to any cache in the system, content does not necessarily have to be cached at the closest cache. For the proposed SABR architecture, we distinguish between two approaches. In the first case, a cache miss leads to caching the missing segment at all caches in the system, whereas in the other case, the content is pushed to the cache that is nearest to the requesting user given that the object is not stored at any other cache in the system.

For cache eviction strategies, we evaluate the standard LRU policy and the TTL policy. We decided to make use of the TTL eviction policy, as it allows the differentiation of segments and their qualities in a more fine-granular manner. Details on the parameterization of the cache algorithms and on the handling of the interaction of caches and SABR are given in Section 5.

4 SDN-ASSISTED STREAMING CLIENT

Next, we describe an SABR streaming application utilizing the architecture from Sections 2 and 3. First, we formalize the video quality adaptation problem before reviewing basic classes of established adaptation algorithms. Finally, we describe our modifications to the ABR quality adaptation algorithms to make use of the network assistance.

4.1 The Quality Adaptation Problem

Here, we consider a graph $G = (V, E)$ that abstracts a given network topology, where V is a set of vertices (i.e., network nodes) and $E \subseteq V \times V$ is a set of links between the nodes. Each link is associated with a capacity $C(i, j)$, where i, j are the indexes of the vertices spanning the link. The nodes are divided into three types: clients, caches, and switches. For the sake of brevity, we consider a simplistic example of an ABR streaming scenario of only one video that is divided into N segments where each segment is available in K qualities (i.e., bitrate levels). Each segment carries l seconds of video, whereas the nth segment of the kth quality has the size $X_{n,k}$ in bits. We will use k_n to denote the quality level of segment n and drop the subscript when obvious. This description can easily be expanded to include an arbitrary set of available videos in the network. We assume

[2]Using a maximum likelihood estimator.

established routing such that there exists at least one path (i.e., a set of links in E) between every client and every cache. Further, we assume that the network carries other traffic, and hence $C(i, j)$ denotes the available bandwidth on the link between vertices i and j. We assume that the available bandwidth is slowly varying with respect to the monitoring frequency of the OF switch.

Given established routing, the available bandwidth along each path between a client i and cache j is described by

$$R(i, j) = \min_{(\kappa,\iota)\in S(i,j)} C(\kappa, \iota), \tag{1}$$

where $S(i, j)$ is the set of all links belonging to the path between the nodes i and j. Given the ABR streaming application, the time needed by client i to fetch the nth video segment of quality-level k from cache j is given by

$$T_{n,k} = \frac{X_{n,k}}{R_{n,k}(i, j)}, \tag{2}$$

where $R_{n,k}(i, j)$ denotes the available bandwidth during the download time of segment $X_{n,k}$.

4.2 SDN-Assisted Quality Adaptation

SABR provides clients with accurate in-network available bandwidth information and, respectively, ARIMA-based predictions for $R_{n,k}(i, j)$. In addition, providing caching and available bandwidth information to the ABR streaming application through the Northbound interface gives the client the opportunity to minimize the fetch times $T_{n,k}$ as

$$T_{n,k} = \min_{j\in G(n,k)} \frac{X_{n,k}}{R_{n,k}(i, j)}, \tag{3}$$

where $G(n, k)$ is the set of caches that possess the segment n in quality k. This corresponds to always choosing the cache with the highest available bandwidth. Intuitively, our approach draws its strength from the statistical multiplexing gain of combining network path information to independent caches such that it is less likely that worst-case conditions occur on all paths at the same time. Hence, the gain is stronger the more disjoint links the paths possess.

Next, we illustrate different classes of quality adaptation algorithms showing the benefits of including SDN assistance information. Note that the modifications due to SDN assistance are orthogonal to most of the adaptation algorithms known to us such that we generally expect a performance gain, in terms of the average quality bitrate, across them.

The basic quality adaptation problem can be formalized as finding the set of segments in given qualities $\{k_1, \ldots, k_N\}$ that maximize the average bitrate $\frac{1}{N}\sum_{i=1}^{N} k_i$ subject to $B(n) > 0$ for $n \in \{1, \ldots, N\}$, where $B(n)$ is the playout buffer filling after fetching segment n, where buffer fill is measured in seconds. Stricter versions of the quality adaptation problem aim to also minimize additional QoE metrics, such as the quality variations. The quality adaptation problem is hard since clients have only smeared estimates of the available bandwidth $R_{n,k}(i, j)$ for a history of segments $\{k_1, \ldots, k_n\}$ with little to no information on the actual in-network dynamics [51]. In the following section, we discuss three classes of quality adaptation algorithms that make use of (1) and (2) to optimize QoE before showing how SDN assistance can significantly improve their performance.

Rate-based adaptation algorithms: Rate-based adaptation utilizes estimates $\hat{R}_{n,k}(i, j)$ that are obtained at the client to determine the quality of the next segment to be fetched. Let us consider a basic algorithm that greedily downloads the next segment at the highest sustainable quality (i.e., the bitrate) that is just lower than the download rate of the previous segment. If the buffer is full, the client idles until the end of the currently played segment. This algorithm is presented in a slightly modified fashion as the `VLC` algorithm in Müller and Timmerer [39]. Rate-based adaptation benefits from SABR, as it receives much more accurate available bandwidth information (i.e., the

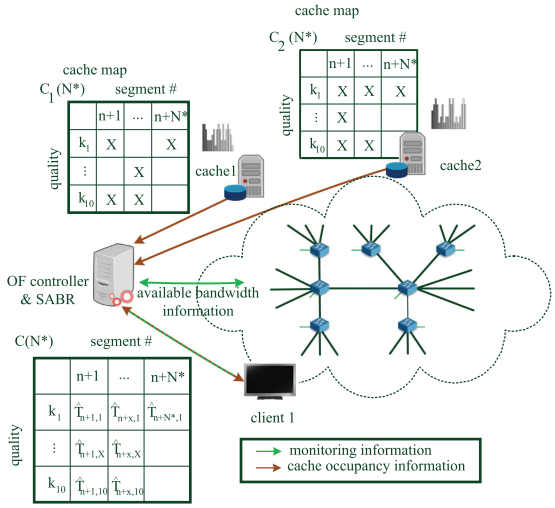


Fig. 2. SDN-assisted quality adaptation by providing the client with available bandwidth and cache occupancy information.

ARIMA estimates of $R_{n,k}(i,j)$). These estimates outperform the empirically obtained application layer estimates at the client.

Buffer-based adaptation algorithms: Buffer-based adaptation algorithms solely use the buffer filling $B(n)$ when deciding on the next segment quality k_{n+1}. Examples for this class of algorithms include Huang et al. [25] and Sodagar [47], where the buffer space is sliced into zones that correspond to different quality adaptation behavior. SABR improves the performance of such algorithms by providing more accurate estimates for (2)—that is, the fetch time of one segment as the buffer naturally drains due to playback during this time.

Hybrid adaptation algorithms: Hybrid adaptation algorithms take both rate and buffer information into account when deciding on the quality of the next segment (e.g., [51]). Such algorithms do not only benefit from a higher accuracy in (1) and (2) but also utilize (3) to find the cache with highest available bandwidth. In general, all considered classes of quality adaptation algorithms benefit from SABR by obtaining a so-called cache map $C_j(N^*)$ for every cache $j \in G$, which indicates the availability of the next N^* segments in the different quality levels at the corresponding caches. This is schematically depicted in Figure 2. The client combines the cache maps of the different caches into a joint cache map $C(N^*)$, which comprises the minimum estimated fetch times $\hat{T}_{n,k}$ for the next N^* segments in different qualities. Required estimates or lower bounds for the fetch times can be calculated from the combination of the segment sizes $X_{n,k}$ and the provided ARIMA-based available bandwidth estimates for the next segments. Different quality adaptation algorithms may utilize the cache map $C(N^*)$ in various ways—for example, to optimize QoE metrics such as the average quality bitrate or the quality variation while fetching the next N^* segments. Note that SABR provides the clients with additional information, namely $C_j(N^*)$, but it does not control the clients' decision on which quality to fetch, which is entirely autonomous. Hence, the QoE perceived at the client fully depends on how the client makes use of the SABR information. We will further discuss this argument in Section 8. Throughout the rest of this article, we use the term *Baseline* to denote various non-SABR client algorithms that belong to the preceding classes of quality adaptation algorithms.

4.3 The Client Implementation

To best represent a real-world ABR streaming application, we implement our SDN-assisted adaptation algorithm as part of an existing open source Python-based DASH client emulator [27]. Since each client uses HTTP by definition, we decided to let clients use a REST interface provided by MongoDB to minimize the implementation overhead caused by our approach. Overall, the client makes the following requests: (i) the initial HTTP GET request to the server or the nearest cache to retrieve the media presentation description (MPD) file for requested video; (ii) the HTTP GET request to SABR for a list of qualities of next segment(s) and the available bandwidth information to every advertised cache (see Section 3.2); and (iii) the HTTP GET request to the selected cache to retrieve the desired segment. Algorithm 1 details the procedure used by the client to fetch segments.

ALGORITHM 1: The Client - Segment Fetcher

```
function DOWNLOAD SEGMENT(segment_no, list_caches, qual_level)#Where segment_no - segment number to be downloaded, list_caches - list of available caches, qual_level - quality level to download
    for cache in list_caches do
        Min_BW=Maximum Capacity of Link
        bw, qual=get_list_info(cache)
        if bw <= Min_BW and qual_level in qual then
            Min_BW=minimum(Min_BW, bw)
            Cache_IP=cache
    get_segment(Cache_IP)                    #(get_segment - retrieve the next segment from chosen cache)
    Return    #(get_list_info - get list of caches along with the bottleneck bandwidth information and list of available qualities)
```

The client parses these responses to obtain segment sizes, available bandwidth to each cache, and cache occupancy information to feed it as needed to one of the algorithms described in Section 4.2. To evaluate our system, we consider a miniature CDN topology. Details on the evaluation environment and experiment results are given in Sections 6 and 7.

5 CACHING ALGORITHMS WITH SABR

Since SABR provides a cache map $C(N^*)$, which indicates the availability of the next N^* segments in different quality levels at the different caches, we define a systemwide cache miss as the event when the client requests a segment from the server. This event may arise either due to the absence of that particular segment at all caches or due to insufficient bandwidth to all caches. Such cache miss events prompt the caches to request segments according to one of the strategies described in this section. To analyze the performance of different caching algorithms with SABR, we implement the content placement strategies described in Sections 5.3. First, we introduce the cache eviction strategies that the SABR system uses.

5.1 Least Recently Used

The LRU algorithm is one of the main content eviction strategies in caches due to its simplicity and performance. Basically, it describes which elements to replace when given a full fixed-size cache. When a cache miss occurs (i.e., a noncached content is requested), the cache replaces the LRU object in the cache with the new object [42]. The rationale behind LRU is to leverage the temporal locality of the requests. Although LRU is simple to implement and very low in complexity, it does not allow any fine tuning of content eviction, as only the "time of the last request" decides if content is evicted or not. However, LRU caching is not optimized for the sequential nature of segment requests that is typically observed in ABR client sessions.

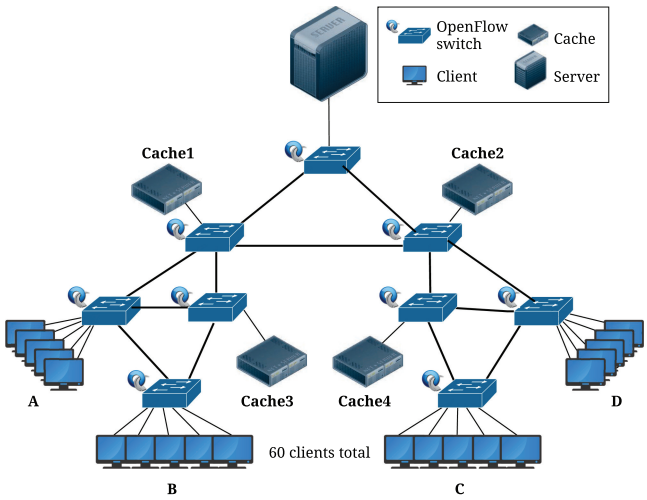


Fig. 3. CloudLab topology used for evaluating SABR. Each client group {A,B,C,D} includes 15 clients.

5.2 Time-to-Live

TTL caches have a timer-based object eviction strategy—that is, requested objects are annotated with the current timestamp and with a calculated eviction time. In the simplest case, the TTL (i.e., the offset between eviction and request time) can be constant for all objects. Analytical relations between TTL caching strategies and capacity-constrained caching strategies such as LRU or FIFO have been established in recent years. Most prominent is the connection between TTL caches and LRU, first discovered in Che et al. [17] for Poisson request processes and generalized later in Martina et al. [36]. Object TTLs can be optimized based on the object popularities to maximize the cache hit rate [19]. Note that TTL caches are not capacity constrained.

5.3 Compound Caching Algorithms in SABR

The following caching algorithms combine either the LRU or TTL algorithm at the individual caches with the global network view of SABR. Next, we differentiate on the one hand between local and global caching and on the other hand between quality-based and popularity-based caching.

ALGORITHM 2: Content Placement Approaches: Local Caching

```
1: function LOCAL CACHING(C_id, map_caches, cl_ip)   #Where c_id - content metadata, map_caches - map of nearest
   caches to clients, cl_ip - Client IP address
2:     s_ip = map_caches[cl_ip]
3:     if C_id not in get_content_list(s_ip) then
4:         Insert content into s_ip
5:     else
6:         Return
                                   #(get_content_list() - retrieves the list of existing content metadata)
```

Local Caching [Algorithm 2] : The SABR system detects a miss by monitoring requests to the server and prompts the cache that is geographically closest to the requesting client to download the relevant segment. In Figure 3, this translates to the following: *Cache1* and *Cache2* are assigned to clients in *GroupA* and *GroupC*, respectively. Similarly, *Cache3* and *Cache4* are assigned to clients

in *GroupB* and *GroupD*. We use this caching strategy in all experiments described in Section 7.1, unless stated otherwise, to demonstrate the advantage of SABR versus a fixed cache allocation for all Baseline algorithms. This caching strategy is denoted as "Local."

Global caching: This approach assumes global knowledge of the cache architecture. This knowledge is provided by SABR through the cache map introduced in Section 4.2. In the first variant, denoted "$\text{Global}_{\text{FR}}$" (FR = Full Replication) [Algorithm 3], when a systemwide cache miss occurs all caches request the missing segment. In a second variant, denoted "$\text{Global}_{\text{NR}}$" (NR = No Replication) [Algorithm 4], only the nearest cache requests the missing segment if it is not cached already on any of the other caches. Our basic setting comprises global caching $\text{Global}_{\text{NR}}$ or $\text{Global}_{\text{FR}}$ in conjunction with the LRU algorithm.

ALGORITHM 3: Content Placement Approaches: Global Caching (Full Replication)

```
function GLOBAL CACHING - FULL REPLICATION(C_id, map_caches, cl_ip)   #Where c_id - content metadata, map_caches - map of nearest caches to clients, cl_ip - Client IP address
    s_ip = map_caches[cl_ip]
    if C_id not in get_content_list(s_ip) then
        for cache in map_caches do
            if C_id not in get_content_list(cache) then
                Insert content into cache
    Return
```

ALGORITHM 4: Content Placement Approaches: Global Caching (No Replication)

```
function GLOBAL CACHING - NO REPLICATION(C_id, map_caches, cl_ip)   #Where c_id - content metadata, map_caches - map of nearest caches to clients, cl_ip - Client IP address
    insert_content = True
    for cache in map_caches do
        if C_id in get_content_list(cache) then
            insert_content = False
    if insert_content is True then
        s_ip = map_caches[cl_ip]
        Insert C_id into s_ip
    Return
```

Global caching with a fixed TTL: This approach combines both previous variants with a uniform TTL for all segments. This TTL is generally used at all caches for segment eviction. In the $\text{Global}_{\text{NR}}$ case (i.e., when only one cache is selected for segment insertion), we set the TTL to be equal to the TTL used in the $\text{Global}_{\text{FR}}$ case multiplied with the number of caches in the system. The intuition here is to provide segments with the same *time in cache* on average to collect requests/hits.

Popularity-based TTL caching: Here, the global caching approach with an adaptive and differentiated TTL modifies the TTLs of single segments according to their temporal clustering—that is, after a cache miss, the newly cached segment is annotated with a base TTL_0 that in turn is increased with every subsequent hit before being forgotten after the segment is evicted. For this increase, we devise the rule for the new TTL on the ith hit as $TTL_i = (i + 1)\, TTL_{i-1}$ for $i \geq 1$. We evaluate differentiated TTL caching with both cases (i.e., the $\text{Global}_{\text{NR}}$ and $\text{Global}_{\text{FR}}$ cases).

Quality-based caching [Algorithm 5]: This is a special approach, as the cache space that each quality can occupy is now restricted to a subset of the global cache size. Inspired by Maggs and Sitaraman [35], this caching system consistently maps the segments to different caches (i.e., the

three lowest qualities, Q_1 to Q_3, are cached in *Cache1* and *Cache2*, whereas the higher qualities, i.e., Q_4 and Q_5, are cached on *Cache3* and *Cache4*). Consistent mapping is known to perform equally good as a single contiguous cache under Zipfian and independence assumptions.

ALGORITHM 5: Content Placement Approaches: Quality-Based Caching

```
1: function QUALITY CACHING(C_id, list_caches_low, list_caches_high, cl_ip, SD_qual, HD_qual)   #Where
   c_id - content metadata, list_caches_low - list of SD caches, list_caches_high - list of HD caches, cl_ip - Client IP address,
   SD_qual - list of low quality levels, HD_qual - list of high quality levels
2:     if get_quality(C_id) in SD_qual then
3:         for cache in list_caches_low do
4:             if C_id not in get_content_list(cache) then
5:                 Insert C_id into cache
                                                   #Where get_quality() - quality level of content from ID
6:     else if get_quality(C_id) in HD_qual then
7:         for cache in list_caches_high do
8:             if C_id not in get_content_list(cache) then
9:                 Insert C_id into cache
10:    Return
```

For all experiments, we prepopulate caches using 30 experiment runs prior to actual measurements.

6 EVALUATION ENVIRONMENT

Next, we present the experimental environment used to evaluate SABR. After introducing the CloudLab testbed, we describe the deployed topology and caching algorithms.

6.1 CloudLab Testbed

CloudLab [45] is a geographically distributed testbed for the development, deployment, and validation of cloud-based services. The CloudLab infrastructure consists of several different racks of varying compute and storage sizes designed to provide isolated performance and support experiments at scale. SDN is supported through the deployment of OF switches. This highly virtualizable infrastructure is a miniature representation of SDI.

6.2 Topology

Next, we describe the topology of the CloudLab testbed, which we use to evaluate our CDN architecture. Figure 3 shows the topology, which comprises four different node types and layer-2 links of capacity 100Mbps that connect the nodes. All nodes run Ubuntu 14.04 inside Xen virtual machines. We will identify the nodes involved in the individual experiment descriptions in Section 7. In the following, we describe the configuration of each node type.

Cache nodes: Cache nodes in Figure 3 represent CDN caches serving client requests. Note that the origin server in Figure 3 is of the same type but in contrast contains the entire video library that may be streamed in the scenario. Cache nodes run a vanilla Apache2 Web server along with a HTTP packet sniffer and a MongoDB database. Together, they emulate a Web server gateway interface (WSGI) that implements the cache replacement policies specified in Section 5. The Apache2 server allows persistent HTTP connections.

Client nodes: Client nodes run the different ABR algorithms we implemented in a DASH client [27] that supports SABR streaming as described in Section 4.

Table 1. Experiment Parameters

No. of Clients	60
Size of Dataset	50 videos of 300s duration and segment size of 2s
Available Quality Representations [31]	$\{89k, 0.26M, 0.79M, 2.4M, 4.2M\}bps$
Maximum Size of Caches	70% of dataset
Port Statistics Sampling Interval	1s
ARIMA Window Size	10s

OVS nodes: In this topology, we use software-based OF switches that give us more flexibility in the topology generation within the testbed compared to the use of hardware OF switches. All OVS nodes run Open vSwitch 2.3.1 and communicate with a single OF controller.

OF controller and SABR: The OF controller and SABR as depicted in Figure 2 are installed in the same virtual machine to minimize the REST API query, search, and response time. Note that the SABR framework is logically separated from the controller and can be installed at any desired location. The associated database as described in Section 3.1 supports a distributed implementation.

6.3 Evaluation Metrics and Node Setup

6.3.1 Evaluation Metrics. First, we introduce the deployed performance metrics that are partially based on Zink et al. [53].

Average quality bitrate (AQB): One of the objectives of quality adaptation algorithms is to maximize the average quality bitrate of the streamed video. For a comprehensive QoE representation, we need to combine this metric with the number of quality switches, which is explained in the following.

Number of quality switches ($\#QS$): This metric is used together with AQB and the magnitude of quality switches to draw quantitative conclusions about the perceived quality (QoE). For example, for two streaming sessions with the same AQB, the session with the lower $\#QS$ will be perceived better by the viewer.

Spectrum (H) [53]: The spectrum of a streamed video is a centralized measure for the variation of the video quality bitrate around the average bitrate. A lower H indicates a better QoE.

Rebuffering ratio (RB): The average rebuffering ratio is given by $RB = \mathsf{E}[\frac{t_a - t_e}{t_e}]$, where t_a is the actual playback time and t_e is the video length in seconds, respectively.

Cache hit rate (C_{hr}): The cache hit rate is the average number of video segment requests that are served by the caching system, for instance, by any cache in the network divided by the overall number of requests. This ratio is usually interpreted as the probability that a video segment request is served by the caching system. It is used to assess the efficiency of the caching system.

Network utilization (N_{util}): The average network utilization per link is given by $N_{util} = \mathsf{E}[\frac{T_p}{C}]$, where T_l represents the measured traffic (in bit/s) on link p and C represents the homogeneous link capacity. We only measure the downstream traffic, as its magnitude indicates the amount of video traffic generated in the network.

Server load ratio (S_{load}): The average server load ratio is the amount of video traffic (in Bytes) served from the server divided by the overall amount of video traffic received by the clients—that is, $S_{load} = \mathsf{E}[\frac{T_s}{T_{tot}}]$, where T_s is the amount of traffic served by the server and T_{tot} is the total video traffic. The server load ratio is a measure for the (Byte) efficiency of the caching system.

For an extensive analysis of the SABR approach, we compare the performance of three quality adaptation algorithms. We decided to compare three algorithms that map to the different categories outlined in Section 4.2 to better analyze the interplay between each algorithm and our network assisted approach. The algorithms are described next.

VLC [39]: As one of the earliest DASH players, its simple rate-based quality adaptation algorithm uses (i) the current playout buffer filling and (ii) the average download rate of previous segments. If the buffer filling is below 25%, the client downloads the lowest quality. Otherwise, it downloads the highest quality that is sustainable based on the average download rate. If the buffer is full, the client waits for the playback duration of one segment before requesting the next one.

SQUAD [51]: This hybrid adaptation algorithm is based on the spectrum metric for QoE [53]. It uses a combination of buffer- and rate-based quality adaptation that accounts for the dynamics of TCP on different timescales. SQUAD possesses three states of operation: *decreasing*, *increasing* and *steady states*, which are defined in relation to a sustainable quality bitrate. SQUAD avoids sudden quality changes under fast varying available bandwidth by sacrificing buffer filling if the requested quality bitrate is sustainable.

BOLA [48]: BOLA is a buffer-based quality adaptation algorithm. It uses a Lyapunov technique for renewal processes to decide on the quality of the next segment to be fetched. Whereas BOLA(U) aims to maximize a playback utility metric that is a weighted combination of quality bitrate and smoothness (related to average rebuffering time), BOLA(O) tries to minimize the oscillation in the average quality bitrate by sacrificing buffer filling without the risk of rebuffering to maintain the previously downloaded quality.

6.3.2 Node Setup. In the following, we report the results from experiments of *partial caching*, where a video might only be partially cached (in terms of segments of certain qualities at the different caches in Figure 3). Partial caching of videos occurs when so-called write-through caching is performed—that is, only the segments that are requested by the client in a specific quality are cached. In this scenario, the client may have to switch between caches not only because of bottleneck bandwidth fluctuations but also because of the unavailability of requested segments in a specific quality. Decisions for the latter case can be made because the SABR architecture provides the client with a cache map (see Section 4.2) describing which cache possesses the desired segments at which quality levels.

7 EVALUATION RESULTS

Next, we report results from a series of experiments for various content placement strategies using either LRU or TTL policy in all caches.

7.1 Quality Adaptation Algorithm Performance With SABR

In this experiment, we run 60 clients in four groups as depicted in Figure 3. Clients start with a time offset of $w_c = 1$ second and a client group offset of $w_g = 3$ seconds. Each client streams 10 consecutive movies, where the movies are independently and identically distributed sampled from a Zipf popularity distribution with parameter $\alpha = 1$. Each movie is available in five different qualities (Q_1 to Q_5), as shown together with other configuration parameters in Table 1. We parameterize the user video streaming behavior—that is, the session length using session abandonment statistics obtained from a real-world trace used in Krishnappa et al. [30]. This parameterization is motivated by the fact that, according to Krishnappa et al. [30], only 60% of the videos are completely requested. We conduct experiments with and without session abandonment where a client

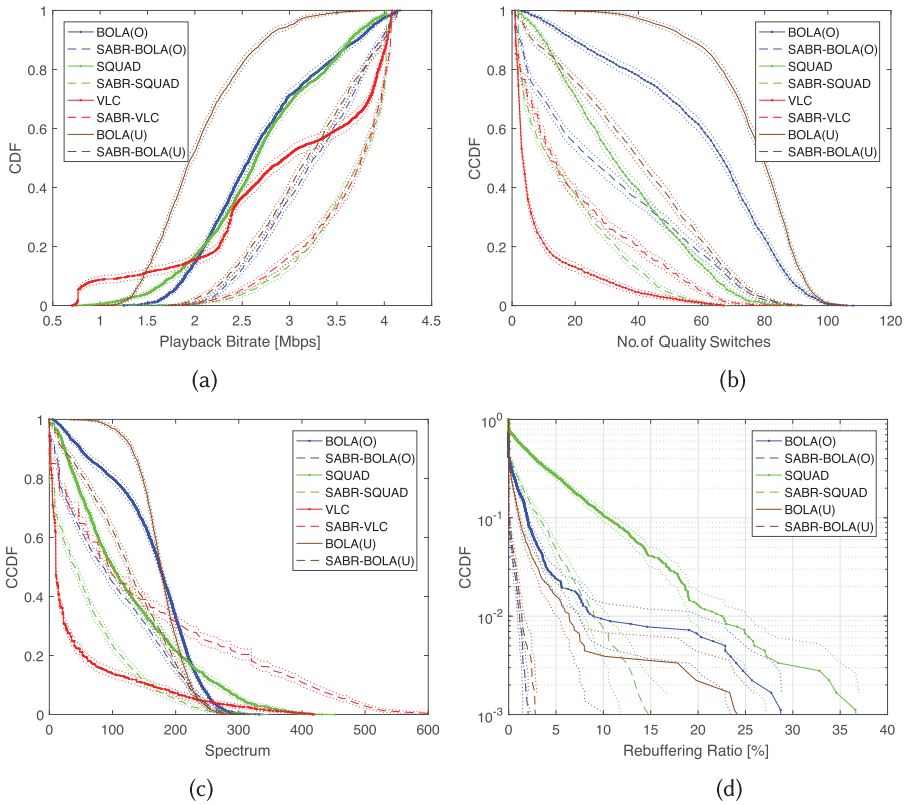


Fig. 4. Fixed cache assignment (Baseline algorithms) versus flexible cache requests with SABR. CCDFs are given with 0.95 confidence intervals. (i) SABR increases the playback rate (in (a)). (ii) Giving the client multiple segment sources substantially impacts the quality oscillations depending on the deployed adaptation algorithm (in (b) and (c)). Note that for (c), a lower spectrum implies less quality variations (i.e., a better QoE). (iii) SABR also significantly reduces rebuffering (in (d)).

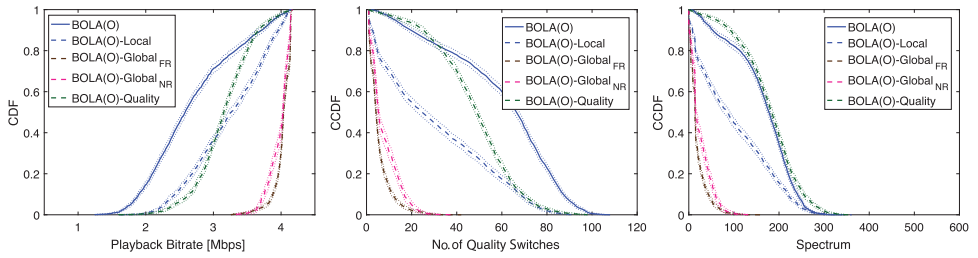
requests a video according to the Zipf popularity distribution. However, for the session abandonment case, client sessions may be only partially watched.

In the first experiment, we compare the streaming performance and the QoE metrics from Section 6.3 for SABR against the Baseline system of fixed client-to-cache assignments. In the Baseline system, we assign each client to the nearest cache (in number of hops) and forward cache misses to the origin server. In the case of SABR, we consider the caching system as one distributed cache, as clients may request segments from different caches or the server. In this experiment, we only consider the local caching version explained in Section 5.

Figure 4 shows the QoE evaluation metrics for a series of 30 experiments. In Figure 4(a), we clearly show that the SABR performance dominates the Baseline approach with respect to the average quality bitrate *AQB* for all evaluated quality adaptation algorithms. We observe that SABR provides a systematic gain in the streamed average quality bitrate. This gain is about 50% to 100% for the clients suffering from the lowest bitrates in the Baseline cases. We attribute this performance gain to two main factors: the flexibility in choosing the segment source and the accurate bottleneck bandwidth information that is provided to the client through network assistance. Providing bottleneck bandwidth information to the clients allows a better utilization of the network,

Table 2. System Performance: Network Utilization and Server Load

	VLC	SABR-VLC	SQUAD	SABR-SQUAD	BOLA(U)	SABR-BOLA(U)	BOLA(O)	SABR-BOLA(O)
N_{util} (%)	42.5	71.2	44.9	74.6	42.5	68.2	46.4	65.7
S_{load} (%)	28.2	21.3	34.5	19.5	28.7	22.5	33.8	23.6



(a) (b) (c)

Fig. 5. Caching strategies with BOLA(O): QoE metrics for content placement strategies using SABR.

as is evident from Table 2. As shown in Table 2, using SABR reduces server load and thus improves server offloading and potentially reduces the operational expenses of a CDN. Although SABR ensures a significant gain in AQB, the improvements in the number of quality switches $\#QS$, spectrum H, and the rebuffering ratio RB depend on how the algorithms utilize the information provided by SABR.

Although the quality adaptation algorithm SQUAD aims to minimize the spectrum H, BOLA(O) focuses on reducing the oscillation in playback bitrate. Figure 4(b) shows a strong reduction in quality switches $\#QS$ with the use of SABR for SQUAD, BOLA(O), and BOLA(U). Figure 4(c) shows that the spectrum H of SABR variants is consistently lower than that of the Baseline algorithms. As shown in Figure 4(d), the rebuffering ratio RB is significantly lower with the use of SABR. We attribute this to the fact that SQUAD, unlike the other algorithms, uses a sliding window of segment download rate history to estimate the available bandwidth. SABR provides ARIMA-based available bandwidth predictions using a sliding window of previous available bandwidths, thus giving a more accurate estimate of the network utilization during playback. Note that the basic VLC algorithm, which is not designed to minimize quality switches $\#QS$ or the spectrum H, shows a perceptible improvement in the overall magnitude and variance of AQB with the use of SABR, but at the cost of increased quality oscillations. From the AQB results for VLC in Figure 4(a), we see that nearly 10% of the clients experience a low QoE and 40% of the clients are served high-quality videos. We observe that SABR improves the overall spread of playback rates for VLC.

7.2 LRU Caching With SABR

Next, we evaluate the performance of different caching strategies when combined with SABR network assistance. Here, we will concentrate on one quality adaptation algorithm (i.e., BOLA) after observing its significant QoE benefits with SABR. For the evaluation, we will resort to Figure 5 to describe the QoE metrics in combination with different caching strategies for a series of 30 experiments. In addition, we present the system performance metrics for different caching strategies in Figure 6 and Table 3. Note that we do not include the results for the rebuffering ratio, RB, as we noted this to be less than 0.1% across all experiments.

Global caching: QoE metrics in Figure 5 indicate that BOLA(O) obtains maximum advantage from adopting a global caching strategy with no replication denoted $\text{Global}_{\text{NR}}$. The number of requests

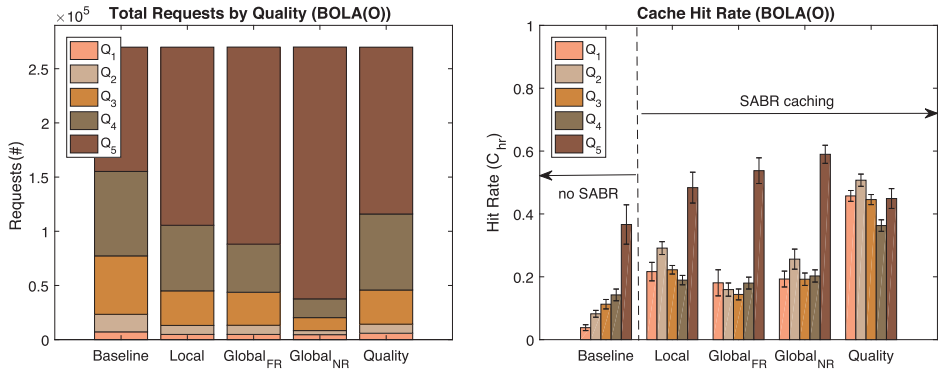


Fig. 6. Absolute number of requests per quality (a) and cache hit rates (b): Comparison of fixed content placement (Baseline) with various content placement strategies for SABR.

Table 3. Caching System Performance: Network Utilization and Server Load

	BOLA(O)			
	Local	$\text{Global}_{\text{FR}}$	$\text{Global}_{\text{NR}}$	Quality
N_{util} (%)	65.7	83.3	82.9	77.1
S_{load} (%)	23.6	20.5	20.4	22.4

for global caching in Figure 6(a) show that BOLA(O) has the highest total requests for the highest quality Q_5. The hit rates C_{hr} for other qualities are distributed almost equally. The network utilization N_{util} given in Table 3 combined with the C_{hr} in Figure 6(b) demonstrates that cache hit rates increase while increasing the utilization of the cache network. Note that the utilization increases naturally as the average quality bitrate is significantly increased, as can be seen from Figure 5(a).

Quality-based caching: The quality-based caching strategy studies more closely the segment quality request pattern of BOLA(O) clients. From Figure 6(b), we see that quality-based caching provides a consistently uniform and relatively high average hit rate C_{hr} for all qualities. We conclude that although quality-based caching improves the hit rate of low qualities in the caching system, it provides only a tolerable QoE and system performance for BOLA(O).

The evaluation that we present in this section investigates an essential trade-off between optimizing the caching architecture, specifically the hit rates, and providing optimal QoE for the end user. Although it is evident from the results that SABR provides a vast improvement in the client streaming quality bitrate and the sever load (i.e., increasing the overall cache system hit rate), we note that a carefully selected caching strategy adopted by the CDN can also contribute significantly to the improvement of all QoE metrics.

7.3 TTL Caching With SABR

In the following, we evaluate the performance of different TTL caching strategies as outlined in Section 5 in combination with network assistance provided by SABR.

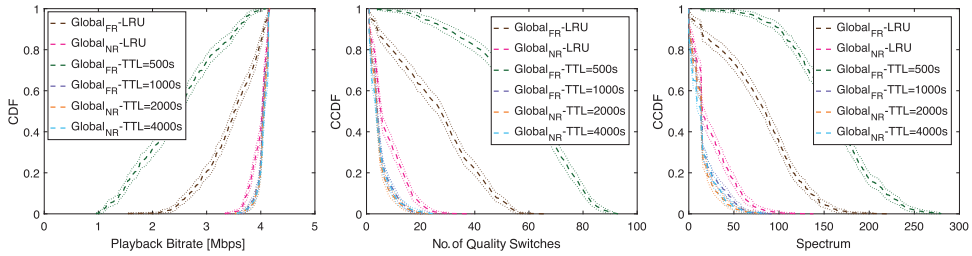


Fig. 7. Fixed TTL Caching with BOLA(O): QoE metrics for content placement strategies using SABR. Increasing the fixed TTL leads to better QoE. The subscripts NR and FR denote global caching with No Replication and Full Replication, respectively.

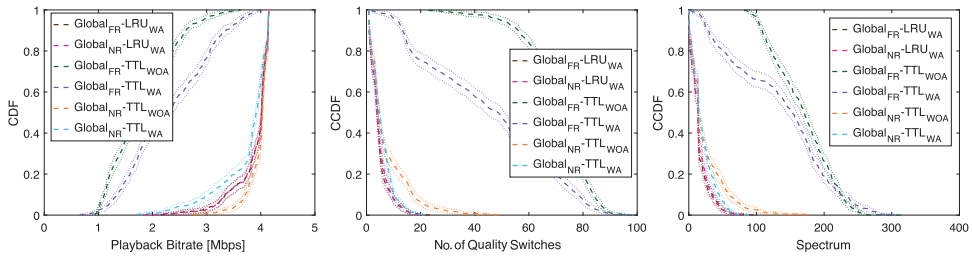


Fig. 8. Adaptive TTL Caching with BOLA(O): QoE metrics for content placement strategies using SABR. Adaptive TTL with no replication provides the higher QoE, as TTLs are prolonged with segment request aggregation, whereas full replication wastes cache space and weakens the TTL prolongation, as requests can be directed to various caches. The subscripts WA and WOA denote a video session with and without user session abandonment behavior, respectively. The subscripts NR and FR denote global caching with No Replication and Full Replication, respectively.

Fixed TTL: Here, we consider TTL caching in SABR where the TTL is fixed for all video segments across all caches. We performed a series of experiments with different TTL values, and the results shown in Figure 7 reveal that an increasing TTL value leads to a significant increase in QoE. This is expected since the performance for increasing TTLs corresponds to the performance of LRU caches given larger cache sizes under fairly general assumptions on the request traffic. However, finding a suitable fixed TTL value is not trivial, as choosing too small TTL values flushes popular segments before they can collect cache hits, which is also seen in Figure 7. To this end, we devise in the following section an adaptive approach to per-object TTLs. For better comparison to the LRU caching policy, we replot the LRU results from Figure 5 in Figure 7 as well. As discussed in Section 5.3, we set the TTLs in the case of "no replication" (Global$_{NR}$) as N times the TTLs of the full replication case (Global$_{FR}$) where N is the number of caches in the system.

Differentiated TTL: Next, we consider TTL caching in SABR with adaptive, differentiated TTLs, where the TTLs of single segments are modified according to the temporal clustering of their requests. With each cache hit, the TTL of the corresponding segment is increased as given in Section 5. These results are presented in Figure 8, which indicates a drastic difference over all QoE metrics for the Global$_{NR}$ and Global$_{FR}$ cases. Whereas the former exhibits high QoE with differentiated TTL values with user session abandonment (*WA*) and without abandonment

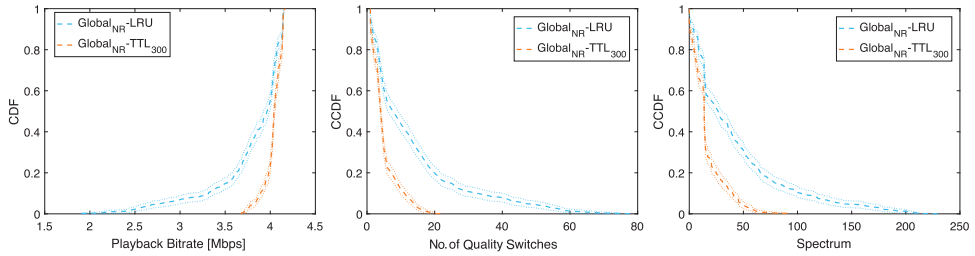


Fig. 9. Adaptive TTL versus LRU caching with BOLA(O): QoE metrics for content placement strategies using SABR. Per-segment differentiated TTL caching provides better QoE than standard LRU for a comparable cache size. Note that we set the LRU cache sizes equal to the average cache sizes observed when running adaptive TTL caching for comparability.

(WOA), the latter experiences a heavy degradation in QoE with this type of TTL assignment. This is mainly caused by the fact that in the Global$_{FR}$ case, multiple copies of a video segment are placed in the overall cache space. TTL differentiation and prolongation is hence strengthened in case of Global$_{NR}$ due to request aggregation, whereas it is weakened in case of Global$_{FR}$ due to the fact that a request can be directed to various caches. Hence, we only consider Global$_{NR}$ in the sequel. Since the Global$_{NR}$ approach exhibits high QoE performance for both TTL and LRU approaches, we modify the cache sizes for the LRU-based policy according to the average size observed in the TTL emulations. As observed in Figure 9, when the cache sizes are set according to the average cache sizes obtained from TTL$_0$ = 300s, LRU caching provides significantly worse QoE, thus making it more attractive to use the adaptive TTL approach.

8 DISCUSSION

8.1 Requirements of Assisted Streaming

Following our discussion of the results in Section 7, we clearly see that exploiting the information provided to the client video player by SABR or generally any network-assisted streaming architecture is a nontrivial optimization problem. ABR streaming clients have been crafted in the past years to be thin and most importantly to run autonomously for scaling reasons. Network-assisted streaming requires (i) a well-defined API between the network and the video player, (ii) modifications to client algorithms to exploit this additional information, and (iii) an appropriate selection of caching algorithms. For example, quality adaptation algorithms such as those in Spiteri et al. [48] and Wang et al. [51] depend on end-to-end bandwidth estimation methods to decide on the appropriate streaming quality. This estimate, which is known to be very hard to obtain cleanly [34], can be significantly improved using network assistance. Buffer-based algorithms [25] make underlying statistical assumptions on the fetched video stream that stem from the notion of point-to-point communication that does not hold anymore if it is possible to find the best source for a video segment using network assistance.

It is crucial to critically assess the impact of network assistance on different streaming algorithms. On the one hand, SABR results presented in Section 7 show that a better streaming quality can be achieved by providing clients with network information on the available bandwidth to different potential caches. On the other hand, the quality adaptation algorithm at the client may have a negative impact on QoE if it is not appropriately adapted to network assistance. For example, in the case of algorithms that do not actively minimize bitrate switchings, such as VLC, network

assistance introduces a higher number of quality switches, which impairs QoE. For algorithms such as SQUAD or BOLA, this information helps in avoiding rebuffering events by providing alternative video segment sources. We also argue that SABR has the tendency to benefit some algorithms more than others. If we consider the examples of SQUAD and BOLA, which use a combination of rate- and buffer-based approaches, BOLA shows higher QoE improvement with SABR. This difference can be attributed to the SQUAD algorithm, which uses a moving window of segment download rate history, whereas BOLA uses the previous segment download rate to decide on the next segment quality. This implies that BOLA benefits more from the temporal information provided by the ARIMA model.

8.2 Scalability and Partial Deployment

Obviously, one concern for a system such as SABR is scalability. Although the main goal of this article is to demonstrate the applicability and benefits of SABR through evaluation in a real-world testbed, we will further investigate its performance in much larger scenarios in future work. For large-scale distributed systems such as CDNs, scalability is always a concern. Therefore, we briefly discuss some potential scalability issues and how their impact can be diminished. This discussion is followed by the evaluation of several approaches for a partial deployment of SABR in a CDN.

Regional approach: Although many CDNs operate on a global scale, our approach is designed to enable operating on a regional level. Thus, several SABR systems can be used in a regional CDN instead of a global one. This will reduce the load on the system and prevent a single point of failure. In addition, our decision to use REST APIs allows a straightforward extension to a hierarchy of SABR systems (i.e., regional systems coordinated by higher-level systems).

Information on cached videos: SABR uses segment size and cache occupancy information, which may seem to be a large amount of data. Luckily, only information about currently requested videos has to be retrieved. This information can be retrieved from caches using the Bloom filter method from Maggs and Sitaraman [35] in the case of co-locating the distributed database with caches. Note that this information set is much smaller than the set of all contents stored on all caches. With MongoDB, we have chosen a database for our system that is highly scalable and in use in large-scale systems.

Monitoring of ports: The monitoring load of the system is reduced by only monitoring ports that are on the paths between clients and caches. In addition, the available bandwidth predictions are centrally calculated once for all clients in a given prediction window. If a SABR reply is not delivered within a certain time interval, for example, the client can always retreat to unassisted bitrate selection. Thus, our approach allows for a natural fallback into standard DASH operation at the client should SABR calls fail.

Partial and fallback deployment: Since it is unrealistic to expect an "overnight" deployment of SABR or a SABR-like approach in a large CDN, we evaluate three different approaches of a partial SABR deployment. One way to perform such a partial deployment is the enabling of SABR in certain regions of the CDN. We replicate such a scenario (called *Split*) in the topology shown in Figure 3 by enabling client groups A and B with SABR to use caches 1 and 3, whereas client groups C and D are not SABR enabled and are assigned to caches 2 and 4, respectively. Additionally, we evaluate an approach (denoted *Startup*) in which all clients are SABR enabled but use SABR only for the initial part of the streaming session, after which they fall back to traditional streaming.

The evaluation results are shown in Table 4. For this evaluation, we only regard the BOLA quality adaptation algorithm, as it performed best in the evaluations presented in Section 7. The first two columns shown in Table 4 are for reference, as they represent a summary of the results in Figure 4. The results for the *Split* approach are shown in two separate columns, *Split (Baseline)*

Table 4. Partial Deployment Study (BOLA(O)): QoE Metrics

BOLA(O)	Baseline	Global$_{norep}$	Split (Baseline)	Split (Global$_{norep}$)	Startup
AQB (Mbps)	2.69 ± 0.6	3.83 ± 0.3	2.72 ± 1.3	3.20 ± 1.3	4.11 ± 0.1
QS	62.54 ± 24.0	14.05 ± 12.8	51.03 ± 38.3	30.39 ± 34.2	6.48 ± 5.8
H	165.45 ± 66.2	42.09 ± 37.4	79.85 ± 55.9	54.87 ± 64.2	16.80 ± 19.8
RB (s)	0.78 ± 2.4	0	2.25 ± 3.0	1.24 ± 2.1	0

and *Split (Global$_{norep}$)* , respectively. The *Split (Baseline)* column represents the results for the part of the topology that is not SABR enabled, whereas the *Split (Global$_{norep}$)* represents the SABR-enabled part. These results demonstrate the feasibility of an incremental deployment. The QoE metrics for the non-SABR clients are comparable to the Baseline approach, whereas for the SABR clients, the QoE metrics improve. Note that the best results are obtained if all clients employ SABR (*Global$_{norep}$*).

The second approach (*Startup*) focuses on scalability. Here, all clients use SABR for the first 150 seconds of the streaming session. This approach significantly reduces the number of clients that use SABR in parallel and reduces the overall load on SABR. Surprisingly, *Startup* performs as well as the approach in which clients use SABR throughout the total length of the streaming session (*Global$_{norep}$*). Further analysis of the results reveals that SABR has the most significant impact at the beginning of a streaming session, where the information on which cache to fetch a segment from and the available bandwidth on the links to the caches significantly speeds up the slow-start phase of the quality adaptation algorithm.

9 RELATED WORK

QoE of ABR streaming: Kleinrouweler et al. [28, 29] consider approaches to providing network assistance to ABR clients for improved QoE. Inan earlier work [29], an Erlang multirate loss modeling approach is shown to fit well to video delivery trees with a central proxy that intercepts the connections between the ABR clients (leafs) and the video source. In more recent work [28], the authors show an SDN-based architecture for DASH assistance using a modified version of OF for QoS differentiation. The idea behind this approach is close to ours; however, the explicit assistance that entails nontrivial modifications to the DASH clients and the SDN-based QoS differentiation are beyond our goals.

Hemmati et al. [24] and Yin et al. [52] model the contention of DASH clients for network bandwidth from a stochastic control perspective. Hemmati et al. [24] consider rate-adaptive video streaming using a decentralized partially observable Markov decision process approach where agents (i.e., DASH clients) pick video rates leading to different states of network congestion. These states are, however, observed through a proxy variable (i.e., the packet loss rate). Although this model is very elegant to capture the network resource sharing with multiple DASH clients, it relies on Markovian assumptions, and most importantly, the problem is known to be NEXP-complete. A more practical approach by Yin et al. [52] shows a robust approximative model-predictive control (MPC) algorithm that is based on offline enumeration. In contrast to our work, the solution in Hemmati et al. [24] requires the offline calculation of the optimal transmission policy beforehand (assuming Markovian network dynamics), whereas the work in Yin et al. [52] does not consider network assistance.

Cofano et al. [18] implement a network control plane (NCP) to allocate and monitor a channel per video stream. In contrast, SABR shows vast improvement in QoE, which we demonstrate

through measurements in a large, real-world testbed without incurring the overhead of per-client QoS management. Bentaleb et al. [10] use SDN capabilities to dynamically allocate network resources based on QoS policies to improve QoE of ABR streaming. Their work uses a client-side probe [32] along with a DPI component to estimate the available bandwidth in a network, which leads to an overhead both for the network and control plane, respectively. Thomas et al. [49] provide an architecture for server- and network-assisted DASH, denoted as SAND. This work introduces a messaging protocol that allows QoS signaling from server to client. Similarly, Nam et al. [41] use network function virtualization–based MPLS traffic engineering to improve streaming QoE.

Bouten et al. [14, 15] consider dynamic server selection for HTTP adaptive streaming and optimizing streaming services using a network-side quality restriction that is based on an integer linear programming approach. This work is related in the context of making use of the bottleneck bandwidth on each path. In contrast, we use time series forecasting for available bandwidth prediction and provide cache information to assist quality adaptation to maximize QoE.

CDN for ABR delivery: Works that investigate the use of an OF control plane for improving video delivery in a CDN include the one by Mukerjee et al. [38], which considers the use of the control plane for load balancing to improve QoE of live video delivery. Here, the authors design, implement, and evaluate a DNS load balancing system with a hybrid (distributed and centralized) control system for live video streaming. Our work is different from this approach, first in the realization through OpenAPIs via a REST interface and an OF controller, which simplifies client integration as seen in Section 4.3, and second in our consideration of the interplay of quality adaptation mechanisms, network bandwidth information, and caching strategies.

Ganjam et al. [22] and Liu et al. [33] propose a coordinated control plane for routing video in the Internet. This work describes a client-based monitoring and control system where the client makes intelligent decisions based on the information gathered by the monitoring system. Unlike our system, this architecture is based on a multi-CDN deployment, where decisions have to be made on a client state basis using global CDN models that do not take cache occupancy into account, thus contributing to considerable overhead for the control plane. In SABR, we dispense with this centralized decision model and provide assistance to ABR clients that make intelligent decisions based on the network status. Georgopoulos et al. [23] use OF to assist a caching system, denoted OpenCache, to facilitate a client redirection to a cache based on network conditions while implementing a caching strategy of minimal hop count. Although SABR is generally able to redirect clients to caches, we additionally provide network and cache status information to the clients, allowing them to make a well-informed decision on content requests. We further analyze the interplay of the client-side quality adaptation and the deployed caching strategies with respect to standard QoE metrics and various performance metrics for caching systems.

The analysis of performance metrics of caching systems has evolved in recent years to generalize the well-known Che's approximation for LRU caches with Poisson object requests [17]. This approximation decouples the interaction of object requests through the use of the notion of a characteristic time that essentially captures the time spent by an arbitrary object in cache. The works by Berger et al. [11], Fofack et al. [20], Fricker et al. [21], and Martina et al. [36] generalize this approach to different arrival processes and show that TTL caches may very well capture the hit rate performance of different capacity-constrained caching algorithms such as LRU, FIFO, and random eviction [11, 20, 21]. Basically, TTL caches annotate the objects admitted to the cache with a timer, which sets the eviction time of the just-admitted object. Variants of this strategy include refreshing the timer upon object hits or sampling the timer values from probability distributions.

In this work, we use the concept of TTL caches in conjunction with SABR to show QoE gains compared to standard LRU caching. We show that by differentiating and adapting the TTLs on an object basis, we maximize the user QoE through prolonging the TTLs of objects with temporally clustered requests. Our results show that SABR provides a user QoE given adaptive TTLs that is significantly higher than in the LRU caching case while setting the LRU cache sizes equal to the measured TTL cache sizes for comparability.

10 CONCLUSIONS AND FUTURE WORK

This work leverages the potential of SDI to provide an SDN control plane architecture that assists ABR video streaming applications in CDNs. This architecture, which we denote as SABR, essentially, provides streaming clients with refined information on network conditions and available video sources, which is made queryable through an SDN architecture. Nonetheless, clients retain full control of the streaming decisions, including quality selection algorithms and video source switching. Our evaluation in a real-world, geographically distributed testbed shows significant improvement in quantitative metrics of QoE, which we mainly attribute to improved estimation accuracy of network characteristics, and statistical multiplexing gains. Further, we show that carefully selected caching strategies can significantly improve streaming QoE and overall system performance. Our future work will address the performance evaluation of SABR for large-scale networks including edge caching in wireless systems.

REFERENCES

[1] Home Page. Retrieved March 9, 2018, from http://www.adobe.com/products/hds-dynamic-streaming.html.

[2] 2018. Home Page. Retrieved March 9, 2018, from https://developer.apple.com/resources/http-streaming/.

[3] Sandvine. 2016. Global Internet Phenomena: Latin America and North America. Retrieved March 9, 2018, from https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america-and-north-america.pdf.

[4] Netflix Open Connect. 2018. ISP Partnership Options. Retrieved March 9, 2018, from https://openconnect.netflix.com/en/delivery-options/.

[5] Microsoft. 2018. Smooth Streaming. Retrieved March 9, 2018, from http://www.iis.net/downloads/microsoft/smooth-streaming.

[6] MongoDB. 2018. Home Page. Retrieved March 9, 2018, from https://www.mongodb.org/.

[7] Sleepy Mongoose (MongoDB). 2018. Home Page. Retrieved March 29, 2018, from https://github.com/mongodb-labs/sleepy.mongoose.

[8] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang. 2012. Unreeling Netflix: Understanding and improving multi-CDN movie delivery. In *Proceedings of the IEEE INFOCOM Conference*. 1620–1628.

[9] S. Agarwal, M. Kodialam, and T. V. Lakshman. 2013. Traffic engineering in software defined networks. In *Proceedings of the IEEE INFOCOM Conference*. 2211–2219.

[10] Abdelhak Bentaleb, Ali C. Begen, and Roger Zimmermann. 2016. SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking. In *Proceedings of the 2016 ACM on Multimedia Conference (MM'16)*. ACM, New York, NY, 1296–1305. DOI: http://dx.doi.org/10.1145/2964284.2964332

[11] Daniel S. Berger, Philipp Gland, Sahil Singla, and Florin Ciucu. 2014. Exact analysis of TTL cache networks. *Performance Evaluation* 79, 2–23.

[12] Mark Berman, Jeffrey S. Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. 2014. GENI: A federated testbed for innovative network experiments. *Computer Networks* 61, 0, 5–23.

[13] Divyashri Bhat, Amr Rizk, and Michael Zink. 2017. Network assisted content distribution for adaptive bitrate video streaming. In *Proceedings of the 8th ACM Multimedia Systems Conference (MMSys'17)*. 62–75.

[14] Niels Bouten, Maxim Claeys, Bert Van Poecke, Steven Latré, and Filip De Turck. 2016. Dynamic server selection strategy for multi-server HTTP adaptive streaming services. In *Proceedings of the 2016 12th International Conference on Network and Service Management (CNSM'16)*. IEEE, Los Alamitos, CA, 82–90.

[15] Niels Bouten, Steven Latré, Jeroen Famaey, Werner Van Leekwijck, and Filip De Turck. 2014. In-network quality optimization for adaptive video streaming services. *IEEE Transactions on Multimedia* 16, 8, 2281–2293.

[16] K. P. Burnham and D. R. Anderson. 2003. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer.

[17] Hao Che, Ye Tung, and Zhijun Wang. 2002. Hierarchical Web caching systems: Modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications* 20, 7, 1305–1314. DOI: http://dx.doi.org/10.1109/JSAC.2002.801752

[18] Giuseppe Cofano, Luca De Cicco, and Saverio Mascolo. 2014. A control architecture for massive adaptive video streaming delivery. In *Proceedings of the ACM Workshop on Design, Quality, and Deployment of Adaptive Video Streaming*. 7–12.

[19] Andrés Ferragut, Ismael Rodriguez, and Fernando Paganini. 2016. Optimizing TTL caches under heavy-tailed demands. *SIGMETRICS Performance Evaluation Review* 44, 1, 101–112.

[20] Nicaise Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. 2014. Performance evaluation of hierarchical TTL-based cache networks. *Computer Networks* 65, 212–231.

[21] C. Fricker, P. Robert, and J. Roberts. 2012. A versatile and accurate approximation for LRU cache performance. In *Proceedings of the 24th International Teletraffic Congress (ITC'12)*. 1–8.

[22] Aditya Ganjam, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar, and Hui Zhang. 2015. C3: Internet-scale control plane for video quality optimization. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. 131–144.

[23] Panagiotis Georgopoulos, Matthew Broadbent, Arsham Farshad, Bernhard Plattner, and Nicholas Race. 2015. Using software defined networking to enhance the delivery of video-on-demand. *Computer Communications* 69, 79–87.

[24] Mahdi Hemmati, Abdulsalam Yassine, and Shervin Shirmohammadi. 2015. A Dec-POMDP model for congestion avoidance and fair allocation of network bandwidth in rate-adaptive video streaming. In *Proceedings of the IEEE Symposium Series on Computational Intelligence*. 1182–1189.

[25] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4, 187–198.

[26] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, et al. 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review* 43, 4, 3–14.

[27] Parikshit Juluri, Venkatesh Tamarapalli, and Deep Medhi. 2015. SARA: Segment-aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In *Proceedings of the IEEE ICC QoE-FI Workshop*.

[28] Jan Willem Kleinrouweler, Sergio Cabrero, and Pablo Cesar. 2016. Delivering stable high-quality video: An SDN architecture with DASH assisting network elements. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys'16)*. Article No. 4.

[29] Jan Willem Kleinrouweler, Sergio Cabrero, Rob van der Mei, and Pablo Cesar. 2015. Modeling stability and bitrate of network-assisted HTTP adaptive streaming players. In *Proceedings of the 2015 27th International Teletraffic Conference (ITC-27)*. 177–184.

[30] Dilip Kumar Krishnappa, Michael Zink, and Ramesh K. Sitaraman. 2015. Optimizing the video transcoding workflow in content delivery networks. In *Proceedings of the 6th International Conference on Multimedia Systems (MMSys'15)*. 37–48.

[31] Stefan Lederer, Christopher Müller, and Christian Timmerer. 2012. Dynamic adaptive streaming over HTTP dataset. In *Proceedings of the 3rd International Conference on Multimedia Systems (MMSys'12)*. 89–94.

[32] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C. Begen, and David Oran. 2014. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications* 32, 4, 719–733.

[33] Xi Liu, Florin Dobrian, Henry Milner, Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. 2012. A case for a coordinated Internet video control plane. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'12)*. 359–370.

[34] R. Lübben, M. Fidler, and J. Liebeherr. 2014. Stochastic bandwidth estimation in networks with random service. *IEEE/ACM Transactions on Networking* 22, 2, 484–497.

[35] Bruce M. Maggs and Ramesh K. Sitaraman. 2015. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review* 45, 3, 52–66.

[36] V. Martina, M. Garetto, and E. Leonardi. 2014. A unified approach to the performance analysis of caching systems. In *Proceedings of the IEEE INFOCOM Conference*. 2040–2048. DOI: http://dx.doi.org/10.1109/INFOCOM.2014.6848145

[37] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2, 69–74.

[38] Matthew K. Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. 2015. Practical, real-time centralized control for CDN-based live video delivery. In *ACM SIGCOMM Computer Communication Review* 45, 4, 311–324.

[39] C. Müller and C. Timmerer. 2011. A VLC media player plugin enabling dynamic adaptive streaming over HTTP. In *Proceedings of the ACM Multimedia Conference*. 723–726.

[40] Mubashar Mushtaq, Toufik Ahmed, and Djamal-Eddine Meddour. 2006. Adaptive packet video streaming over P2P networks. In *Proceedings of the ACM International Conference on Scalable Information Systems (InfoScale'06)*.

[41] Hyunwoo Nam, Kyung-Hwa Kim, Jong Yul Kim, and Henning Schulzrinne. 2014. Towards QoE-aware video streaming using SDN. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM'14)*. 1317–1322.

[42] Stefan Podlipnig and Laszlo Böszörmenyi. 2003. A survey of Web cache replacement strategies. *ACM Computing Surveys* 35, 4, 374–398. DOI: http://dx.doi.org/10.1145/954339.954341

[43] R Core Team. 2014. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. http://www.R-project.org/.

[44] Barath Raghavan, Martín Casado, Teemu Koponen, Sylvia Ratnasamy, Ali Ghodsi, and Scott Shenker. 2012. Software-defined Internet architecture: Decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks (HotNets-XI)*. 43–48.

[45] Robert Ricci, Eric Eide, and the CloudLab Team. 2014. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *The USENIX Magazine* 39, 6. https://www.usenix.org/publications/login/dec14/ricci.

[46] D. Simeonidou, R. Nejabati, and M. P. Channegowda. 2013. Software defined optical networks technology and infrastructure: Enabling software-defined optical network operations. In *Proceedings of the Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC'13)*. 1–3.

[47] I. Sodagar. 2011. The MPEG-DASH standard for multimedia streaming over the Internet. *IEEE MultiMedia* 18, 4, 62–67. DOI: http://dx.doi.org/10.1109/MMUL.2011.71

[48] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *Proceedings of the IEEE INFOCOM Conference*. 1–9.

[49] E. D. R. Thomas, M. O. van Deventer, T. Stockhammer, A. C. Begen, and J. Famaey. 2015. *Enhancing MPEG Dash Performance via Server and Network Assistance*. IET, Stevenhage.

[50] Niels L. M. Van Adrichem, Christian Doerr, and Fernando Kuipers. 2014. OpenNetMon: Network monitoring in OpenFlow software-defined networks. In *Proceedings of the IEEE Network Operations and Management Symposium (NOMS'14)*. 1–8.

[51] C. Wang, A. Rizk, and M. Zink. 2016. SQUAD: A spectrum-based quality adaptation for dynamic adaptive streaming over HTTP. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys'16)*. ACM, New York, NY, 1:1–1:12.

[52] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. *SIGCOMM Computer Communication Review* 45, 4, 325–338.

[53] M. Zink, J. Schmitt, and R. Steinmetz. 2005. Layer-encoded video in scalable adaptive streaming. *IEEE Transactions on Multimedia* 7, 1, 75–84.

Received June 2017; revised November 2017; accepted January 2018