FLARE: Coordinated Rate Adaptation for HTTP Adaptive Streaming in Cellular Networks

Youngbin Im,* Jinyoung Han,† Ji Hoon Lee,‡ Yoon Kwon,§
Carlee Joe-Wong,¶ Ted "Taekyoung" Kwon,‖ and Sangtae Ha*
*University of Colorado Boulder †Hanyang University ‡Juni Korea
§Kakao ¶Carnegie Mellon University ‖Seoul National University
Emails: {youngbin.im, sangtae.ha}@colorado.edu, jinyounghan@hanyang.ac.kr, brandon.lee@juniglobal.com
alberto.kwon@kakaocorp.com, cjoewong@andrew.cmu.edu, tkkwon@snu.ac.kr

***Abstract*—Fog computing is an emerging architecture that aims to run applications on multiple devices that lie on a continuum from cloud servers to personal user smartphones. These architectures allow applications to optimize over the information stored at and functionalities run on each device, based on individual device capabilities. We demonstrate the benefits of this approach for mobile video streaming. Existing HAS (HTTP adaptive streaming) techniques often suffer from problems like unstable video quality and suboptimal resource utilization. We find that a lack of coordination prevents both client- and network-side HAS techniques from solving them. However, our fog approach can exploit existing telecommunication APIs, which expose network capabilities to applications, in order to coordinate between clients and the network. Our coordinated HAS solution, FLARE, optimizes the total utility of all clients in a cell while maintaining stable video quality and supporting user- and device-specific needs. We implement FLARE on a commodity LTE femtocell and use the implementation to conduct the first comparison of HAS players on an LTE femtocell. By conducting extensive experiments using the ns-3 simulator, we also demonstrate that FLARE (i) enhances the average video bitrate, (ii) achieves stable video quality, and (iii) balances the throughput of simultaneous video and data flows, compared to other representative HAS solutions.**

I. Introduction

Traditional computing applications have largely been run from either the client–e.g., mobile applications' triggering content downloads from a remote server–or from a centralized server, e.g., cloud-based fusion of data collected by multiple distributed sensors. More recently, however, fog and edge computing approaches have begun to develop hybrid architectures in which application functionalities are divided between the client and server [1], [2]. Such architectures are likely to become increasingly popular as more and more devices emerge along the "cloud-to-things continuum" between remote cloud servers and personal devices like smartphones or smartwatches. We can thus envision applications that are cooperatively run on multiple devices along this continuum.

A fog architecture can bring significant benefits compared to a purely client- or server-based approach: for instance, putting some control at the client makes it easier to customize the application's configuration according to individual clients' preferences. Client-based control can also allow these configurations to more rapidly adapt to changes in client preferences or background information. A central server component, on the other hand, can improve coordination between different clients who might compete for resources, e.g., a cellular network setting in which multiple applications share network bandwidth. Thus, fog computing allows applications to optimize over both client and server information, leading to an intelligent division of labor between the client and server.

Cellular networks are an especially compelling example of fog computing architectures. Network operators like Vodafone, for instance, are beginning to expose their network APIs (application programming interfaces) as a way to find new revenue opportunities and help third-party app developers [3]. These APIs can reveal statistics such as network availability and congestion in near real-time, which can then be used to optimize the bandwidth requested by different client flows. OneAPI [4], for instance, allows UE applications to specify their QoS (quality-of-service) requirements and to be notified of the QoS level that the network can provide. Thus, OneAPI can be used to develop fog applications that have a presence at both the client and the network server, leveraging information from both to optimize network bandwidth allocation. In this work, we focus on one particular application that often dominates bandwidth usage: video streaming.

A. HTTP Adaptive Streaming (HAS)

Recent advances in mobile communications and video compression technologies have led to significant growth in video traffic over mobile networks. Many major video service providers, including Netflix and YouTube, employ HTTP adaptive streaming (HAS) to deliver this video content, largely due to its adaptability to time-varying network conditions [5]–[8] and ability to use existing HTTP web server infrastructure. The basic idea of HAS is to divide a video into multiple segments over the time dimension, e.g., of a few seconds each. Each segment can be encoded with multiple bitrates (e.g., 640p, 1080p), and in each time interval, HAS selects the bitrate of a corresponding segment. Information about each segment, such as its sequence, timing, bitrate, and URL, is stored in a Media Presentation Description (MPD) file that is sent to the client streaming the video. Either the client or a server in the network can then choose the segment bitrates.

The choice of which video segment to download depends on a variety of both client and network factors, including the

client device capabilities, the time-varying amount of available network bandwidth, and the relative priority of non-video flows in the network. Thus, HAS is uniquely suited to a fog computing approach that takes advantage of client and network information. Clients cannot obtain information about the available network resources, the number of devices in the network, etc., while network entities alone cannot access some device information that impacts the optimal video bitrates. Indeed, network-side and middleware solutions cannot handle encrypted video traffic, which constitutes a large share of today's data traffic [9] and obscures even the identity of video traffic from any entity placed in the network.

This fog-based philosophy distinguishes our work from existing HAS systems, which generally select the segment bitrates based only on information at the user equipment (UE) clients or inside the network. We overcome these challenges by designing and implementing FLARE (Fair and Link-Aware RatE adaptation), *the first HAS system that uses both individual client information and newly standardized network APIs that expose network information*.

B. Related Work: Existing HAS Techniques

As suggested by their name, client-side HAS approaches run an adaptation algorithm on each UE. At each given time, the algorithm estimates the available network bandwidth based on the throughput history of recently downloaded segments. It then chooses a desired bitrate for the next video segment based on this estimate, the video encoding information, the amount of video data in its buffer, and the previously selected bitrate [10]–[12]. For instance, the FESTIVE algorithm [13] considers tradeoffs between stability of the video quality, fairness, and efficiency. Tian *et al.* [14] similarly seek to balance the smoothness (i.e., stability) of the video rate with high bandwidth utilization. However, both algorithms can yield unstable video quality in LTE wireless networks, due to highly variable link bandwidth (cf. Section IV).

Li *et al.* [10] showed that the discrete nature of video bitrates makes it hard for clients to find their fair share of available bandwidth, proposing an adaptation algorithm akin to TCP congestion control. Yin *et al.* [11] introduced a model predictive control algorithm in order to optimally combine throughput and buffer occupancy information. Xie *et al.* [12] proposed a cross-layer approach in which the PHY-layer information of the LTE network is used to estimate available bandwidth. However, all client-side techniques suffer from the fact that a UE has no information about the overall radio resources in the cell. Consequently, clients may suffer from frequent re-buffering, unfair bitrate assignments, and abrupt bitrate changes. Network-side HAS techniques can overcome these problems, but have their own limitations.

Network-side techniques collectively consider individual clients' channel conditions and utilities and then choose clients' bitrates so as to maximize their aggregate utility [15]–[18]. These bitrates are enforced by setting the guaranteed bit rate (GBR) and/or maximum bit rate (MBR) for each flow using the base station (BS) scheduler or a resource slicing technique. For instance, Chen *et al.* [15] proposed AVIS, an in-network resource management framework that schedules HTTP-based video flows in cellular networks. However, clients' realized bitrates may still be suboptimal: these approaches assume that the bitrates for the segments requested by the UE adaptation modules will quickly converge to the bitrates chosen by the network entity, which may not occur in practice (cf. Section IV-B). Moreover, the network entity is oblivious to some UE characteristics, such as the buffer memory, computation capability, client preferences, etc., that may affect clients' desired bitrates and utilities. Indeed, if the video traffic is encrypted the network may not even be able to identify HAS flows. Some network-side HAS techniques (e.g., [15]) also suffer from a static division of resources among video and other data flows, which can cause severe underutilization of the overall radio resources.

C. FLARE: A Fog Computing Approach to HAS

We implement FLARE as a light-weight client-side plugin (e.g., using Javascript) that can be easily embedded in video service providers' HAS players. FLARE coordinates with OneAPI [4] to obtain network information. Given client information from FLARE's client-side UE (user equipment) plugin, the OneAPI server can choose a desired bitrate for each video flow based on information obtained from the network BSs, UEs, and the network's Policy, Charging, and Rules Function (PCRF). The PCRF manages and monitors all flows in the network; thus, it can provide the OneAPI server with all relevant network information, such as the number of non-video flows, for assigning network bitrates. Since the OneAPI server is provided by the network operator, it can easily interface with both our UE plugins and the PCRF.

We divide the bitrate selection and enforcement between the UE plugins and OneAPI network server, guaranteeing their coordination. While the OneAPI server can easily allocate bandwidth between video and non-video flows, it does not directly control the actual bandwidth allocations. Thus, we use the FLARE plugin to enforce the bitrates chosen by the OneAPI server, avoiding convergence inefficiencies due to mis-coordination between the UE and the network. We limit the information exchanged between the OneAPI server and the plugin to the minimum needed to decide the bitrates, protecting clients' privacy; they can individually choose to disclose additional information to the OneAPI server.

FLARE introduces three fundamental improvements to existing HAS systems. Not only are bitrates chosen based on both the network and client information, but FLARE unifies the resource allocation of video and data flows into a single framework. Thus, FLARE optimizes the total utility of *all* clients, avoiding AVIS's occasional inefficiencies in resource utilization caused by static resource partitioning between video and data flows. Finally, FLARE ensures a stable video quality by taking a stateful approach to the rate selection and ensuring that UEs always utilize the bitrates assigned by the HAS network entity. Thus, we avoid fluctuations in the selected rates as well as instabilities caused by a mismatch between the

UE's requested and assigned bitrates. We demonstrate these improvements through the following contributions:

We propose **FLARE, a coordinated rate adaptation approach** that uses a fog computing architecture for selecting client bitrates in cellular networks. FLARE incorporates both client- and network-side information and guarantees coordination between network- and client-side bitrate selection. For ease of deployment, FLARE requires minimal modifications to run on existing HAS video players. We introduce and develop a plugin-style module that can be easily embedded on video players, can communicate with the network entity without causing privacy issues, and can work with cryptographic protocols such as Secure Sockets Layer (SSL).

We **develop FLARE on a commodity LTE femtocell BS**, implementing its functionalities in the MAC layer. We then evaluate FLARE's performance in the testbed and show that it outperforms existing HAS players. Our **extensive simulation study** demonstrates that FLARE significantly improves on both client- and network-side algorithms in various settings in terms of the average video bitrate (up to a 69% and 66% improvement compared to AVIS [15] and FESTIVE [13], respectively), stability of video quality (up to 85% and 95% improvement compared to AVIS and FESTIVE, respectively), and balance of resources allocated to video and data flows.

We document the benefits of a fog computing approach to HAS by comparing purely client- and network-based approaches to FLARE. We find that client-side HAS algorithms suffer from (i) throughput imbalances between data and video flows [13], (ii) unstable video selection due to a lack of network-side information [13], [19], and (iii) frequent rebuffering and drastic bitrate changes due to aggressive rate selection [19] (Figures 4 and 5). Network-side HAS solutions, on the other hand, suffer from (i) instability in the selected bitrates due to indirect rate enforcement [15], [16] and (ii) suboptimal flow utilities over time due to static resource allocation among video and data flows [15] (Figures 6 and 7).

We describe FLARE's design principles and rate adaptation algorithms in Section II before discussing FLARE's operations and LTE femtocell testbed implementation in Section III. We then evaluate FLARE compared to other HAS solutions in Section IV. We finally discuss some of FLARE's deployment challenges before concluding in Section V.

II. FLARE Design

We first discuss the design principles used in our coordinated HAS system, FLARE, in Section II-A, and then present our rate adaptation algorithm in Section II-B.

A. Design Considerations

We design FLARE's client- and network-side components and their interactions by following four main principles.

Compatibility with existing systems: Most major video service providers (e.g., YouTube) have adopted cryptographic protocols to deliver their videos safely. It is therefore difficult to detect HAS video flows and enforce their bitrates in the middle of the end-to-end path as proposed by [15], [16].

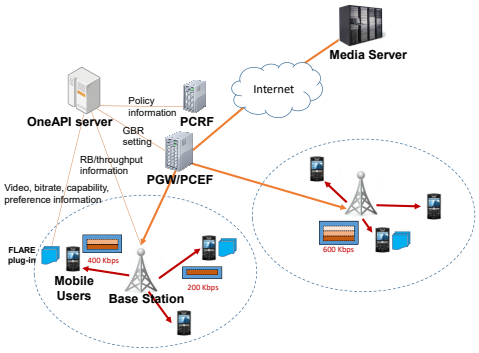


Fig. 1. FLARE's main entities in an LTE network are the OneAPI server and UE plugins (circled), which decide the UE bitrates depending on information like their link conditions, utility, and user/device status.

To handle this challenge, we adopt an existing standard, OneAPI [4], to coordinate between the network and clients. Clients can indicate which flows contain video traffic by communicating with OneAPI. We implement FLARE as a light-weight client-side plugin using a Javascript file, which can be easily embedded in existing video services.

Minimal modification: To ensure that our system is deployable, we minimize the changes required in the current cellular network core. Instead, we create a control overlay on top of existing infrastructure.

Privacy protection: While individual client information such as video clickstream logs can help FLARE assign clients the optimal bitrates, clients may not wish to reveal this information due to privacy concerns. We thus minimize the information required from each client, though clients may reveal more information at their discretion.

Scalability: Optimizing the bitrate selections for all clients in a cell requires a significant amount of computational power: since there are a finite number of possible bitrates for each video segment, the optimization problem is a discrete, NP-hard problem (cf. Section II-B). We propose a continuous optimization framework to reduce the computational complexity in order to accommodate a large number of clients.

Figure 1 shows FLARE's architecture. The FLARE plugin at each UE decides which client information to send to the OneAPI server. The OneAPI server, which has access to the current network state from the BS, then decides clients' bitrates with a scalable, continuous optimization framework, enforcing these decisions through the network PCRF (policy, charging, and rules function), PCEF (policy, charging, and enforcement function), and the FLARE UE plugin. A single OneAPI server can manage multiple BSs, though the bitrates are calculated independently for each network cell. We next describe the details of the OneAPI server's bitrate optimization before outlining FLARE's operations in Section III.

B. FLARE Coordination

We now explain how the network and client coordinate to decide the optimal bitrates. We assume that all video flows follow HAS, all data flows use TCP, and the OneAPI server

Algorithm 1: Algorithm for calculating the video bitrates and allocating resources for video and data flows.

for $i > 1$ **do**
- Gather client information;
- Find $\left\{b_u^{i-1}, n_u^{i-1}, L_u^{i-1} * | u \in U\right\}$ from BAI $i-1$;
- Solve (3–4) or its relaxation for $r^*, R_u^{i\,*}$, adding constraints from client information as necessary;
- $L_u^{i\,*} = \max_k \left\{k \in \mathbb{Z}_+ | r_u(k) \leq R_u^{i\,*}\right\}$;
- **if** $L_u^{i\,*}, L_u^{i-1\,*}, \ldots, L_u^{i-\delta\left(L_u^{i-1}\right)^*} = L_u^{i-1} + 1$ **then**
  - $L_u^i = L_u^{i-1} + 1$;
- **else**
  - $L_u^i = \min\left\{L_u^{i-1}, L_u^{i\,*}\right\}$;
- $R_u^i = r_u\left(L_u^i\right)$ for each $u \in U$;

knows the number of data flows in each network cell through its connection with the PCRF (cf. Figure 1).

We suppose that the OneAPI server runs a bitrate optimization algorithm once per bitrate assignment interval (BAI). In each BAI, FLARE attempts to maximize the total utility of video and data flows, which we define as

$$\sum_{u \in U} \beta_u \left(1 - \frac{\theta_u}{R_u^i}\right) + \alpha \sum_{u \in D} \log \frac{T_u^i}{\theta_u}, \tag{1}$$

following [16], [20], subject to capacity and stability constraints. Here U denotes the set of video flows, D the set of data flows, and R_u^i and T_u^i the bitrate and throughput respectively of client, or flow, u for BAI i. The client-specific parameters β_u and θ_u represent the importance of video traffic to client u and the screen size (a larger screen requires a higher bitrate for good resolution) respectively. We use a logarithmic utility function for the data flows to account for the fact that larger throughputs yield smaller marginal increases in utility. Video bitrates similarly exhibit decreasing marginal utility as they increase, but we constrain the maximum utility achieved to 1 in order to model the fact that, once the bitrate can support the device resolution, users notice very little difference in the video streaming. The parameter α controls the importance of data relative to video flows. We can simplify (1) by defining r, the fraction of resource blocks (RBs) allocated to video flows:

Lemma 1: Suppose that the sum of the throughputs of all data flows is proportional to $1-r$, the portion of RBs allocated to the data flows, and that each data flow receives a fixed fraction of the total data throughput throughout its lifetime. Then optimizing (1) is equivalent to optimizing

$$\sum_{u \in U} \beta_u \left(1 - \frac{\theta_u}{R_u^i}\right) + n\alpha \log\left(1 - r\right). \tag{2}$$

Proof: We find from our assumptions that

$$\begin{aligned}\sum_{u \in D} \log \frac{T_u^i}{\theta_u} &= \sum_{u \in D} \left(\log\left(X_u^i\left(1-r\right)\right) - \log\left(\theta_u\right)\right) \\ &= n \log\left(1-r\right) + \sum_{u \in D} \left(\log\left(X_u^i\right) - \log\left(\theta_u\right)\right)\end{aligned}$$

where X_i^u is flow u's throughput with no video flows ($r = 0$) and n is the total number of data flows. Since X_u^i and θ_u are constants, we can neglect their sums when maximizing (1), instead choosing r and the R_u^i to maximize (2). ∎

We now formulate FLARE's network capacity and bitrate stability constraints. We let $r_u = \{r_u(1), r_u(2), \ldots, r_u(M_u)\}$ denote the bitrates available for the video on flow u, with $r_u(k) \leq r_u(k+1)$ for all k, and choose $R_u^i \in r_u$. We use L_u^i to denote the index of the chosen R_u^i, i.e., $R_u^i = r_u\left(L_u^i\right)$.

We constrain the total allocated RBs for video flows to be no more than the total available for video flows: $\sum_{u \in U} \frac{BR_u^i}{b_u^{i-1}} n_u^{i-1} \leq rN$, where N is the total number of RBs, B is the length of the BAI, and n_u^i and b_u^i are respectively the numbers of RBs assigned and bytes transmitted to client u in BAI i. Note that our constraint incorporates knowledge from the previous BAI, $i-1$, to estimate the flow's required RBs in BAI i. We also leverage this past information to introduce a stability constraint that prevents large increases in flows' bitrates in consecutive time intervals: $R_u^i \leq r_u\left(L_u^{i-1} + 1\right)$, for $i > 1$. We do, however, permit large drops in the flow's bitrate if necessary to maximize (2), e.g., several new clients enter the system. The main idea is then to solve

$$\max_{r \in [0,1], R_u^i \in r_u} \sum_{u \in U} \beta_u \left(1 - \frac{\theta_u}{R_u^i}\right) + n\alpha \log\left(1 - r\right) \tag{3}$$

$$\text{s.t.} \sum_{u \in U} \frac{BR_u^i}{b_u^{i-1}} n_u^{i-1} \leq rN,\ R_u^i \leq r_u\left(L_u^{i-1} + 1\right) \tag{4}$$

Since (3–4) is a mixed integer optimization problem that may include many flows, it is difficult to solve efficiently. A full examination of solution algorithms for (3–4) is beyond the scope of this work, which focuses on building FLARE, but we propose one possible solution: using a continuous relaxation in order to make the problem tractable.

Proposition 1: The continuous relaxation of (3–4), in which we replace $R_u^i \in r_u$ with the constraint $r_u(1) \leq R_u^i \leq r_u(M_u)$, is a convex optimization problem.

Proof: We note that the constraints (4) are linear in the optimization variables r and R_u^i; thus, it suffices to show that (3) is a concave function of r and R_u^i. Omitting the detailed calculations, concavity follows from the fact that $\log x$ and $-x^{-1}$ are both concave functions of x. ∎

We can use a standard convex optimization solver to solve our relaxation. Given a solution $r^*, R_u^{i\,*}$, we discretize each $R_u^{i\,*}$ by rounding it down to the nearest available bitrate, which we denote as $r_u\left(L_u^{i\,*}\right)$.[1] Algorithm 1 formalizes this method of calculating the bitrate. Note that it can well-approximate the optimum if there are a large number of available bitrates. We add two extra steps beyond solving (3–4): gathering information from the client in order to formulate

[1]We can accommodate client-side bandwidth limitations that can occur in wired networks by rounding $R_u^{i\,*}$ to the minimum of $r_u\left(L_u^{i\,*}\right)$ and x_u, where x_u is client u's maximum bandwidth for HTTP video streaming.

our optimization problem, and an additional constraint on the chosen bitrate to ensure stability.

Enhancing stability: Since frequent bitrate changes can adversely affect video users' experience [21], [22], the last step of Algorithm 1 ensures that the bitrate does not increase from the previous BAI unless the increase has been recommended (i.e., $L_u^{i\,*} = L_u^{i-1} + 1$) for the previous $\delta\left(L_u^{i-1} + 1\right)$ BAIs. Here δ is a parameter that controls the frequency of bitrate increases, with a slower increase for higher bitrates [13]. While this limiting can lead to less efficient resource utilization, it ensures a fair rate allocation by preventing any single client from suddenly hogging system resources through receiving a higher bitrate.

Incorporating client information: Since we allow each client to choose which information to send to the OneAPI server (cf. Section II-A's design principles), we do not explicitly specify how the server uses this information. Clients can choose to send information like their buffer memory and computation capability, or to send specific bitrate preferences. For example, if the current amount of buffered video (in the client) is relatively small or the client wishes to limit its mobile data costs, the client can specify an upper bound on its bitrate to quickly fill the buffer or limit the amount of data transferred. On the other hand, if a client decides to send its video clickstream data without specific preferences to the OneAPI server, the server might detect that the client is skimming the video (e.g., frequent clicks of forward/backward buttons) and select the minimum bitrate. Since we use an optimization framework to choose the bitrates, we can easily incorporate these client preferences as additional constraints.

III. FLARE Operations and Implementation

We now present our realization of Section II's design principles and rate adaptation algorithm. We first discuss FLARE's operations before describing our implementation.

A. FLARE Operations

Figure 1 shows the overall architecture on which FLARE's operations take place. FLARE's video streaming begins when a UE sends an HTTP GET Request message to request a Media Presentation Description (MPD) file and the media server sends the MPD file to the UE. The MPD is parsed to extract the available bitrates for each video flow, and the FLARE plugin at the UE client sends these bitrates to the OneAPI server, after removing any information that can be used to identify the video. The plugin can also send information on the clients' preferences or device/buffer status.

In each BAI, the OneAPI server runs Algorithm 1 to select each client's bitrate based on the client information, as well as the number of resource blocks available, RB assignment history, selected bitrate history, available video bitrates for clients' videos, video traffic policy, and user/buffer/device information. Each selected video bitrate is transferred to the FLARE plugin at the corresponding UE, and the video player uses the bitrates for selecting the next video segment. When transmitting the video bitrates, the OneAPI server communicates with the PCEF to set the GBR of each video flow according to its selected bitrate, thus providing stable service to the UE. Note that these message exchange procedures can be standardized by extending related existing standards for telecommunications APIs such as [23]; the detailed design of these protocols, however, is out of the scope of this paper.

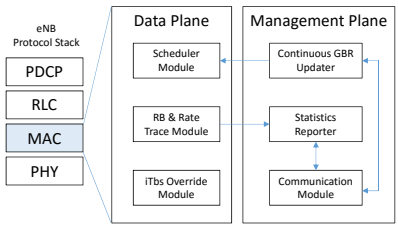
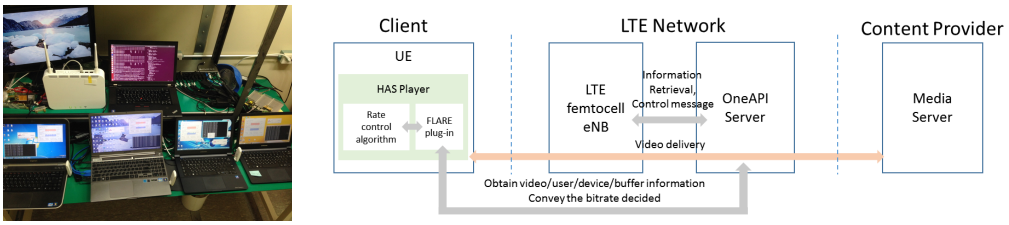


Fig. 3. New modules implemented in the LTE eNodeB.

B. LTE Femtocell Testbed Implementation

Our testbed setup is illustrated in Figure 2. It consists of four entities: UEs, an LTE femtocell eNodeB, a OneAPI server, and a media server. We modify the MPEG-DASH/Media Source demo player [19] to implement FESTIVE [13] and FLARE players. We develop an LTE base station using a commercial LTE femtocell eNodeB (eNB), JL-620 [24]. It supports 10 Mhz-bandwidth FDD operations on E-UTRA Band 7 [25] with 20 dBm transmission power, and 50 RBs are available per transmission time interval (TTI), which is 1 ms. We add the following modules in the medium access control (MAC) layer of the eNB, as illustrated in Figure 3:

- The **Scheduler Module** performs GBR-based per-TTI scheduling for video traffic in two phases. In Phase I, the scheduling algorithm uses GBR-based scheduling for video flows, and in Phase 2, the remaining RBs are allocated to both video and data flows with a legacy proportional fair scheduling algorithm.
- The **RB & Rate Trace Module** traces the RB and rate records for each video flow. The RB records are updated by the Scheduler Module. We use the rate records to calculate the transmitted bytes for FLARE operations.
- The **iTbs Override Module** allows us to emulate time-varying link bandwidth by changing the index of the Transport Block Size (iTBS). Each TBS index (iTbs) defines its own modulation and coding scheme [26], and hence this allows us to set the transmission rate.
- The **Continuous GBR Updater** dynamically changes the GBR, which is normally [27] assigned only when the traffic bearer is set up.
- The **Statistics Reporter** collects the RB and rate records of video flows from the RB & Rate Trace Module and sends them to the Communication Module.
- The **Communication Module** is located in the eNodeB and enforces GBR rates received from the OneAPI Server. It also sends a periodic statistics report (on the per-client RB utilization and throughput) to the OneAPI Server.



(a) Our FLARE testbed.

(b) Testbed architecture.

Fig. 2. Our FLARE testbed consists of UEs, an LTE femtocell base station, a OneAPI server, and a media server.

The OneAPI server receives the per-client RB utilization and throughput information from the eNB's communication module and the video information (e.g., video rate, duration) from the FLARE UE plugin when each video begins streaming. Based on the received information, it decides the bitrate (or GBR) for each video client using Algorithm 1 and sends the chosen bitrates to the eNodeB and FLARE plugin for enforcement in each BAI. We use KNITRO [28], a well-known solver for nonlinear optimization, to implement the bitrate calculation in the OneAPI server.

IV. Evaluation

We evaluate FLARE on the LTE femtocell testbed described in Section III-B before presenting simulation results that evaluate our bitrate selection algorithm in various environments. Our testbed evaluation (Section IV-A) compares our FLARE implementation to two client-side approaches, in both a static scenario and a dynamic scenario in which the network conditions change over time. Since the testbed considers only one network cell and thus cannot fully emulate UE mobility, we then use the ns-3 simulator [29] to compare FLARE's bitrate adaptation algorithm to both client- and network-side algorithms in mobile scenarios (Section IV-B). This simulation setup also allows us to investigate FLARE's bitrate adaptation algorithm in more detail. We find that, in both the testbed and simulation scenarios, FLARE yields higher average bitrates with fewer bitrate changes than the algorithms we compare against. Moreover, FLARE can efficiently calculate near-optimal bitrates, and its algorithm parameters can flexibly balance the relative priorities of data and video flows.

A. Testbed Performance Evaluation

Experiment setup. We compare FLARE with two client-side rate control algorithms: the MPEG-DASH/Media Source demo player, which we call GOOGLE [19], and FESTIVE [13]. We modify the MPEG-DASH/Media Source demo player to implement the FESTIVE and FLARE players. We implement tracing functions on the HAS player to log the segment transmissions, selected bitrates, and buffer status. GOOGLE makes two link bandwidth estimates, b^l and b^s, based respectively on the long- and short-term histories of recently received segments and selects the highest available video rate that is $\leq 0.85 \min\{b^l, b^s\}$. Unless otherwise stated, we solve the exact bitrate optimization problem (3–4).

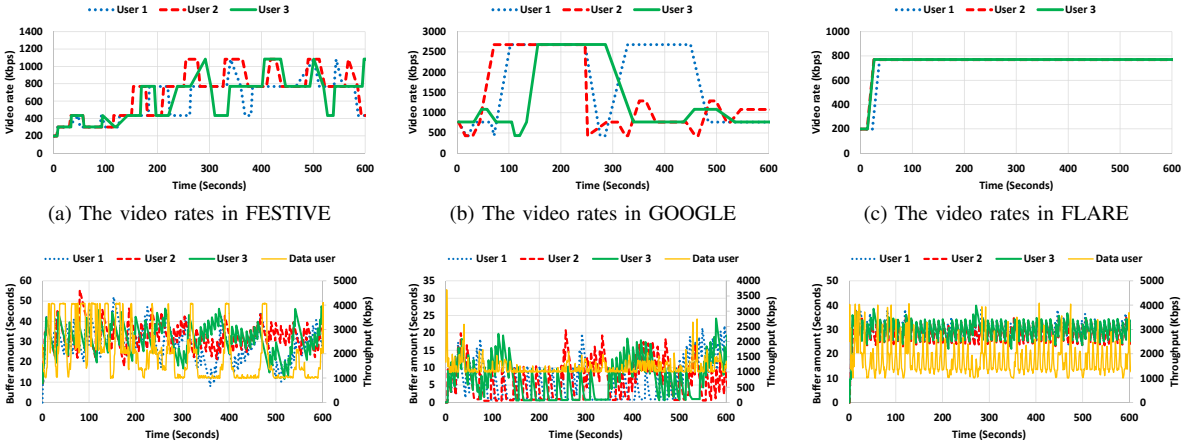
For our experiments, we encode a video into 200, 310, 450, 790, 1100, 1320, 2280, and 2750 Kbps. We generate one video flow each on three devices by running the HAS players, and one data flow on a device by running Iperf [30]. We consider a static scenario in which the Modulation and Coding Scheme (MCS) of each device does not change, and a dynamic scenario in which the MCS changes over time. For the static scenario, we set the iTbs value to 2. For the dynamic scenario, we change the MCS by gradually increasing the iTbs from 1 to 12 for the first 2 minutes, decreasing it back to 1 for the next 2 minutes, and repeating this cycle for 10 minutes. To model UE heterogeneity, each UE starts the cycle with a different offset. We use a commercial EPC emulator [31] to emulate signaling (e.g., bearer setup) with the core network entities.

TABLE I
Summary of the static scenario results.

	FESTIVE	GOOGLE	FLARE
Average video rate (Kbps)	638	1151	726
Average time that the buffer is underflowed (sec)	0	185.3	0
Average number of bitrate changes	20.3	9.7	1
Jain's fairness index of average video rates	0.998	0.990	0.999
Average throughput of data flow (Kbps)	2512	1140	1800

Stable rate selection in the static scenario. We first show the performance of FESTIVE, GOOGLE, and FLARE in the static scenario in terms of the video rates, buffer sizes of the video flows, and throughput of the data flow in Figure 4. FESTIVE selects the video rate conservatively, resulting in slow convergence to a stable state, and hence achieves a relatively large throughput for the data flow. Due to its unawareness of the overall radio status, video flows' bitrates change frequently. On the other hand, GOOGLE selects the video rate aggressively, causing frequent re-buffering interruptions whenever the amount of buffered video data falls below 1 second. GOOGLE assigns the fewest radio resources to the data flow, yielding lower overall data flow throughput.

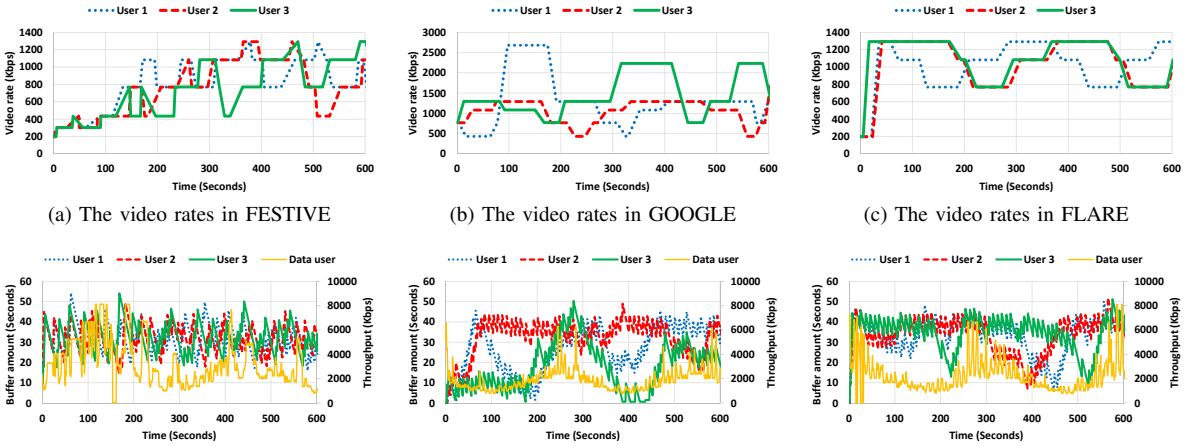
Unlike FESTIVE and GOOGLE, FLARE, as shown in Figure 4c, selects the video rates in a stable manner. FLARE initially increases the video bitrate conservatively, and then consistently chooses a video rate of 790 Kbps while maintaining stable buffer sizes during the measurement period. The data flow also receives a stable, moderately high throughput. Table I summarizes the static scenario's results. While FLARE achieves a slightly lower throughput for the data flow, it dramatically reduces the number of bitrate variations and buffer



(a) The video rates in FESTIVE (b) The video rates in GOOGLE (c) The video rates in FLARE

(d) Video flows' buffers and data flow's throughput in FESTIVE (e) Video flows' buffers and data flow's throughput in GOOGLE (f) Video flows' buffers and data flow's throughput in FLARE

Fig. 4. FESTIVE, GOOGLE, and FLARE are compared in terms of the video rates, buffered amounts of the video flows, and throughput of the data flow in the static scenario. FLARE selects the bitrate very stably, and maintains a larger buffer, compared to FESTIVE and GOOGLE.



(a) The video rates in FESTIVE (b) The video rates in GOOGLE (c) The video rates in FLARE

(d) Video flows' buffers and data flow's throughput in FESTIVE (e) Video flows' buffers and data flow's throughput in GOOGLE (f) Video flows' buffers and data flow's throughput in FLARE

Fig. 5. FESTIVE, GOOGLE, and FLARE are compared in terms of the video rates, buffered amounts of the video flows, and throughput of the data flow in the dynamic scenario. FLARE selects bitrates similar to the link capacity, with less fluctuation, compared to FESTIVE and GOOGLE.

TABLE II
SUMMARY OF THE DYNAMIC SCENARIO RESULTS.

	FESTIVE	GOOGLE	FLARE
Average video rate (Kbps)	839	1297	1025
Average time that the buffer is underflowed (sec)	0	10.7	0
Average number of bitrate changes	22.7	14	11.3
Jain's fairness index of average video rates	0.998	0.997	0.998
Average throughput of data flow (Kbps)	3870	1870	2300

underflows compared to FESTIVE and GOOGLE respectively.

Fast bitrate adaptation in the dynamic scenario. We next compare FESTIVE, GOOGLE, and FLARE in the dynamic scenario, as shown in Figure 5. FESTIVE shows a similar pattern to that of the static scenario: the video bitrates oscillate frequently and with little visual correlation with the two-minute MCS cycles, while the data flow throughput also changes frequently but is high on average (Figures 5a and 5d). GOOGLE shows less bitrate oscillation, but more frequent rebuffering, as in the static scenario (Table I). Indeed, these results incorporate changes that we have made to reduce GOOGLE's frequent rebuffering: we send an HTTP GET request for the next segment when the remaining buffered video length reaches 40 seconds instead of the static scenario's 15 seconds. FLARE, however, does not require rebuffering and rapidly adapts its bitrates to the network conditions. As shown in Figure 5c, the changes in bitrates for FLARE are similar to the MCS change pattern configured for the dynamic scenario. Table II summarizes the results in the dynamic scenario; only GOOGLE suffers from rebuffering, while FESTIVE yields the highest data throughput but more bitrate changes than FLARE.

Even though FLARE does not directly consider the buffered amounts of video when selecting the bitrates, it never causes

TABLE III
SIMULATION SETTINGS.

Simulator	ns-3 3.18.1
Simulation time	1200 seconds
Territory	2000m x 2000m
Number of video UEs	8 (32 to 128 for the experiments in Figure 9)
Placement of UEs	random
Fading model	trace based model
Video segment duration $= B$	10 seconds
Video bitrates	100, 250, 500, 1000, 2000, 3000 Kbps (100, 200, ..., 1200 Kbps for the experiments in Figures 8 to 10)
TCP	Westwood
Scheduler	Priority Set Scheduler [32]

TABLE IV
DEFAULT PARAMETER VALUES FOR FLARE, FESTIVE, AND AVIS (α AND δ ARE TAKEN FROM FIGURES 11 AND 12, θ_u AND β_u FROM [16], AND FESTIVE AND AVIS PARAMETERS FROM [13], [15]).

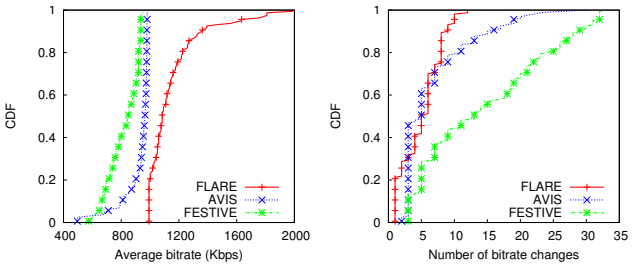
FLARE				FESTIVE			AVIS	
α	δ	θ_u	β_u	k	p	α	α	W
1.0	4	0.2 Mbps	10	4	0.85	12	0.01	150

a buffer underflow, even in the worst channel condition (cf. Table II). We believe that this is due to two reasons. First, FLARE considers the channel status of each UE in its optimization framework; thus, it is less likely to request segments whose bitrates are larger than the available bandwidth. Second, while video segments are serviced with the GBR, the data traffic is serviced with non-GBR in FLARE. Thus, the Scheduler Module can opportunistically use the RBs of data traffic for video flows when the OneAPI server's bitrate optimization algorithm cannot keep pace with wireless link dynamics.

B. Simulation of Bitrate Adaptations

Simulation setup. We finally conduct an extensive simulation study on our proposed solution, FLARE, using ns-3 [29] with the LTE module. The simulator will allow us to explore FLARE's bitrate adaptation algorithm for richer mobility settings and different video and data flow requirements, compared to our femtocell testbed. For comparison, we also evaluate a client-side algorithm (FESTIVE [13]) and a network-side algorithm (AVIS [15]). For AVIS, we run a simple rate adaptation algorithm on a UE that requests the highest possible rate based on the estimated throughput, and set the GBR/MBR using the scheduler in the BS instead of resource slicing techniques.

Table III summarizes our simulation settings. We modify the Priority Set Scheduler module in ns-3 to add the MBR assignment and to retrieve information about each client's assigned RBs and transmitted bytes. We set the parameters for each scheme as shown in Table IV. In most cases, there are 8 clients in each run, and we carry out 20 runs for each plot. Since HAS operates on top of TCP, we do not use traditional performance metrics like the Peak Signal-to-Noise Ratio (PSNR), which is



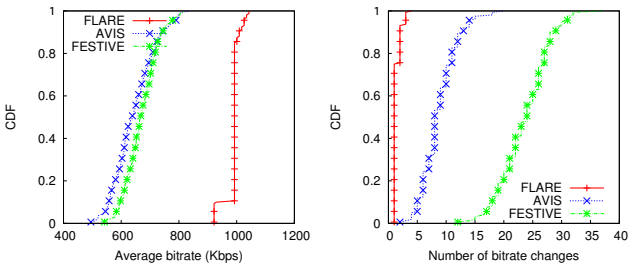
(a) Average client bitrate values. (b) Rate changes per client.

Fig. 6. Performance CDFs in static scenarios over 160 clients. FLARE shows higher average bitrates and better stability than AVIS or FESTIVE.

mostly useful in lossy environments with UDP. Instead, we use the average bitrate, number of bitrate changes, and Jain's fairness index (for actually transmitted bitrates), which more accurately reflect users' qualities-of-experience in HAS.

Stable, high bitrates in static scenarios. We first compare our ns-3 results to the testbed implementation by considering a static scenario with stationary UEs. Figures 6a and 6b show the CDFs (cumulative distribution functions) of the average bitrate values and numbers of bitrate changes over 160 clients for each scheme. FLARE's average bitrates respectively exceed those of AVIS and FESTIVE by 24% and 39%, with 26% and 66% fewer bitrate changes than AVIS and FESTIVE, which is consistent with the testbed's static scenario results in Table I.

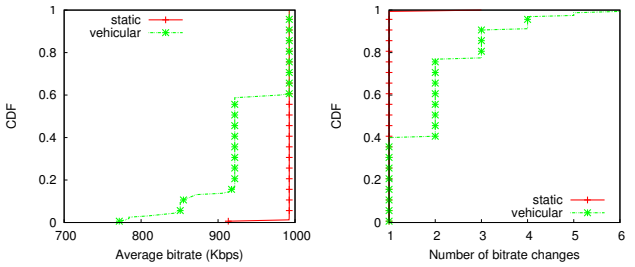
In AVIS, the network sets only the GBR/MBR, while the rate controller in the UE selects the actual video bitrate, resulting in frequent mismatches between the bitrates set by the network and the ones selected by the UEs. FESTIVE performs worse than the others due to its unawareness of the link conditions in a cell. FLARE's fog computing approach eliminates both of these shortcomings. The average Jain's fairness index is comparably high at 0.989, 0.989, and 0.986 for FLARE, AVIS, and FESTIVE, respectively, meaning that all three schemes are very fair across time.



(a) Average client bitrate values. (b) Rate changes per client.

Fig. 7. Performance CDFs in the mobile scenarios over 160 clients. The stability performance difference between FLARE and the other schemes is larger than that of the static scenario (Figure 6b).

Stable bitrate selection in mobile scenarios. We next consider mobile scenarios where the UE operates in vehicles. FLARE's advantages are even more pronounced in this scenario compared to the static one, as shown in Figures 7a and 7b. FLARE shows a 53% and 47% improvement in the average bitrate compared to AVIS and FESTIVE, and the average numbers of rate changes decrease by 85% and



(a) CDF of the average bitrate values over 160 clients.

(b) CDF of the numbers of rate changes over 160 clients.

Fig. 8. FLARE with continuous bitrate optimization. The average bitrate is reduced by less than 15% for the two scenarios, and the stability is retained.

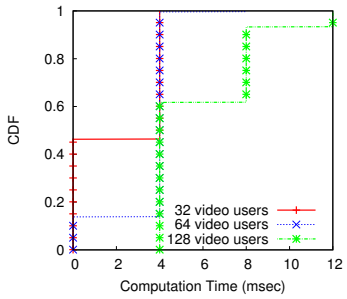
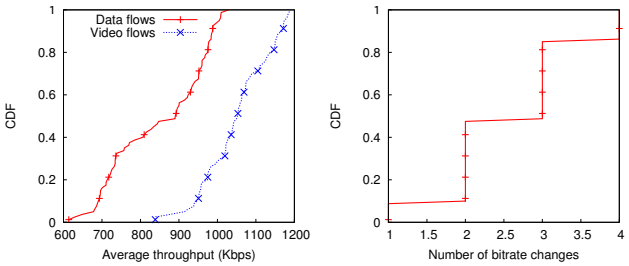


Fig. 9. CDFs of computation times for the bitrate selection with 32, 64, and 128 video clients in a cell. The computation time increases with the number of video clients, but remains much smaller than a segment duration.

95% compared to those of AVIS and FESTIVE, respectively. FLARE's bitrate stability constraints in Algorithm 1 ensure that rate increases occur less frequently (than FESTIVE and AVIS) in the presence of the mobile scenarios' highly varying link bandwidths, which results in conservative use of the wireless resources compared to the static scenario (Figure 6). The average fairness indices, however, show a similar pattern to the static case: 0.999, 0.988, and 0.993 for FLARE, AVIS, and FESTIVE, respectively.

Scalable bitrate selection. We now evaluate the optimality and scalability of FLARE's bitrate selection. Figures 8a and 8b show the bitrates chosen when we use the continuous relaxation approximation to solve (3–4) in Algorithm 1. The average throughput is 14% and 6% less for the static and mobile scenarios compared to those in the original FLARE algorithm. The stability varies but is generally retained: the number of bitrate changes decreases by 80% in the static scenario, but increases by 48% in the mobile scenario (while remaining under 6). The average Jain's fairness indices are 0.999 and 0.997 for the static and mobile scenarios, respectively. The computation time for each bitrate assignment is less than 4 ms for most cases (7135 out of 7140). Only 5 cases took around 4 ms, which is negligible compared to a segment duration on order of seconds.

As the number of video clients increases, we find that the computation times for the bitrate selection algorithm remain low. We plot the CDFs of the computation times with 32, 64, and 128 video clients in a cell, respectively, in Figure 9. Due to coarse-grained measurement, some computation times are plotted as zero. The computation time increases as the number of video clients increases, but remains much smaller (up to 12 ms) than a segment duration (1-10 seconds).



(a) CDF of the throughput of video and data flows in FLARE.

(b) CDF of the numbers of bitrate changes for video flows in FLARE.

Fig. 10. When video and data flows coexist, FLARE balances the throughput of video and data flows while maintaining bitrate and throughput stability.

Flexibility to different video and data flow requirements. We finally investigate the throughput balance between video and data flows with FLARE. We simulate 8 video and 8 data clients and show the resulting bitrates and throughputs in Figure 10a. There is no noticeable difference in the number of video bitrate changes compared with our previous experiments, as shown in Figure 10b. FLARE consistently prioritizes video flows over data flows.

FLARE can adjust to different data and video flow priorities by changing the value of α, which balances the throughput of data flows compared to video flows (cf. (3) in Section II-B). As α increases from 0.25 to 4, Figure 11 shows the average throughput and standard deviation for each type of flow. We observe that the average throughput of the data flows smoothly increases and that of the video flows decreases as α increases.

We finally examine the effect of the parameter δ on the relative priorities of achieving a higher bitrate versus bitrate stability. Recall that in Algorithm 1, recommended bitrate increases are not used until the next higher bitrate index is selected for $\delta(L_u^{i-1}+1)$ BAIs. To see the impact of different values of δ in practice, we increment δ from 1 to 12 in our experiments. Figure 12 shows the average bitrate and number of bitrate changes as δ varies. Overall, the average bitrate decreases as δ increases, since a higher δ means that the rate is increased more conservatively. Likewise, the stability increases as δ increases, indicating that FLARE can successfully adjust to different bitrate selection criteria.

V. Conclusions

In this paper, we propose a new HAS technique called FLARE that uses a fog computing approach to address existing HAS techniques' lack of coordination between clients and networks. In FLARE, a network entity and clients cooperate to decide the bitrate of a video flow. FLARE takes advantage of existing telecommunications APIs to minimize its required modifications at the network core, while minimizing changes at the client by embedding a plugin module on the HAS video player. FLARE optimizes the total utility of video and data flows in a cell, while stable in the video quality. Experimental evaluations with ns-3 and an LTE femtocell implementation demonstrate that FLARE significantly enhances the average

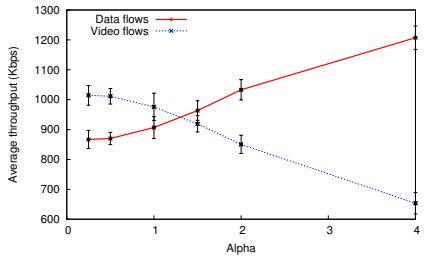


Fig. 11. Average flow throughputs with different α values, showing the tradeoff between data flows' throughputs and video flows' bitrates.

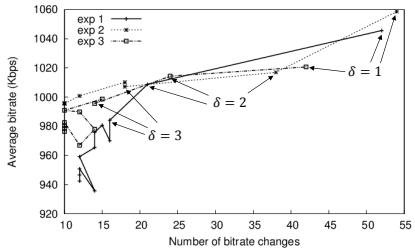


Fig. 12. Average client bitrate with different δ values. The average bitrate decreases as δ increases, i.e., the bitrate becomes more stable.

throughput (or bitrate) and stability of video quality compared to state-of-the-art client-side and network-side solutions (FESTIVE and AVIS), while flexibly balancing video and data flows according to their relative priorities.

In a practical deployment, FLARE can coexist with conventional HAS players by servicing their traffic like other data traffic without any bitrate guarantees; traffic from these HAS players would then not interfere with FLARE. Users would then have an incentive to adopt FLARE in order to receive GBR video rates. Moreover, FLARE can be easily extended to uplink video streaming with minor modifications. Thus, it represents a practical HAS techniques that not only improves on existing HAS techniques, but also demonstrates the feasibility of fog computing approaches to network applications.

ACKNOWLEDGMENTS

This work was partly supported by NSF grant CNS-1525435, an Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korean government (MSIP) (B0190-16-2013, Development of Access Technology Agnostic Next-Generation Networking Technology for Wired-Wireless Converged Networks), and the research fund of Hanyang University (HY-2017-N). The snu-samsung smart campus research center at Seoul National University provides research facilities for this study.

REFERENCES

[1] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. PP, no. 99, 2016.
[2] "Openfog architecture overview," White Paper, OpenFog Consortium Architecture Working Group, Feb. 2016, https://www.openfogconsortium.org/wp-content/uploads/OpenFog-Architecture-Overview-WP-2-2016.pdf.
[3] "Making Money Through API Exposure," http://www.oracle.com/us/industries/communications/comm-making-money-wp-1696335.pdf.
[4] "OMA OneAPI Profile of RESTful Network APIs V4.0," http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oneapiprofilerest-v4-0.
[5] "MPEG DASH standard." http://dashif.org/mpeg-dash.
[6] "Adobe HTTP Dynamic Streaming." http://www.adobe.com/products/hds-dynamic-streaming.html.
[7] "Apple HTTP Live Streaming." https://developer.apple.com/streaming.
[8] "Microsoft Smooth Streaming." http://www.iis.net/downloads/microsoft/smooth-streaming.
[9] R. Hackett, "Most Internet traffic will be encrypted by year end. Here's why." Fortune, 2015, http://fortune.com/2015/04/30/netflix-internet-traffic-encrypted/.
[10] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 719–733, 2014.
[11] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 325–338, 2015.
[12] X. Xie, X. Zhang, S. Kumar, and L. E. Li, "pistream: Physical layer informed adaptive video streaming over lte," in *ACM MOBICOM*. ACM, 2015, pp. 413–425.
[13] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," in *ACM CoNEXT*, 2012.
[14] G. Tian and Y. Liu, "Towards agile and smooth video adaptation in dynamic HTTP streaming," in *ACM CoNEXT*, 2012.
[15] J. Chen, R. Mahindra, M. A. Khojastepour, S. Rangarajan, and M. Chiang, "A scheduling framework for adaptive video delivery over cellular networks," in *ACM MOBICOM*, 2013.
[16] D. De Vleeschauwer, H. Viswanathan, A. Beck, S. Benno, G. Li, and R. Miller, "Optimization of HTTP adaptive streaming over mobile cellular networks," in *IEEE INFOCOM*, 2013.
[17] S. Cicalo, N. Changuel, R. Miller, B. Sayadi, and V. Tralli, "Quality-fair http adaptive streaming over lte network," in *IEEE ICASSP*. IEEE, 2014, pp. 714–718.
[18] W. Pu, Z. Zou, and C. W. Chen, "Video adaptation proxy for wireless dynamic adaptive streaming over http," in *2012 19th International Packet Video Workshop (PV)*. IEEE, 2012, pp. 65–70.
[19] "MPEG-DASH / Media Source demo." http://dash-mse-test.appspot.com/.
[20] M. Uchida and J. Kurose, "An Information-Theoretic Characterization of Weighted alpha-Proportional Fairness," in *IEEE INFOCOM*, 2009.
[21] N. Cranley, P. Perry, and L. Murphy, "User perception of adapting video quality," *International Journal of Human-Computer Studies*, vol. 64, no. 8, pp. 637–647, 2006.
[22] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, "A quest for an internet video quality-of-experience metric," in *ACM HotNets*, 2012.
[23] "OMA RESTful Network API for Quality of Service V1.0," http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-restful-network-api-for-quality-of-service-v1-0.
[24] "JL-620 LTE Enterprise Indoor Small Cell," http://www.juniglobal.com/products/products_lte_jl620_d.asp.
[25] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception," Sep. 2009, Rel-9 v9.1.0. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/36101.htm
[26] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures," Sep. 2009, Rel-8 v8.8.0. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/36213.htm
[27] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2," Sep. 2009, Rel-9 v9.1.0. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/36300.htm
[28] "KNITRO," http://www.ziena.com/knitro.html.
[29] "ns-3." http://www.nsnam.org/.
[30] "Iperf," http://sourceforge.net/projects/iperf/.
[31] "Accelerate Development of LTE Evolved Packet Core Products with Aricent Solutions." http://www.aricent.com/pdf/Aricent_Solution_Brief_LTE_EPC.pdf.
[32] G. Monghal, K. I. Pedersen, I. Z. Kovacs, and P. E. Mogensen, "QoS oriented time and frequency domain packet schedulers for the UTRAN long term evolution," in *IEEE VTC*, 2008.