Spring 2018

FlexStream: SDN-Based Framework for Programmable and Flexible Adaptive Video Streaming

Ibrahim Ben Mustafa
Old Dominion University

FLEXSTREAM: SDN-BASED FRAMEWORK FOR PROGRAMMABLE AND FLEXIBLE ADAPTIVE VIDEO STREAMING

by

Ibrahim Ben Mustafa

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
May 2018

Approved by:

Dr. Tamer Nadeem (Director)

Dr. Ravi Mukkamala (Co-Director)

Dr. Stephan Olariu (Member)

Dr. M'Hammed Abdous (Member)

ABSTRACT

FLEXSTREAM: SDN-BASED FRAMEWORK FOR PROGRAMMABLE AND FLEXIBLE ADAPTIVE VIDEO STREAMING

Ibrahim Ben Mustafa
Old Dominion University, 2018
Director: Dr. Dr. Tamer Nadeem
Co-Director: Dr. Dr. Ravi Mukkamala

With the tremendous increase in video traffic fueled by smartphones, tablets, 4G LTE networks, and other mobile devices and technologies, providing satisfactory services to end users in terms of playback quality and a fair share of network resources become challenging. As a result, an HTTP video streaming protocol was invented and widely adopted by most video providers today with the goal of maximizing the user's quality of experience. However, despite the intensive efforts of major video providers such as YouTube and Netflix to improve their players, several studies as well as our measurements indicate that the players still suffer from several performance issues including instability and sub-optimality in the video bitrate, stalls in the playback, unfairness in sharing the available bandwidth, and inefficiency with regard to network utilization, considerably degrading the user's QoE. These issues are frequently experienced when several players start competing over a common bottleneck. Interestingly, the root cause of these issues is the intermittent traffic pattern of the HTTP adaptive protocol that causes the players to over estimate the available bandwidth and stream unsustainable video bitrates. In addition, the wireless network standards today do not allow the network to have a fine-grain control over individual devices which is necessary for providing resource usage coordination and global policy enforcement. We show that enabling such a network-side control would drive each device to fairly and efficiently utilize the network resources based on its current context, which would result in maximizing the overall viewing experience in the network and optimizing the bandwidth utilization.

In this dissertation, we propose FlexStream, a flexible and programmable Software-Defined Network (SDN) based framework that solves all the adaptive streaming problems mentioned above. We develop FlexStream on top of the SDN-based framework that extends SDN functionality to mobile end devices, allowing for

a fine-grained control and 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 WiFi and cellular links and can effectively alleviate common problems such as player instability, playback stalls, large startup delay, and inappropriate bandwidth allocation. FlexStream offloads several tasks such as monitoring and policy enforcement to end-devices, while a network element (i.e., Global Controller), which has a global view of a network condition, is primarily employed to manage the resource allocation. This also alleviates the need for intrusive, large and costly traffic management solutions within the network, or modifications to servers that are not feasible in practice. We define an optimization method within the global controller for resource allocation to maximize video QoE considering context information, such as screen size and user priority. All features of FlexStream are implemented and validated on real mobile devices over real Wi-Fi and cellular networks. To the best of our knowledge, FlexStream is the first implementation of SDN-based control in a live cellular network that does not require any internal network support for SDN functionality.

ACKNOWLEDGMENTS

All Praise is Due to Allah (God)
To all who have helped me succeed in my study:
Thank You!

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The Internet has recently witnessed a tremendous increase in video traffic fueled by smartphones, tablets, 4G LTE networks, and other mobile devices and technologies. In 2012, it was reported that video traffic generated by YouTube and Netflix alone constituted more than 50% of the peak download traffic in the USA [32]. Moreover, Cisco reported that mobile video traffic is growing rapidly and expected to form 75% of total Internet traffic by 2020 as shown in Figure 1. Surprisingly, unlike other type of traffics that are generated throughout the day, video traffic tends to be heavily streamed during evening hours and has a peak time. Consequently, increasing the video traffic would lead to adding more traffic during the peak hours which increases the possibility of creating network bottlenecks, making video flows competition over the bandwidth unavoidable. Figure 2 shows that global busy-hour traffic was 66% higher than the average-hour traffic in 2015 and expected to reach 88% in 2020 [37]. In its 2014 viewer experience report, Conviva [11] analyzed 45 billion viewed videos and found that 26.9% of viewers experienced buffering, 43.3% were impacted by low resolution, and 4.8% of videos failed to start.

The nature of wireless links, on the other hands, adds another challenge. Unlike fixed devices in wired networks, wireless devices can experience severe fluctuation in the network condition which would dramatically impact the throughput, latency, error rates, and other network metrics [17, 24]. Moreover, limited battery power of mobile devices is also another critical issue that should be considered when streaming videos [46, 72]. Therefore, providing satisfactory services to end users in terms of playback quality, fair share of network resources, and low battery consumption becomes challenging. As a result, most video providers such as YouTube [79], Netflix [55], Hulu [36], and Dailymotion [13] have embraced the HTTP Adaptive Streaming as a video delivery approach to assure a high level of QoE to end users. This adoption, as indicated in [70], was motivated by several features and services provided by this technology:

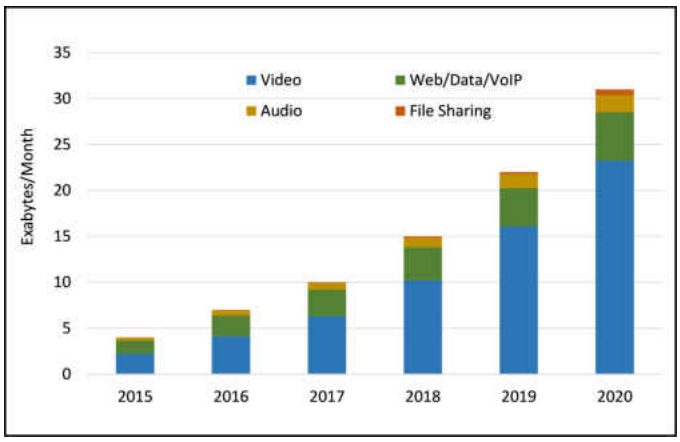

Figure 1: Traffic percentage for different classes of traffic from 2015 to 2020.

1. The ability of the video player, residing on the client device, to dynamically and seamlessly adapt the video bitrate to the network condition.

2. The reliance on the existing Content Delivery Network (CDN) infrastructures for Web content delivery, allowing videos to be streamed from stranded HTTP servers and caches.

3. The seamless and easy streaming via traversing the NAT and firewall without any complication.

4. The popularity and widespread of HTTP and underlying TCP protocol provides reliability and simplicity for video delivery.

1.2 ISSUES WITH HTTP ADAPTIVE STREAMING

Despite the benefits that this protocol brings to video streaming, recent measurements [32, 2, 4, 30] have shown that many widely used adaptive players suffer from multiple performance issues including:

1. Instability and sub-optimality in the video bitrate and thus quality.

2. Interruption and freeze in the Playback.

3. Long startup delay time.

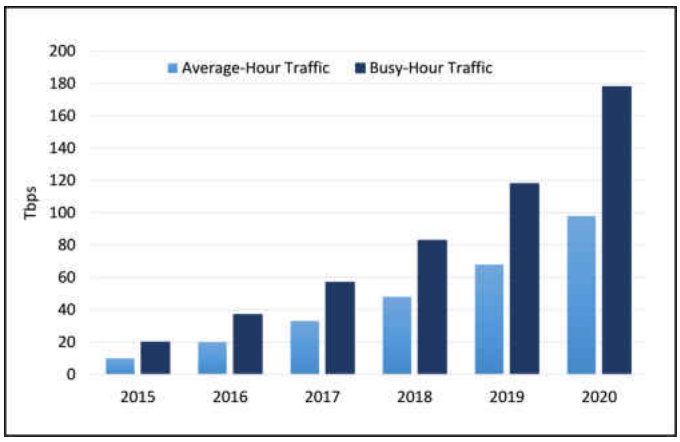

Figure 2: Busy-Hour vs Average-Hour traffic.

4. Unfairness in sharing the network resources, leading to unbalanced Quality of Experience (QoE) among end users.

5. Inefficient use of network resource utilization.

These issues were also confirmed by our measurements of recent versions of players competing over a common bottleneck. Many studies have reported that providing a high quality picture and stable bitrate is very important for maximizing user engagement and repeat viewership, and the instability in the perceived quality, in contrast, can significantly diminish user engagement [15, 12, 7, 52]. Results in [68, 29, 75] show that fluctuation in video quality found to be very annoying to users, degrading video QoE. Consequently, it is reported by [62] that users prefer a lower constant bitrate or quality over unstable bitrate while higher in average. In fact, instability in the video bitrate not only affect the perceived quality, but also causes network underutilization [16, 2]. As HTTP operating over TCP protocol, the bandwidth tend to be distributed equally (assuming same RTT values) among end-devices regardless of their characteristics (e.g., screen size) and context, leading to unfair QoE among the end users.

In fact, the main cause of the previous issues is the well-known unwanted interactions between multiple adaptive players. As players operate in the ON/OFF pattern in the steady state, downloading small chunks of video periodically, the OFF periods of one or more players can cause other players to overestimate the available bandwidth when requesting new chunks of data. Conversely, if multiple players overlap

their ON periods they may perceive lower bandwidth, not accounting for the OFF period. This can ultimately lead to player instability, with frequent video bitrate switching which lowers the video Quality of Experience (QoE). Moreover. If the overestimation of the available bandwidth is considerable, then a major drop in the throughput can also lead to playback freeze (stall). Other issues are long startup delay time and unbalanced video QoE which mostly occur when a new player(s) joins and finds the bandwidth is dominated by other running players. Therefore, looking beyond a single player scenario and accounting for the interaction of several adaptive players is essential for improving the video QoE. It is more probable in the coming years to find many people simultaneously streaming videos from the same access point in the home network, or at some public place such as the airport, shopping mall, or university campus.

While various approaches are used to stabilize bitrate selection of players and prevent stalls, these players typically work in a homogeneous environment where the same adaptation algorithms co-exist and their limited local view of the network can be somewhat mitigated. This implies that for stable and stall-free multi-player streaming experience, the same adaptation algorithm should be run on different devices. This is unrealistic to expect in today's real world where virtually every video service has its own player and a video catalog of titles with a proprietary selection of video bitrate profiles. Also, after considering multiple types of devices, screen sizes, and user needs, we can realize that the only entity that can have a global view and optimally manage many players competing for the bottleneck bandwidth is the network.

1.3 CONTRIBUTION

In this dissertation, we argue that the network should step in and help the players, whenever needed, to stream the best possible video bitrate while ensuring fair distribution of bandwidth (to provide balanced QoE), stable video quality, stall-free playback. The network should also be able to provide some valuable services such priority of certain users, if needed (e.g., emergency responders, premium customers, etc.). We stress that for the purposes of this work, we redefine the notion of fairness. As detailed later, we do not consider equal distribution of available bandwidth to be fair and appropriate in all cases. Instead, the goal is that devices with the same (or similar) screen size should receive similar video bitrates, meaning that larger screens

should receive higher bitrates than smaller screens. In terms of stability, the network should step in when it detects conditions that lead to instability, such as devices imposing higher demand that the network currently cannot sustain. Moreover, if there are higher priority users in the network, they should receive higher bitrates than lower priority users.

To achieve these goals, we propose FlexStream, a flexible and programmable Software-Defined Network (SDN) based framework that automates the process of monitoring and managing bandwidth of video users. We select this approach for the following reasons:

1. examine the ability to extend the SDN paradigm to end user devices to perform automated management and control tasks.

2. SDN promises a standardized framework for programmable control and can be implemented as a kernel functionality.

3. offers a universal approach to work across network technologies and network domains (cellular, WiFi, home networks, etc.).

FlexStream is a system that we developed on top of the framework that extends the SDN to mobile devices. FlexStream enables centralized control of bandwidth allocation via a global controller that specifies a policy, and a local policy implementation via Open vSwitch (OVS) that offloads the fine-grained functionality to the end device. In addition to the bandwidth control policy, the system supports defining various control policies based on the different interests and contexts of the user, device, video, network, and environment (e.g., user priority-based policy, device screen size-based policy), which is often overlooked in practical implementations. Using an optimization function, we demonstrate that all these factors can be effectively accounted for within the policy that allocates bandwidth across devices. Network programmability is also one of the main features of FlexStream, where network policies can be implemented and enforced in real time based on the context (e.g., time, location, flow type). Finally, as end devices running FlexStream can be treated as logical switches with ports acting like the available network interfaces (e.g., WiFi and cellular), they can support multi-path delivery (e.g., MPTCP) according to user-specified interface preferences [25].

We implement FlexStream on commodity Android devices and evaluate its performance using realistic scenarios on both WiFi and cellular networks. Our prototype

implementation uses a controller located in the cloud so that the SDN functionality does not depend on the underlying network support. Our contributions can be summarized as follows:

- We develop FlexStream on top of the SDN-based framework that extends SDN functionality to mobile end devices, allowing for fine-grained control and 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 WiFi 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 best of our knowledge, the first working implementation of the SDN extension to commodity mobile devices that runs in both WiFi and cellular networks without requiring support from the existing network infrastructure.

1.4 ORGANIZATION

The reminder of this thesis is organized as follow. Chapter 2 provides a background on HTTP adaptive streaming and SDN, while Chapter 3 presents traffic analysis for adaptive players. We introduce the FlexStream framework in Chapter 4. Then, in Chapter 5, we describe the implementation of FlexStream. While Chapter 6 presents our optimization module for resource allocation, Chapter 7 is dedicated to the system evaluation. Finally, Chapter 8 concludes the dissertation and highlights the future work.

1.5 SUMMARY

A tremendous increase in mobile data traffic in the recent years increases the possibility of having more network bottlenecks. These bottlenecks would have a

negative impact on the performance of adaptive streaming technology. When these adaptive video players compete over bandwidth, several performance issues that degrades the users QoE appears such as instability in the perceived quality and stalls in the playback, among others. It is found that the root cause of these issues is the intermittent of the adaptive traffic that often causes video players to misestimate the available bandwidth. To overcome these issues and improve the performance of adaptive players, we proposed FlexStream, a flexible and programmable Software-Defined Network (SDN) based framework that automates the process of monitoring and managing video sessions in the network. We explained that FlexStream also supporting defining various control policies based on the different interests and contexts of user, device, video, network, and environment.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 BACKGROUND

2.1.1 VIDEO STREAMING OVER IP NETWORKS

Videos is a sequence of frames or pictures which when played they generate an illusion of motion. The number of frames ranges typically between 24 to 60 per seconds. Low number of frames (< 24) can result in unsmooth motion, while a high number can produce an adequate motion but with larger video size. There are typically three types of video frames, Intra (I), Bidirectional (B), and Predicted (P) frames, generated by video compression algorithms. These frames are different in size, with the I-frame is larger than the other two frames since I-frames use only intra frame compression, while B and P frames use previous I-frames for size reduction. Most of the videos are encoded with variable video bitrates (VBR) at the servers side before being transmitted through the Internet to the client devices. However, the network can only support streaming videos on a best-effort basis, which means that if the available bandwidth is not sufficient for streaming the requested video biterate, then the decoder at the client side will consume the video data on a higher rate than the receiving rate supported by the network, leading to a drop in the video quality in addition to rebuffering events (video stalls). To avoid these undesirable events, several solutions have been proposed in the literature aiming to provide simple but effective mechanisms to match the video biterate to the available network bandwidth. These solutions can be summarized as follow:

1. Using playback buffer: This buffer allows the player to pre-fetch and to store the data in advance to absorb any short term variations in the network bandwidth.

2. Transcoding-based solutions: These solutions are computationally intensive as they require to change one or more compression algorithm parameters to accommodate the video bitrate and the available bandwidth.

3. Scalable Video coding: In this technique, the video is encoded into a base layer and multiple enhancement layers that can be partially or totally truncated to adjust the video bitrate to the network bandwidth. However, this solution has not been adopted by video providers as it is also computationally intensive.

4. Adaptive Streaming Solution: Video data with this technique is processed or encoded into multiple bitrates and stored on the server. These different bitrates can then be requested by the video player according to the available bandwidth. Most of the video providers have adopted this solution, with a playback buffer to avoid any bandwidth variation impact. The following section introduces this technology in details.

2.1.2 EVOLUTION OF VIDEO STREAMING TECHNOLOGIES

Since the Internet was not originally designed to support data delivery of bandwidth-intensive applications such as video streaming, most of the early efforts on improving streaming services focused on techniques that enable resource reservations and Quality of Service (QoS). Real Time Transport Protocol (RTP) [38], Real Time Streaming Protocol (RTSP) [63], RTP Control Protocol [20], Resource ReSer-Vation Protocol [81], and Session Description Protocol (SDP) [28] are examples of the protocols that were developed and proposed to support real-time streaming over UDP. The server in these protocol controls and configures the end systems that support and initiate the video streams. However, these protocols and techniques require dedicated servers and network infrastructure, incurring high deployment cost and adding major complexities. I addition, these protocols have an issue in traversing NATs and firewalls. Moreover, as these techniques use UDP as a transport protocol, the congestion control and reliability remain open issues. Finally, These techniques can not provide real-time adaptability to network condition. Consequently, HTTP adaptive streaming protocol over TCP is proposed to overcome all the above issues.

2.1.3 HTTP ADAPTIVE BITRATE STREAMING

HTTP Adaptive Streaming (HAS) has become the prevalent paradigm for video delivery in today's Internet. In Adaptive Bitrate Streaming (ABR), video content is encoded into multiple bitrate profiles (also known as tracks or quality levels), with each having different screen resolution, frame rate, and other encoding parameters.

With increasing picture quality, each track has a higher bitrate, i.e. higher bandwidth requirement for delivery. The purpose of multiple bitrates is to allow the client player to adapt to varying network conditions.

As illustrated in Figure 3, the content of each bitrate profile in ABR is further split into small segments, where each segment represents a short duration of playback time (typically 2 to 10 seconds). Segment boundaries are the same in each profile, giving the player an opportunity to switch profiles at each segment boundary. These segments are made available for downloading on a conventional HTTP web server via a standard HTTP request. A manifest file, which describes the available bitrate profiles, segment URLs, and other parameters, is downloaded by the player prior to the streaming session. Typical player behavior is to seek the highest available bitrate if bandwidth allows.

The examples of the most widely used protocols for HAS streaming today are Apple HTTP Live Streaming (HLS) [5], Microsoft Smooth Streaming (MSS) [80], Adobe HTTP Dynamic Streaming (HSD) [1], and Dynamic Adaptive Streaming over HTTP (DASH). DASH is a standard (ISO/IEC 23009-1) [69], while HLS, MSS, and HSD are proprietary protocols. All these examples are client-centric solutions, meaning all decisions are made on the client, leaving the server to only respond to the client's requests. This enables rapid deployment through the existing CDN infrastructure, which is the key enabler responsible for the prevalence over ABR over HTTP. The client can dynamically adapt to the change in network conditions by adjusting the video bitrate, typically according to measured bandwidth. For instance, the player can request a lower profile when it encounters a major drop in the available bandwidth to avoid possible stalls in the playback if it had stayed with the same quality profile. One of the major challenges in designing the adaptive algorithm of the video players is the adaptation logic that maximizes the viewing experience. In fact, most of the work in the literature tries to enhance the performance of video players by designing a better adaptation algorithm.

HAS protocol has two distinct states: buffering state and steady state as depicted in Figure 4. Initially, the player enters the buffering state to fill the buffer with video frames and as soon as the buffer is filled, it switches to the steady state in which the player starts generating ON/OFF traffic patterns. Typically, the adaptive player maintains two thresholds, an upper and lower thresholds. The player pauses downloading video chunks as soon as the buffer reaches the upper threshold, and

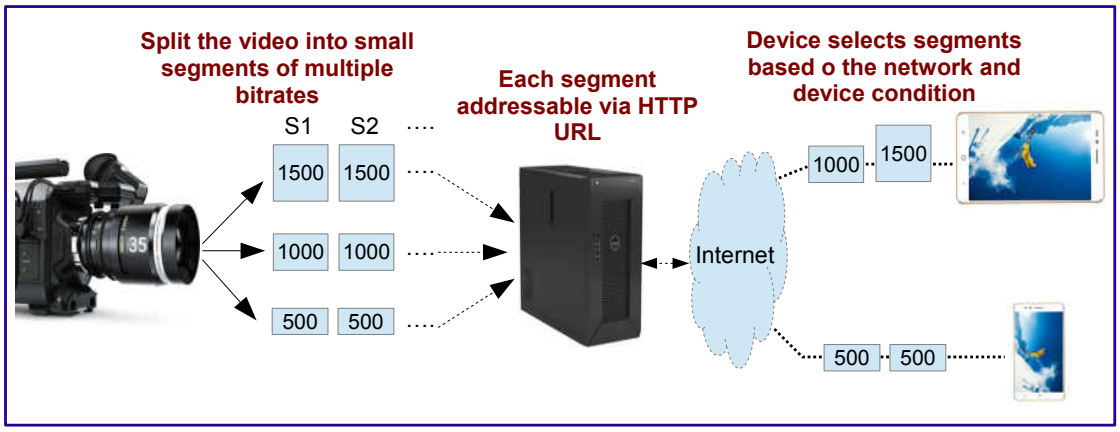

Figure 3: Adaptive Video Streaming.

it resumes downloading once the buffer drops to the lower threshold. The main purpose of using a limited buffer size is to avoid downloading unnecessary content and thus save network resources when the user abandons watching the video before completion.

The intermittent traffic pattern (ON/OFF periods) of the HAS players in the steady state introduces a major challenge for competing players to accurately estimate the available bandwidth, as any one player may perceive drastically different network condition depending on whether it competes with another player during a segment download or not [2]. To further understand how the intermittent pattern of HAS players can cause them to over estimate the available bandwidth, Figure 5 shows the traffic pattern of two competing players of a real experiment that we conduct over the WiFi network. As we can observe, when the two players start downloading at the same time before second 307, they tend to fairly share the bandwidth. However, when Player B goes off after that time for about 15 seconds, Player A starts observing higher bandwidth causing it to over estimate the available bandwidth. Similarly, when Player A goes off at time 22, Player B starts overestimating the available bandwidth till Player A wakes up after 17 seconds at which the players start seeing the actual available bandwidth. This behavior is the key contributor to instability in the video bitrate.

TCP-based operation underneath the HTTP leads to another unintended effect.

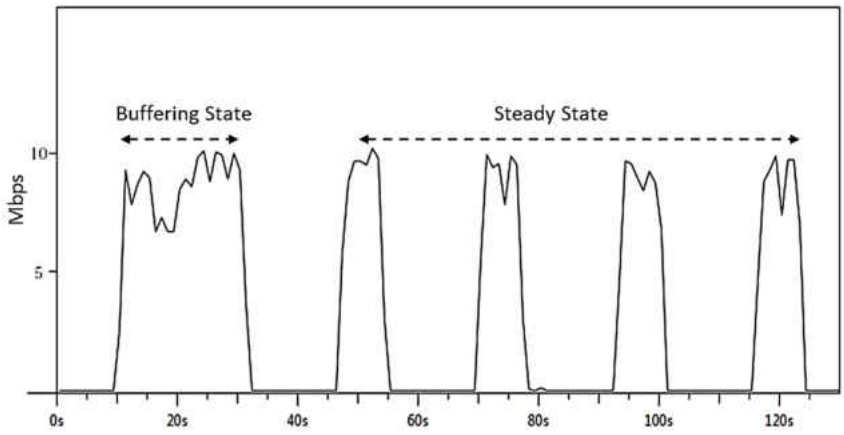

Figure 4: Player's States: Buffering state and steady state.

As TCP flows from competing players attempt to equally share the bottleneck, the players perceive that share and gradually converge to the same video quality. This may not be desirable at all times, given that player requirements may differ due the variations in screen size, type of content, or user preferences and priorities. For example, larger screens, sports and fast action, as well as higher data cap call for higher bitrates. In typical cases, neither clients or servers, nor the network can recognize these requirements across heterogeneous players from different content providers.

2.1.4 TCP FLOW CONTROL

Transport Control Protocol (TCP) [18] is a transport protocol that provides a reliable data transfer over an unreliable network. This is to ensure that the packet reaches the destination intact and in the right order. Also, when data packets are sent from a node to another in the network, the TCP protocol ensures that the sender node is not overwhelming the receiver by sending too many packets. This happens when the receiver node receives packets at a rate faster than it consumes. To prevent this from happening, the TCP protocol allows the receiver to send feedback informing the sender about its buffer condition, to adjust the sending rate. The TCP protocol implements this feedback mechanism by advertising its Receiving Window (rwnd) with every acknowledgment (ACK) packet sent from the receiver to the sender. This rwnd field is included in the TCP packet and contains a value that matches the spare room in the receiving buffer. On the other side, the sender node uses a sliding window protocol to control the sending rate according to the last advertised rwnd

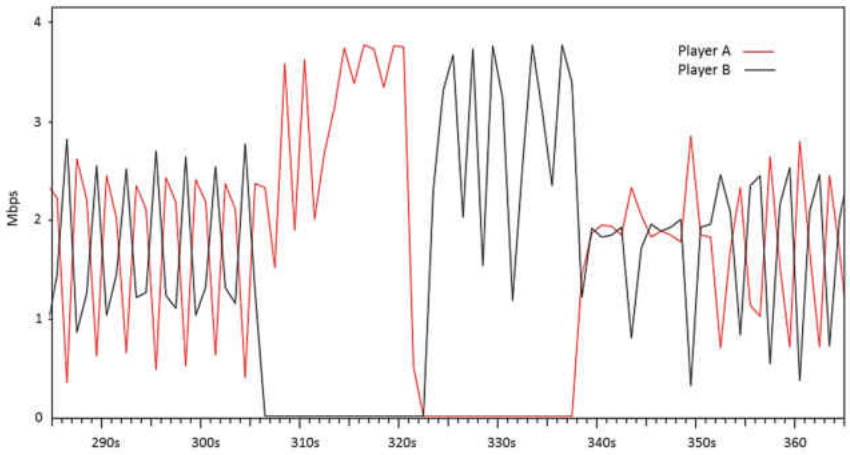

Figure 5: Traffic patterns of two competing Players.

received from the client. This process is intended to control the number of bytes sent by the sender, but have not been acknowledged yet.

2.1.5 SOFTWARE-DEFINED NETWORK

Software-Defined Network (SDN) is a promising technology invented to simplify network operation and management, and also to allow for innovation to promote the network infrastructure. This new technology introduces a new layered network architecture in which the control plane (i.e., control function such as routing, security, etc.) are decoupled from the data plane (forwarding function) of the network devices (e.g., switches), allowing for more sophisticated and flexible traffic management. As shown in Figure 6, an SDN instance mainly consists of three layers: application layer, control layer, and data layer [49, 40, 45].

The application layer is the part that utilizing the decoupling of the data layer and the control layer to achieve specific goals such as data collection or enabling a security mechanism [56]. The application layer communicates with the control layer via APIs using the northbound interface. On the other hand, the control layer is responsible for accomplishing the target application goals by manipulating the forwarding devices via a dedicated controller. In one direction the controller translates and conveys the application requirement down to the data layer, while in the opposite direction it provides the applications with a holistic view of the network topology (e.g., flow

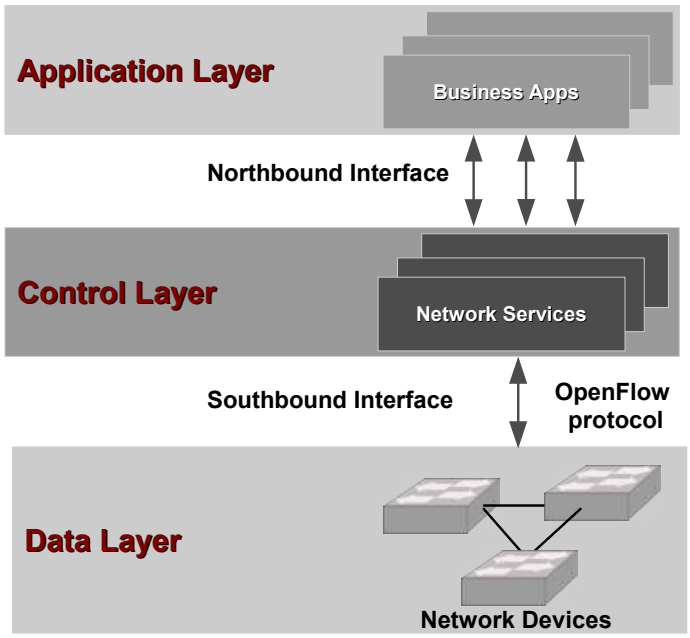

Figure 6: The layered architecture of SDN.

statistics and events), leveraging the link layer discovery protocol messages (LLDP). The southbound interface of the SDN-switch enables the controller to communicate with the data plane via a shard protocol, OpenFlow [50], which determine a set of messages that can be exchanged between these two planes over a secure channel. Therefore the control layer is composed of at least one Northbound-API Agent, the SDN Control Logic, and Southbound-API driver [60]. The data plane handles the actual packets according to the configuration received from the controller. It enables the controller to handle the forwarding operations and to perform other tasks such as advertising its capabilities, reporting traffic statistics, and sending event notification.

In the SDN-enabled switch forwarding rules, contained in the flow table, are associated with ingress packets to look up the port to which the packet should be transmitted. Therefore, once a packet is received, the packet header fields are used to identify the flow and execute one of the following actions: (i) forward the packet to a specific port, (ii) drop the packet, or (iii) send it to the controller for a flow rule installation if there is no matching rule. Using the OpenFlow protocol, the controller

can respond to the query according to the application policy.

2.2 RELATED WORK

2.2.1 CLIENT-BASED SOLUTION

In this section, we demonstrate the efforts done by the industry as well as by the research community to enhance the performance and solve the issues at the client side. The main focus of these solutions is to improve the application layer Adaptive Bitrate (ABR) algorithms. In general, the aim is to make this adaptive algorithm maximizing and the video bitrate in addition to avoiding stalls in the playback. In fact, achieving all these goals simultaneously is quite difficult and some tradeoffs have to be considered.

Performance of Commercial Players

Despite the continued efforts of major video providers such as Google and Netflix in improving their ABR algorithms, many studies, as well as our measurements, have revealed several performance issues with these players. For instance, studies such as [4, 61] have shown that users with several state-of-art players such as Netflix, Smooth Streaming, and Hulu can experience several performance issues including instability in perceived quality that dramatically degrades the QoE. In fact, this instability in the video quality has two adverse effects: first, switching too frequently is most likely to disturb and annoy users [12], and second, a considerable amount of duplicated video frames with different bitrates (for the same scene) are unnecessarily streamed by the end devices, adding an additional burden on the network [65]. In most cases, this causes a significant waste of both network resources and end-device resources, which might be scarce (e.g., mobile device). It is reported, for instance, by several studies [65, 66] that the YouTube player can download up to 40% of redundant video data which are discarded and not displayed to the user. As we explained in the previous section, the root cause of this oscillation is the released bandwidth at the steady state which may cause several players to over estimate the available bandwidth which would result in falling into unsustainable quality switches (to higher profiles) making some or all of them to switch back to the previous qualities, or in some cases even to lower qualities.

Recent studies [6] have extensively examined the behavior and the performance of six up-to-date ARB algorithms used by the most popular video players nowadays including YouTube, Netflix, Vimeo [76], and Bitmovin [8] players. They evaluated these players using several important quality metrics, which have the most impact on the video QoE such as instability, stalls, and startup delay time under different scenarios and network settings. Their findings show that the bitrate selection of these players are not stable nor fair under the competing scenario. In addition, some players can end up stalling for a considerable amount of time when tested under the dynamic bandwidth scenario. Moreover, YouTube player exhibits an aggressive behavior when competing with other players, thus causing these players to stream low quality videos comparing to YouTube player, leading to unfair and unbalanced QoE.

It is worth mentioning that our experiments also reveal that some video players including the YouTube player may still experience very long and terrible stalls in the video playback especially with live streaming. This undesired event usually happens when the video player encounters a sudden and major drop in the available bandwidth. When a stall event occurs, the player would have an extreme reaction by making a significant quality reduction to prevent further stalls. This major drop in the quality is usually unnecessary as in many cases the available bandwidth can allow the player to stream a higher quality without any issue. Instability and quality degradation are not the only the issues with this player, but also unfairness among the competing players is also confirmed by our experiments. Other widely used commercial players such as Adobe OSMF and Smooth Streaming suffer from similar issues as reported by [41, 3].

Research Efforts

Most of the existing studies such as [41, 35, 48, 47] attempt to improve the player's performance through improving the ABR algorithm of video players which is primarily used to determine the rate of the next video chunk. In general, the client-side ABR algorithms in the literature can be classified into three main categories: rate-based, buffer-based, and hybrid solutions.

To decide about the bitrate of the next video segment, rate-based solutions estimates the future bandwidth based on the past observation. Then the player selects the highest bitrate based on the estimated throughput. Rate-based solutions basically

depend on applying different throughput averaging techniques, such as exponential or weighted averaging to avoid the impact of outliers and thus stabilizing the quality. However, if the averaging period is too short, the instability might not be avoided, while if the averaging period is too long, the resulting estimation can not correctly reflect or adapt to the current network condition [52]. The algorithm proposed in [48] and [41] (FESTIVE) are among the well-known and effective rate-based algorithms in the literature used to cope with the stability, fairness, and efficiency issues. The main idea with the adaptive algorithm proposed in [48] is to use a smoothed HTTP throughput measurement based on the segment fetch time (SFT) to determine the bitrate of the next request segment, rather than using instantaneous TCP transmission rate. The algorithm compares the segment fetch time with the segment playback duration to probe the spare network capacity and detect congestion. For probing spare network capacity, an adaptive increase is used to choose a higher bitrate profile, while an aggressive multiplicative decrease is used once detecting network congestion to avoid stalls in video playback. FESTIVE on the other hand uses the harmonic mean of the download speed computed over the last 20 video segments. Other techniques such as [51, 52] are also throughput-based approaches which have similar ideas in estimating the throughput. However, in the highly dynamic network condition, having a good estimation of future network capacity becomes challenging. Consequently, works such as [34, 33, 67] propose to ignore throughput estimation and use an approach purely based on the buffer status. In other words, the adaptive algorithm picks up the video bitrate by only looking at the current buffer occupancy. Generally speaking, the algorithm selects a high bitrate if the buffer is full or near full. Otherwise, it picks a low bitrate to avoid stalls.

A hybrid approach is more popular and used by many commercial players nowadays such as YouYube. It uses both throughput estimation and buffer occupancy to decide about the next bitrate. PANDA [47] and SQUAD [] are among the most known and effective hybrid algorithms in the research work. The key idea with PANDA adaptive algorithm is to follow a probe-and-adapt approach in which the algorithm periodically increments the requested bitrate to probe the available bandwidth, while using the buffer fullness in addition to target bitrate to schedule next request.

Despite these intensive efforts to improve the performance of adaptive algorithms at the client side, the major issues that impact the video QoE remains unsolved

with these solutions as indicated by our measurements and also by several studies mention above[6]. We believe that this is due to the intermittent pattern of adaptive players, in addition to the inability of a player to realize both the current condition of the network and the existing of other competing players, thus the estimation algorithm clearly will not lead to an accurate estimation of the network capacity. Another downside of client-based solutions is the lack of the flexibility that prevents the network administrator to apply some policies and achieve specific requirements.

2.2.2 SERVER-BASED SOLUTION

The authors in [3] proposed server-based traffic shaping techniques to primarily overcome the instability problem. The main idea behind their technique is to adjust the streaming rate at the server side to be too close from the requested bitrate to avoid the OFF periods and thus to stabilize the bitrate. However, modifying a standard HTTP server, as their techniques required, may not be an attractive technique. Furthermore, their solution adds a significant overhead on the server since it requires the server to monitor the behavior of all connected clients, and then performing traffic shaping once the oscillation is detected. The work in [27], on the other hand, propose a system that involves a modification of both server and client device to achieve one of two goals: improving the video QoE, or reducing the cost (e.g., battery consumption) with negligible QoE degradation.

2.2.3 EDGE-BASED SOLUTIONS

To assist video players in selecting appropriate bitrates and co-existing gracefully inside the network, several approaches are proposed. One such approach is to implicitly cause the player to adapt by shaping the flows of the video streams at the router or DASH-aware proxy, including rewriting HTTP requests [31, 43, 9]. For example, shaping traffic at home gateway was firstly proposed in [30] to also deal with the unfairness issue. Their idea based on performing traffic shaping with two modes: "static mode" in which the shaping decision remains fixed till one player finishes downloading the whole video, and "flexible mode" in which the players are allowed to utilize any extra bandwidth. However, their proposed techniques can only work with unencrypted videos. They assumed that the home gateway can always intercept the manifest file and get all the information about the requested video including the available playback rates. In fact, this assumption is no longer valid with the major

video providers including YouTube and Hulu have already adopted HTTPS protocol in streaming videos. Moreover, their static shaping technique (which is their main technique) can leave a significant portion of the bandwidth unutilized causing bandwidth underutilization, while "flexible mode" can cause players to compete over the released bandwidth.

When multiple bottlenecks exist, a set of coordinating proxies is proposed as a mitigating solution, which includes information sharing between clients [58]. An SDN-assisted solution uses an OpenFlow-enabled system with an orchestrating element that explicitly informs the players which bitrate they should select [23]. Another proposal using SDN approach employs in-network caches to reduce the load of many unicast flows on parts of the network and mitigate some causes of the poor QoE [22]. The aforementioned approaches are implemented in the network, at the proxies or upstream elements, may use intrusive approaches to detect and manage video QoE, such as deep-packet inspection, bandwidth shaping, or request rewriting, and collaboration between the player and the network is at the application layer. The DASH standard has a proposal on Server and Network Assisted DASH (SAND) that specifies the control messages between DASH Aware Network Element (DANE) and players [71]. The goal is to formally specify interaction and establish a framework for co-operation between network and players.

In [21], a centralized control plane, deployed on the top of multiple CDNs, is proposed to assist players in selecting the optimal video bitrate in addition to optimizing video delivery in CDNs. This goal is achieved by monitoring the buffer state and experienced throughput via a thin layer deployed at the client device, which also sends periodic updates to the control plane and executes control decision. A similar idea of using a centralized node in a CDN to improve video QoE is proposed in [53]. Although these solutions conclude that a centralized controller can significantly help to improve QoE, such solutions, in contrast to our work, are not designed to manage a group of video players sharing a bottleneck wireless link. Consequently, the overall optimal QoE and fair share of network resources cannot be achieved, as these solutions are agnostic to the exact network state (the number of competing streams, background traffic, etc).

The key difference between our system and related proposals are that we extend the SDN paradigm to the client device so that it directly communicates with the network in a more fundamental SDN sense (via Open vSwitch), as opposed to messaging

the player application. Then, we implement the system on the real smartphones that uses the cellular interface, which is considerably more challenging than a simulation or WiFi implementation. Finally, we present the use case that addresses previously ignored aspects of bandwidth sharing by competing players.

2.3 SUMMARY

In this chapter, we started with presenting a background about video streaming technologies focusing on the most recent technology, HTTP adaptive streaming. We also briefly introduced the TCP flow control mechanism. Then we gave a background about the SDN technology and its main layers in addition to its roles in simplifying the network operations and management. The second part of this chapter was dedicated to describe the research and industry efforts toward improving the video QoE. We presented three main classes of solutions: Client-Side, Edge-Side, and Server-Side solutions, and demonstrated the downsides of each of these class.

CHAPTER 3

TRAFFIC AND PERFORMANCE ANALYSIS OF HTTP ADAPTIVE STREAMING

In this chapter we analyze the traffic pattern and evaluate the performance of commercial adaptive players via testing a recent version of one of the most common adaptive video players "YouTube". We limit our analysis on YouTube as the performance of other commercial players were extensively studied in [4, 61] and the performance issues were detected and highlighted. Moreover, in contrast to other popular players such as Netflix, YouTube uses encryption (HTTPs) to preserve their users privacy, thus our goal of this analysis is not only to reveal some performance issues, but also to understand the traffic pattern of adaptive players and to get some insights on how to build an efficient streaming system that can also work with encrypted traffic.

We divide our analysis into two main scenarios: non-competing scenarios, where only one video player is streaming, and competing scenarios where two or more video players are competing for the bandwidth over the same bottleneck.

3.1 EXPERIMENT SETUP

In the experiments, we use a laptop as Wi-Fi Access Point (Wi-Fi AP) running Ubuntu OS. This Wi-Fi AP is also connected to the Internet using Ethernet interface. In the Wi-Fi AP, we installed OpenvSwitch (OVS) [74] and added the wireless interface (wlan0) of AP as a port with the OVS bridge. Consequently, all the traffic coming or going to any of the connected smartphones should pass through this OVS. In addition, we use Linux Traffic Control (TC) [14] of the Wi-Fi AP to control or limit the bandwidth of the video traffic. In the experiment setup, we have used three Android smartphones (two Samsung S5 and one Nexus 5) which are all connected with the Wi-Fi AP. Moreover, we use iperf [73] to generate UDP traffic as a background traffic. In our smartphones, we have installed and used the latest version of YouTube app. This YouTube app comes with a "stats for nerds" option that enable

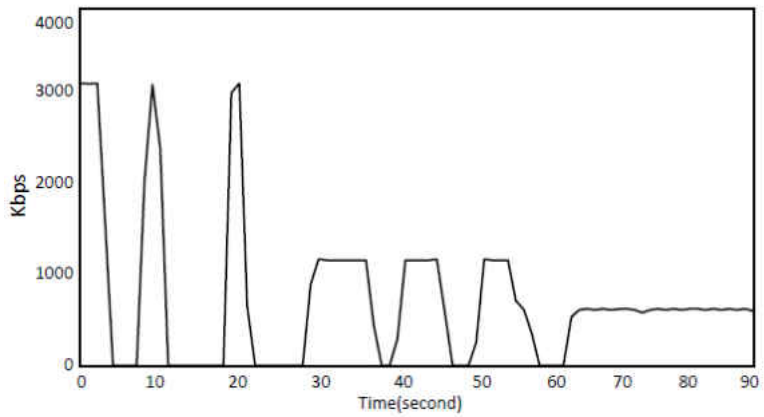

Figure 7: The correlation of ON/OFF periods with video encoding rate.

us to observe the quality requested by each player in addition to the buffer status and the estimated throughput value.

3.2 NON-COMPETING SCENARIO

In this scenario, we capture and study the traffic patterns generated by only one video player with no competition from other players. We use different videos encoded with different bitrates and make the player streams from one of the YouTube servers. For the first experiment, we stream a video with a 480p quality resolution which encoded at 650kbps (selected manually among different available resolutions) and set the network capacity at different values to see its impact on the video traffic patterns. We use the Linux TC at the Wi-Fi AP to control the available bandwidth of the video player running on the smartphone. In the beginning, we allow the video player to stream at 3200kbps, and then we reduce the bandwidth after 27 seconds to 1200kbps for 30 seconds before reducing it again to 650kbps. As we can note from Figure 7, the duration of OFF period at the steady state shrinks as the throughput rate drops and getting closer to the video encoding rates, results in totally vanishing the OFF periods in the last 30 seconds. Therefore, there is a strong correlation between the length of the OFF period and the video streaming and encoding rates. The existence of the OFF periods clearly indicates that the playback of the current quality profile is utterly stable, and experiencing degradation in the viewing quality (e.g., switching to a lower quality or stalling in the playback) is basically not possible as long as

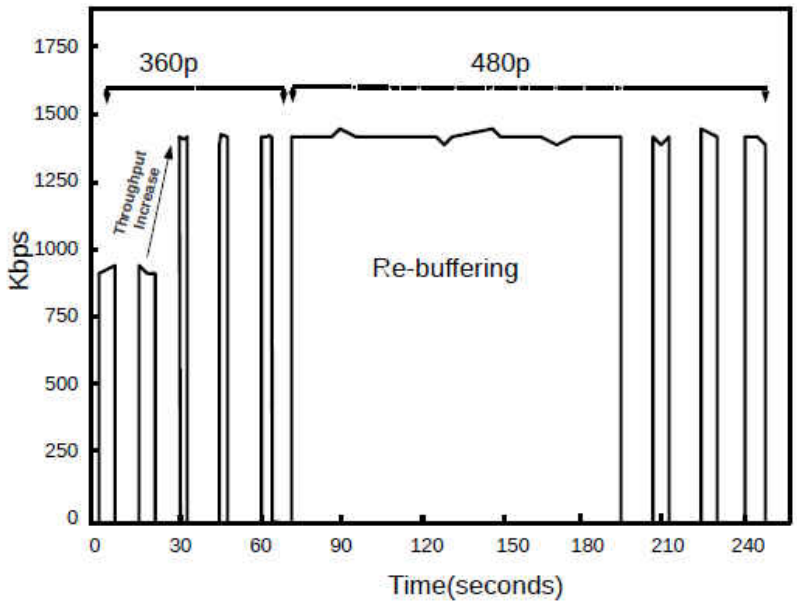

Figure 8: An increase in the throughput causes a change in the traffic pattern.

there is no drop in the throughput. On the other hand, the disappearance of the idle period of the video stream as a result of a drop in the throughput may indicate that either the throughput is equal (or slightly above), or below the encoding rate. In the former case, the video playback will not be affected, while in the latter case, the video playback may or may not be affected depending on some factors such as the throughput drop level, buffer condition, and the length of the remaining playback duration. Therefore, it is extremely important for the performance to distinguish between these two cases and determine whether the drop in the bandwidth would affect the playback rate. One technique for distinguishing between the two cases is to determine the video average bitrate.

Since the video traffic is encrypted, knowing the exact value of the video bitrate is not possible. However, an estimated value would be obtained if we can calculate the average chunk size and divide it by the average duration of ON and OFF periods. In fact, this becomes quite possible with the developing of OpenvSwitch which allows for collecting statistical information about the traffic flows in real time. For example, we see in Figure 7 that at second 57 the throughput drops from 1200kbps to about 650kbps which completely removes the OFF periods from the traffic patterns. Now by dividing the average chunk size (800KB) by the average length of both ON and

OFF periods (10 seconds), the estimated bitrate will be 650kbps which is too close to the actual value (600kbps). Therefore, the drop in this example can definitely affect the video traffic in a long play, and a well-designed assistant system should react to prevent such possible degradation in the video viewing quality. Having this situation, it will be more efficient to estimate the number of seconds in the buffer at the time of the drop. This is very important for the performance of the overall streaming system. For instance, if we get an estimated value of the video encoding rate and know that the buffer has about x seconds of video frames at the time of the drop, then the system will be able to infer when the buffer can turn empty and start harming the viewing quality. In case the degradation would happen after a considerable amount of time, the system can safely defer its intervention for quite long time waiting for the condition to be inherently improved (e.g., waits for one stream to finish downloading a video).

As a matter of fact, each streaming application has its own setting for the buffer. For instance, our measurement reveals that YouTube app uses a 20MB buffer. This means that at the time of the drop, the player can continue playing the current quality for about 20MB divide by an estimated average bitrate. Thus, when the available bandwidth drops below the encoding rate, we can have an estimated knowledge of when the player might switch the quality or be subjected to playback stalls. It is worth mentioning that the disappearing of OFF periods not only happen when the throughput drops close to video bitrate.

In fact, we identify another scenario that causes the idle periods to vanish. Figure 8 shows the intermittent traffic pattern of a video player streaming video of 400kbps bitrate disappears for a considerable period of time when the available bandwidth increased from 900kbps to 1400kbps. The increase in the throughput happens at second 30, and starting from second 68 the OFF periods disappear for nearly 140 seconds (persistent pattern) before showing up again at time 207. By looking at the "stats for nerds" option, this is interpreted as the player increases the quality and re-enters the buffering state in which some of the low quality packets in the buffer get replaced by high quality packets for the same scene. This quality change can be inferred from the traffic pattern by looking at the chunk size before and after the buffering state. We can clearly see from Figure 8 that the video chunks streamed after the persistent period is much larger in size than the chunks streamed before. This change in the average chunk size is a clear sign of an increase in video quality

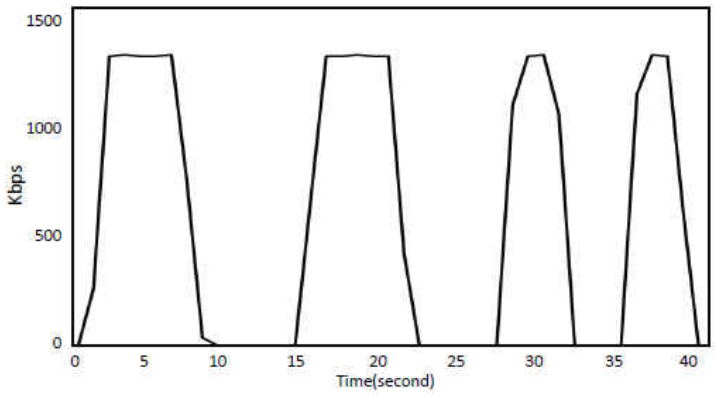

Figure 9: Traffic pattern of high motion video.

as the chunks of a high quality profile typically have higher bitrate and size than the lower quality. In case that the OFF periods vanish with no change in the throughput value, the system can interpret this as the client jumps to time offset in the video playback.

While switching to a higher video quality, the player, as we mentioned before, flushes all the buffered packets of lower quality and replaces them with a higher quality for the same scene. This behavior aims to enhance the QoE by switching and playing a better quality as soon as the network condition gets improved. However, switching from low to high quality would introduce a significant waste in the device and network resources reaches up to 33% of redundant traffic, which would also

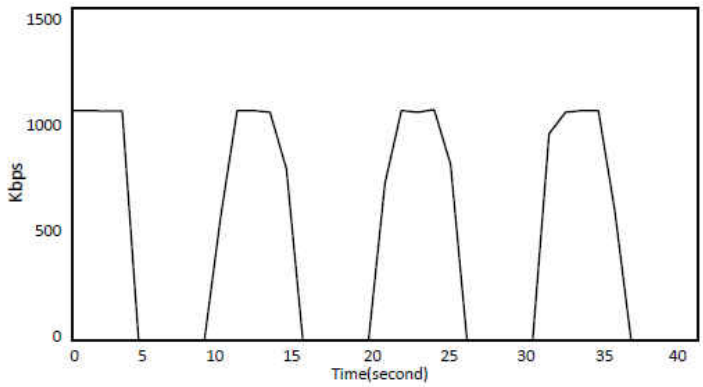

Figure 10: Traffic pattern of low motion videos.

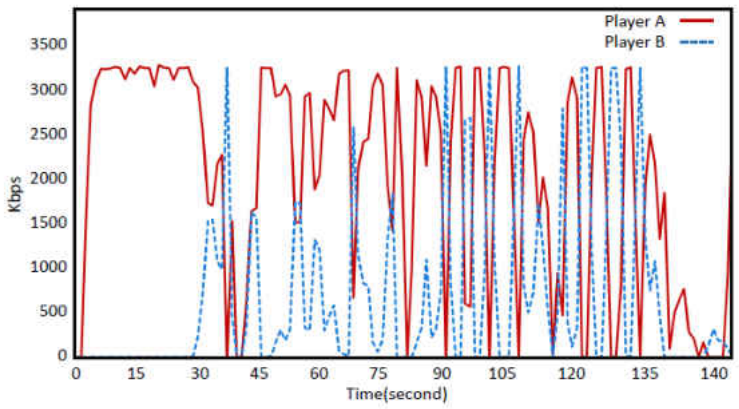

Figure 11: Two video players compete for bandwidth over the same bottleneck.

increase the load on the server [65]. Note that several quality switches can lead to considerable amount of redundant traffic, resulting in an unacceptable increase in the cost on users with limited cellular plans.

In videos, we see a mixture of slow and high motion clips such as sports games or action movie, while in other videos, we see slow motion clips such as news. Note that, the type of a motion clip in a video has a direct impact on the video encoding rate. Similarly, the regularity of the ON/OFF period also depends on the variability of the video encoding rate. For example, in Figure 9, the first two chunks of the video are for high motion scenes, and have higher encoding rates and data size compare to the following two chunks with slower motion scenes. Thus the change between the high and slow motion clips changes the ON/OFF periods to have different length as in Figure 9. On the other hand, the video chunks represented in Figure 10 have slow motion scenes with almost the same encoding rates and date size. Thus, in this case, we observe no changes in the length of ON/OFF periods.

3.3 COMPETING SCENARIO

In this section, we start analyzing the video traffic in more realistic scenarios where multiple players stream videos concurrently over the same wireless access point. Figure 11 shows the flow patterns of two devices playing video over the same wireless AP. To ensure that both devices are exactly under the same condition, we place both devices at the same distance from the AP and set the players to request the same

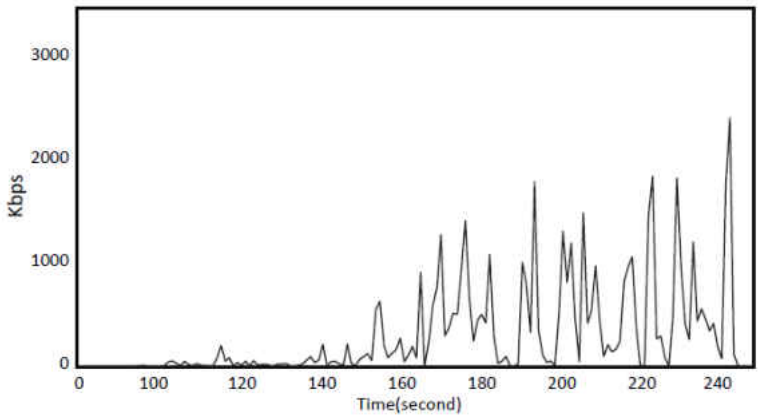

Figure 12: The flow of one player competing with two other players.

video. In addition, we generate background traffic in the wireless network using Iperf running on a third device to mimic a real-life scenario and make the flows compete. We first start Player A, and after 30 seconds we start Player B, so at the beginning Player A achieves high throughput, around 3200kbps, which permits to stream a high quality profile. However, as soon as the competing flow of Player B shows up, the throughput temporally drops to 1700kbps. Figure 11 shows aggressive competition between the two flows, resulting in extreme fluctuation in the throughput. Player A is clearly getting much higher throughput in average than the Player B which is experiencing a very low throughput. This unfairness in sharing the bandwidth lasts for about two minutes before eventually and slowly converse to a fair state as a result of using TCP as the transport protocol. This slow increase in the throughput not only reduces the user engagement and affect the video QoE, but also can lead Player B to pass through all the quality levels before reaching to the final quality that fits with the fare portion of the bandwidth. This several quality switches results in flushing out most the packets from the buffer and replacing them with higher quality at every switch, wasting the network and device resources.

We also examine the influence of the competition between three players. We start running two players and after 80 seconds we start the third player. For clarity, Figure 12 shows only the flow pattern of the third player. As can be seen from the figure, the third player can only obtain a very small portion of the bandwidth, making the player unable to buffer enough packets to start the playback for about 40 seconds. Under this circumstance, the user could get frustrated, and decide not to stream

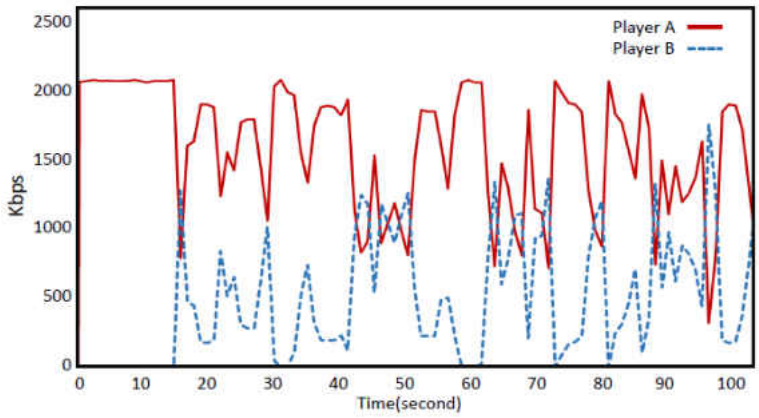

Figure 13: The traffic pattern of two players streaming different video bitrates while competing over the same bottleneck.

video over the network. Note that having a short start-up delay time is one of the most important metrics for the QoE. Therefore, it is essential to have a mechanism that assures a good performance for all players.

Figure 13 shows the impact of different video bitrates on the flow competition between two players, Player A and Player B. We disable the YouTube auto quality selection on both players, and manually set the quality levels at 720p (130kbps) for Player A and 360p (55kbps) for Player B respectively. We start both players at the same time under the same conditions (i.e., same video and same distance from Wi-Fi AP). Figure 13 shows that Player A with a higher bitrate stream wins the competition and dominates the bandwidth. This explains, why Player A in the first experiment has higher throughput than Player B which starts later. The reason is that when Player A starts, it gets enough throughput to request high quality video, while Player B does not find much bandwidth available, thus ends up requesting low quality video.

Typically, the wireless link conditions of different devices in the same network vary according to different conditions such as their distances from the AP. Figure 14 shows the result of an experiment in which we use two devices with different link conditions. At the beginning, we start both players (A and B) while placing both devices close to the AP, and then we slowly start moving the device that runs Player B away from the AP. As a result, Player B starts to observe throughput reduction around time 170s, and the player also loses the competition against Player

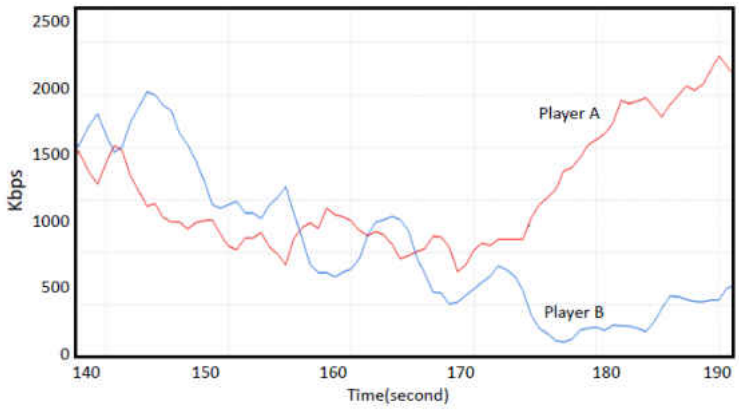

Figure 14: The impact of the wireless link condition on throughput competition.

A which starts to experiencing higher throughput. Note that, in this case, we have two factors contributing to the quality drop of Player B: the link condition and the competition between the video flows. Although, we have no control over the link condition, smarter network management can address the competition issue to have an acceptable video quality for all users.

Now we turn our attention to understand the impact of the intermittent traffic pattern of HTTPs adaptive streaming on the QoE. We study and analyze the traffic pattern of two players at the steady state. Figure 15 shows the smoothed throughput of these two players. Before second 350, both players (A and B) were stable and achieved good and fair throughput values, but after 350 seconds, we observe a drop in their throughput for about 100 seconds followed by a dramatic increase in the throughput of Player A and a major drop in throughput for Player B.

To understand why Player B loses the competition, Figure 16 zooms into the first 40 seconds of the previous figure. This figure shows how both players were utilizing the OFF period of each other for the first 25 seconds, and because of a long idle period of Player B (between second 315 and 325), Player A experiences a huge increase in the bandwidth during that period. This increase in the throughput causes Player A to switch to a higher quality profile and to reenter in a buffering state, causing Player B to lose its idle periods. Therefore, we can infer that Player B was relying on the Player As released bandwidth (at OFF periods) in maintaining the current quality level. As a result, both players start competing for bandwidth causing their throughput values

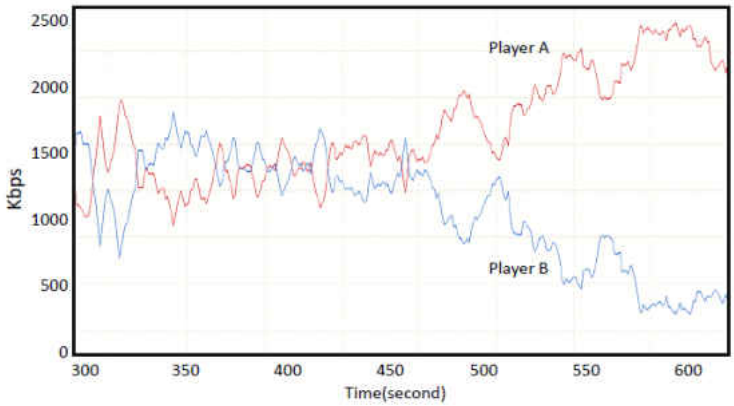

Figure 15: Smoothed throughput averages of two video streams competing at steady state.

to go below their current video bitrates. Consequently, their buffers start quickly draining, and because Player A can slightly gain more bandwidth than Player B in the competition, the buffer of Player B gets drained before Player A. This causes Player B to switch down three quality levels at one time (from 720p resolution, encoded at 1200kbps, to 240p encoded at 400kbps as confirmed by examining the change in the player's playback resolution) in order to prevent stalls in the playback. This experiment highlights a problem raised from the intermittent traffic pattern of HTTP adaptive players and confirms the need for a mechanism to enhance the performance.

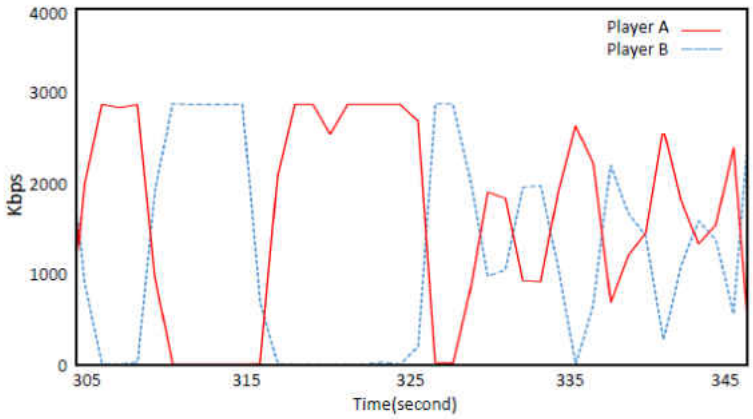

Figure 16: Player A utilizing the huge bandwidth of Player B causing Player A to switch and dominating the bandwidth.

3.4 SUMMARY

In this chapter, we studied the traffic pattern of HTTP adaptive streaming protocol through YouTube player. The aim was to analyze and to understand the ON/OFF pattern in estimating the video bitrate and buffer size for encrypted video streams. This traffic analysis was presented as a stand-alone scenario, while in the competing scenario we examined the performance of the video players when as they compete over the bandwidth. Our focus in evaluating the performance was mainly on instability, unfairness, and start-up delay time. As our results showed, under this competing scenario, the YouTube player was found to be unstable and can cause long-startup delay time. In addition, we found that competing over bandwidth can also lead to an unfair share of network resources, results in unbalanced video QoE.

CHAPTER 4

FLEXSTREAM FRAMEWORK

In this chapter, we introduce and describe our proposed system FlexStream which is particularly designed and developed to maximize the overall video QoE in the network. This can be achieved by managing the network resources in an efficient way according to the network condition and various context information. We start with giving a short overview of how the system actually works and how different system components interact and cooperate to optimize the QoE before providing a detailed description of the role of each system component.

4.1 SYSTEM OVERVIEW

FlexStream is designed to maximize QoE in the access network by allocating the highest sustainable bitrates to adaptive players while ensuring: (i) minimal variations in the quality, (ii) minimum number of stalls, and (iii) well-balanced and fair QoE. FlexStream achieves this goal using a hybrid approach, which takes advantage of both centralized and distributed components. While a network element, which has a global view of network conditions, is primarily employed to manage resource allocation for video flows, monitoring and policy enforcement tasks are offloaded from the network to end-devices, via lightweight software agents. This alleviates the need for intrusive, large and costly traffic management solutions within the network, or modifications to servers that are not feasible in practice [3, 10, 44].

Figure 17 shows the high-level overview of our proposed system "FlexStream", where the end device simultaneously opens two communication channels with two different network entities, the Global Controller (GC) and the media server (HTTP server). The control channel is used by the end-device to connect to the GC to receive commands that control and manages the data rate of the second channel (the data channel) according to the global optimization policy. Intuitively, the data channel is used to stream video content from the video server. To enable the control over bandwidth, we deploy both SDN data plane (extended OVS) and the control plane (SDN controller) on mobile devices. The OVS is installed with a new action

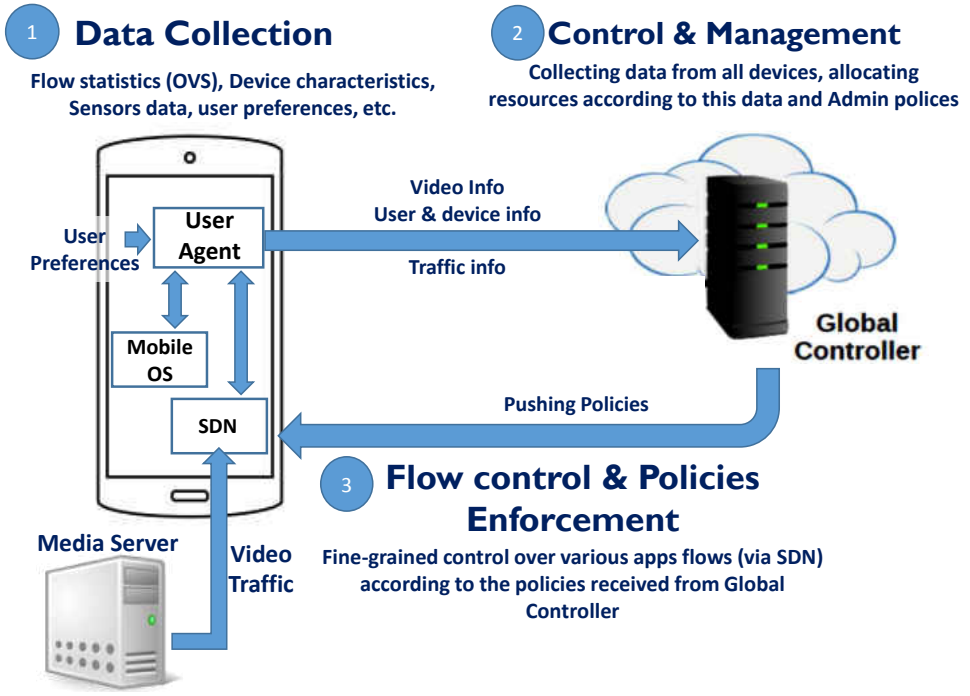

Figure 17: FlexStream Overview.

added to the kernel module that compels the device to reduce the download rate to a certain value dictated by the SDN Local Controller (LC). This control of the bandwidth consumption at the end device provides the ability to stabilize the video quality through avoiding the competition among video players.

As the LC is deployed on the end device and hence does not have any knowledge about network conditions beyond its single link, we utilize the GC in the cloud (or in the wireless network infrastructure) to have a global view of the network condition. GC receives feedback from the end devices and possibly from the network infrastructure too (in case of a cellular network), so it can acquire a good knowledge of various context information about the users and end-devices in addition to learning about active streaming sessions and network load. Consequently, GC utilizes these information to detect bottlenecks and activate the bandwidth control when necessary. Therefore, the GC has a centralized role in managing the bandwidth according to optimization policies that maximize the QoE in the network or achieves any other objectives. The device agent, on the other hand, plays an important role in facilitating the communication between the local SDN components and the GC. It also

provides the GC with the necessary information to make the best possible management decisions, while translating GC's commands and policies to be understood and enforced by the local SDN components.

We now can conclude the process of managing video sessions as follows: The video session management process starts with data collection at end devices. The DA collects information related to the video session, device characteristics, and context information. This information is then reported to GC for processing. Upon receiving information from the DAs in the network, and possibly from the network infrastructure too (in case of cellular or enterprise networks), GC can detect bottlenecks and take control over network resources, if necessary. In this case, the GC uses the received information as input to some optimization policy that maximizes QoE in the network or achieves any other objective. Bitrate allocation along with some dynamic policy rules, such as location or time context restrictions, are then sent to DAs for enforcement. Therefore, one of the primary tasks of DAs is to enforce GC policies.

As we mentioned before, Flexstream takes into consideration different context information as inputs to the global policy implemented inside GC in managing the bandwidth to provide a fair share of network resources. This context information can be classified into four main categories: (i) user context: priority class, preferences, and location, (ii) device context: screen size and battery level, (iii) network context: link condition and traffic types, and (iv) environment context: surrounding luminance. In fact, considering context information is crucial for achieving a high and balanced QoE in the network. For example, if we have multiple users with two different priorities (e.g., high and regular class) streaming videos over a shared bottleneck, thus allocating more resources to those with high priority (e.g., emergency services) to ensure higher video quality is essential for enabling service differentiation. Similarly, assigning resources to devices in accordance with the type of the generated traffic (video or background) can substantially improve the overall users experience.

To further realize the importance of considering various context information in optimizing the service, consider the scenario in Figure 18 where four clients simultaneously stream videos over a shared bottleneck. As client A has higher priority (e.g., paying more money for the service or has an high rank in an organization) than the other three clients, the system should provide better service (i.e., video quality) to client A by assigning more bandwidth (or guarantee a minimum rate) to A's player.

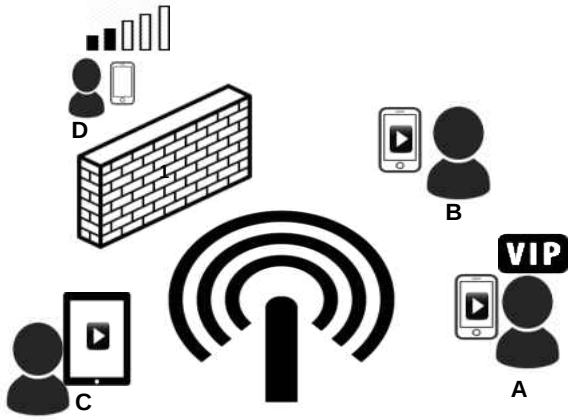

Figure 18: Example Scenario.

Similarly, to maintain a balanced QoE in the network, client C might get a higher quality than clients B and D due to its bigger screen size. On the other hand, since client D device experiences a poor signal because of its distant location from the AP along with several obstacles in the path, the system should be aware of this situation and react to prevent other competing players from starving it. Therefore, considering all these factors when managing the bandwidth can considerably help FlexStream improving the overall QoE in the network.

We note that GC could be deployed in the public or network operator clouds, or at the network edge. We envision that the GC can be operated by content providers, network operators, or individuals in their private networks. Deployments at the wireless access points for home or enterprise networks, or at mobile edge nodes, would give GC a better view of network conditions and device contexts, enabling it to manage bandwidth over the shared bottlenecks more effectively. Furthermore, it enables us to easily conduct service differentiation among classes of devices. In addition, through communication between the end devices and GC, Flexstream takes into consideration different context information as inputs to the global policy implemented inside GC in managing the bandwidth to provide fair share of network resources.Different device contexts, comprised of video stream information, screen sizes, priority levels, radio signal characteristics etc., are used as inputs to the global policy implemented inside GC.

4.2 SYSTEM ARCHITECTURE

Figure 19 shows the main system components and their key modules. Here we briefly describe each component and its main tasks.

4.2.1 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 functions of several modules, in addition to mediating communication between local components and the GC. Crucial to the operation of FlexStream, the DA listens to all important events (e.g., start streaming, bitrate switch) and monitors local context related to the video stream, and promptly informs the GC for appropriate action. To reduce the overhead and keep the DA as light as possible, we set the DA to run every 2 seconds, which approximates the minimum interval between two consecutive video segment requests generated by adaptive players in the steady state [4]. However, this sleeping period can be set dynamically by observing the player OFF period average length (using SDN data plane) to further reduce the overhead. At each run, the DAs send an update to the GC. An alternative approach is that reporting an update would only be triggered by a change in the context including the average throughput, to reduce bandwidth overhead and avoid overloading the GC. However, even with periodic updates, the overhead would be limited with such time interval. In addition, since most adaptive players use 30 seconds of playback buffer or more [55, 35], the GC can always react before the QoE would be impacted in practice.

The DA consists of several modules. The *Context Monitor* is initially responsible for observing and reporting a streaming event by combining `netstat` log data with the data fed by *HTTP Inspector*. The main task of *HTTP Inspector* is to report video encoding rates by inspecting the manifest file sent to the player prior to the streaming session. The HTTP Inspector then keeps monitoring and inspecting the HTTP requests sent by the video player to the server to know about the bitrate of the next video segment, but only notifies the QoE Monitor module if there is a change in the requested bitrate. If the session is encrypted, then the *Bitrate Estimator* module is alternatively invoked to estimate the bitrate, which initially relies on recognizing the streaming application and then following the bitrate guidelines of its provider. Once the player reaches the steady state, the *Bitrate Estimator* starts a process of

monitoring video flow statistics and patterns (segment size and ON/OFF duration), through SDN components, to make another estimation and adjust the assigned resources accordingly. The aim of this adjustment is to avoid inefficiency in utilizing network resources, when the assigned bandwidth is too high, and also to avoid any degradation in QoE, when the assigned bandwidth is too close from the video bitrate. Note that this module uses the length of ON/OFF periods of the adaptive player as an indicator of the error in bitrate estimation. For instance, if the players generates a long ON period (long download) followed by a short OFF period (e.g., ON=10 s and OFF=1 s), then we consider the allocated bandwidth is inappropriately tight, and vice versa.

In addition, *Context Monitor* oversees the device context, which may contain both physical device characteristics (e.g., screen size, radio capabilities, network interface), user preferences (e.g., preferred interface), and administrative context (e.g., priority class). It provides the function to monitor and report the device context to the GC. 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 to ensure the sustainability of the current bitrate. Therefore, there is no 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. Note that the throughput is estimated by monitoring the flow statistics through SDN components (local controller and OVS). The *QoE Monitor* is also required to report any switch in the video bitrate requested by the player to the GC.

At the higher level, the *Policy Engine* is responsible for maintaining and enforcing the policies received from the GC. 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 GC can restrict streaming HD videos (due to high bitrate) at specific times or locations (base stations), or set the bitrate of background traffic to different values based on the application type that generates the traffic.

Since bandwidth control is a mechanism to implement a policy in our use case, the *Rate Handler* module translates the bandwidth share assigned by the GC into whatever the local implementation requires. As detailed later in the paper, we use adjustment of the TCP receive window (*rwin*) to limit the TCP connection throughput, as one example mechanism of control applicable to the use case of video streaming.

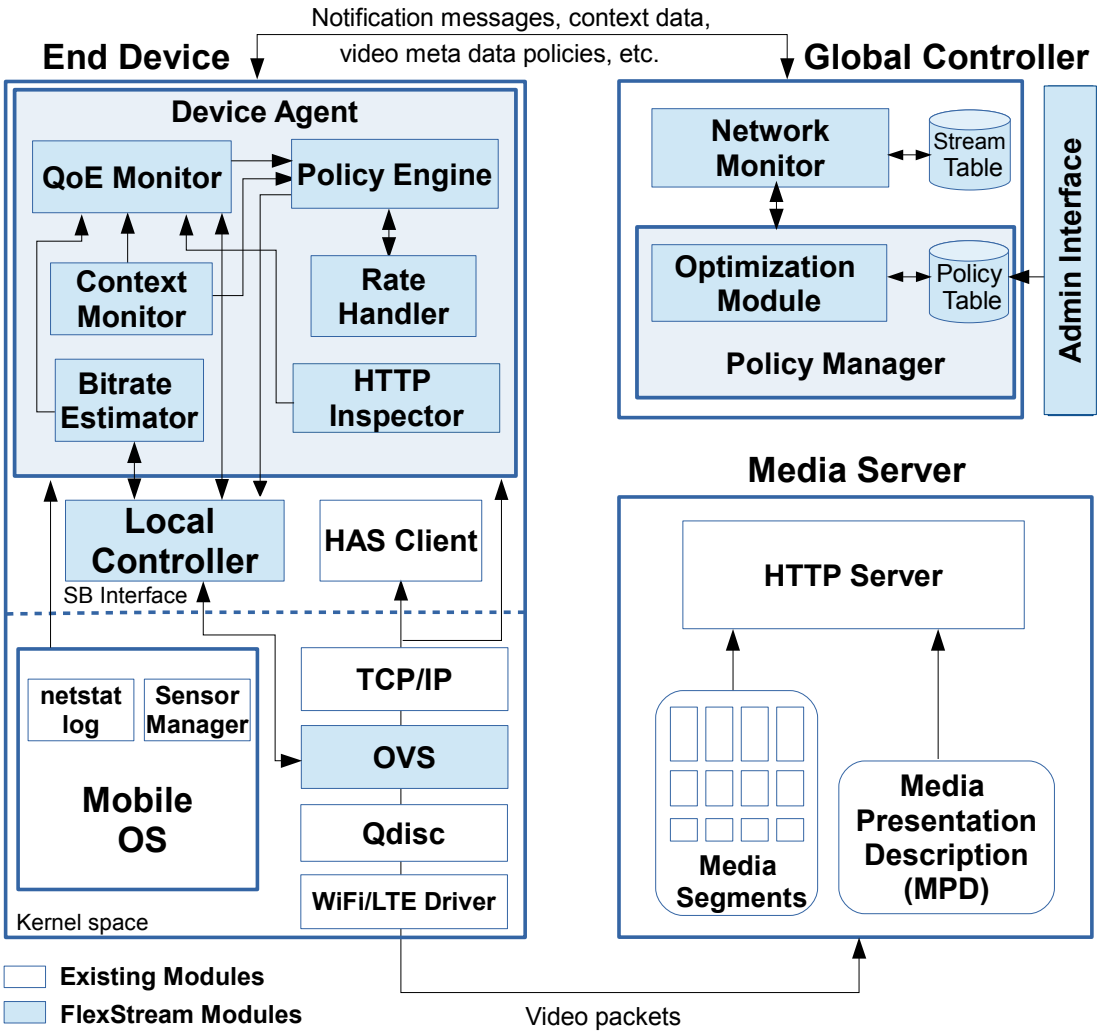

Figure 19: FlexStream system architecture.

Thus, the Rate Handler's role is to derive the appropriate *rwin* from Round Trip Time (RTT) of video stream packets either upon request or if there is a significant change in the RTT that could impact throughput ($\pm$ 100 Kbps).

To modify TCP packet *rwin* in the data plane, we leverage OVS on the end device. OVS is a programmable software switch which acts as the data plane in the kernel space of the system, controlled and managed by the local controller via OpenFlow protocol. In FlexStream, OVS uses the extended OpenFlow protocol to receive the *rwin* modification action and 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. Note

that in addition to this newly added action, OVS is leveraged by FlexStream 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 flow on the same device, one of the options that FlexStream uses to maintain high QoE is to utilize the OVS in routing 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, the controller can instruct the OVS to return the flow information maintained in its flow 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.

4.2.2 GLOBAL CONTROLLER COMPONENTS

The GC consists of the *Network Monitor* and *Policy Manager*. The *Network Monitor* tracks network conditions and device states, such as video QoE of all devices under control. To ensure stable and fair QoE in the network, the GC collects and maintains device context including video meta-data, such as bitrate profiles, in the *Stream Table*.

As part of monitoring task, the GC allows DAs to send several notification messages in regards to the performance. The GC, in fact, can have different interpretation of the same notification messages. For instance, if a considerable drop in the average throughput is reported while the wireless channel of the reported device is in good shape, the GC interprets it as an indication of network load increase which can lead to players competition. However, if the drop in the throughput is reported in conjunction with a bad wireless link from only a single device, the GC interprets this drop as a result of week wireless signal. In this case, the GC may decide to activate the control over the bandwidth in order to prevent other players from getting its fair share of network resources, which would harm the user experience.

Note that some of the context information is only required at the beginning of

the streaming session, such as video bitrate profiles and device characteristics, but 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. For admitting a new stream, GC also runs an admission control algorithm [77] to ensure that there is enough resources that supports at least the lowest available video bitrate. This is always done right after inspecting the manifest file to get video meta data.

4.3 SUMMARY

This section introduced FlexStream Framework with the focus on defining its operation process and main modules. We started the chapter by giving an overview of the FlexStream and how the process of managing the video sessions is initiated and managed. In addition, we presented the two SDN components that we deployed on the mobile device and their main role of enforcing the GC policies. Then we described the system main components/modules in details including DA and GC. We also described the main modules of these two main components (DA and GC) and how these modules interact to achieve certain tasks.

CHAPTER 5

IMPLEMENTATION

We start this chapter by describing the video session workflow, and then we move to explain some implementation details of FlexStream's main components.

5.1 SESSION WORKFLOW

As soon as the client joins the network, the device agent establishes a communication channel with the GC through sending a registration request message. This message should include the wireless access point unique identifier in addition to client's account information. Note that in cellular networks, it is possible for a client to get the tower ID number, while in the WiFi networks, we can use the MAC or IP address, which should be public, of the wireless AP as a network identifier. The GC in turn retrieves the client profile from its internal database to determine the client's privileges. The client then is added to the active client table leaving a streaming flag unset.

During the initial startup phase, certain information such as client device capabilities are shared with the GC. Since the user agent is designed to remain active in the background, it is leveraged to detect the start of the streaming event via monitoring the active network sockets (netstat logs) and their bindings, which is available in the Android OS. Upon detecting the streaming events, different device agent components are activated which initially send a streaming notification message to the GC. This message carries different pieces of information such as video meta data and context information. The QoE inspector parse the Media presentation Description (MPD) file to get video meta data and send it to the GC. It also continues inspecting the HTTP requests and reports any change in current video bitrate or in the average throughput. Similarly, the context monitor shares any major changes in the device or client context with the GC. On the other side, once the start streaming message reaches the GC, it adds the new stream to the active streaming table and checks the network condition. If the network becomes saturated and the players start competing for the bandwidth, the controller immediately activates the control over

the bandwidth by invoking the optimization function. This function distributes the bandwidth among the end devices according to the optimization policy, and then sends each device its fair portion. On the end device, once the rate limiting handler receives the assigned rate, it calculates the TCP receiving window which will be then passed to the SDN local controller through the API northbound. The SDN controller in turn uses the OpenFlow protocol to add a new flow to the OpenFlow table on OVS with our new action that modifies the TCP receiving window to limit the media server sending rate.

5.2 IMPLEMENTING OVS BINDING ON ANDROID

It is challenging to bind OVS to the local network stack and add the physical network interface (e.g., WiFi or 3G/LTE) as a port to OVS. This requires us to remove the IP address of the physical interface and assign it to the OVS to enable forwarding traffic to its internal device, as is the case with other Linux bridges. Otherwise, the traffic will stop at this interface and will not reach OVS. While possible to successfully implement such configuration on the WiFi 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 is unattainable as it 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 WiFi access point as a mediator. However, this approach limits the practicality of such a solution. To overcome this challenge and allow for direct experimentation over cellular network, we should find a way to properly bind the cellular interface (i.e., adding it as a port to OVS) without dropping its connection with the base station while allowing the traffic to go through OVS. This is achieved by adding 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 hits 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.

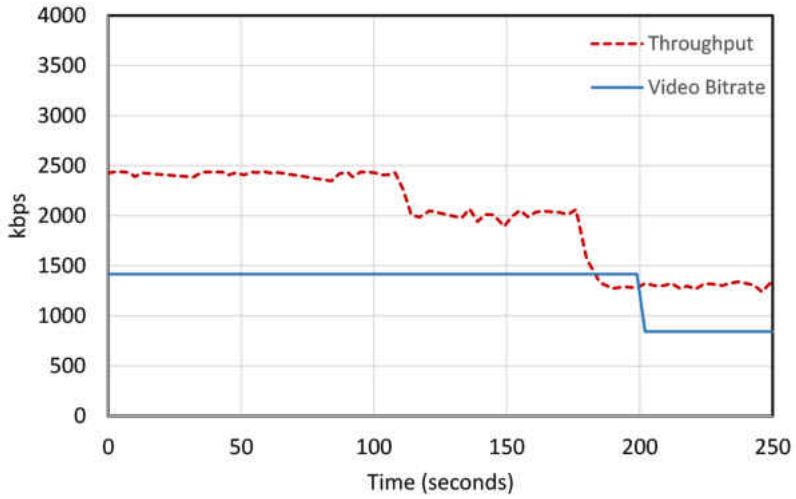

Figure 20: Performance of the TCP flow control in shaping the traffic.

5.3 RATE LIMITING APPROACH

The popular technique to limit the bandwidth for ingress traffic on Linux based devices is to use TC Qdisc policing of iproute2 utility package. However, this tool implements the policing via Token Bucket Filter which works by discarding the traffic when exceeding a certain limit. This implies that successfully received packets will be dropped at the end device which lead to unacceptable waste in the device and network resources. Moreover, this technique of limiting the traffic can cause the server to take a long time to respond to the shaping action, and before responding the packets will keep coming at the same rate causing more packet loss. These two drawbacks (packet loss and rate limiting delay) makes the TC tool not ideal for limiting the rate by FlexStream. To avoid these drawbacks and provide a more efficient way of shaping the traffic, we decided to limit the bandwidth using the TCP receiving window. In fact, the main advantage of this technique is that the server can be notified to reduce the rate fast enough. When the traffic shaping is activated at the end device, the FlexSteam starts intercepting the outgoing packets (sent by the video player to the server) and modifying the TCP receiving window according to the desired limit. In addition to the fast response by the server, no traffic loss can be incurred by this technique.

In order to use the TCP receiving window to limit the traffic, the round trip time

between the client device and the server should be computed before sending the next request. Therefore, we add a function running in the background which periodically inspects the RTT value. Although we can modify the OVS to calculate the RTT for the actual video packets, we simply choose to send ICMP packets to periodically ping the server prior to sending the chunk request. This decision of using the ICMP packets stems from the fact that the adaptive video players have idle periods followed by active periods. As a result, when the player switches ON after a long idle period to fill its video buffer, the last computed RTT value of the last received packets might not reflect the actual value. In our implementation, we use the following formula for calculating the TCP receiving window assuming that the wireless channel is perfect and does not incur any packet loss:

$$Rwnd = (RTT * Ratelimit)/scalingfactor \tag{1}$$

The TCP window scaling factor can be easily determined from the three way handshake packets exchanged between the client and the server. Figure 20 shows a real experiment that is conducted to show the performance of rate limiting using the TCP flow control mechanism. To make the experiment as realistic as possible, we set the player to stream a real Dash video from a server on the internet via a large U.S. cellular carrier. Starting out, the player streams at 2500 Kbps, and after 120 seconds we set the TCP receiving window to limit the traffic to 2000 Kbps for 60 seconds before reducing it to 1300 Kbps. The calculated RTT value during the experiment was around 100 ms, thus to limit the rate to 2000 Kbps and 1300 Kbps, the TCP window (with disabling the scaling factor) is set to 26000 and 16000 bytes respectively. Even with high fluctuation in the RTT values ($\pm$20 seconds), it is clear from the figure that the achievable throughput by the player is reasonably stable and too close to the desired rate. Moreover, the player is successfully forced to reduce the requested bitrate when limiting the rate to 1300 kbps. Note that, we implement our system such that the TCP window is updated whenever there is a significant change in the RTT value.

5.4 EXTENDING SDN PLANES

As FlexStream is designed to shape the traffic by modifying the TCP receiving window, we are required to extend the SDN data plane to rewrite the TCP receiving window field in the TCP header for the outgoing packets sent from the player to

the video server. Therefore, to force the traffic to go through the OVS, we bind the OVS to the local network stack and add the physical network interface (e.g, WiFi or 3G/LTE) as one of OVS ports. In FlexStream, the OVS works by forwarding packets between two ports: OVS internal port and the physical interface which now functions as a port within the OVS. When a packet arrives at the OVS kernel datapath, which contains part of the flow table, coming from the wireless interface, it checks whether the packet matches any flow entries. If the match found, then the corresponding action is executed. Otherwise, (in case of a new flow) the packet is forwarded to the ovs-vswitchd, userspace daemon that implements the switch, to learn the about the desired action. These actions may specify forwarding, dropping, modifying, or sampling the packet. Packet modification is in our interest to enable the header modification.

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 here is that the action field for "*set*" instruction in the flow table entry comes with a limited size, and adding a new action to this field requires a change in the size of the data structure to not only the flow table in the datapath, but the modification needs to propagate to the user space and OpenFlow protocol. Whenever a packet matches with a flow entry, the *execute_set_action* function, which responsible for modifying packet header fields in the datapath, is invoked. This function in turn calls and passes socket buffer "*sk_buff*" and the new window value, to be written, to our new added function "*set_tcp_window*", which perform the TCP modification field after makes the TCP header in the *sk_buff* writable. We also modified the SDN controller on a mobile device to support this new action. As the OpenFlow protocol is used by the controller to speak to the OVS user space, the OpenFlow protocol has also been extended to support the TCP window modification action as well.

5.5 ESTIMATING VIDEO BITRATES FOR MANAGING ENCRYPTED STREAMING SESSIONS

The functionality of FlexStream is already extended to also manage encrypted streaming sessions. This becomes an essential feature since several video providers have already adopted encryption for video delivery. However, managing encrypted sessions is extremely challenging since the video meta data can no longer be obtained by intercepting the manifest file which is typically sent by the server to the player

at the beginning of the streaming session. Moreover, the HTTP requests sent by the player to the server is also encrypted. This makes detecting the bitrate of the newly requested segments by checking the HTTP get request header is impossible. One solution to overcome this challenge is to assign an amount of bandwidth to the encrypted session equal to those sessions which have similar priority, screen size, and other contextual factors. However, even though this solution looks fair enough to be adopted, it can lead to an extreme case of bandwidth underutilization. For instance, suppose that the fair share of bandwidth assigned to an encrypted session by the system is 2 Mbps. If the maximum bitrate profiles of this streaming video is much less than this value (2 Mbps), say 0.5 Mbps, then we end up wasting about 1.5 Mbps for just one session. If we have more similar situations, then the bandwidth underutilization will be considerably increased. However, this waste in the bandwidth does start until the player transits from the buffering state to the steady state, in which the player starts generating ON/OFF patterns.

Fortunately, during the buffering state, assigning high throughput compared to the video bitrate can rather reduce the startup latency and make the player converge to the target (highest possible) bitrate much faster, without causing any waste in the bandwidth. Consequently, we only need now to eliminate the bandwidth waste during the steady state through adjusting the traffic shaping value and then reassigning the resulting (extra) bandwidth to other sessions by re-invoking the optimization module. To this end, we need to estimate the current video bitrate as soon as the player switches to the steady state. We utilize the OVS on the mobile device to calculate the length of the ON and OFF periods as well as the segment size downloaded during the ON period.

For determining the total size of the video segment downloaded by the player, we leverage the byte counting feature of the flow table which is inherently supported by the OVS datapath. The total number of received bytes of all received chunks then can be retrieved through the SDN controller using any query command such as “dump-flows”. By comparing the number of bytes before and after receiving a video segment, we can accurately obtain the segment size. The OVS datapath also supports several flow related timers such as “idle_age”, which gives the amount of time that has passed without observing any packets of the video flow passing the flow table. We wrote a script using C language which developed to interact with the SDN controller to utilize some of these timers and find out the length of ON and OFF

periods. By knowing the segment size downloaded during the ON period and also the length of this ON period in addition to the length of subsequent OFF period, the video bitrate can be determine by dividing the segment size by the total time of these two consecutive periods. FlexStream then sets the throughput rate of the video session slightly above the video bitrate in order to be sustained as we describe in the next chapter.

5.6 HEURISTIC BANDWIDTH MANAGEMENT ALGORITHM

In this section, we begin with introducing a heuristic algorithm for bandwidth allocation which can give a solution to the problem close to the optimal solution within a reasonable time complexity. However, we dedicate the following chapter to formulate an optimization problem and introduce an optimal solution to this bandwidth allocation problem.

Our system is triggered by two events: when the total video bitrate of all players reaches a predefined threshold, or when one of the players unfairly gets a lower quality (e.g., due to bad wireless link). Algorithm 1 shows the pseudo code for bandwidth allocation. The algorithm takes as inputs C: the total bottleneck capacity, C_{thresh}: the reached bottleneck capacity that triggers bandwidth control, N: the number of active players streaming a video, $\{b_1,...,b_K\}$: bitrate profiles (we use the same profiles across devices for simplicity), $\{z_1,...,z_L\}$: device screen resolutions, and $\{c_1,...,c_K\}$: the current bitrate requested by all players.

We consider that the end devices come with different screen sizes and each device does not request bitrate with a resolution higher than its screen resolution. Moreover, to enable priority assignment, devices are classified into two main classes, high and regular class. We also assume that the number of devices with high priority M is much smaller than the number of regular users $N - M$. We set the threshold C_{thresh} to be 75% of C, where we empirically observe that players usually start competing for bandwidth which can lead to instability.

Therefore, if the total requested bitrate does not exceed this value, algorithm returns without initiating the bandwidth control (L2); otherwise, it initially assigns the highest quality profile to each device, whether it has a high or a regular priority (L3). Then, it gradually reduces the bitrate profiles one level at a time for regular users only, starting with the smallest screen size among those devices getting the highest bitrates, until the network capacity can accommodate the total bitrate profiles of all

Algorithm 1 : Bandwidth Allocation

INPUT: C, C_{thresh}, N, M, $\{b_1,...,b_K\}$, $\{z_1,...,z_L\}$, $\{c_1,...,c_K\}$
OUTPUT: r_i - rate limits

1: Calculate $b_{total} = \sum_{n=1}^{N} c_n$
2: **if** $b_{total} < C_{thresh}$ & $cb_n == b_K, \forall n = 1, ..., N$ **then return**
3: Set target bitrate $t_n = b_K, \forall n = 1, .., N$.
4: Calculate the new bandwidth capacity $C_{New} = C - M \times b_K$.
5: For the $(N - M)$ regular users only do the following:
6: **while** $\sum_{i=1}^{N-M} (t_i) > C_{New}$ **do**
7: Find the session j with the smallest screen size from those which have the highest bitrates.
8: Reduce t_j one bitrate level.
9: Calculate $r_i = C \,/\, \sum_{n=1}^{N}(t_n) * t_i, \forall i = 1, .., N$.

sessions (L6-8). Note that the devices with larger screens are allowed higher bitrates only when demand exceeds C_{thresh}, i.e., competition starts and does not allow all devices to stream at the same bitrate. Further, the algorithm ensures that the maximum difference in bitrates across regular devices can not exceed one level regardless of the size of the screen. Finally, once total demand is within capacity, the rest of the bandwidth is distributed based on the assigned bitrate (L9). Once r_i is computed by GC, it is sent to DAs on each mobile device. The GC may release the control over the bandwidth or reallocate the bandwidth as some players join or finish streaming.

We set the threshold to be 75% of the total network capacity C. Above this value, we have observed, through several experiments, that players may start competing for bandwidth which can cause instability in video quality for some players. If the traffic load reaches the threshold, the algorithm initially assigns the highest quality profile to each session, and then it starts reducing the bitrate profiles one level for regular users only, one at a time, starting devices with lower screen sizes till the network capacity can accommodate the total bitrate profiles of all sessions. Once r_j is computed by GC, it will be sent to user agents on mobile devices for bandwidth enforcement. The GC may release the control over the bandwidth or reallocate the bandwidth as some players finish or start streaming.

5.7 SUMMARY

This chapter stated with explaining the session work flow of FlexStream. We explained how the data move from one components to another to achieve the target tasks. Then we turn into describing some implementation details and some of the challenges that we encounter during the development of FlexStream such as binding OVS with LTE interface and extending OVS function for rate limiting. In addition, we described our methodology for traffic shaping which uses TCP flow control mechanism. Finally, we introduced a heuristic algorithm for network resources allocation to manage video sessions.

CHAPTER 6

FLEXSTREAM OPTIMIZATION MODULE

6.1 INTRODUCTION

As we mentioned before, our system Taking control over network resources by GC is generally triggered by two events: when the total video bitrate of all players reaches a predefined threshold or one of the players unfairly gets lower quality (e.g., due to bad wireless link). Once the control is activated, the *optimization module* is invoked to manage network resources. The main objective of this module is 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 a 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.

6.2 PROBLEM FORMULATION

Let B be the total link capacity that is shared by N active video sessions. 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, and we define it as:

$$u_{ij} = \prod_{l=1}^{a} \beta_{il}.\log(r_{ij}) \tag{2}$$

Our choice of the logarithmic utility function comes primarily from its properties of diminishing returns as the bitrate increases. This property ensures a proportional share of network resources among all users. Note that the log function is widely used as a utility function for rate control in wireless networks [42, 64]. To account for

context factors in the utility, we weight the bitrate with a set of positive parameters corresponding for the considered context factors. For example, if we want to consider for 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_{i1} is used to assign different priorities to different users while 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). Similarly, as the surrounding luminance has a considerable impact on the perceived image quality [78], β_{i3} can be set to have larger values for those devices located in the dark environment (e.g., indoor) than those in a bright environment (e.g., outdoor). In addition, β_{i4} is used to assign more value to the video flow than the background flow. Once again, the aim of this utility adjustment is to guarantee balanced QoE in the network in addition to enabling quality of service differentiation based on traffic types and different contexts. Given this utility function, the optimization problem is defined as the maximum sum of the utility functions u_{ij} across all video sessions as follows:

$$\max_{x_{ij}} \sum_{i=1}^{N} \sum_{j=1}^{K_i} (u_{ij} - \mu\delta_{ij}) x_{ij} \tag{3}$$

$$\text{subject to} \sum_{i=1}^{N} \sum_{j=1}^{K_i} (\epsilon r_{ij}) x_{ij} \leq B \tag{4}$$

$$\sum_{j=1}^{K_i} x_{ij} = 1,\ x_{ij} \in 0, 1\ \forall i \tag{5}$$

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:

$$\delta_{ij} = \begin{cases} |r_{ij} - r_{ic}| s_i + (m - \lceil \frac{t_i}{k} \rceil), & t < t_{thresh} \\ |r_{ij} - r_{ic}| s_i, & t \geq t_{thresh} \end{cases} \tag{6}$$

This definition of δ_{ij} takes into consideration a number of important factors that impact the stability aspect of video QoE. First, 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. In other words, jumping across several bitrates at

once is not recommended and will result in a larger penalty. This stepwise decrease in the video bitrate, in turn, prevents a large drop in the assigned bandwidth to any video session, hence reducing the possibility of having stalls in practice. Moreover, this penalty increases as the total number of switches s_i of session i increases. We also account for the time period between bitrate switches. We aim to ensure that there is enough time t_{thresh} between any two switches that might be experienced. The term $(m - ti/k)$, where m is the maximum penalty that can be applied while k is a scaling factor, will result in a significant penalty when the time period from the last switch t_i is short. Note that this penalty function requires each DA to maintain a history of the number of switches s_i as well as recording the time of the last bitrate change t_i. In addition, this penalty is not applied to background flows as it is only used for stabilizing the video session.

In the objective function (3), we include a tunable parameter μ to tradeoff between delivering high bitrate and stability. This parameter allows the network operator to customize or balance the objective between maximizing the bitrate delivery 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} is equal 1 when bitrate version j is selected, and 0 otherwise. The inequality (4) indicates that the optimization formula in (3) is restricted by the total available bandwidth at the AP. In addition, we use a constant ϵ in (4) 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. Although different players may use different margins, we notice that most of commercial players converge to the target version when the available bandwidth reaches 35% above the video bitrate. Therefore, FlexStream initially sets ϵ=1.35, and then this value can be reduced once the target bitrate is reached, in order to improve resource utilization.

To ensure high resource utilization and to maximize the QoE in the network, we take into consideration the maximum achievable TCP throughput for each client. According to [57], this value is subject to the following factors: Round Trip Time RTT,

Maximum window size $rwin_{max}$, probability of packet loss p, the number acknowledged packets b, and average retransmission timeout T_0 and can be approximated by the following model:

$$G_i(p) \approx min\Bigg(\frac{rwin_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 min(1, 3\sqrt{\frac{3bps}{8}})p(1+32p^2)}\Bigg) \tag{7}$$

Where G_i denotes the maximum achievable throughput for client i. Note that this maximum throughput is limited by the first term when probability of packet loss p is low. Therefore, G_i should be calculated by all DAs and reported to the GC before the optimization module is invoked. Then, to avoid assigning infeasible video bitrate j to session i, due to limited throughput, the following constraint is added to each session i in the optimization problem:

$$\sum_{j=1}^{K_i} r_{ij}x_{ij} \leq G_i \quad ,\forall\ i \tag{8}$$

6.3 PROBLEM SOLUTION

6.3.1 MAPPING THE OPTIMIZATION INTO MULTIPLE-CHOICE KNAPSACK PROBLEM

The optimization problem (3) can be clearly map-ped to a multiple-choice knapsack problem, in which one item in each class of items must be selected with the objective of maximizing the profit without exceeding the knapsack capacity. Each video profile in our problem corresponds to a class of items while each bitrate that belongs to the video profile 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.

6.3.2 DYNAMIC PROGRAMMING ALGORITHM

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 in which we use to discretize the total link

capacity B into a series of Z incremental capacity 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:

$$y(i,z) = \max_{1 \le j \le K_i} \{h_{ij} | zs \ge w_{i,j}\}, \quad \forall\ 0 \le z \le Z = B/s \tag{9}$$

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 (3) and (4), respectively. Note that the constraint in (8) excludes any infeasible video bitrate. Then, we can solve the problem via DP in a bottom-up fashion using the following recurrence:

$$Y(i,z) = \begin{cases} y(i,z), & i = 1, \forall z \\ \max\limits_{0 \le a \le z} \{Y(i-1,a) + y(i,z-a)\}, & 2 \le i, \forall z \end{cases} \tag{10}$$

where $Y(i, z)$ is the total maximum utility that can be obtained for all i video sessions when the available bandwidth is zs. Note that 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 lead to the optimal solution. Algorithm 2 lists the summary of the dynamic program used by GC for the network resource allocation to video sessions.

6.3.3 ALGORITHM COMPLEXITY AND OVERHEAD

Given number of possible bitrates for a video session is quite limited in practice (e.g., $|K_i| \le 10$), and careful implementation of the dynamic programming steps, the complexity of this solution will be $O(NZ)$. Given that the running time depends on N and $|z|$, its overhead is within sub-second level for the typical practical large values. For instance, when $N = 400$, $K = 8$, $B = 200$ Mbps, and $s = 100$ Kbps, the execution time on a single-core Intel 2.20 GHz processor is about 400 ms. As we pointed out before, this small time overhead would allow the GC to react in time before the QoE can be affected. However, if the execution time occasionally becomes larger and above certain threshold, then FlexStream can speed up the execution time

Algorithm 2 : Algorithm for solving the optimization problem using Dynamic Programming

Variables: Number of video sessions: N, Video session index: i, Bitrate index: j
INPUT: Video bitrate profiles r_{ij}, Utility y_{ij}, Bandwidth steps: z, Safety margin: ϵ, Total system capacity:B
OUTPUT: Selected bitrate profile for each video session $x_{ij}, x_{ij} \in [0, 1]$
Repeat: Each time there is a video session starts or finishes or there is a considerable change in the network condition

for i from 1 to N **do**
    for j from 1 to K **do**
        $w_{ij} = \epsilon \times r_{ij}$
sort $w_{i1} \leq w_{i2} \leq ... \leq w_{i,K_i}$
if $\sum_{i=1}^{N} w_{i1} \geq B$ **then**
    assign the lowest bitrate profile to each video session and exit.
for i from 1 to N **do**
    for j from 0 to B, j=j+z **do**
        $Y(i, j) = 0$
for i from 1 to N **do**
    for j from 0 to B, j=j+z **do**
        for j from 1 to K **do**
            $Y(i, z) = max(Y(i - 1, z - w_{i,j}) + y_{i,j}, Y(i, z))$
ourput the solution that produces Y(N,Z)

by reducing the granularity of z by increasing the bandwidth step size s without empirically sacrificing the optimality of the solution, or by utilizing one of the well-known approximate algorithms [59, 54] that guarantees faster execution to be within this sub-second level at the cost of deviating from the optimal solution.

6.4 CONCLUSION

This chapter introduced an optimization problem that optimizes the network resources allocation to different video sessions. The aim is to enable the players to stream the highest possible set of video bitrates while maintaining a fair share of the network resources with minimum quality variations and stalls. In addition, we take into consideration several context factors in the formulation of the optimization problem including user priority, device capability, surrounding luminance, link condition and traffic types to ensure high, well-balanced, and fair video QoE. The optimization problem is defined as the maximum sum of the utility functions (log

functions) across all video sessions. We mapped the problem into a multiple-choice knapsack problem in which we select one video bitrate from each video session profile that maximizes the total streamed bitrates within the system capacity, then we used dynamic programming to solve this optimization problem.

CHAPTER 7

EVALUATION

In this section, we evaluate FlexStream through real implementation on mobile devices in WiFi and cellular networks, in addition to emulated environment for larger scale experiments. Since our implementation does not involve any modification to the network infrastructure, the testbed described below is the same for both networks (WiFi and cellular).

7.1 EVALUATION METRICS

Several objective tests have been conducted to evaluate the performance of the proposed techniques in the literature using several QoE metrics. We select five of these evaluation metric that are most important for the users QoE. The following are the description of these metrics used in our evaluation:

1. Instability: Previous studies have shown that the instability in the video bitrate during the playback has significant impact on the user's watching experience. We simply measure the instability as the number of switches that occurs throughout the video playback duration. Intuitively, the lower the number of switches, the better viewing experience the user can get. Therefore, the goal of our system is to minimize the number quality switches.

2. Inefficiency: One of the ultimate goals of FlexStream is to make streaming videos more efficient by maximizing the average video bitrate taking into consideration the available bandwidth for video sessions in the network. We measure the quality by calculating the average bitrate of all video chunks requested by the video player.

3. Playback Fluency: It is defined as the number of playback freezes and the total duration of all these freezes during the video streaming time. The aim here for FlexStream is to minimize the number of stalls and the duration of each stall if it could not completely avoid them.

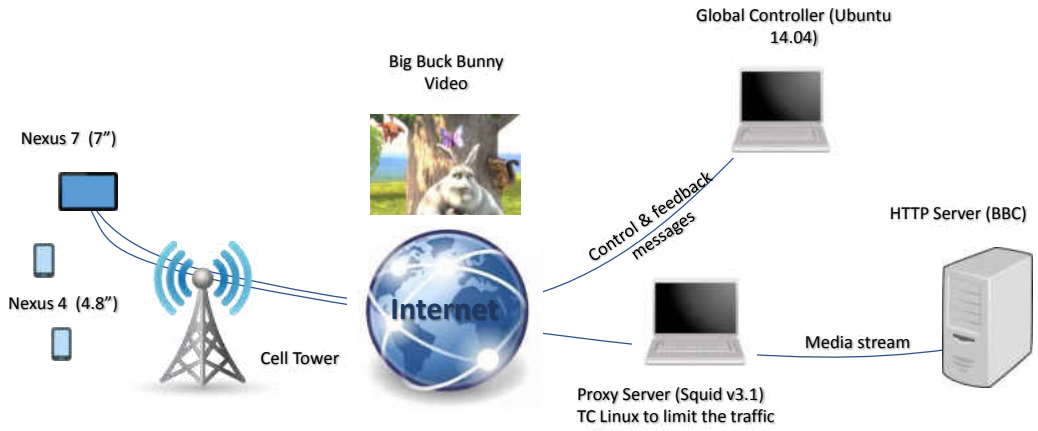

Figure 21: Testbed used in conducting the experiments.

4. Startup Latency: The startup latency is defined as the time between the first segment request sent by the player and when the playback starts. At this stage, we defer computing this startup latency to future work.

5. Unfairness: To measure the unfairness, we use the difference in the video bitrate between the players, but here we also consider the screen size of different devices. In the extended experiments, we use Jain's Fairness Index [39] of the request video bitrates over all players. A higher value of this metric implies a more fair QoE among users.

7.2 EXPERIMENTAL SETUP AND DESIGN

In our evaluation, we use two experiment setups, one with real players and the other one with emulated players and server. We start by conducting relatively small-scale, but real-world experiments using several mobile devices, to validate, analyze and understand FlexStream behavior under different and realistic scenarios. We install and run the GC on a laptop running Ubuntu 14.04 connected to the Internet via a public IP address. We use three Android mobile devices with different screen sizes (Nexus 7 and two Nexus 4) sharing the bandwidth over the same wireless access point. To create a bottleneck and make the devices compete for bandwidth in the way that works for both WiFi and cellular scenarios, we place a Squid proxy

Encoded Frame size	Frame Rate	Approx Bitrate	Representation ID
512x288	25	449Kbps	512x288p25
704x396	25	843Kbps	704x396p25
896x504	25	1416Kbps	896x504o25
1280x720	25	2656Kbps	1280x720p25

Table 1: Encoding settings of video segments used in the experiments.

server (version 3.1) running on Ubuntu 12.04 between the end devices and the media server. The proxy controls the bandwidth using Linux Traffic Control (TC). All video segments requested by video players on the end devices are forwarded through the proxy server. On the end devices, four main components are installed: FlexStream, SDN-Controller, OVS, and GPAC video player[1]. FlexStream is implemented in Java and communicates with GC over TCP. The SDN controller uses OpenFlow v1.2 on the southbound to control and configure the OVS. We install our modified version of OVS v1.11.0 after cross compiling it for our Android devices, and then we bind it to the wireless interface (wlan0/rmnet0) corresponding to the target network (WiFi/cellular). This binding allows the traffic to pass through the OVS and perform the TCP window adjustment. The Android version of GPAC player v0.6.2-DEV (Osmo4) is the adaptive player that is used to stream videos on all end devices. To make the experiment more realistic, a well-known Big Buck Bunny[2] video is streamed from a large Internet portal. The video, as shown in Table 1, comes with several bitrate profiles ranging from 449 Kbps to 2656 Kbps and lasts for about nine minutes.

To evaluate the actual performance of FlexStream in the presence of a sufficiently larger number of concurrent competing video flows, we develop an emulator of a real video player. Similar to the real player, the emulator, implemented as a Java application, generates real traffic by sending HTTP requests to a test server, which responds with dummy video segments equivalent in the size and distribution to those used in the real experiment. We implement relevant adaptation algorithms and operations in the emulator, including scheduling segment downloads and tracking the buffer occupancy, with the exception of actually decoding and playing video

[1]https://gpac.wp.imt.fr/player/
[2]https://peach.blender.org/

segments. As with the real experiments, all emulated players connect to the server through a WiFi AP running on a Linux machine which in turn forwards the requests to the server over the Internet. In all experiments, 12 emulated players, representing 8 phones and 4 tablets are used to stream over five different AP capacities {7, 10, 13, 16, and 19 Mbps}, and set to randomly start during the first two minutes. For each capacity, we repeat the experiment 10 times with and without FlexStream focusing on five different QoE metrics: bitrate, stability, fairness, playback stalls, and startup delay. Finally, we examine the impact of both background traffic and wireless link conditions on the video QoE. We believe that 12 concurrent players reflect well a highly loaded home network, a moderately loaded public WiFi scenario, or a lightly to moderately loaded LTE radio cell with one video service under control and management of FlexStream.

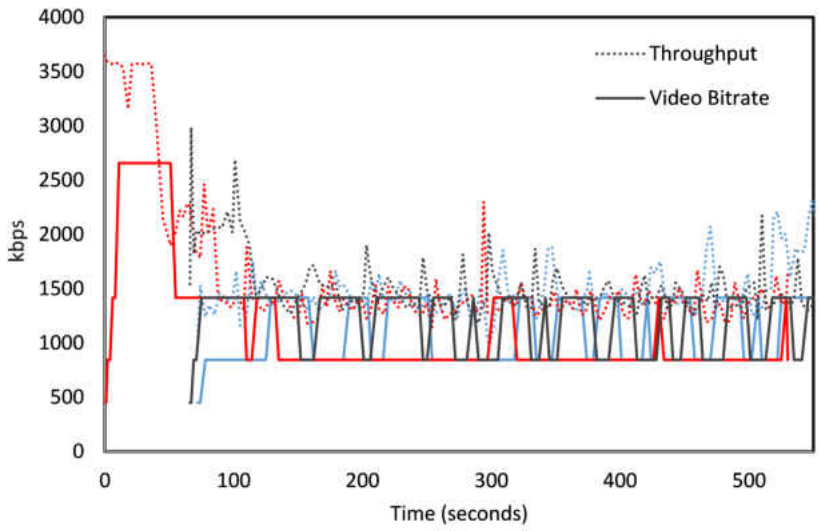

Figure 22: Throughput and requested video bitrates of competing streams.

In these two experiment setups, the evaluation is split up into multiple experiments using both static and dynamic network capacities. The first experiment is set to inspect the impact of FlexStream on stabilizing and improving the video QoE. We then look at the ability of our system in providing quality-based fairness in the network. This is to assign more bandwidth to the devices based on the screen sizes which would balance the observed video quality among all users in the network. In another experiment, we show how FlexStream can differentiate between different user classes where some users can have better video quality when the network gets

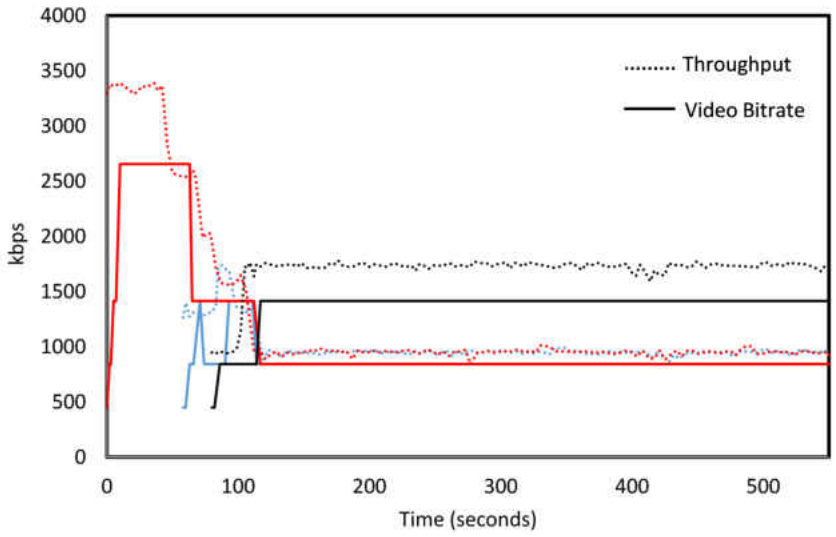

Figure 23: Throughputs and requested video bitrates of competing streams when FlexSteam is activated.

overloaded. Furthermore, we examine the impact of background traffic on the video QoE and evaluate the performance of FlexStream when one or more devices start downloading a large file (e.g., apps updates) while streaming videos.

7.3 PERFORMANCE UNDER STATIC BANDWIDTH

Instability and Inefficiency: It is well-known that adaptive video players suffer from instability in the observed video quality which negatively impacts the QoE. This issue frequently appears when the network gets overloaded and the players start competing for bandwidth. Figure 22 demonstrates the impact of players competition on throughput and videos bitrates. In the beginning, there is only one player streaming a video at the highest bitrate (2650 Kbps) since the network capacity is set to allow up to 3800 Kbps. However, after 60 seconds two other players join and start competing over the bandwidth causing a significant drop in download speed of the first player. In theory, players are expected to equally share the bandwidth as a result of using TCP, but due to the intermittent traffic of adaptive streaming and selfish behavior of the adaptive algorithms of the players, we can notice large oscillations in the download speed for all players causing them to frequently switch between 1416 Kbps and 843 Kbps video bitrates.

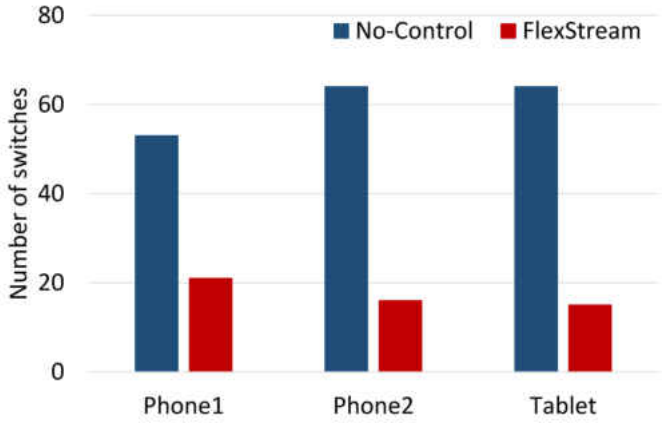

Figure 24: Comparison of the number of quality switches per device with and without FlexStream.

FlexStream, on the other hand, is designed to prevent such fluctuations and provide better stability. Figure 23 shows the result of running the same experiment using FlexStream. In this experiment, as soon as the second player starts at 60 seconds, the FlexStream activates the control of bandwidth as the capacity is oversubscribed with the highest bitrates. Therefore, players adapt to 1416 Kbps bitrate profile as FlexStream splits the bandwidth equally between the players, but after 30 seconds the FlexStream has to reallocate the bandwidth for the second time as the third player joins.

Note that FlexStream is designed to distribute the bandwidth in such a way that maximizes the requested bitrates. Hence, FlexStream assigns about 1850 Kbps for one device, which is enough for streaming 1416 Kbps bitrate profile, while assigning about 975 Kbps to other two devices which are capable of requesting 843 kpbs bitrate profile. These are the maximum bitrates that the players can request without affecting the stability. After 120 seconds, all players can adapt to new rate limits and switch to the target bitrates and maintain them throughout the video sessions without any switches. FlexStream reduces the total number of quality switches for all players from 66 in the first experiment to only 15 switches while maintaining approximately the same overall quality (about 1125 Kbps on average in both experiments). Note that GPAC is designed to initially request the lowest bitrate profiles to reduce the startup latency and then gradually increases the bitrate as bandwidth allows. Therefore, the 15 switches that were recorded with FlexStream are mostly

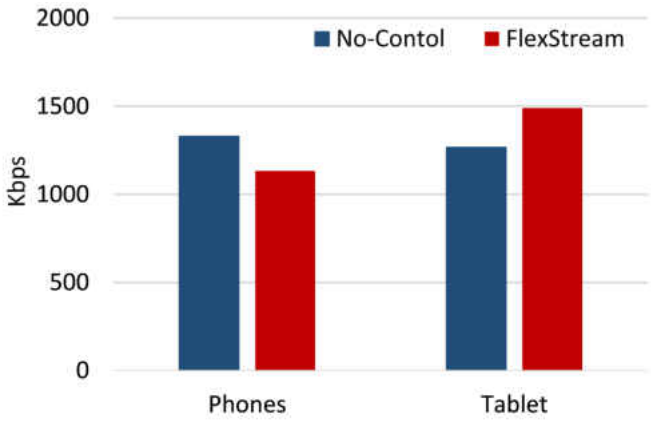

Figure 25: Comparison of the Average bitrate requested by each player with and without FlexStreams.

occurring upon startup, not because of fluctuation in throughput.

To better evaluate the effect of using FlexStream on the stability compared with the non-controlled scenario, we repeat the previous two experiments many times using different network capacity starting at 2500 Kbps with an increase of 1500 Kbps each time until no competition between the players is observed, at 8500kpbs. The average number of switches for each device is recorded and showed in Figure 24 for both scenarios. It can be seen from the figure that the average number of switches for non-controlled scenario ranges between 50 and 70 switches which are far more than the number of switches when FlexStream was used, with only between 10 to 20 switches. Therefore, the advantage of using Flexstream in reducing the fluctuation and stabilizing the video quality over the non-controlled case is obvious. However, this instability reduction should not cause a big drop in the video quality. Figure 25 shows that FlexStream could maintain almost the same overall average video bitrate with a clear enhancement of the tablet over the phones while they are still getting high video bitrate (over 1 Mbps).

To evaluate FlexStream over a cellular network, we conduct the experiments on a major U.S. cellular carrier with two mobile devices, tablet, and phone. The OVS is now bound to the cellular interface (rmnet0) instead of WiFi interface. We set the devices to forward the traffic toward our proxy and avoid the mobile network proxy to have better control over the experiments, and also to be able to limit the capacity at the desired level, 2600 Kpbs. The phone starts first, and the tablet joins at time

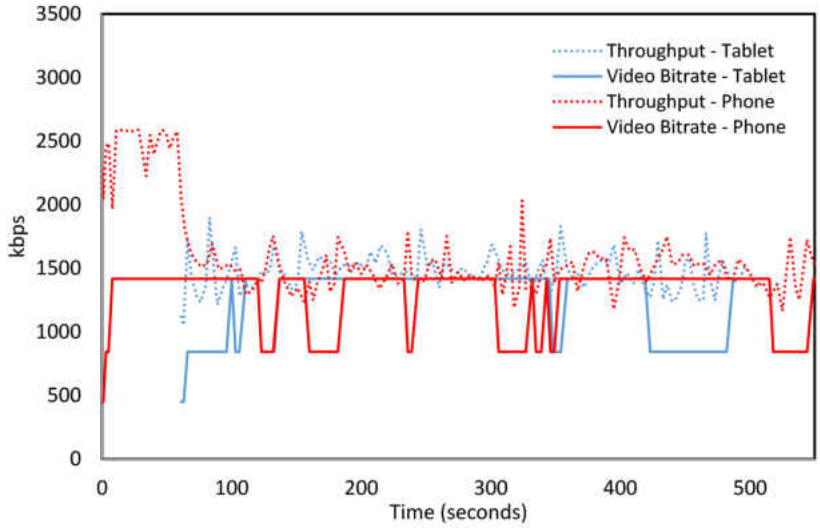

Figure 26: Throughput and bitrates of competing players over Cellular network without controlling the bandwidth.

60. Figure 26 shows the result of the uncontrolled experiment when players compete. We observe expected bitrate fluctuation. Figure 27 shows that FlexStream works again by stabilizing players. In this case, to maximize overall QoE, screen size-based allocation is needed. Therefore, FlexStream works in a similar manner on both WiFi and cellular networks.

This fluctuation in the throughput results in frequent bitrate switches between 843 Kbps and 1416 Kbps for both players which negatively impacting the stability and the overall QoE in the network. On the other hand, Figure 27 shows that the performance of the player is considerably improved with FlexStream regarding stability and fairness. When the flow of the tablet shows up, FlexStream intervenes and manages the bandwidth by assigning more bandwidth to the tablet than the phone. Therefore, the player on the tablet can stream 1416 Kbps while the player on the phone gets the next bitrate profile, 843 Kbps. From these two experiments, we can conclude that the results for both WiFi and Cellular networks are too similar. This makes us refrain from repeating other experiments that we have been conducted on WiFi and consider these two experiments are sufficient to show the visibility of our system on the cellular network.

In the previous experiments, we used real-world experiments but at small scale.

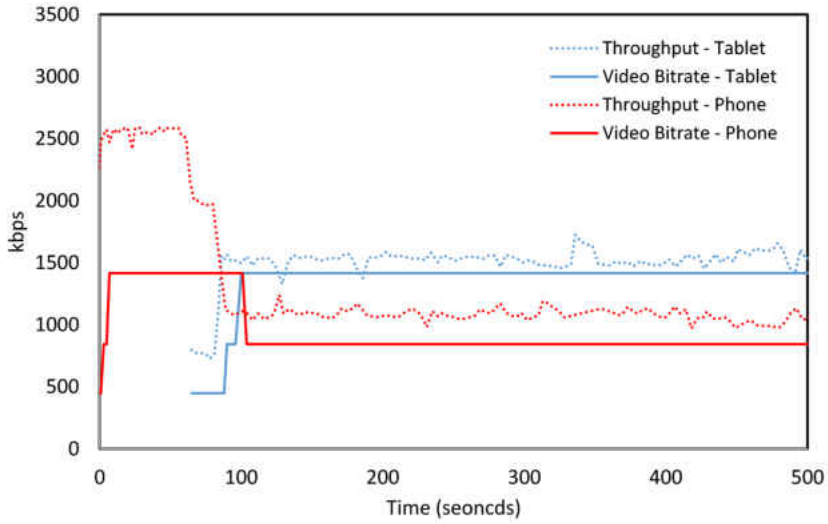

Figure 27: Performance of FlexStream with players compete over the cellular network.

Now to evaluate the stability of our system with larger scale, we use extended experiments with emulated players as we explained in the previous section. Figure 28 compares stability of players with and without FlexStream for each AP bottleneck capacity. Table 2 summarizes 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. FlexStream substantially improves stability in each case, with the average number of switches reduced by 81% from 19.1 to 3.6 (Table 2). Again, the switches with FlexStream occur only at the beginning of the streaming sessions while players ramp up and briefly while the GC redistributes bandwidth as new players join.

Playback Fluency: Player competition for bandwidth not only causes instability and degradation in the video bitrate, but can also lead to playback stalls (playback freezes/rebuffering events) resulting in a severe QoE degradation. This undesired event usually occurs when there is a major drop in the bandwidth, and at the same time, there are not enough packets in the buffer. Therefore, this event can be encountered more with live streaming than on-demand streaming. This is because that when the video player is used to stream live events it typically use much smaller buffer size than if it is used to stream an on-demand video. This small buffer size

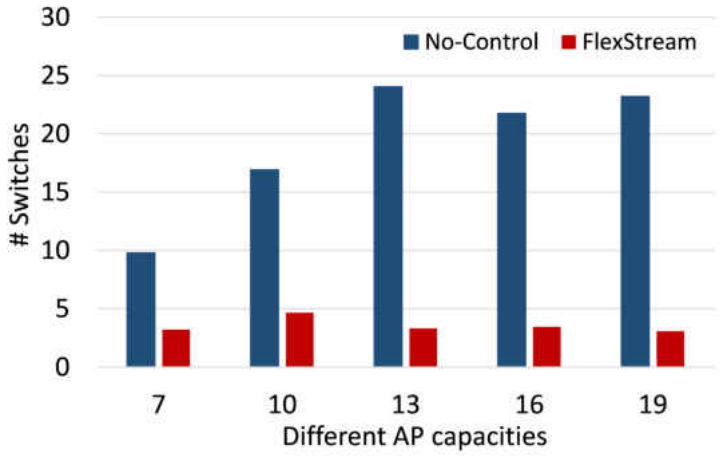

Figure 28: FlexStream significantly reduces switching.

does not allow the player to react in time before the stall happens. With on-demand streaming, the stall can happen when the player overestimates the available bandwidth and starts requesting and streaming high bitrate segments while still no many video segments are already buffered. If the player, in this case, experiences a major drop in the available bandwidth, the stall event is likely to occur. To check how often this event can happen with no control scenario and also to evaluate our system in preventing or minimizing it, we conduct several experiments with different network capacities. Figure 29 shows the average number of stalls per bottleneck capacity. The decreasing trend with higher capacity is expected. The average duration of stalls is also unacceptably high and follows the same trend, as shown in Figure 30. 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).

Startup Latency: Table 2 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). The largest improvement was observed for the lowest capacity and the smallest for the highest capacity (Figure 31). It is also noticeable that startup delay with FlexStream is similar across capacities except for 10 Mbps. Similarity or relative independence of startup delay from capacity can be explained by the avoidance of competition. As soon as the new player notifies the controller, all players get a rapid notification to adjust their bandwidth usage to allow the new

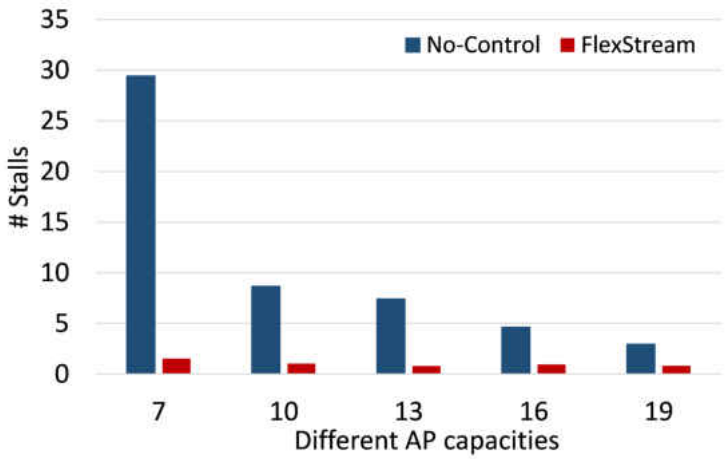

Figure 29: FlexStream reduces number of stalls.

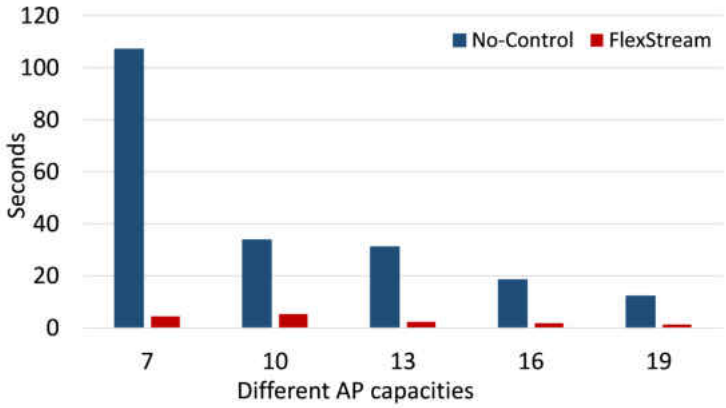

Figure 30: Comparison of average stall duration.

player to start up quickly.

To understand why we have such a large delay at the beginning of uncontrolled streaming sessions, we record the startup delay time for all players for the experiment with 19 Mbps capacity and present the players in the order of their starting time in Figure 32. As can be observed, only players which started afterward have experienced a large delay, starting with the eighth player and dramatically increased for the rest. One way to explain this is that when later players join, they find most of the bandwidth is already utilized and dominated by earlier players causing them to struggle to get enough bandwidth to stream the first video segments. This situation not only leaves these players subject to long stalls at the beginning of the streaming

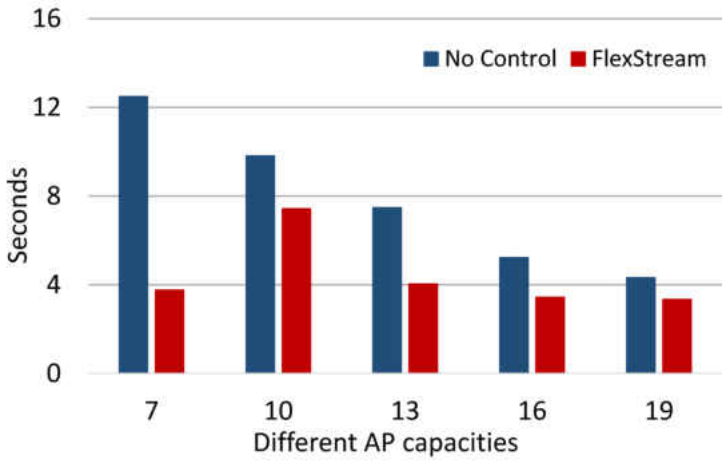

Figure 31: Comparison of average startup delay times.

Table 2: 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

session, but also can lead to extremely unfair scenario where the first set of players can always receive higher bitrates.

Unfairness: We examine bitrate unfairness among uncontrolled players in Figure 33, where the average bitrates of the first six players to start is about 600 Kbps higher than the average of the last six players. Specifically, those users with larger screens (tablets) that belong to the latter group are most impacted by the unfair share of bandwidth.

FlexStream further addresses fair allocation of available bandwidth among video players taking into consideration their screen sizes. The *Tablets bitrate difference* entry in Table 2 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 compared to only 20 Kbps in uncontrolled case. Therefore, FlexStream

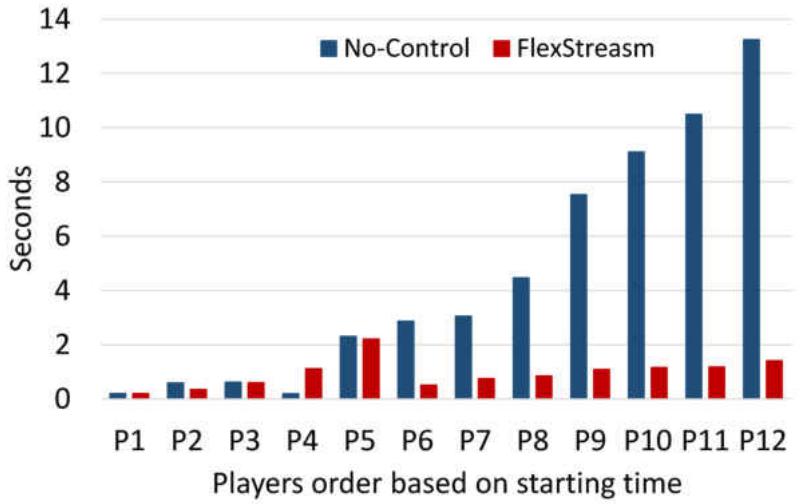

Figure 32: Startup delay times of video players ordered based on their start streaming time.

considerably alleviates this type of unfairness.

Moreover, FlexStream improves fairness among devices of the same screen size (e.g., phones or tablets). Figure 34 compares fairness in the average requested bitrate among phones using Jain's Fairness Index (JFI), where higher values mean better fairness. Across all capacities, FlexStream improves JFI, and significantly so for 7, 10, and 13 Mbps bottleneck. For 16 and 19 Mbps, JFI marginally improves because under the capacity constraints, some devices with the same screen size received a higher share of bandwidth to maximize average bitrate in the system, as per optimization function. The overall JFI across all devices (including phones and tablets) and scenarios are increased by 0.06 as indicated in Table 2.

7.4 PERFORMANCE UNDER DYNAMIC BANDWIDTH

In the previous experiments, we set the network capacity at fixed rates. However, in practice, the network capacity can be highly dynamic. Therefore, to evaluate our system on a more realistic scenario, we shape the traffic at the proxy server based on a real bandwidth trace. We run the experiments with and without activating FlexStream to show its performance on WiFi network. Figure 43 shows the scenario without enabling FlexStream. Note that the dotted line at the top represents the available bitrate which extremely bounces between 14 Mbps 4.3 Mbps. This extreme

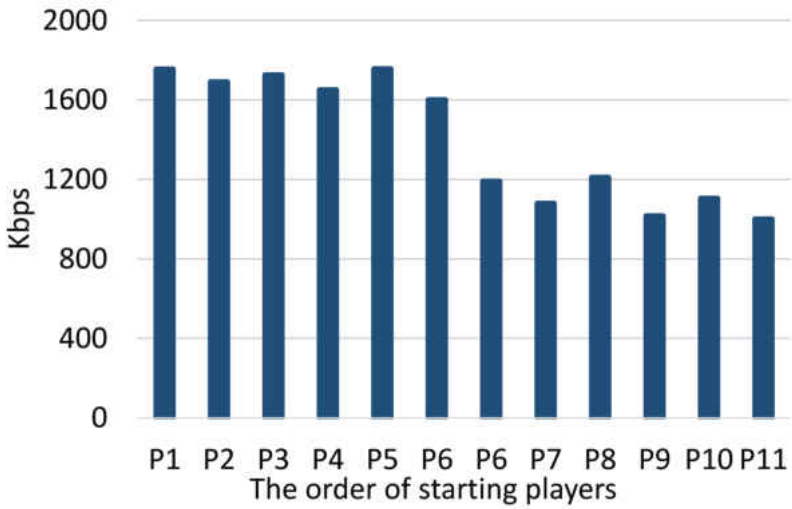

Figure 33: Comparison of average bitrate of players listed in the order of their starting time.

fluctuation in the available bandwidth causes large variations in the download speeds and makes the players compete from time to time till a significant drop occur after about 180 seconds. As a result, the players start aggressively competing over the bandwidth and consequently start switching between 2656 Kbps and 1416 Kbps video bitrates. The total number of switches recorded in this experiment is considerably large reaches to 42 switches for all players. On the other hand, when FlexStream was used, the number of switches is significantly reduced to only 13 switches as shown in Figure 36. When the large drop occurs around second 180, the FlexStream starts controlling and reallocating the bandwidth. Since the Bandwidth is dynamic, the FlexStream uses the throughput reported by video players to estimate the total available bandwidth at that time. Therefore, it assigns approximately 3500 Kbps to the tablet which becomes able to stream 2656 Kbps bitrate profile, while assigning about 2000 Kbps to both phones which could adapt to 1443 Kbps video bitrate.

Now we turn into evaluating FlexStream with an entirely real scenario in which we impose no cape on the bandwidth. In other words, we need to check whether the performance of FlexStream is impacted by uncontrolled background traffic when the network is naturally overloaded without any interference to limit the capacity (using a proxy server). We have done a considerable number of download speed tests over one major cellular network provider in our campus in order to determine whether the base stations (towers) get overloaded and at which times. Our results show

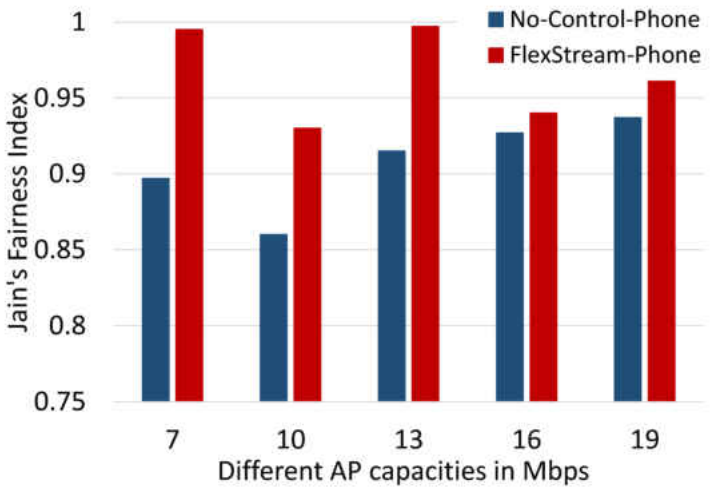

Figure 34: Comparison of average Jain's Fairness Index.

that during specific times during the day (e.g., during the lunch time) the network experiences a considerable amount of traffic, creating several bottlenecks. Figure 37 shows the average number of switching of five experiments conducted during those times. For each listed experiment, we run one experiment with no control immediately followed by one with FlexStream enabled. As can be seen, even with no control over background traffic, FlexStream still provides good stability compared with no control scenario. The results are too similar to the static scenario with a slightly higher number of switches. We believe that as the number of mobile phones controlled by FlexStream, the performance will definitely get improved.

7.5 CONSIDERING VARIOUS CONTEXT INFORMATION

7.5.1 DEVICE CHARACTERISTICS

In fact, FlexStream does not only improves the QoE via stabilizing the video quality, but also improves the fairness and overall QoE in the network through favoring the devices with larger screen sizes over the devices with small screen (e.g., tablet over phones). In the previous experiment depicted in Figure 23, when FlexStream allocates the bandwidth to the players, it assigns the highest bandwidth (1850 Kbps) to the tablet to have better quality than the other two phones. Therefore, the player on the tablet could stream 1416 Kbps bitrate and have good quality comparable to

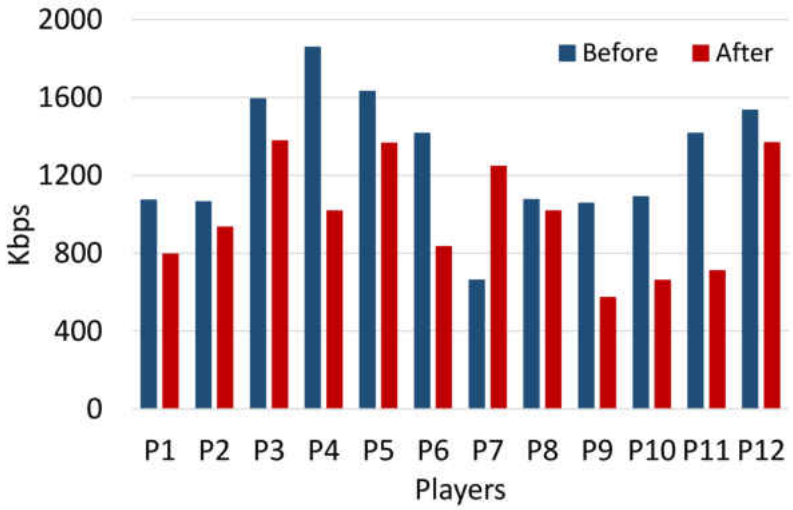

Figure 35: Background flow degrades video bitrates.

the quality the players on phones get, which correspond to 843 Kbps bitrate. Balancing the quality among the clients is a substantial feature of FlexStream as for some scenarios such as home network, it is possible to encounter a situation where video streams generated by phones with only four or five inches screen size competes with another stream generated by a TV with tens of inches screen size.

7.5.2 SERVICE DIFFERENTIATION

FlexStream is also designed to differentiate between two different classes of users, regular and high priority users. Figure 42 show how FlexStream is capable of assigning the bandwidth to the players based on their users priority level. As can be seen from the figure, players on the phone and a tablet with regular priorities started at time 0 and 50 respectively, and both adapt to stream the highest quality (at 2656 Kbps) till a third player on another phone with high priority class joins at time 100. Consequently, as the network capacity can not allow all players to stream the highest bitrate profiles, FlexStream activates the control over the bandwidth and allocates higher bandwidth to the player with higher priority. The amount assigned to the high priority user by FlexStream guarantees to stream the highest bitrate profile, 2656 Kbps. The rest of the bandwidth is then divided between the regular users taking into consideration the screen size. As shown in the figure, even though that the player on the phone with high priority started after the other two players and its device

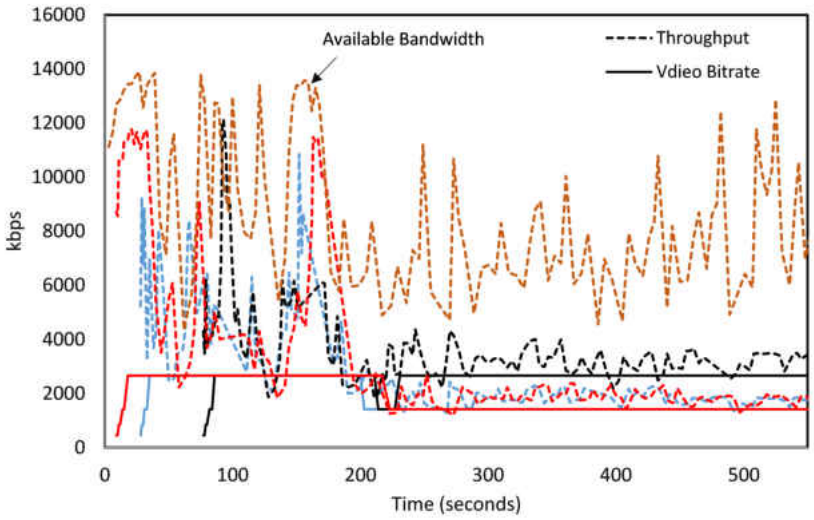

Figure 36: Throughput and bitrates of video streams with real bandwidth trace with using FlexSteam.

has smaller screen size than one of them, it could stream the highest bitrate profile at 2656 Kbps once it joins, while the tablet and the phone with regular privileges could stream at 1416 Kbps and 843 Kbps respectively. Therefore, this experiment not only shows the great advantage of using FlexStream in supporting different user classes, but also shows how FlexSteam improve the stability in addition to providing QoE-Based fairness through considering the devices' screen sizes.

7.5.3 LINK CONDITION

In the previous experiments, we set the devices hosting the players at equal distance from the AP to maintain similar radio signal. In reality, players are at varying distances with different radio signals. To study the impact of the wireless link conditions on the video QoE, we move one player further away to weaken the signal. Figures 38 and 39 show the throughput and video bitrates requested by this player without and with FlexStream, respectively. The drop in the throughput at time 380 when the device is moved is much worse in the uncontrolled case than with FlexStream. In uncontrolled case, the player lowers the bitrate to the lowest level compared to only one level and better stability with FlexStream. The reason for the difference is that the bandwidth share of the player is protected with FlexStream although its throughput is affected by the radio signal and not also by the overall

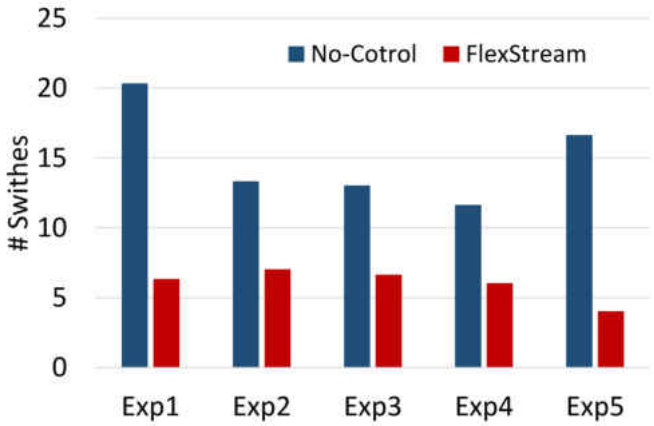

Figure 37: Performance comparison between FlexStream and No-Control scenario over cellular network with no cape on the network capacity.

competition like in the uncontrolled case.

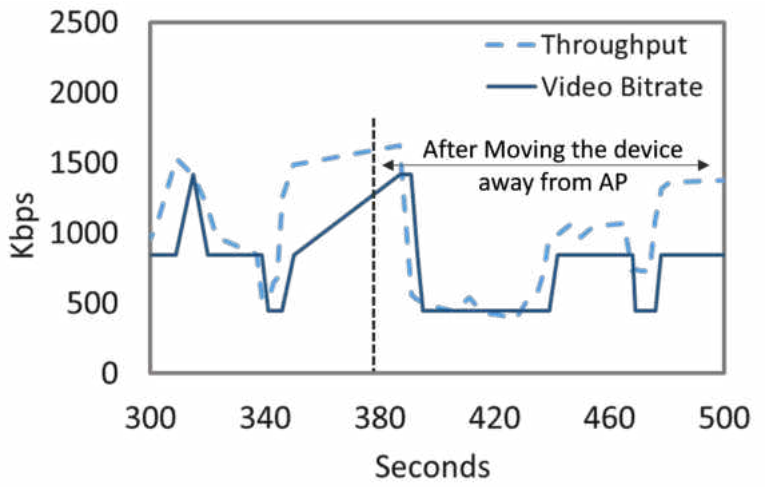

Figure 38: Impact of radio link on QoE (no control).

7.6 IMPACT OF BACKGROUND TRAFFIC ON VIDEO QOE

In practice, the drop and fluctuation in the video quality can be also caused by background traffic. Figure 40 shows the negative impact of the background traffic (e.g., auto apps update) on the throughput and video qualities. We set the capacity of the network to 10 Mbps which is enough for all players to stream the highest quality profile without causing them to compete. After started up all players, we use iperf on one of the devices to initiate a TCP connection with an external device

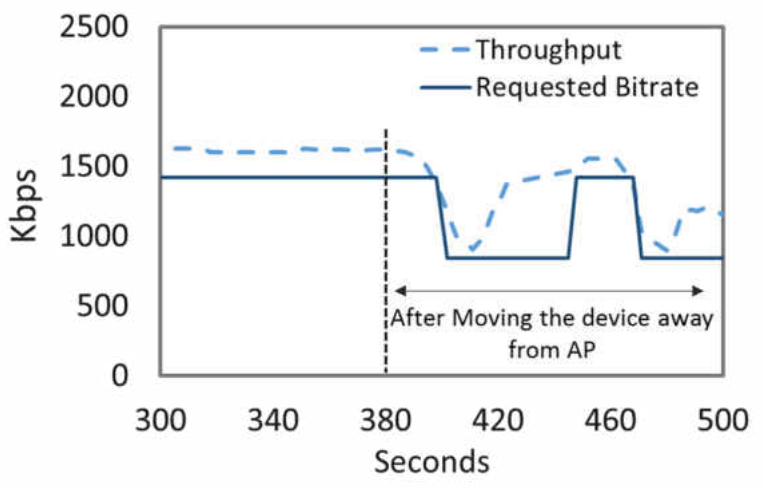

Figure 39: Impact of radio link on QoE (FlexStream).

for two minutes starting at seconds 210. Note that the download speed of this connection is not controlled or capped and only tuned by the TCP congestion control. Thus it is expected that the background traffic can get a quarter of the bandwidth because of the TCP fair share mechanism. Consequently, from the figure, we can notice a significant drop in the players'throughputs once the background traffic starts causing the players to start competing over the bandwidth and perform poorly. This competition between these four TCP connections over the bandwidth makes the players not able to maintain the highest qualities and start bouncing up and down several times between 2656 Kbps and 1416 Kbps bitrate profiles till the background traffic ends at second 330. We repeat the previous experiment but now with using FlexStream, which is set to limit the background traffic to be under 200 Kbps via adjusting the TCP receiving window.

Figure 41 shows the result of this experiment where all players could obviously maintain the same quality since no clear impact of the background traffic on the video streams can be observed throughout the duration of the background traffic. Therefore, having a control on the background traffic at the time of streaming videos is crucial for maintaining good player performance. As part of our system resides on the end devices, one of the advantages of FlexStream is that the application initiating the background traffic can be identified. Therefore, it is possible to set different policies for different applications. For instance, we can set OVS to completely block the background traffic generated by a game app and allow it after the player finishes

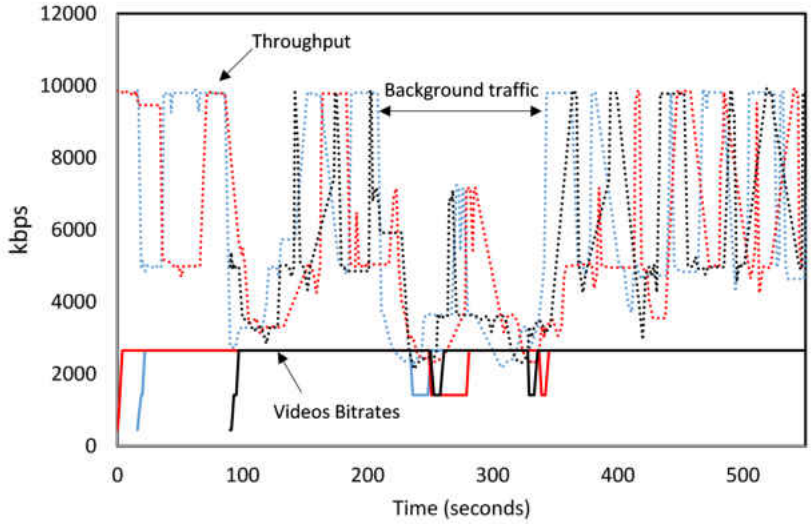

Figure 40: The Throughputs and requested video bitrates of competing video streams in the presence of background traffic which initiated at second 210 and lasts for two minutes.

steaming. On the other hand, apps such as email and bank apps are much more important and should be allowed to send and receive data at reasonable rates. Another factor that plays an important role in setting the rate for the background traffic is the network condition which can be inferred from the observed download speed.

7.7 FLEXSTREAM OVERHEADS

In this section, we evaluate four types of overheads to demonstrate and confirm the practicality of our solution. We consider the CPU overhead of deploying the DA on a mobile device, communication overhead between the DA and GC, and computation overhead resulting from executing the optimization function.

7.7.1 CPU OVERHEAD

As we mentioned before, we have three main components are deployed on a mobile device: DA, SDN Contoller, and SDN. In this section, we evaluate the overhead of running these components on the mobile device. To keep the overhead as low as possible, we design DA to sleep and wake up with the video player keeping some safety margin for variable encoding bitrate of the streamed video which might incur some changes in the length of OFF period. For the evaluation, we use Google

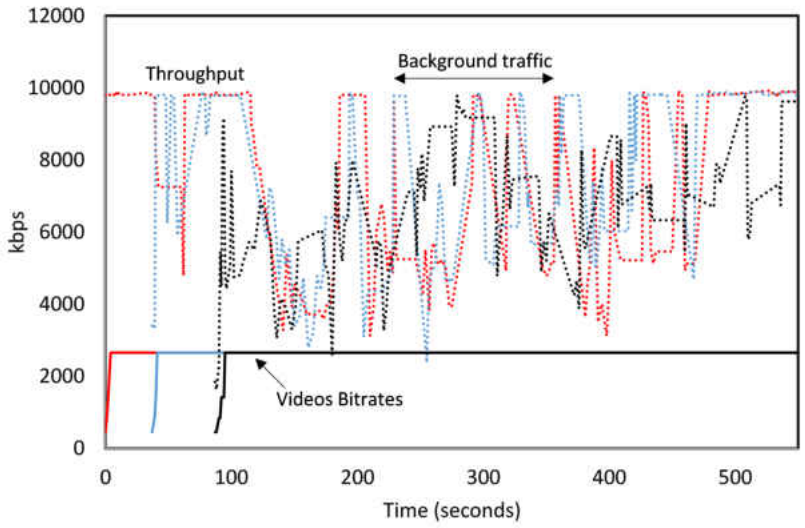

Figure 41: Throughput and requested video bitrate of competing streams in the presence of background traffic when FlexStream is active. The background traffic initiated at second 210 and lasts for two minutes.

Nexus 7 device that comes with 1.2 GHz quad-core processor. We use GPAC on a mobile phone to stream a video encoded at 1.4 Mbps while DA is running in the background. We start by streaming the video with tight bandwidth to trigger the control over bandwidth by GC and trigger the DA to start monitoring the performance, communicating with GC, and enforcing the GC policies through interacting with the SDN controller, which in turn translates these policies into flow rules and actions installed into OVS flow table. The CPU Monitor tool that comes with Android Studio is used in this experiment to monitor the CPU usage overhead caused by our system. As expected, we find that DA incurs a negligible CPU overhead which found to be around 1%. This overhead is on quite an old device, and we expect the overhead to be much less than this value for newer devices.

7.7.2 COMMUNICATION OVERHEAD

Now we turn into evaluating the communication overhead resulting from the messages exchange between the DA and GC. This is the extra bandwidth needed by our system to work in addition to the video data. We measure it as the average number of bits downloaded or uploaded at the end device. We use the same experimental setup that is used in measuring the CPU overhead including the video bitrate profile.

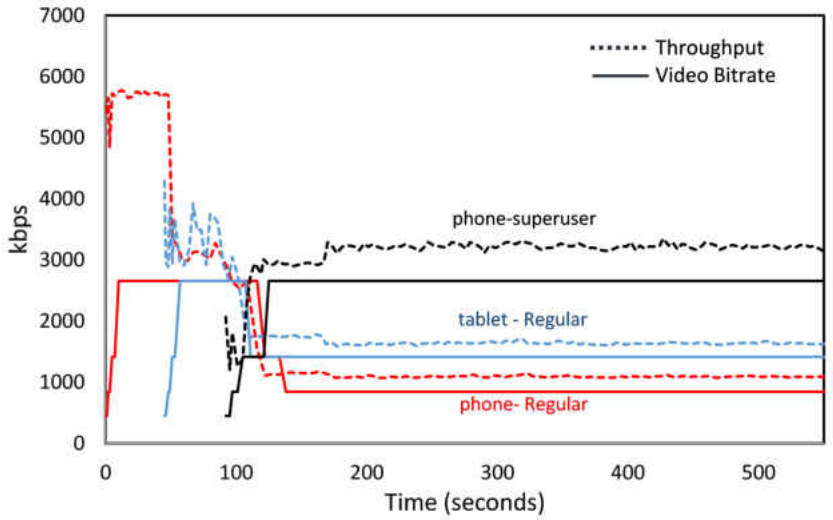

Figure 42: Differentiating between different user classes. FlexStream guarantees better service to users with superuser privileges when the network gets overloaded.

Our results indicates that there is only about 800 bits/second on average added to the bandwidth used by video traffic, which constitutes less than 0.00004% of the total bandwidth. Therefore, the communication overhead caused by our system is also negligible.

7.7.3 COMPUTATION OVERHEAD

In chapter 5, we show that the computation overhead resulting from executing the optimization function is practically very low and sufficient for GC to react in real time. Given the number of possible bitrates for a video session is quite limited in practice, and careful implementation of the dynamic programming steps, the complexity of this solution will be the product of the number of video sessions and available bandwidth, which divided into a series of Z incremental values. Given that the running time depends on this product, its overhead is within sub-second level for the typical practical large values. For instance, as we show before, when $N = 400$, $K = 8$, $B = 200$ Mbps, and $s = 100$ Kbps, the execution time on a single-core Intel 2.20 GHz processor is about 400 ms. As we pointed out before, this small time overhead would allow the GC to react in time before the QoE can be affected. However, if the execution time occasionally becomes larger and above a certain threshold,

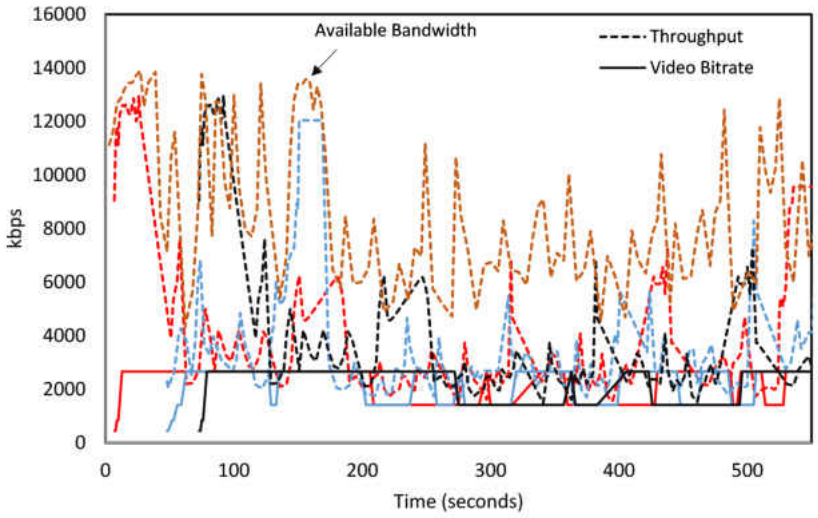

Figure 43: Throughput and requested bitrates of video competing streams with real bandwidth trace. FlexSteam is not enabled.

then FlexStream can speed up the execution time by increasing the bandwidth step size in the optimization function without empirically sacrificing the optimality of the solution, or by utilizing one of the well-known approximate algorithms [59, 54] that guarantees faster execution to be within this a sub-second level at the cost of deviating from the optimal solution. However, we believe that running our optimization function on a newer and more powerful device with mutli-core processor (as it is the case with most of the servers and also with most data centers) can produce much better results which makes the system capable of managing thousand of users within the required time frame (sub-second level).

7.8 CONCLUSION

In this section, we evaluated FlexStream using real experiments setup on mobile devices over both WiFi and cellular networks, in addition to emulated environment for larger-scale experiments. In our evaluation, we mainly focused on five metrics which have the most impact on User's QoE: Instability, inefficiency, playback fluency, and Startup latency, and unfairness. We evaluated our system using two main scenarios: under stable network scenario and when the network is highly dynamic (the available bandwidth for video players is not stable and changes dramatically). These two scenarios were conducted over WiFi networks as well as over a cellular network

at ODU campus. We found that FlexStream reduces the bitrate switching by 81%, stall duration by 92%, and startup delay by 44%, while improving fairness among players. Moreover, this chapter revealed the impact of the background traffic on the watching experience and we presented how FlexStream could effectively minimizing the impact of background traffic. We also showed that Flexstream is capable of supporting different user priorities and considering different screen sizes. To balance the watching experience among users, we showed that FlexStream allows devices with larger screen sizes to stream higher bitrates. Finally, this chapter highlights the impact of having different link condition of the streaming devices. Our experiments showed that devices with poor wireless signals could lose their fair share of bandwidth to other devices having better signals. When FlexStream was activated, this issue was totally overcome and the devices with poor wireless signals can sustain the best possible video bitrate.

CHAPTER 8

CONCLUSION

This chapter summaries the motivation behind this dissertation, the problem that has been addressed, and the solution that we proposed to overcome this problem. It also summaries our the results that we obtain from running our system in addition to listing our plan for future work. This chapter is organized as follows: Section 1 summaries the problem and main motivation of this dissertation. Section 2 briefly summaries our proposed solution and lists the contributions of this dissertation. Section 3 summaries the evaluation part of this dissertation. Finally, our plans for future works are described in Section 4.

8.1 SUMMARY

With the huge increase in mobile data traffic caused by the large increase in the number of smart devices in addition to several bandwidth-intensive applications such as streaming applications (e.g., Youtube, NetFlix, Facebook, etc.) and online games applications, continue providing the end users with satisfactory services becomes challenging. We found that video streaming applications, in particular, are impacted the most when the bandwidth in the network becomes scarce. This is still the case with all video applications despite the huge efforts of the research community and major video providers such as YouTube and Netflix in improving their players. Among the efforts and attempts to improve the performance of streaming videos is the invention and adoption of HTTP video streaming protocol by most video providers nowadays which enable the player to dynamically and seamlessly adapt to the change in the network condition in addition to easily traversing NATs and firwalls as well as eliminating the need of dedicated multimedia servers. However, despite this nice feature of this new technology, our measurements as well as several other studies indicate that most of the state-of-the-art players still suffer from several issues including instability and sub-optimality in the video quality, unfairness in sharing the available bandwidth, and inefficiency with regard to network utilization which would have a negative impact on the video QoE. These performance issues are

frequently experienced when several players start competing over a common bottleneck. Interestingly, it is found that the root cause of these issues is the intermittent traffic pattern of HTTP adaptive protocol that causes the players to over estimate the available bandwidth and starts fluctuating between several qualities.

Several solutions are proposed in the literature in order to overcome these issues and provide seamless and nice streaming experience. For example, works such as those proposed in [41, 35, 48, 47] attempt to improve the player user's performance through improving the ABR algorithm of video players. This ABR algorithm is primarily used to determine the rate of the next video chunk through estimating the available bandwidth using either the throughput measurements of the last couple of received video segments, buffer condition, or hybrid technique which considering both the throughput measurements and buffer fullness in estimating the network condition. Despite the intensive efforts to improve the performance of adaptive algorithms at the client side, the major issues that impact the video QoE remains unsolved with these solutions as indicated by our measurements and also by several studies mentioned above[6]. This is due to the inability of a player to realize both the current condition of the network and the existing of other competing players. Thus the estimation algorithm clearly will not lead to an accurate estimation of the network capacity. Moreover, client-based solutions lack the flexibility that precludes the network administrator to apply some policies and achieve specific requirements.

Another attempt to address the performance issues with the adaptive application is at the server side. For instance, the works in [3] proposed server-based traffic shaping techniques which adjust the streaming rate at the server to be too close from the requested bitrate to avoid generating the OFF periods and thus stabilizing the bitrate. However, modifying a standard HTTP server, as their techniques required, is definitely not an attractive technique in addition to adding significant overhead on the server through monitoring and traffic shaping functions. Other works try to improve the performance at the network edge such as [23, 58]. However, most of these works are either expensive to adopt or application dependent.

Therefore, the research in this dissertation aims to address these performance issues and maximizes the user's QoE. We proposed the design, implementation, and evaluation of FlexStream, a system that we developed on top of the framework that extends the SDN to mobile devices. FlexStream enables centralized control of bandwidth allocation via a global controller that specifies a policy, and a local policy

implementation via Open vSwitch (OVS) that offloads the fine-grained functionality to the end device. This means that FlexStream uses a hybrid approach, which takes advantage of both centralized and distributed components. While a network element (GC) is primarily employed to manage resource allocation for video flows, monitoring and policy enforcement tasks are offloaded from the network to end-devices, via lightweight software agents. This alleviates the need for intrusive, large and costly traffic management solutions within the network, or modifications to servers that are not feasible in practice [3, 10, 44]. FlexStream is designed to maximize QoE in the access network by allocating the highest sustainable bitrates to adaptive players while ensuring: (i) the minimal variations in the quality, (ii) minimum number of stalls, and (iii) well-balanced and fair QoE.

In addition to the bandwidth control policy, the system supports defining various control policies based on the different interests and contexts of the user, device, video, network, and environment (e.g., user priority-based policy, device screen size-based policy), which is often overlooked in practical implementations. Using an optimization function, we demonstrate that all these factors can be effectively accounted for within the policy that allocates bandwidth across devices. Network programmability is also one of the main features of FlexStream, where network policies can be implemented and enforced in real time based on the context (e.g., time, location, flow type). Finally, as end devices running FlexStream can be treated as logical switches with ports acting like the available network interfaces (e.g., WiFi and cellular), they can support multi-path delivery (e.g., MPTCP) according to user-specified interface preferences [26].

8.2 CONTRIBUTION

The following are the main contributions that have been made by this dissertation:

- We develop FlexStream on top of the SDN-based framework that extends SDN functionality to mobile end devices, allowing for fine-grained control and 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 WiFi 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 best of our knowledge, the first working implementation of the SDN extension to commodity mobile devices that runs in both WiFi and cellular networks without requiring support from the existing network infrastructure.

8.3 EVALUATION

We evaluated FlexStream using real testbeds including real mobile devices and players over both WiFi and cellular networks. In our evaluation, we considered two main scenarios: (1) when the network bandwidth available to video players is stable, (2) when the bandwidth is highly dynamic. In these two scenarios we focus on five main evaluation metrics which we believe that are the most important for users' QoE including instability in the video bitrate, playback fluency, video quality, startup latency, and unfairness in utilizing the network resources which leads to unbalanced and unfair QoE among the end users. We summarize our findings as follows:

- Most of the state-of-the-art adaptive players still suffer from several performance issues with players compete for bandwidth over the same network bottleneck.
- Under both stable and dynamic scenarios, we found that player competition causes extreme instability in the video bitrate which not only impacts the user's QoE but also causes a significant waste in the bandwidth reaches up to 40%.
- Adaptive video players can still encounter rebuffering events (playback freeze) especially under dynamic scenario and also when there is no sufficient data in the buffer to observe the drop in the network condition.
- When multiple video players simultaneously stream over the same network bottleneck, unfairness in utilizing the network resources is observed. This creates an unfair situation in which some users can have nice watching experience while other can not.

- When some players start streaming over already overloaded access point, these player tend to have a long startup latency as the bandwidth is usually dominated by other existing players.
- Background traffic found to have an impact on the video sessions. It can cause instability and other issues depending on the number of the flows.
- We found that the variation in the link conditions among the end devices can lead to unfair usage of network resources, especially over the cellular network, leading to a significant drop in the quality for devices with poor wireless signals.
- Our proposed Systems, FlexStream, could minimize all the performance issues with adaptive video players and provide high watching experience to end users.
- FlexStream can reduce the bitrate switching by 81%, stall duration by 92%, and startup delay by 44%, while improving fairness among players and eliminating the impact of link condition variation in addition to minimizing the impact of background traffic.
- Flexstream is capable of supporting different user priorities and considering different screen sizes, allowing devices with larger screen sizes to stream higher bitrates.

8.4 FUTURE WORK

As streaming video primarily comes with two modes, live and on-demand streaming, and the fact that video players typically set the buffer size according to the type of the streaming mode, the need for an accurate buffer size estimation technique can considerably enhance the performance. When streaming live videos, a video player tends to maintain a relatively small buffer to ensure the freshness of the data. This small buffer size makes the video playback more prone to quality switches and playback stalls. Therefore, it is essential for good streaming performance to differentiate between users steaming live events and those streaming on-demand videos. To this end, as part of our future works, we will focus on developing a new function for estimating the player buffer fullness and then incorporate it with other FlexStream modules. This can be achieved by utilizing the SDN on mobile devices in monitoring and calculating the amount of data received by the player in addition to intercepting

the HTTP requests sent by the video player to get the requested video bitrate of the requested video segments.

Another avenue for extending our system is to make it fully capable of managing encrypted streaming sessions. Several video providers have already adopted encryption for video delivery. Therefore the need for the extension to work and mange encrypted video sessions becomes essential. Note that FlexStream comes now with the ability to estimate the current video bitrate of an encrypted video session, and thus we will only need to extend its functionality to estimate the buffer size, and possibly other bitrate profiles. As indicated before, the traffic pattern of adaptive players reveals several pieces of information about the player status. We will also utilize the OVS on a mobile device to collect video traffic statistics and use them in estimating the buffer size.

In addition, as part of our future work, we would like to investigate and study the impact of the competition of heterogeneous adaptive players and understand how this is different than the case of having only one type of video players. We will also check how FlexStream would perform under this real scenario in which different players would generate different ON/OFF periods and may use different adaptive techniques. We will focus on improving the resource management module aiming to minimize the bandwidth underutilization resulting from the traffic shaping, which will result in maximizing the video QoE. Another possible extension that can improve the watching experience is to check the possibility of scheduling the background traffic during the players sleeping times. The goal is to improve the bandwidth utilization and to entirely remove the impact of background traffic on the video sessions. It is worth to re-mentioning that FlexStream is already developed with the function of controlling and shaping the background traffic on supported devices in addition to have the capability of directing the background traffic to be received from another network interface which totally eliminates its impact on the running video sessions.

I would also like to investigate and study the advantages and downsides of removing the OFF periods of adaptive players at the client side, using our DA and SDN components. This elimination of these periods is quite easy to implement for unencrypted traffic while needs current bitrate estimation for encrypted sessions. In fact, this idea stemmed from our conclusion that the root cause of all problems with adaptive players is their intermittent traffic pattern. Although we believe that this approach can further stabilize the video bitrate, the major gain will be maximizing

the video bitrate in the network. However, this approach requires that the network interface on the mobile device will constantly be on which might increase the battery consumption. Thus we need to measure the power consumption of this approach to check its practicality. Also, we need to investigate the impact on the HTTP server as now the server needs to continually sending video packets to the client, which might increase the overhead.

Many people nowadays stream video to solely listen to the audio (mostly while walking or driving) with no interest in watching. This implies that reducing the video quality for those non interested in watching would not lead to any impact on their QoE. Therefore, we are studying the possibility of utilizing this knowledge to reallocate the network resource based on the user interest in watching the video. This will enable FlexStream to efficiently manage the network resources and maximize the videos QoE when the network gets overloaded. For example, suppose that a user is deriving while streaming a video to only listen to the music (not interested in watching). Also, suppose that he is joining a new cell tower of a cellular network that provides a service to hundreds of people who are attending a football game at a city stadium. Since people at such an event are usually overloading the network by doing live broadcasting, streaming replies, ..etc., the bandwidth at this time becomes very scarce. Therefore, it will be more efficient for QoE that FlexStream considers lowering the video quality on the driver to release some bandwidth that can be used to improve the QoE for other users. Since the driver is not showing any interest in watching and also the reduction would only last for a small period of time (as cars are moving very fast between towers), the QoE for this user will not be impacted. As we saw in the previous example, considering user interest in watching video would save much bandwidth. However, incorporating this context information is not an easy task in practice. There are several challenges that need to be addressed:

1. Detecting whether the user is interested in watching the video.
2. Predicting the possibility that the user will not be interested again in the near future.
3. When the user becomes suddenly interested, the video quality needs to be switched up immediately to avoid user frustration.
4. Switching from low to higher quality may cause waste in network and device resource.

Another possible way to extend the system is to integrate FleXstream with Multipath TCP - Fortunately, an extension of the standard TCP, MultiPath TCP (MPTCP) [19] have been recently launched by Internet Engineering Task Force (IETF) which is currently considered the de-facto multipath solution enabling applications to transparently and simultaneously utilize several paths for receiving data. However, this protocol does not take into account the user preferences in using different network interfaces. To this end, we are examining the possibility of integrating MPTCP with FlexStream to enable simultaneous multipath streaming over both WiFi and cellular taking into consideration the users' preferences in using network interfaces. The idea is to leverage the SDN and TCP flow control mechanism to control and manage the amount of data that should be streamed in parallel from each interface.

We would also like that our system can consider a situation when the network is overloaded with too many streams in which some users might not be able to get enough bandwidth to stream good quality videos or even can not stream at all. This situation is most likely encountered at (very crowded) public places such as airports or stadiums. Such a low QoE is likely to cause users frustration, and thus reducing their engagement. In or order to minimize this negative impact and improve the QoE when experiencing a severe bottleneck, we propose to coordinate between the end-users in using the network resources. This requires an explicit interaction between end-users and the network along with some incentives to persuade and encourage some users to defer their streaming for some time. The incentives can be bandwidth points that are increased with the time. For instance, the user can get one minute of high bandwidth (that ensures streaming high quality video) for every two minutes of waiting time. In fact, this coordinating was motivated by the fact that most of the streaming is not life which implies that deferring the watching will not lead to missing any scene. Moreover, some users may prefer to wait several minutes to watch a high quality video than immediate watching with low QoE (re-buffering events, low quality, etc.).

CHAPTER 9

PUBLISHED WORK

1. Ibrahim Ben Mustafa, and Tamer Nadeem. " FlexStream:Towards Edge-Based SDN Architecture for Programmable and Flexible Adaptive Video Streaming" (In preparation for ACMMultimedia, ACM, 2018).

2. Ibrahim Ben Mustafa, Mostafa Uddin, and Tamer Nadeem. "Understanding the intermittent traffic pattern of HTTP video streaming over wireless networks." Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2016 14th International Symposium on. IEEE, 2016.

3. Mostafa Uddin, Ibrahim Ben Mustafa, and Tamer Nadeem. "EdgeEye: Fine Grained Traffic Visibility at Wireless Network Edge." Edge Computing (SEC), IEEE/ACM Symposium on. IEEE, 2016.

4. Ibrahim Ben Mustafa, and Tamer Nadeem. "Dynamic traffic shaping technique for http adaptive video streaming using software defined networks." Sensing, Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on. IEEE, 2015.

5. Ibrahim Ben Mustafa, Uddin Mostafa, and Tamer Nadeem, "Extending SDN to Cellular Network End Devices". Demo at AT&T Research Academic Summit 2016.

REFERENCES

[1] H. D. S. . Adobe Systems Inc. http://www.adobe.com/products/hds-dynamic-streaming.html.[Online; accessed 06-Mar-2017].

[2] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. What happens when http adaptive streaming players compete for bandwidth? In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, pages 9–14. ACM, 2012.

[3] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 19–24. ACM, 2013.

[4] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 157–168. ACM, 2011.

[5] H. L. S. . Apple Inc. https://developer.apple.com/.[Online; accessed 06-Mar-2017].

[6] I. Ayad, Y. Im, E. Keller, and S. Ha. A practical evaluation of rate adaptation algorithms in http-based adaptive streaming. *Computer Networks*, 133:90–103, 2018.

[7] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. A quest for an internet video quality-of-experience metric. In *Proceedings of the 11th ACM workshop on hot topics in networks*, pages 97–102. ACM, 2012.

[8] Bitmovin. http://bitmovin.com.[Online; accessed 14-Mar-2018].

[9] N. Bouten, J. Famaey, S. Latr, R. Huysegems, B. D. Vleeschauwer, W. V. Leekwijck, and F. D. Turck. Qoe optimization through in-network quality adaptation for http adaptive streaming. In *2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualization management (svm)*, pages 336–342, Oct 2012.

[10] N. Bouten, J. Famaey, S. Latré, R. Huysegems, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. QoE optimization through in-network quality adaptation for HTTP adaptive streaming. In *Proceedings of the 8th International Conference on Network and Service Management*, CNSM '12, pages 336–342, 2013.

[11] Conviva. http://www.conviva.com.[Online; accessed 27-Mar-2017].

[12] N. Cranley, P. Perry, and L. Murphy. User perception of adapting video quality. *International Journal of Human-Computer Studies*, 64(8):637–647, 2006.

[13] Dailymotion. http://dailymotion.com.[Online; accessed 03-Mar-2017].

[14] I. Delchev. Linux traffic control. In *Networks and Distributed Systems Seminar, International University Bremen, Spring*, 2006.

[15] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 362–373. ACM, 2011.

[16] J. Esteban, S. A. Benno, A. Beck, Y. Guo, V. Hilt, and I. Rimac. Interactions between http adaptive streaming and tcp. In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, pages 21–26. ACM, 2012.

[17] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 281–287. ACM, 2010.

[18] K. R. Fall and W. R. Stevens. *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011.

[19] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. Tcp extensions for multipath operation with multiple addresses, draft-ietf-mptcp-multiaddressed-09. *Internetdraft, IETF (March 2012)*, 2012.

[20] T. Friedman, R. Caceres, and A. Clark. Rtp control protocol extended reports (rtcp xr). Technical report, 2003.

[21] A. Ganjam, F. Siddiqui, J. Zhan, X. Liu, I. Stoica, J. Jiang, V. Sekar, and H. Zhang. C3: Internet-scale control plane for video quality optimization. In *NSDI*, volume 15, pages 131–144, 2015.

[22] P. Georgopoulos, M. Broadbent, A. Farshad, B. Plattner, and N. Race. Using software defined networking to enhance the delivery of video-on-demand. *Comput. Commun.*, 69(C):79–87, Sept. 2015.

[23] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. Towards network-wide qoe fairness using openflow-assisted adaptive video streaming. In *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, FhMN '13, pages 15–20, New York, NY, USA, 2013. ACM.

[24] A. Gouta, C. Hong, D. Hong, A.-M. Kermarrec, and Y. Lelouedec. Large scale analysis of http adaptive streaming in mobile networks. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*, pages 1–10. IEEE, 2013.

[25] B. Han, F. Qian, S. Hao, and L. Ji. An anatomy of mobile Web performance over MultiPath TCP. In *Proceedings of the The 11th International Conference on emerging Networking EXperiments and Technologies*, CoNEXT '15, New York, NY, USA, 2015. ACM.

[26] B. Han, F. Qian, S. Hao, and L. Ji. An anatomy of mobile Web performance over MultiPath TCP. In *Proceedings of the The 11th International Conference on emerging Networking EXperiments and Technologies*, CoNEXT '15, New York, NY, USA, 2015. ACM.

[27] B. Han, F. Qian, L. Ji, V. Gopalakrishnan, and N. Bedminster. Mp-dash: Adaptive video streaming over preference-aware multipath. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 129–143. ACM, 2016.

[28] M. Handley, C. Perkins, and V. Jacobson. Sdp: session description protocol. 2006.

[29] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner. Assessing effect sizes of influence factors towards a qoe model for http adaptive streaming. In *Quality of*

Multimedia Experience (QoMEX), 2014 Sixth International Workshop on, pages 111–116. IEEE, 2014.

[30] R. Houdaille and S. Gouache. Shaping http adaptive streams for a better user experience. In *Proceedings of the 3rd Multimedia Systems Conference*, pages 1–9. ACM, 2012.

[31] R. Houdaille and S. Gouache. Shaping http adaptive streams for a better user experience. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 1–9, New York, NY, USA, 2012. ACM.

[32] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 225–238. ACM, 2012.

[33] T.-Y. Huang, R. Johari, and N. McKeown. Downton abbey without the hiccups: Buffer-based rate adaptation for http video streaming. In *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*, pages 9–14. ACM, 2013.

[34] T. Y. Huang, R. Johari, N. Mckeown, M. Trunnel, and M. Watson. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *SIGCOMM*, SIGCOMM, pages 187–198, 2014.

[35] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review*, 44(4):187–198, 2015.

[36] Hulu. http://hulu.com.[Online; accessed 03-Mar-2017].

[37] C. V. N. Index. Global mobile data traffic forecast update 2016–2021. white paper c11-520862. *Available on http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html*.

[38] V. Jacobson, R. Frederick, S. Casner, and H. Schulzrinne. Rtp: A transport protocol for real-time applications. 2003.

[39] R. Jain, D.-M. Chiu, and W. R. Hawe. *A quantitative measure of fairness and discrimination for resource allocation in shared computer system*, volume 38. 1984.

[40] Y. Jarraya, T. Madi, and M. Debbabi. A survey and a layered taxonomy of software-defined networking. *IEEE Communications Surveys & Tutorials*, 16(4):1955–1980, 2014.

[41] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 97–108, Dec 2012.

[42] F. Kelly. Charging and rate control for elastic traffic. *Transactions on Emerging Telecommunications Technologies*, 8(1):33–37, 1997.

[43] J. Kleinrouweler, S. Cabrero, R. van der Mei, and P. Cesar. Modeling stability and bitrate of network-assisted http adaptive streaming players. In *Teletraffic Congress (ITC 27), 2015 27th International*, pages 177–184, Sept 2015.

[44] J. W. Kleinrouweler, S. Cabrero, R. van der Mei, and P. Cesar. Modeling stability and bitrate of network-assisted http adaptive streaming players. In *Teletraffic Congress (ITC 27), 2015 27th International*, pages 177–184. IEEE, 2015.

[45] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[46] X. Li, M. Dong, Z. Ma, and F. C. Fernandes. Greentube: power optimization for mobile videostreaming via dynamic cache management. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 279–288. ACM, 2012.

[47] 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*, 32(4):719–733, 2014.

[48] C. Liu, I. Bouazizi, and M. Gabbouj. Rate adaptation for adaptive http streaming. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 169–174. ACM, 2011.

[49] N. McKeown. Software-defined networking. *INFOCOM keynote talk*, 17(2):30–32, 2009.

[50] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[51] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz. Adaptation algorithm for adaptive streaming over http. In *Packet Video Workshop (PV), 2012 19th International*, pages 173–178. IEEE, 2012.

[52] R. K. Mok, X. Luo, E. W. Chan, and R. K. Chang. Qdash: a qoe-aware dash system. In *Proceedings of the 3rd Multimedia Systems Conference*, pages 11–22. ACM, 2012.

[53] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang. Practical, real-time centralized control for cdn-based live video delivery. *ACM SIGCOMM Computer Commu nication Review*, 45(4):311–324, 2015.

[54] Y. E. Nesterov and M. J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations research*, 22(1):1–42, 1997.

[55] Netflix. http://Netflix.com.[Online; accessed 03-Mar-2017].

[56] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634, 2014.

[57] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling tcp throughput: A simple model and its empirical validation. *ACM SIGCOMM Computer Communication Review*, 28(4):303–314, 1998.

[58] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck. Qoe-driven rate adaptation heuristic for fair adaptive video streaming. *ACM Trans. Multimedia Comput. Commun. Appl.*, 12(2):28:1–28:24, Oct. 2015.

[59] D. Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, 83(2):394–410, 1995.

[60] F. M. Ramos, D. Kreutz, and P. Verissimo. Software-defined networks: On the road to the softwarization of networking. *Cutter IT journal*, 2015.

[61] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz. A comparison of quality scheduling in commercial adaptive http streaming solutions on a 3g network. In *Proceedings of the 4th Workshop on Mobile Video*, pages 25–30. ACM, 2012.

[62] D. C. Robinson, Y. Jutras, and V. Craciun. Subjective video quality assessment of http adaptive streaming technologies. *Bell Labs Technical Journal*, 16(4):5–23, 2012.

[63] H. Schulzrinne. Real time streaming protocol (rtsp). 1998.

[64] H. Shi, R. V. Prasad, E. Onur, and I. Niemegeers. Fairness in wireless networks: Issues, measures and challenges. *IEEE Communications Surveys & Tutorials*, 16(1):5–24, 2014.

[65] C. Sieber, A. Blenk, M. Hinteregger, and W. Kellerer. The cost of aggressive http adaptive streaming: Quantifying youtube's redundant traffic. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1261–1267. IEEE, 2015.

[66] C. Sieber, P. Heegaard, T. Hoßfeld, and W. Kellerer. Sacrificing efficiency for quality of experience: Youtube's redundant traffic behavior. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2016*, pages 503–511. IEEE, 2016.

[67] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2016.

[68] N. Staelens, J. De Meulenaere, M. Claeys, G. Van Wallendael, W. Van den Broeck, J. De Cock, R. Van de Walle, P. Demeester, and F. De Turck. Subjective quality assessment of longer duration video sequences delivered over http adaptive streaming to tablet devices. *IEEE Transactions on Broadcasting*, 60(4):707–714, 2014.

[69] T. Stockhammer. Dynamic adaptive streaming over HTTP –: standards and design principles. In *ACM conference on Multimedia systems*, MMSys '13, pages 133–144, 2011.

[70] T. Stockhammer. Dynamic adaptive streaming over http–: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144. ACM, 2011.

[71] E. Thomas, M. van Deventer, T. Stockhammer, A. C. Begen, and J. Famaey. Enhancing mpeg dash performance via server and network assistance. 2015.

[72] G. Tian and Y. Liu. On adaptive http streaming to mobile devices. In *Packet Video Workshop (PV), 2013 20th International*, pages 1–8. IEEE, 2013.

[73] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. Iperf: The tcp/udp bandwidth measurement tool. *htt p://dast. nlanr. net/Projects*, 2005.

[74] A. Unknown. Open vswitch, an open virtual switch. *Date Unknown but prior to Dec*, 30(2), 2010.

[75] B. J. Villa, K. De Moor, P. E. Heegaard, and A. Instefjord. Investigating quality of experience in the context of adaptive video streaming: findings from an experimental user study. *Akademika forlag Stavanger, Norway*, 2013.

[76] Vimeo. http://vimeo.com.[Online; accessed 14-Mar-2018].

[77] H. Vin, P. Goyal, and A. Goyal. A statistical admission control algorithm for multimedia servers. In *Proceedings of the second ACM international conference on Multimedia*, pages 33–40. ACM, 1994.

[78] J. Xue and C. W. Chen. Mobile video perception: New insights and adaptation strategies. *IEEE journal of selected topics in signal processing*, 8(3):390–401, 2014.

[79] YouTube. http://youtube.com.[Online; accessed 03-Mar-2017].

[80] A. Zambelli. Iis smooth streaming technical overview. *Microsoft Corporation*, 3:40, 2009.

[81] L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (rsvp)–version 1 functional specification. *Resource*, 1997.

VITA

Ibrahim Ben Mustafa
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

Ibrahim Ben Mustafa received his Bachelor's Degree in Computer Science from University of Tripoli in 2001. He also earned his masters degree in Computer Science from New Mexico State University in 2011. Currently, he is a PhD candidate in the Computer Science Department at Old Dominion University. His research interest focuses on wireless networks, adaptive video streaming, and software defined network. In particular, my current research activities revolve around enhancing the performance of adaptive video players on mobile devices under various scenarios and network conditions.