FlexStream: Towards Flexible Adaptive Video Streaming on End Devices using Extreme SDN

Ibrahim Ben Mustafa
Old Dominion University
iben@cs.odu.edu

Tamer Nadeem*
Virginia Commonwealth University
tnadeem@vcu.edu

Emir Halepovic
AT&T Labs – Research
emir@research.att.com

ABSTRACT

We present FlexStream, a programmable framework realized by implementing Software-Defined Networking (SDN) functionality on end devices. FlexStream exploits the benefits of both centralized and distributed components to achieve dynamic management of end devices, as required and in accordance with specified policies. We evaluate FlexStream on one example use case – the adaptive video streaming, where bandwidth control is employed to drive selection of video bitrates, improve stability and increase robustness against background traffic. When applied to competing streaming clients, FlexStream reduces bitrate switching by 81%, stall duration by 92%, and startup delay by 44%, while improving fairness among players. In addition, we report the first implementation of SDN-based control in Android devices running in real Wi-Fi and live cellular networks.

KEYWORDS

SDN; DASH; ABR; video streaming

ACM Reference Format:
Ibrahim Ben Mustafa, Tamer Nadeem, and Emir Halepovic. 2018. FlexStream: Towards Flexible Adaptive Video Streaming on End Devices using Extreme SDN. In *2018 ACM Multimedia Conference (MM '18), October 22–26, 2018, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3240508.3240676

1 INTRODUCTION

HTTP Adaptive Streaming (HAS) is the dominant approach to deliver over-the-top video today. The adaptive aspect of HAS allows the same content to be encoded in multiple quality levels so it can serve a variety of client devices with different screen sizes, capabilities, and network conditions. HAS exploits the existing Content Delivery Network (CDN) infrastructure to provide video content to applications, which adapt by requesting the video quality appropriate for their network conditions.

Unfortunately, HAS comes with several drawbacks, especially in mobile environments [1]. HAS clients (players) are shown to exhibit instability, stalls and poor quality when they compete over the bottleneck link [6, 8]. While this issue is applicable to all network environments, including stable links, it is especially common on cellular links, where radio channel conditions and throughput are highly variable.

Another issue, specific to devices with smaller screens, such as smartphones on cellular network, is that the video players are greedily trying to reach the highest available quality, even when that is not needed to provide an excellent Quality of Experience (QoE) [27, 32]. Since high video quality requires high bitrates, this leads to data waste, both in terms of excess image quality imperceptible on small screens, and depletion of users' data allowance, typical in cellular networks. Finally, rapid increase in overall video demand (projected to become 78% of total mobile data traffic [3]) can lead to overall QoE unfairness and sub-optimal experience for majority of users.

While different approaches were proposed to address some of these issues, they are typically invasive (require all players to follow specific adaptation logic), specific to adaptive video (not generic enough to serve other applications and services), or require large and costly special-purpose network infrastructure [12, 23, 24, 27, 41]. They also generally fail to provide a comprehensive and flexible solution due to a lack of global view of the network or the relevant part of the network (e.g., a bottleneck). Considering the current variety of devices, video service designs, CDNs, players with their adaptation algorithms, and user needs, network is the only entity with the "big picture" view that can optimally manage multiple video clients competing for bandwidth. Therefore, a comprehensive video management solution needs to involve the intelligent component in the network.

To that end, we design, implement, and evaluate FlexStream, a framework that leverages: (i) a *centralized* or *edge* manager component in the network that specifies a policy controlling resource allocation (e.g., bandwidth), and (ii) a *distributed* SDN component, which implements that policy via Open vSwitch (OVS), essentially offloading the fine-grained functionality to the end device. We refer to these SDN components on mobile end devices as ***extreme SDN*** [1], *to differentiate them from the traditional SDN used in the network core. In our approach, the proposed extreme SDN works independently and without any collaboration or support from network core SDN.*

FlexStream considers the growing trend in embracing Bring Your Own Device (BYOD) approach, in which the network can have some level of control over the end hosts, for example to enable network policies and bring additional services to the client devices [13]. In addition, FlexStream assumes that the policy manager could be operated by either content providers, network operators or individuals. FlexStream supports various management policies based on the different contexts of the user, device, video, network, and

*Work done while author was affiliated with Old Dominion University.

MM '18, October 22–26, 2018, Seoul, Republic of Korea

ACM ISBN 978-1-4503-5665-7/18/10…$15.00
https://doi.org/10.1145/3240508.3240676

[1] Recently, "extreme" has been adopted in networking and mobile computing community to refer to network edge devices such as wireless access points or user devices [10]. We adopt this definition and use "extreme SDN" for the SDN capability that is deployed and used on end devices (e.g. smartphones, tablets, etc.).

environment (e.g., user priority-based or device screen size-based policy), which is often overlooked in practical implementations. Using an optimization function for adaptive video use case, we show that all these factors can be effectively accounted for within the policy that allocates bandwidth to devices.

The benefits of FlexStream approach are to: (i) support open and flexible framework to implement network policies on a set of end devices, such as those using the same application, have the same owner (a "fleet"), or have a specific temporary role on the network (BYOD); (ii) offload intrusive, costly or resource-demanding tasks from the network to end devices, enabling more intelligent management and promoting users privacy, (iii) implement network policies when an end host becomes a network node, (e.g., smartphone becomes a Wi-Fi hotspot); and possibly many other functions.

We implement FlexStream on commodity Android devices and evaluate it using realistic scenarios in Wi-Fi and cellular networks, using a real video player and a policy manager in the cloud.

Our contributions can be summarized as follows:

- We develop FlexStream, a framework that uses SDN functionality on mobile end devices (i.e., extreme SDN), allowing for fine-grained management of bandwidth based on real time context-awareness and specified policy.
- We demonstrate that FlexStream can be used to manage video delivery for a set of end devices over Wi-Fi and cellular links and can effectively alleviate common problems such as player instability, playback stalls, large startup delay, and inappropriate bandwidth allocation.
- We define an optimization method to practically improve video QoE considering context information, such as screen size and user priority, and validate it using real experiments, including reductions in quality switching by 81%, stalls by 92%, and startup delay by 44%.
- We introduce, to the best of our knowledge, the first working OVS implementation on commodity mobile devices that runs in both Wi-Fi and cellular networks.

The rest of the paper is organized as follows. Section 2 provides a background on HAS and SDN. We present the FlexStream overview in Section 3, architecture in Section 4, implementation in Section 5, and evaluation in Section 6. Section 7 concludes the paper.

2 BACKGROUND AND RELATED WORK

HTTP Adaptive Streaming (HAS). In HAS, videos are encoded into multiple quality profiles, determined by bitrate, screen resolution, frame rate, and other encoding parameters. A manifest file describing the profiles is downloaded by the player when starting the streaming session. Each profile is split into segments, typically 2 to 10 seconds long, which are served by conventional web servers via HTTP. Streaming players have the opportunity to switch between profiles, typically at segment boundaries.

Multiple bitrates and segmentation allow players to adapt to varying network conditions, typically using throughput measurements and buffer state, in order to maximize the user experience. As players try to maintain a full buffer of pre-fetched segments, the sequential segment downloads naturally create the intermittent traffic pattern (ON/OFF periods). This introduces a major challenge for competing players to accurately estimate the available bandwidth, as any one player may perceive drastically different network conditions depending on whether it competes with other players during a segment download [6]. This behavior is the key factor of unstable video quality for most commercial HAS players today.

TCP behavior underneath HTTP(S) leads to another unintended effect. As TCP flows from competing players attempt to equally share the bottleneck, they converge towards the same throughput and video bitrate. This may not always be desirable, given that player requirements may differ due to the variations in screen size, type of content, or user preferences and priorities.

Software-Defined Networks. Software-Defined Networks (SDN) is invented to improve control and management of data networks, as well as reduce cost and spur network service innovation [33]. SDN introduces a layered network architecture in which the control plane (i.e., control function such as routing, security, etc.) is decoupled from the data plane (forwarding function) of the network devices (e.g., switches), allowing for more sophisticated and flexible traffic management. Network applications can explicitly and directly communicate their network requirements and desired behavior to the control plane via APIs utilizing a *northbound* interface. The control plane is responsible for accomplishing the goals of these applications by manipulating the forwarding devices as well as providing the applications with a holistic view of the network state. The *southbound* interface of the SDN switch enables the control plane to communicate with the data plane via a shared protocol, *OpenFlow*. The data plane handles the actual packets according to the configuration received from the control plane and performs other tasks such as reporting.

In the SDN-enabled switch, such as Open vSwitch (OVS) [38], the forwarding table is extended to the *flow table* that stores flow rules defining different actions (e.g., forwarding or dropping packets, modifying header fields, etc) to be applied on the incoming packets. Although SDN is typically deployed in data centers [22] and core networks [18], we develop and implement an OVS-based SDN functionality on end devices, which we refer to as *extreme SDN*. The concept of extending SDN to the wireless edge and end devices is proposed as a mobile extension of SDN (*meSDN*) [28], to enable Wi-Fi LAN virtualization. Similarly, *TrafficVision* is proposed to enhance visibility into the network traffic [43].

FlexStream draws inspiration from these works and leverages the OVS actions to perform fine-grained control over corresponding application flows on end devices to optimize video QoE. For example, FlexStream utilizes OVS action for packet-header modification on a streaming flow to control data rate, while it may use the *drop* action when the policy dictates to completely block a background flow at specific times.

Related Work. Most prior works address HAS player performance issues as client-based solution by improving adaptation algorithms of video players [20, 23, 29, 30]. These algorithms are either throughput-based [34], buffer-based [19, 20], or hybrid (using rate and buffer) [29] used to avoid the impact of throughput variations and thus stabilize the quality. However, a very recent study shows that these techniques fail to address the main issues, including stability, stalls and fairness [9]. Moreover, a pure client-based solutions lack the flexibility to enable network policy implementation and achieve specific requirements.

Server-based solutions are also proposed to overcome instability and improve fairness, such as adjusting the sending rate at

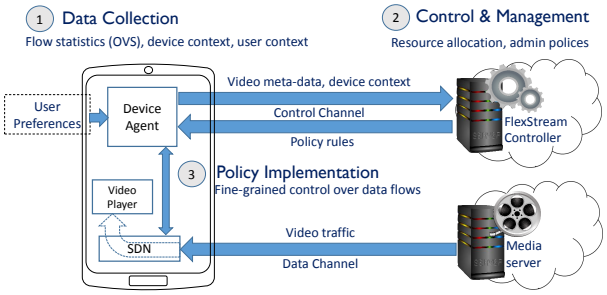

Figure 1: FlexStream overview.

the server to nearly match the requested bitrate and avoid the OFF periods [7]. A similar idea, but involving both client and server modification, is proposed to improve the video QoE or reduce the cost (e.g., battery consumption) with negligible QoE degradation [17]. However, such solutions are agnostic to network conditions, can not provide real-time bitrate adaptability, and incur significant overhead to the server.

Consequently, several network-based solutions are proposed to improve performance. For instance, flow shaping is applied at the router or DASH-aware proxy, including explicitly rewriting HTTP requests [12, 27]. When multiple bottlenecks exist, a set of coordinating proxies is proposed as a mitigating solution, which also includes information sharing between clients [37]. SDN-assisted solutions propose orchestrating network elements that either explicitly recommend bitrates to video players or dynamically control the traffic from within the network [11, 15, 16, 26]. However, these proposals require intrusive (URL rewriting), costly (bandwidth-shaping network elements), or infeasible in practice (CDN edge server changes) approaches to detect and manage video QoE. In addition, most of these proposals are application-dependent and assume the player can interact with these systems. In contrast, FlexStream provides a practical and feasible approach that works without requiring player interactions nor SDN support from the wireless infrastructure. Extreme SDN approach adopted by FlexStream allows for offloading several expensive tasks from the network to end devices which makes it an inexpensive solution for network operators to adopt. Moreover, unlike existing proposals, FlexStream leverages end device capabilities to support important features such as real time adaptability and context-awareness.

Finaly, a centralized control plane [14, 35], deployed on top of multiple CDNs, is proposed to assist players in selecting the optimal video bitrate in addition to optimizing video delivery in CDNs. These goals are achieved via buffer state and throughput feedback from a player plug-in to the controller that makes decisions. Although these works conclude that a centralized controller can help to improve QoE, such solution, in contrast to our work, is not designed to manage a group of video players sharing a bottleneck link. Consequently, the optimal QoE and fair share of network resources cannot be achieved, as these solutions are unaware of the underlying network conditions.

3 FLEXSTREAM OVERVIEW

Figure 1 shows the system overview of FlexStream. At a high level, FlexStream consists of centralized and distributed components deployed in the network and end devices, respectively. The centralized FlexStream Controller (FC) uses information periodically provided by end devices to manage resource allocation based on defined policies. Monitoring and policy implementation tasks are offloaded to end devices themselves, via lightweight software agents referred to as Device Agents (DAs). This alleviates the need for network-wide, intrusive and costly traffic management solutions or impractical modifications to servers.

The DA opens a *control channel* to the FC, dedicated for reporting device state and context to the FC, and delivery of optimization policy decisions to the DA. In HAS use case, the process of managing video sessions starts with the DA collecting and reporting information related to the video session, device characteristics, and context information. FC utilizes this information to detect bottlenecks and take control over network resources, if necessary. The FC uses the received information as input to an optimization policy that achieves the target objective. For HAS, this means maximizing QoE by allocating the highest sustainable bitrates to video sessions while ensuring stability, fairness and minimal stalls. Bitrate allocation along with some dynamic policy rules, such as location or time context restrictions, are then sent to DAs for implementation, using the deployed SDN data plane (extended OVS) and the control plane (Local Controller (LC)) on end devices (i.e., *extreme SDN*). OVS is installed with a new action added to the kernel module that compels the device to adjust the bandwidth consumption to a certain value, as directed by the LC according to the policy, to avoid player competition and improve video experience.

The FC could be deployed in the public or network operator clouds, or at the network edge. It is envisioned that the FC could be operated by content providers, network operators, or individuals in their private networks. Deployments at the wireless access points for homes or enterprises, or at mobile edge nodes, would give FC a better view of network conditions and device contexts, enabling it to manage bandwidth over the shared bottlenecks more effectively. FlexStream does not require the bottleneck to be the last mile or wireless link, and it works the same way for devices it controls that share any bottleneck. It assumes knowledge of which devices share a bottleneck so the policy can be applied to that set of devices.

FlexStream takes different context information as inputs to the global policy implemented inside FC. Context information can be classified into: (i) user context: priority class, preferences, and location, (ii) device context: screen size and battery level, (iii) network context: link condition and traffic mix, and (iv) environment context: surrounding luminance. In fact, considering context information is crucial for achieving high and balanced QoE in the network. For example, if multiple users have two different priorities (e.g., high and regular class) while streaming over a shared bottleneck, allocating more resources to those with high priority (e.g., emergency services) to ensure higher video quality is essential for enabling service differentiation.

4 FLEXSTREAM ARCHITECTURE

Figure 2 shows the main system components and their key modules, whose roles and operations are described next.

4.1 End-Device Components

There are three main components deployed on the end device: the DA, LC, and OVS. The DA runs in the background and oversees the

functions of several modules, in addition to mediating communication between local components and the FC. The DA also listens to all important events (e.g., streaming start, bitrate switch) and monitors local context related to the video stream, and promptly informs the FC for an appropriate action. To reduce overhead, the DA runs every 2 seconds, which is usually the minimum interval between two consecutive video segment requests generated by HAS players in the steady state [8]. At each run, the DA sends an update to the FC. Alternatively, updates could be triggered by context changes (e.g., throughput drop). Even with periodic updates, the FC can react quickly and before QoE is impacted in practice, since most HAS players' buffers are at least around 30 seconds [5, 20].

The DA consists of several modules. The *Context Monitor* is initially responsible for observing and reporting a streaming event by combining data from the `netstat` log and *HTTP Inspector*. The main task of *HTTP Inspector* is to report video encoding rates by inspecting the manifest file prior to the streaming session. In some cases, it also inspects the HTTP requests sent by the player to learn the bitrates of the newly requested segments. If the session is encrypted, then the *Bitrate Estimator* is invoked to estimate the bitrates from packet-level traffic (via SDN components). Recently proposed estimation techniques are suited for this purpose [31, 42].

In addition, *Context Monitor* oversees the device context which may contain both physical device characteristics, user preferences and administrative context, and reports it to the FC. Once the player starts streaming, the video quality is monitored by *QoE Monitor* module. To reduce overhead, the *QoE Monitor* is only required to periodically check the average throughput (through SDN components) to ensure the sustainability of the current bitrate. This eliminates the need for the *HTTP Inspector* to constantly inspect each HTTP request for the requested bitrate, as long as the average throughput does not fall below the target bitrate. The *QoE Monitor* is required to inform the FC about any switch in the requested video bitrate.

At the higher level, the *Policy Engine* is responsible for maintaining and implementing the policies received from the FC. This is achieved by instructing the LC to install new actions in OVS flow table. Note that these policies are dynamic and programmed based on the context. For instance, the FC can restrict streaming HD videos (due to high bitrate) at specific times or locations (base stations), or set the bitrate of background traffic to a different value based on the application. Adjustment of TCP receive window (*rwin*) is one of the techniques that we use to limit the TCP connection throughput. Thus, the *Rate Handler* module's role is to derive the appropriate *rwin* from Round Trip Time (RTT) of video stream packets either upon a request or when there is a significant change in RTT that could impact throughput (± 100 Kbps).

To modify the TCP packet *rwin* in the data plane, we leverage OVS. In FlexStream, OVS uses the extended OpenFlow protocol to receive the *rwin* modification action and then insert it into the action field entry in the flow table. Once a match in the flow table is detected, then the *rwin* field in the TCP header of the matched packet is modified to the received value. OVS is also leveraged to provide other functions including routing traffic between different network interfaces and collecting flow-based statistics. In our implementation, when a background flow (e.g., an app update) starts competing with video flows, FlexStream may instruct the OVS to route the background traffic over another network interface, if possible (according to user preferences). Finally, to control OVS from the user space, we employ the *Local Controller (LC)* as a separate component from DA to enable other systems to leverage SDN data plane for their own services. The LC represents the control plane in the SDN architecture. Its main function is to manage the OVS and handle the flow rules within the flow table, in addition to retrieving the flow information maintained in this table. In FlexStream, the LC communicates with the DA through the northbound API in order to receive the control commands, and then uses the extended OpenFlow protocol through the southbound API to send a rate limiting action (and *rwin*) to the OVS.

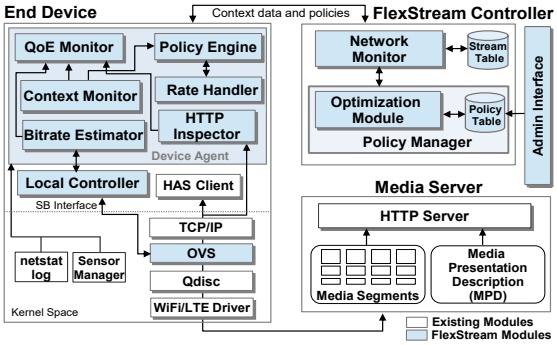

Figure 2: FlexStream system architecture.

4.2 FlexStream Controller Components

The FC consists of the *Network Monitor* and *Policy Manager*. The *Network Monitor* tracks network conditions, device contexts and device states, including video QoE, for all devices under control. The FC collects and maintains this information, including video meta-data, such as bitrate profiles, in the *Stream Table*.

Unlike some context information, which is required at the beginning of the streaming session (e.g., device characteristics), other information is needed periodically or upon significant events, such as pause or end of stream, throughput change, etc. The *Policy Table* is used to hold the optimization policies set by the administrator. When bandwidth control is required to improve performance, the *Optimization Module* is invoked on the policy by the *Policy Manager*. It uses the information maintained in both tables to assign bandwidth to video flows according to the optimization policy.

4.3 Optimization Module

Once the control is required over the bandwidth due to player competition, the *optimization module* is invoked to optimize resource allocation over a set of performance metrics that maximize the overall QoE across all users. Specifically, we formulate an optimization problem to determine the highest possible set of video bitrates across all sessions that guarantees a fair share of resources with minimum quality variations and stalls. To ensure well-balanced and fair QoE, we consider several context factors in the formulation of the optimization problem, including user priority, device capability (screen size), surrounding luminance, link condition and traffic type, to differentiate between video and background traffic.

4.3.1 ***Problem Formulation.*** Let B be the total bandwidth capacity that is available for N active video sessions. Note that B is determined from throughput feedback received from all DAs, hence

FlexStream does not require visibility into the total bottleneck capacity. We assume that each requested video i is encoded at K_i bitrates such that r_{ij} denotes the bitrate j of video i. We define a utility u_{ij} as a function of video bitrate that returns the value of selecting a bitrate j for video session i:

$$u_{ij} = \prod_{l=1}^{a} \beta_{il}.\log(r_{ij}) \tag{1}$$

Our choice of the logarithmic utility function comes primarily from its properties of diminishing returns as the bitrate increases, to ensure a proportional share of network resources among all users [25, 40]. To account for context factors in the utility, we weight the bitrate with a set of positive parameters corresponding to the considered context factors. For example, if we want to consider four factors – user priority, screen size, surrounding luminance and background traffic for video session i, we choose a to be 4 and assign positive weights to β_{i1}, β_{i2}, β_{i3}, and β_{i4} respectively. In this example, β_{i2} expresses how much more value is assigned to a device with a large screen size (e.g., tablet) than to the one with a small screen size (e.g., phone). β values are typically determined by the network administrator according to the weights assigned to different network contexts by the policy. For instance, in our experiments, we set β_{i2} to 1 for phones with small screens (e.g., 4" to 5") and 1.5 for tablets with larger screens (e.g., 7" to 10"). Such settings result in higher utility value for larger screens that translates into higher bandwidth assignments, thereby enabling tablets to stream higher bitrates. Given this utility function, the optimization problem is defined as the maximum sum of the utility functions u_{ij} across all video sessions:

$$\max_{x_{ij}} \sum_{i=1}^{N} \sum_{j=1}^{K_i} (u_{ij} - \mu\delta_{ij}) x_{ij} \tag{2}$$

$$\text{subject to} \sum_{i=1}^{N} \sum_{j=1}^{K_i} (\epsilon r_{ij}) x_{ij} \leq B \tag{3}$$

$$\sum_{j=1}^{K_i} x_{ij} = 1,\ x_{ij} \in 0, 1\ \forall i \tag{4}$$

where δ_{ij} is a penalty function that we use to minimize the fluctuation in the bitrate. As we show later, our definition of δ_{ij} can also assist in reducing stalls. The δ_{ij} function is of the form:

$$\delta_{ij} = \begin{cases} |r_{ij} - r_{ic}|s_i + (m - \lceil \frac{t_i}{k} \rceil), & t < t_{thresh} \\ |r_{ij} - r_{ic}|s_i, & t \geq t_{thresh} \end{cases} \tag{5}$$

This definition of δ_{ij} takes into consideration a number of important factors that impact the stability aspect of video QoE. The term $|r_{ij} - r_{ic}|$, where r_{ic} denotes the current bitrate of video session i, ensures that the penalty increases in line with the amount of bitrate variation. Moreover, jumping several bitrates at once is not recommended and will result in a larger penalty which in turn reduces the possibility of having stalls by preventing a large cut in the assigned bandwidth to any video session. This penalty increases as the total number of switches s_i of session i increases. We also take into account that there is enough time t_{thresh} between any two switches. The term $(m - t_i/k)$ will result in a significant penalty when t_i is short. Here, m determines the maximum penalty assigned to a quality switch, while k impacts how quickly the switching penalty diminishes with elapsed time (larger k means higher penalty if switched recently). The switching penalty diminishes with elapsed time since the last switch (t_i) and approaches t_{thresh}.

We found experimentally that the best values for parameters m, t_{thresh}, and k are 3, 60 and 20, respectively. To implement the penalty function, each DA maintains a history of the number of switches s_i as well as the time of the last switch t_i.

In the objective function (2), we include a tunable parameter μ that allows a trade-off between high bitrate and stability. For instance, if the objective is to minimize the number of switches, then μ should be set to a large value, while assigning a small value to μ will result in higher average bitrate at the expense of stability. In our implementation, we set μ=1. The indicator variable x_{ij} in the objective function is used to represent the selected video bitrate for session i such that $x_{ij} = 1$ when bitrate j is selected, and 0 otherwise. The inequality (3) indicates that the optimization formula in (2) is restricted by the total available bandwidth at the bottleneck. In addition, we use a constant ϵ (e.g., ϵ=1.35) in (3) to account for the conservative behavior of adaptive players. Most adaptive players tend to keep a safety margin between the requested bitrate and available bandwidth to avoid any unnecessary bitrate variations.

4.3.2 ***Solution.*** The optimization problem (2) can be mapped to a multiple-choice knapsack problem where one item in each class of items must be selected such that the profit is maximized without exceeding the knapsack capacity. Each set of video profiles in our problem corresponds to a class of items, while each individual profile or bitrate represents an item within that class. Similarly, the total available resource maps to the knapsack capacity, while the utility function represents the profit of selecting an item.

Intuitively, an exact solution for this problem can be obtained using Dynamic Programming (DP) within pseudo-polynomial time complexity. In order to apply DP, we start by defining a bandwidth step size s, which we use to discretize the total link capacity B into a series of Z incremental values $\{0, s, 2s, 3s, \ldots, zs, \ldots, B\}$. Then, for each video session i we calculate $y(i, z)$ for each capacity value zs as follows:

$$y(i, z) = \max_{1 \leq j \leq K_i} \{h_{ij} | zs \geq w_{i,j}\}, \qquad \forall\ 0 \leq z \leq Z = B/s \tag{6}$$

where $y(i, z)$ represents the maximum utility of video session i when the available bandwidth is zs, while h_{ij}=$(u_{ij} - \mu\delta_{ij})$, and $w_{i,j}$=(ϵr_{ij}) as defined in (2) and (3), respectively. Then, we can solve the problem via DP in a bottom-up fashion using the following recurrence:

$$Y(i, z) = \begin{cases} y(i, z), & i = 1, \forall z \\ \max\limits_{0 \leq a \leq z} \{Y(i-1, a) + y(i, z-a)\}, & 2 \leq i, \forall z \end{cases} \tag{7}$$

where $Y(i, z)$ is the total maximum utility that can be obtained for all i video sessions when the available bandwidth is zs. At each step, the optimal utility for session i is determined by selecting the highest utility among its K_i bitrates under their bandwidth requirements $zs - w_{ij}$. Using this recurrence, we calculate $Y(i, z)$ in a bottom-up fashion for all i and z until we calculate $Y(N, Z)$ that represent the total maximum utility for all N video sessions when the available bandwidth is B. Once $Y(N, Z)$ is obtained, we then perform a usual trace back to construct the optimal set of video bitrates that leads to the optimal solution. Given that the number of possible bitrates for a video session is quite limited in practice (e.g., $|K_i| \leq 10$), and a careful implementation of the dynamic programming steps, the

complexity of this solution will be $O(NZ)$. Thus if $N = 400$, $K = 8$, $B = 200$ Mbps, and $s = 100$ Kbps, the execution time on a single-core Intel 2.20 GHz processor is about 400 ms. This small time overhead would allow the FC to react in time before the QoE can be affected. However, if the execution time occasionally becomes large, then FlexStream can speed it up by reducing the granularity of z by increasing the bandwidth step size s without empirically sacrificing the optimality of the solution. Alternatives include one of the well-known approximate algorithms [36, 39] that guarantees faster execution to be within similar sub-second level at the cost of deviating from the optimal solution.

5 FLEXSTREAM IMPLEMENTATION

We next describe the implementation of core FlexStream functionalities: extensions to the SDN planes on the end device, OVS binding in Android, and the rate limiting technique.

5.1 Extending SDN Planes

In OVS, when a packet arrives on the OVS kernel data path from the wireless interface, OVS checks whether the packet matches any of its flow entries in the flow table. If the match is found, then the corresponding action is executed. This action may be to forward, drop, modify, or sample the packet. We require packet modification to enable the TCP header update. In fact, OVS is designed to perform a limited number of packet modification actions such as rewriting Ethernet/IP source or destination addresses. The challenge is that the action field for the *set* instruction in the flow table entry comes with a limited size. Adding a new action to this field requires a change in the size of the data structure of the flow table in the data path, but the modification needs to propagate to the user space and OpenFlow protocol.

Whenever a packet matches the flow entry, the *execute_set_action* function, which is responsible for modifying packet header fields in the data path, is invoked. This function in turn calls and passes the socket buffer `sk_buff` and the new *rwin* value, to our newly added function *set_tcp_window*, which modifies the TCP header after making it writeable in the `sk_buff`. We also modified the LC on the mobile device to support this new action by extending OpenFlow to support *rwin* modification action as well.

5.2 Implementing OVS binding on Android

It is challenging to bind OVS to the local network stack and add the 3G/LTE network interface as a port to OVS. While possible to successfully bind OVS to Wi-Fi interface (`wlan0`), this fails on the cellular interface as it uses different technologies and protocols to connect to its base station. In fact, simply moving the IP address of the cellular interface to OVS immediately breaks the connection between the end device and the base station, causing the interface configuration to reset and assign a new IP address. Due to this challenge, a typical way to avoid this problem is to utilize a Wi-Fi access point as a mediator. However, this approach has practical limitations. To overcome this challenge and allow for direct experimentation in cellular networks, we add the cellular interface with its IP address, assigned by the network, as a port to the OVS, which is configured with a different IP address. Then we install a number of rules to the OVS flow table to rewrite the destination IP and MAC addresses (for ingress packets) with OVS addresses, to force the traffic to go (to the upper layers) through OVS internal device once it arrives at the cellular interface. Similar actions are applied for egress traffic, but in this case the IP and MAC source addresses are overwritten with the cellular interface addresses instead. We also enable IP forwarding in the kernel space and make appropriate changes to the routing table.

5.3 Rate limiting approach

We implement rate limiting via TCP *rwin* adjustment, by computing $RTT \times ratelimit$, as a technique well-suited for our use case. It is superior to token bucket used by Linux Traffic Control (TC) Queuing discipline (Qdisc) policing for two key reasons. First, token bucket discards packets, which is a waste of bandwidth when ingress traffic at the end device is concerned, since packets have already traversed all bottlenecks. Second, *rwin* adjustment reaches the server roughly within the sum of one-way delays between the end device, FC and video server, which is much faster and less disruptive to TCP than dropped packets. To determine the *RTT* value, we can modify the OVS to calculate the *RTT* for the actual video packets, but for simplicity we choose periodic ICMP pings.

6 EVALUATION

We evaluate FlexStream through a testbed using real implementations on mobile devices in Wi-Fi and cellular networks, in addition to emulated environment for larger scale experiments.

6.1 Basic Experiments

We start by conducting relatively small-scale, real-world experiments using several mobile devices to validate, analyze and understand FlexStream behavior under different and realistic scenarios.

6.1.1 ***Experimental Setup and Design.*** The FC runs on a laptop connected to the Internet via a public IP address. Mobile clients are three Android devices with different screen sizes (Nexus 7 and two Nexus 4). The outbound bottleneck bandwidth for both Wi-Fi and cellular tests is regulated by a Squid proxy v3.1 downstream from the media server, using Linux Traffic Control (TC). Clients are set up to forward HTTP traffic via the proxy. Clients have four main components installed: DA, LC, OVS, and GPAC player [4]. DA and LC are implemented in Java. LC uses the `ovs-ofctl` library, which provides the functionality to monitor and administer OVS including modifying the flow table. The LC uses OpenFlow v1.2 on the southbound interface to control and configure the OVS. Modified version of OVS v1.11.0 is used after cross-compiling it for our Android devices, and bound to the wireless interface (`wlan0/rmnet0`) corresponding to the target network (Wi-Fi/cellular) to allow the traffic to pass through the OVS and perform the TCP window adjustment. We use a public H.264 encoded video "Big Buck Bunny" [2], streamed from a public server. This 9-minute video is split into 4-second segments and comes with many bitrate profiles, from which we select 449, 843, 1416, and 2656 Kbps. The Android version of GPAC is v0.6.2-DEV (Osmo4). We modify the basic adaptation algorithm of GPAC to use the harmonic mean of measured throughput of the last five received segments for bandwidth estimation.

6.1.2 ***Results.*** We evaluate FlexStream with respect to average bitrate and stability over static and variable capacity bottlenecks, with and without background traffic.

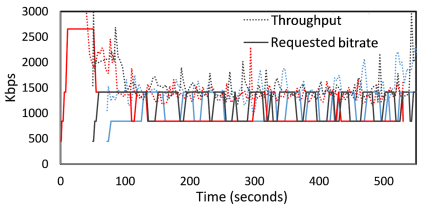

Figure 3: Uncontrolled player competition causes frequent switching.

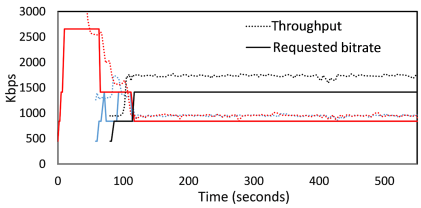

Figure 4: FlexStream stabilizes the requested bitrates of competing players.

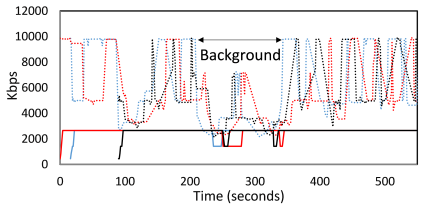

Figure 5: Background traffic causes bitrate reduction in uncontrolled case.

Wi-Fi with static capacity. The uncontrolled case of 3 players competing for 3800 Kbps bottleneck is presented in Figure 3, showing the impact of competition on average throughput and requested bitrates. Even though that TCP-based players are expected to equally share the bandwidth, the intermittent traffic of HAS and selfish behavior of the adaptive algorithms leads to oscillations in throughput of all players, causing frequent switches between 843 and 1416 Kbps bitrates.

FlexStream is designed to prevent such fluctuations and provide improved stability, as shown in Figure 4. Note that FlexStream is also designed to distribute bandwidth in a way that maximizes the requested bitrates. Hence, FlexStream assigns about 1850 Kbps to one device and 975 Kbps to the other two devices allowing them to stream 1416 Kbps and 843 Kbps bitrate profiles, respectively. These are the maximum bitrates that the players can request without affecting stability. FlexStream reduces the total number of bitrate switches for all players from 66 in the uncontrolled experiment to only 15, while maintaining approximately the same overall quality (about 1125 Kbps on average in both experiments). Note that the 15 switches incurred with FlexStream have mostly occurred during startup, not because of fluctuation in throughput. We repeated this experiment several times using different network capacities, starting from 2500 Kbps and increasing by 1500 Kbps until no competition between the players is observed at 8500 Kbps. As expected, FlexStream significantly reduces the average number of switching from over 60 for 2 devices in an uncontrolled scenario, to 20 or less with FlexStream, which confirms its distinct advantage.

Considering device characteristics. FlexStream also improves fairness by implementing screen size-based policy that favors larger screens when needed (e.g., tablet over phone). In the previous experiment (Figure 4), FlexStream assigns more bandwidth (1850 Kbps) to the tablet so it can select higher bitrate (1416 Kbps) than the two phones (843 Kbps each). Therefore, the tablet user can have comparable viewing experience to phone users. When repeating this experiment over a range of bottlenecks, the tablet with FlexStream gets 350 Kbps higher bitrate on average than each of the phones in contrast to the uncontrolled case. Balancing the QoE is an important feature as in some scenarios such as a home network, it is possible to encounter competition between players with a large screen size differences, such as phones, laptops, and TVs.

Robustness to background traffic. In practice, quality degradation and instability on mobile devices can also be caused by background traffic, such as automatic backup, cloud sync, or app updates. Figure 5 shows the negative impact of background traffic on throughput and video bitrates even with enough bandwidth (10 Mbps). After three devices start streaming, we use `iperf` on one of the devices to initiate a TCP connection with an external host which lasts for two minutes, starting at time 210. As the throughput is only tuned by the TCP congestion control, it is expected that the background traffic achieves equal share of about a quarter of the total bandwidth. Consequently, this causes a significant drop in the players average throughputs and leads to competition, resulting in reduces video bitrates and fluctuation between 1416 and 2656 Kbps, until the background traffic stops at time 330. We repeat the previous experiment with FlexStream, configured to limit the background traffic to 250 Kbps, and observe no clear impact on the video QoE (figure omitted due to space limit). If no video is streamed, FlexStream may or may not control the traffic to the device, depending on the policy.

Differentiated services. FlexStream is also designed to differentiate between priority classes of devices, e.g., regular and high priority, and to distribute bandwidth accordingly (Figure 6). Two players, on the phone and tablet with regular priorities, start at time 0 and 50, respectively, and adapt to stream the highest quality (2656 Kbps) until a third player on another phone with high priority joins at time 100. Consequently, as the network capacity is oversubscribed, FlexStream steps in to allocate more bandwidth to the high priority device (a phone) to stream the highest bitrate profile. The rest of the bandwidth is then divided between regular devices, while also considering the screen size.

Cellular network with dynamic capacity. To demonstrate how well FlexStream can work in practice in the presence of uncontrolled background traffic, we conduct the same experiments on a real cellular network with three mobile devices. We select busy hours in a few loaded radio cells for our experiments, to compare uncontrolled players and FlexStream. From Figure 7, we can observe three main issues in the uncontrolled experiment due to the high fluctuation in throughput: a large drop in the video bitrate (sometimes to the lowest bitrate profile), a significant instability, and unfairness in bandwidth utilization, all resulting in unstable viewing experience for users. On the other hand, even under such highly dynamic network conditions caused by competing players and high (and uncontrolled) background traffic, FlexStream can still improve performance and mitigate the issues that appeared in the uncontrolled scenario, as seen in Figure 8.

Wi-Fi with dynamic capacity. We also evaluated FlexStream in Wi-Fi with dynamic capacity and obtained qualitatively similar results to cellular scenarios (omitted due to space limit).

6.2 Extended Experiments

6.2.1 ***Experimental Setup.*** To evaluate the performance in the presence of a sufficiently large number of competing video players, we develop a player emulator that runs on the same testbed as in the basic experiments. Similar to the real player, the emulator, implemented as a Java application, generates real requests to the HTTP server, which responds with dummy video segments equivalent in size and distribution to those used in the real experiment. The

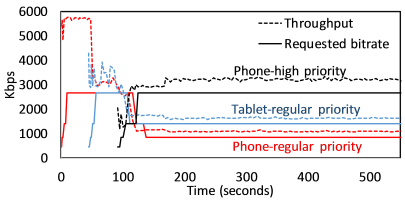

Figure 6: FlexStream implements differentiation between device classes.

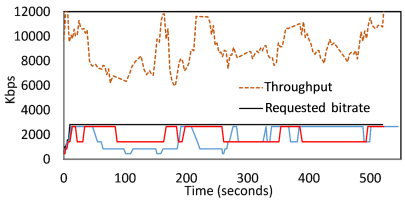

Figure 7: Instability and unfairness in cellular network with no BW control.

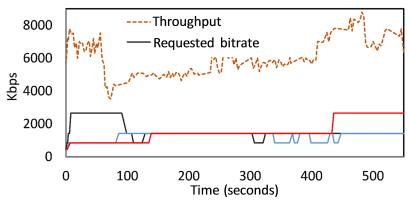

Figure 8: FlexStream provides improved stability in cellular network.

Table 1: Average performance metrics for 12 players.

	No Control	FlexStream
Instability (switches)	19.1	3.6
Number of stalls	10.6	1.0
Stall duration	40.6 s	3.1 s
Startup delay	7.9 s	4.4 s
Tablets bitrate difference	20 Kbps	196 Kbps
Fairness (JFI)	0.90	0.96

emulated player is developed with all the required functions with the exception of decoding and playing video. In all experiments, 12 emulated players, representing 8 phones and 4 tablets are used to stream over five Wi-Fi AP capacities {7, 10, 13, 16, and 19 Mbps}, and set to randomly start during the first two minutes. For each capacity, 10 repetitions and a broader set of QoE metrics are used in the evaluation.

6.2.2 ***Results.*** Figure 9 compares stability of the video quality with and without FlexStream for each AP bottleneck capacity. Table 1 summarizes the performance statistics over all bottleneck capacities. It is clear that players with no control fail to provide a stable viewing experience. The average number of switches per player is between 10 and 24. On the other hand, FlexStream substantially improves stability in each case, with the average number of switches reduced by 81% from 19.1 to 3.6 (Table 1).

Player competition not only causes instability, but can also lead to playback stalls resulting in a severe QoE degradation. Figure 10 shows the average number of stalls per bottleneck capacity. The average duration of stalls is also unacceptably high and follows the same trend, as shown in figure 11. In all experiments, adding FlexStream shows outstanding performance by reducing the average number of stalls by 91% (from 10.6 to 1.0), and lowering the average stall duration by 92% (from 40.6 to 3.1 s). Table 1 also shows that there is a significant improvement in startup delay with FlexStream, with startups becoming faster by 44% on average (from 7.9 to 4.4 s), since the competition among players is now avoided.

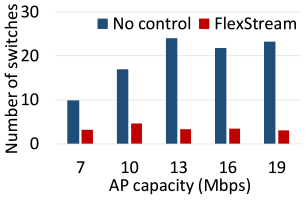

Figure 9: FlexStream significantly reduces switching.

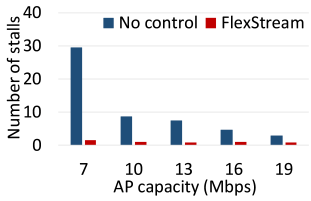

Figure 10: FlexStream reduces number of stalls.

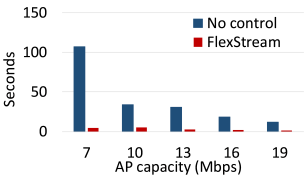

Figure 11: Comparison of average stall duration.

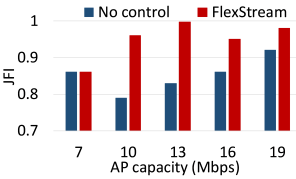

Figure 12: Comparison of average Jain's Fairness Index.

FlexStream further addresses fair allocation of bandwidth among players taking into consideration their screen sizes. The *Tablets bitrate difference* entry in Table 1 refers to the average bitrate difference achieved by tablets over phones, where tablets with FlexStream can obtain 196 Kbps higher bitrate on average than the phones. Moreover, FlexStream improves fairness among devices of the same screen size (e.g., phones or tablets). Figure 12 compares fairness in the average requested bitrate among phones using Jain's Fairness Index (JFI) [21], where higher values mean better fairness. Across all capacities, FlexStream improves JFI, and significantly so for 7, 10, and 13 Mbps bottleneck. The overall JFI across all devices (including phones and tablets) and scenarios is increased by 0.06 (Table 1).

To study the impact of background traffic with many players, the experiments in Figure 14 show the average video bitrate one minute before and after generating a TCP flow via `iperf`. All players are impacted, resulting in an average drop by ≈ 300 Kbps, with up to 800 Kbps for one player. As expected, we do not observe any bitrate switches or a significant drop in average bitrates with FlexStrem, as it controls and shapes the background traffic (figure omitted).

7 CONCLUSION

We present FlexStream, an extreme SDN framework to manage end devices in the network according to specified policies. We demonstrate that FlexStream can achieve appropriate bandwidth distribution by a light-weight controller in the network, as well as significantly improve performance of video streams, by offloading policy implementation to end devices. We also report the first implementation of SDN-based control on commodity mobile devices in live cellular network. Other aspects of the system left for future work include integrating FlexStream into the existing network policy functions and an ability to obtain link capacity and other state from the network directly.

Acknowledgment. This material is based upon work partially supported by the US National Science Foundation under Grants No. CNS-1454285 & CNS-1764185.

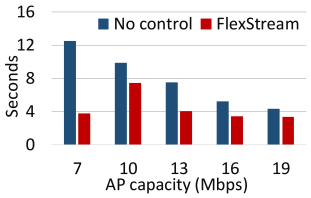

Figure 13: Comparison of average startup delay times.

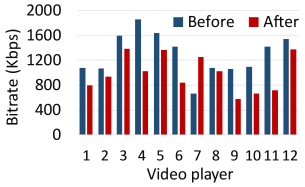

Figure 14: Background flow degrades video bitrates.

REFERENCES

[1] 2015. Conviva. http://www.conviva.com/. (2015).
[2] 2017. Big Buck Bunny. https://peach.blender.org/download/. (2017).
[3] 2017. Cisco Visual Networking Index: Global mobile data traffic forecast update 2016–2021 white paper. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html. (2017).
[4] 2017. GPAC. https://gpac.wp.imt.fr/player/. (2017).
[5] 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 *IEEE International Conference on Computer Communications (INFOCOM)*. 1620–1628.
[6] Saamer Akhshabi, Lakshmi Anantakrishnan, Ali C Begen, and Constantine Dovrolis. 2012. What happens when HTTP adaptive streaming players compete for bandwidth?. In *International workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*. ACM, 9–14.
[7] Saamer Akhshabi, Lakshmi Anantakrishnan, Constantine Dovrolis, and Ali C Begen. 2013. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. ACM, 19–24.
[8] Saamer Akhshabi, Ali C Begen, and Constantine Dovrolis. 2011. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *International Conference on Multimedia systems (MMSys)*. ACM, 157–168.
[9] Ibrahim Ayad, Youngbin Im, Eric Keller, and Sangtae Ha. 2018. A Practical Evaluation of Rate Adaptation Algorithms in HTTP-based Adaptive Streaming. *Computer Networks* 133 (2018), 90–103.
[10] Suman Banerjee, Neil Klingensmith, Peng Liu, and Anantharaghavan Sridhar. 2017. Edge Computing in the Extreme for Sustainability. In *International Conference on Communication Systems and Networks (COMSNETS)*. 93–109.
[11] Abdelhak Bentaleb, Ali C Begen, and Roger Zimmermann. 2016. SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking. In *ACM on Multimedia Conference (MM)*. ACM, 1296–1305.
[12] Niels Bouten, Jeroen Famaey, Steven Latré, Rafael Huysegems, Bart De Vleeschauwer, Werner Van Leekwijck, and Filip De Turck. 2013. QoE Optimization Through In-Network Quality Adaptation for HTTP Adaptive Streaming. In *International Conference on Network and Service Management (CNSM)*. 336–342.
[13] Aaron M French, Chengqi Guo, and Jung P Shim. 2014. Current Status, Issues, and Future of Bring Your Own Device (BYOD). *Communications of the Association for Information Systems* 35 (2014), 10.
[14] 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 *USENIX Conference on Networked Systems Design and Implementation (NSDI)*, Vol. 15. 131–144.
[15] 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 Communication*. 69, C (2015), 79–87.
[16] Panagiotis Georgopoulos, Yehia Elkhatib, Matthew Broadbent, Mu Mu, and Nicholas Race. 2013. Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming. In *ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking (FhMN)*. ACM, 15–20.
[17] Bo Han, Feng Qian, Lusheng Ji, and Vijay Gopalakrishnan. 2016. MP-DASH: Adaptive Video Streaming Over Preference-Aware Multipath. In *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 129–143.
[18] David Ke Hong, Yadi Ma, Sujata Banerjee, and Z Morley Mao. 2016. Incremental deployment of SDN in hybrid enterprise and ISP networks. In *Symposium on SDN Research (SOSR)*. ACM.
[19] Te-Yuan Huang, Ramesh Johari, and Nick McKeown. 2013. Downton abbey without the hiccups: Buffer-based rate adaptation for HTTP video streaming. In *ACM SIGCOMM workshop on Future human-centric multimedia networking (FhMN)*. ACM, 9–14.
[20] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.
[21] Raj Jain, Dah-Ming Chiu, and William R Hawe. 1984. A quantitative measure of fairness and discrimination for resource allocation in shared systems. *Technical Report DEC-TR-301, Tech. Rep.* 38 (1984).
[22] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 3–14.
[23] J. Jiang, V. Sekar, and H. Zhang. 2012. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 97–108.
[24] A. M. Kakhki, F. Li., D. Choffnes, A. Mislove, and E. K. Bassett. 2016. BingeOn Under the Microscope: Understanding T-Mobile's Zero-Rating Implementation. In *ACM SIGCOMM Workshop on QoE-based Analysis and Management of Data Communication Networks (Internet-QoE)*. 43–48.
[25] Frank Kelly. 1997. Charging and rate control for elastic traffic. *Transactions on Emerging Telecommunications Technologies* 8, 1 (1997), 33–37.
[26] Jan Willem Kleinrouweler, Sergio Cabrero, and Pablo Cesar. 2016. Delivering stable high-quality video: An SDN architecture with DASH assisting network elements. In *International Conference on Multimedia Systems (MMSys)*. ACM.
[27] 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 *International Teletraffic Congress (ITC)*. IEEE, 177–184.
[28] Jeongkeun Lee, Mostafa Uddin, Jean Tourrilhes, Souvik Sen, Sujata Banerjee, Manfred Arndt, Kyu-Han Kim, and Tamer Nadeem. 2014. meSDN: Mobile extension of SDN. In *International Workshop on Mobile Cloud Computing & Services (MCS)*. ACM, 7–14.
[29] 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 (2014), 719–733.
[30] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. 2011. Rate adaptation for adaptive HTTP streaming. In *International Conference on Multimedia systems (MMSys)*. ACM, 169–174.
[31] Tarun Mangla, Emir Halepovic, Mostafa H. Ammar, and Ellen W. Zegura. 2018. eMIMIC: Estimating HTTP-based Video QoE Metrics from Encrypted Network Traffic. In *Network Traffic Measurement and Analysis Conference (TMA)*.
[32] Tarun Mangla, Ellen W. Zegura, Mostafa H. Ammar, Emir Halepovic, Kyung-Wook Hwang, Rittwik Jana, and Marco Platania. 2018. VideoNOC: Assessing video QoE for network operators using passive measurements. In *International Conference on Multimedia systems (MMSys)*. ACM, 101–112.
[33] 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 (2008), 69–74.
[34] Ricky KP Mok, Xiapu Luo, Edmond WW Chan, and Rocky KC Chang. 2012. QDASH: a QoE-aware DASH system. In *International Conference on Multimedia Systems (MMSys)*. ACM, 11–22.
[35] 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. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 311–324.
[36] Yu E Nesterov and Michael J Todd. 1997. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations research* 22, 1 (1997), 1–42.
[37] Stefano Petrangeli, Jeroen Famaey, Maxim Claeys, Steven Latré, and Filip De Turck. 2015. QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming. *In ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 12, 2, Article 28 (2015), 24 pages.
[38] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. 2015. The Design and Implementation of Open vSwitch. In *USENIX Conference on Networked Systems Design and Implementation (NSDI)*.
[39] David Pisinger. 1995. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research (EJOR)* 83, 2 (1995), 394–410.
[40] Huaizhou Shi, R Venkatesha Prasad, Ertan Onur, and IGMM Niemegeers. 2014. Fairness in wireless networks: Issues, measures and challenges. *IEEE Communications Surveys & Tutorials* 16, 1 (2014), 5–24.
[41] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 1–9.
[42] Dimitrios Tsilimantos, Theodoros Karagkioules, and Stefan Valentin. 2018. Classifying Flows and Buffer State for Youtube's HTTP Adaptive Streaming Service in Mobile Networks. In *International Conference on Multimedia Systems (MMSys)*. ACM.
[43] M. Uddin and T. Nadeem. 2016. TrafficVision: A Case for Pushing Software Defined Networks to Wireless Edges. In *International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 37–46.