HTTP ADAPTIVE STREAMING ARCHITECTURES FOR VIDEO ON DEMAND AND LIVE TV SERVICES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

YİĞİT ÖZCAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2013

Approval of the thesis:

HTTP ADAPTIVE STREAMING ARCHITECTURES FOR VIDEO ON DEMAND AND LIVE TV SERVICES

submitted by **YİĞİT ÖZCAN** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan
Head of Department, **Electrical and Electronics Engineering**

Assoc. Prof. Dr. Şenan Ece Schmidt
Supervisor, **Electrical and Electronics Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Semih Bilgen
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Şenan Ece Schmidt
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Gözde Akar
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Cüneyt Bazlamaçcı
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Dept., METU

Date:

ABSTRACT

HTTP ADAPTIVE STREAMING ARCHITECTURES FOR VIDEO ON DEMAND AND LIVE TV SERVICES

ÖZCAN, Yiğit
M.S., Department of Electrical and Electronics Engineering
Supervisor : Assoc. Prof. Dr. Şenan Ece Schmidt

August 2013, 57 pages

HTTP Adaptive Streaming (HAS) has become a popular video streaming solution since it both benefits from the ubiquitous HTTP protocol and firewall and NAT traversal capabilities of TCP. HAS aims to provide high Quality of Experience (QoE) to the clients under limited and varying bandwidth by *rate adaptation* algorithms which allow the clients to choose the most appropriate video quality. A rate adaptation algorithm should utilize the available bandwidth. Furthermore, the received video bitrates should not deviate from each other leading to an unfair bandwidth use among the clients. It is also desired to minimize the rate switches as they degrade QoE of the clients. In this thesis, we propose two architectures that operate on HAS. The first architecture is *FEedback based Adaptive STreaming over HTTP* (FEAST). FEAST enables the clients to adapt their rates according to the total number of clients, average video rate and the average bandwidth information provided by the server. These values are computed as moving averages by the server with a small amount of information sent from the clients. The server side computation is simple and not client specific which makes FEAST a scalable solution. The second architecture is *Adaptive LIVE Streaming over HTTP* (ALIVE) which enables a high number of clients to watch live TV channels over HTTP. ALIVE is based on enabling the clients to download the contents from nearby clients instead of the server whenever it is possible. ALIVE employs SVC which makes it possible to adapt the video bitrates of the clients even when they download from other clients. ALIVE decreases the load of the server and accommodates more clients as we demonstrate with simulations.

Keywords: video streaming, HTTP, video on demand, live TV, scalable video coding

ÖZ

ISMARLAMA VİDEO VE CANLI TV SERVİSİ İCİN HTTP ADAPTİF AKIŞ MİMARİLERİ

ÖZCAN, Yiğit
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç Dr. Şenan Ece Schmidt

Ağustos 2013, 57 sayfa

HTTP Adaptif Akışı (HAS), HTTP protokolünün yaygın kullanımı ve TCP protokolünün güvenlik duvarı ve ağ adresi dönüştürücülerinden geçiş yapabilme yeteneğinden dolayı popüler bir video akış çözümü haline gelmiştir. HAS, sınırlı ve değişken bant genişliği olan durumlarda hız adaptasyonu yöntemi ile kullanıcıların kendilerine en uygun video bit hızlarını seçmelerini sağlayarak, kullanıcılara yüksek deneyim kalitesi sunabilmeyi amaçlamaktadır. Hız adaptasyonu algoritmalarında, kullanılabilir bant genişliğinin yararlı bir şekilde kullanılmalıdır. Bunun yanında, kullanıcıların video bit hızları arasında büyük fark olmaması da gereklidir. Kullanıcıların bit hızları arasında yaptığı geçiş sayısının da en aza indirilmesi de deneyim kalitesi için önemlidir. Bu tezin ilk kısmında ısmarlama video servisi için kullanıcıların video bit hızlarını sunucudan aldıkları ortalama video bit hızı, ortalama bant genişliği ve kullanıcı sayısına göre belirlemelerini sağlayan Geri Besleme Temelli HTTP Adaptif Akış Uygulamasını (FEAST) öneriyoruz. Bu ortalamalar sunucu tarafından kullanıcılardan aldığı bilgiye dayanarak ve çok az durum bilgisi tutularak hesaplanır. Bu hesaplamalar basittir ve her kullanıcı için ayrı yapılmamaktadır. Bu da FEAST mimarisini ölçeklenebilir bir çözüm yapmaktadır. Tezin ikinci kısmında çok sayıda kullanıcının canlı yayınları internet üzerinden izlemesine olanak veren HTTP üzerinden Adaptif Canlı Akış (ALIVE) uygulamasını öneriyoruz. ALIVE, kullanıcıların içerikleri uygun olduğu durumlarda sunucu yerine daha yakında bulunan diğer kullanıcılardan indirmesini sağlayan bir sistemdir. ALIVE, SVC kullanarak kullanıcılara içeriği başka kullanıcıdan indirse bile bit hızını ayarlama olanağı sağlar. Simülasyonlarla da gösterdiğimiz gibi ALIVE sunucunun yükünü azaltarak daha fazla kullanıcının desteklenmesini sağlar.

Anahtar Kelimeler: kesintisiz video, HTTP, ısmarlama video, canlı TV, ölçeklenebilir video

To My Father

ACKNOWLEDGMENTS

I would like to express my special thanks to my supervisor Assoc. Prof. Dr. Şenan Ece Schmidt. I could not finish this thesis, even could not start this thesis without her support, guidance and friendship. It was an honor to work with her both as a student and a teaching assistant. Every graduate student must have a supervisor like her.

This thesis is dedicated to my father. It is impossible for me to express my feelings. My dear father, the only thing I will say is I missed you so much and I love you so much. Thank you for always being with me. I owe you everything in my life.

I would also like to thank my mother who never stopped loving and supporting me. I will feel great if this work will make her proud even if just a bit. Thank you for your lifelong support. I am very lucky to have a mother like you. I must also thank to my sister Nazlı with whom I grew up together and learnt the life. She is the person whom I spent most of my life with. She always praised me and made me feel good. I also acknowledge my grandfather who encouraged me and lead the way in my academic life. I thank my grandmother, my aunt Eser and Binhan and my other relatives for supporting me.

I want to thank Vicdan who was with me for a long time to here. It would be impossible to achieve this without her support. Thank you for being with me all the time. Thank you for your unending support and patience. Now it is my turn to support for your thesis.

I must thank my friends Can who I always say that, Burak *the smurf*, Egemen with whom I become another person, Tamer who works seriously but lives in a joy, Ender who knows every thing and does not care about any thing, Ayşegül who will grow up, Tayfun who is my ex-student, Dicle *the bride*, Osman and Hakkı who are not real and all my other friends. All of them encouraged me and made me believe that I can accomplish this work.

TABLE OF CONTENTS

ABSTRACT v
ÖZ vi
ACKNOWLEDGMENTS viii
TABLE OF CONTENTS ix
LIST OF TABLES x
LIST OF FIGURES xi
LIST OF ABBREVIATIONS xiv
CHAPTERS
1 INTRODUCTION 1
2 BACKGROUND 5
2.1 Internet Multimedia 5
2.2 Scalable Video Coding (SVC) 6
2.3 HTTP Adaptive Streaming 7
2.3.1 Microsoft Smooth Streaming 11
2.3.2 Rate Adaptation for Adaptive HTTP Streaming: Liu at al. 11
3 FEAST: FEEDBACK BASED ADAPTIVE STREAMING OVER HTTP . . . 15
3.1 Motivation for FEAST 15
3.2 Performance Metrics 17
3.3 Operation of FEAST 19
3.3.1 Client-Server Information Exchange and Server Side Computations 19
3.3.2 Rate Adaptation on the Client Side 20
3.4 Performance Evaluation of FEAST 23
4 ALIVE: ADAPTIVE LIVE STREAMING OVER HTTP 33
4.1 Motivation for ALIVE 33
4.2 Operation of ALIVE 34
4.2.1 Seed Selection for ALIVE 35
4.2.2 Rate Adaptation for ALIVE Algorithm 42
4.3 Performance Evaluation of ALIVE 44
5 CONCLUSION 51
REFERENCES 53

LIST OF TABLES

TABLES

Table 2.1 Comparison of existing solutions for Internet Multimedia 6

Table 3.1 Rate Decision 22
Table 3.2 Unfairness 31
Table 3.3 Switching Rates 31
Table 3.4 Avg. Video Rates 31

Table 4.1 Applied Encoding Rates 45
Table 4.2 Avg. Video bitrate according to number of channels 49

LIST OF FIGURES

FIGURES

Figure 2.1 IP multicast tree structure 6
Figure 2.2 Illustration of Media Presentation Description File 8
Figure 2.3 A client first downloads MPD and then downloads the successive segments 9

Figure 3.1 ON-OFF periods of two HAS clients: Case1 16
Figure 3.2 ON-OFF periods of two HAS clients: Case2 16
Figure 3.3 ON-OFF periods of three HAS clients 16
Figure 3.4 Average video bitrates and measured bandwidth of the clients 18
Figure 3.5 The network topology that is used in simulations 24
Figure 3.6 Unfairness comparison of HAS algorithms 25
Figure 3.7 Average video bitrate comparison of HAS algorithms 25
Figure 3.8 Unfairness comparison of HAS algorithms in the existance of background traffic 26
Figure 3.9 Average video bitrate comparison of HAS algorithms in the existance of background traffic 26
Figure 3.10 Average unfairness comparison of HAS algorithms according to background traffic rate 27
Figure 3.11 Rate switching comparison of HAS algorithms according to background traffic rate 27
Figure 3.12 Average video bitrate comparison of HAS algorithms according to background traffic rate 28
Figure 3.13 Average unfairness comparison of HAS algorithms according to number of clients 28
Figure 3.14 Average video bitrate comparison of HAS algorithms according to number of clients 29
Figure 3.15 Average rate switch comparison of HAS algorithms according to number of clients 30

Figure 4.1 The topology that denotes the importance of hop count 36
Figure 4.2 Content preparation in the server 37
Figure 4.3 Client downloads the first four segments: Case1 38
Figure 4.4 Client downloads the first four segments: Case2 38
Figure 4.5 Client downloads the first four segments: Case3 39
Figure 4.6 The backbone network that is used in ALIVE simulations 44
Figure 4.7 An example network with 20 clients that is used in ALIVE simulations . . 44

Figure 4.8 Effect of s parameter on the probability mass function of Zipf distribution 46
Figure 4.9 A sample Average video bitrates vs time graph for ALIVE and Liu at al. . 47
Figure 4.10 Comparison of number of clients that download the content from the server 47
Figure 4.11 Average video bitrates of the clients according to different percentage of clients behind NAT 48
Figure 4.12 Average video bitrates of the clients according to different number of clients 48

LIST OF ABBREVIATIONS

3GPP	3rd Generation Partnership Project
AIMD	Additive Increase Multiple Decrease
ALIVE	Adaptive Live Streaming over HTTP
AVC	Advanced Video Coding
CBR	Constant Bit Rate
CDN	Content Distribution Network
DASH	Dynamic Adaptive Streaming over HTTP
FEAST	Feedback Based Adaptive Streaming over HTTP
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IMS	IP Multimedia Subsystem
IP	Internet Protocol
HAS	HTTP Adaptive Streaming
HD	High Definition
HTTP	HyperText Terminal Protocol
MPEG	Moving Pictures Experts Group
MSD	Media Segment Duration
MSS	Microsoft Smooth Streaming
MVC	Multiview Video Coding
NAT	Network Address Translator
NS2	Network Simulator 2
P2P	Peer to Peer
PSNR	Peak Signal to Noise Ratio
QoE	Quality of Experience
QoS	Quality of Service
RTCP	RTP Control Protocol

RTP	Real-time Transport Protocol
SFT	Segment Fetch Time
SIP	Session Initiation Protocol
SR	Switching Rate
SVC	Scalable Video Coding
TCP	Transmission Control Protocol
TV	Television
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VBR	Variable Bit Rate
VoD	Video on Demand
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

In recent years, multimedia applications have become very popular and the amount of multimedia traffic is rapidly growing every year. The most important and widely adopted class of multimedia is video applications. According to the predictions of Cisco [1], 90 percentage of the total Internet traffic will be video related in 2017. With the same predictions, the amount of total video traffic will be about 1.2 zettabytes per year.

With this enormous demand for multimedia, many commercial video tools appeared in the last few years. Youtube [2], Hulu [3], Move Networks [4], Microsoft Smooth Streaming [5], Apple HTTP Live Streaming [6] are some examples. These video tools give both Video on Demand and Live TV services to its clients.

Internet protocol [7] is a best-effort service and it does not guarantee any Quality of Service (QoS) which is essential for video applications. Therefore, some solutions were offered in the past years to cope with the enormous amount of multimedia traffic. One of the important solutions is video compression. Amount of raw video data is so huge that it is impossible to carry it on the network. Video data can be compressed and video traffic can be reduced by compression methods. In addition, network infrastructures are upgraded to carry more traffic with fiberoptic or wireless channels.

Many networking protocols are developed for managing multimedia traffic. Internet Engineering Task Force (IETF) developed RTP [8] (Real-Time Transport Protocol) which works over UDP and carries multimedia traffic. It works in conjunction with RTP Control Protocol (RTCP) which provides feedback information about network conditions. IP Multimedia Subsystem [9] is proposed by 3rd Generation Partnership Project for providing multimedia services over IP in next generation networks. Another solution is IP Multicast [10] which is a promising solution by avoiding redundant traffic. However, these above solutions do not possess scalability and are not easy to deploy in the existing network. Furthermore, firewalls and NAT traversals affect their operation.

HTTP Adaptive streaming (HAS) is a promising solution for multimedia transfer [11]. In HAS, clients download the contents by sending successive GET requests to the server. The server encodes the video into small segments of generally 1 to 10 seconds with different video bitrates. HAS clients choose the most appropriate bitrate each time before sending the request and this property of HAS enables the clients to cope with scarce and varying network bandwidth. The fact that each client chooses its own video quality makes HAS a scalable solution. Moreover, HAS also makes use of firewall and NAT traversal abilities of TCP. Using standard caches, server and clients makes HAS an easier solution when it is compared to other streaming solutions. Many commercial tools such as [4], [5] and [6] started using HAS as their streaming enabler.

In previous researches, a number of problems of HAS are revealed. One important problem of HAS is unfairness which means huge video quality differences between HAS clients. One other problem is instability which is caused by frequent video bitrate changes. These issues degrade client Quality of Experience (QoE) severely. Other problems are due to live streaming. Some researches reveal that the number of clients that HAS servers can support is fewer than expected. This is a serious problem for a live TV architecture where the number of clients are generally high. Principles of HAS and its related problems are given in detail in 2.3.

In this thesis, we propose two architectures that operate using HAS. The first architecture is FEAST which is used for Video on Demand services. FEAST aims to give a fair, stable and efficient service by using feedback information from server. There are three feedback information that are used by clients to adapt their video bitrates which are number of clients, average video bitrates of the clients and average bandwidth measurements of the clients. When the clients adapt their video bitrates by considering their own video bitrates and the feedback information, the unfairness of HAS can be reduced to minimal levels. We performed extensive simulations on NS2 [12] to evaluate the performance of FEAST. From the simulations, it is shown that FEAST outperforms other HAS solutions in terms of fairness and stability while efficiently utilizing the available bandwidth. We also performed experiments to investigate the effects of background traffic, number of clients and number of video resolutions on the performance of FEAST.

Our second architecture is ALIVE for live TV applications. The aim of ALIVE is to support more live TV clients with higher QoE by decreasing the load of the server. The basic principle of ALIVE to enable the clients to download the contents from nearby clients instead of the server whenever it is possible. We used Scalable Video Coding [13] in ALIVE which enables the clients to adapt their video bitrates even when they download from other clients. ALIVE can be considered a combination of HAS, SVC and Peer-to-Peer (P2P) systems. We performed extensive simulations to evaluate the performance of ALIVE. We show that the load of the server decreases significantly and more clients can use ALIVE with respect to other HAS algorithms with a better QoE.

The remainder of the thesis is outlined as follows. In Chapter 2, we will explain the solutions of IP multimedia networks and Scalable Video Coding. HTTP Adaptive Streaming will be defined in detail in the same chapter. In Chapter 3, we propose FEedback based Adaptive STreaming (FEAST) and evaluate the performance of FEAST. The details of Adaptive LIVE

Streaming over HTTP (ALIVE) and its performance evaluation are given in Chapter 4. Finally in Chapter 5, we conclude the thesis and future work of this thesis is explained.

CHAPTER 2

BACKGROUND

2.1 Internet Multimedia

As the multimedia traffic increases, researchers propose methods to cope with this increasing traffic demand. Especially for live video streaming, Real-Time Transport Protocol (RTP), IP Multicast and IP Multimedia Subsystem are some of these solutions.

Real-Time Transport Protocol (RTP) [8] is a UDP based protocol. It is extensively used in entertainment systems such as multimedia streaming. It works in conjunction with RTP Control Protocol (RTCP). RTCP [14] provides feedback information related to QoS and network conditions. Due to the nature of UDP, lighter traffic is generated by RTP which can increase QoS. Disadvantage of RTP is due to UDP's being blocked by firewalls and inability to traverse across Network Address Translators. Moreover, it is a complex structure. It is not cachable in Content Distribution Networks (CDN) and it is difficult to use RTP with CDNs. The server also requires to manage seperate streaming sessions for each client which makes RTP an unscalable solution.

IP Multicast [10] is a multicast technology based on sending the datagrams to a group of a client instead of a single client. This technology significantly decreases the transmission of duplicate data. Higher QoS can be provided by eliminating redundant data transfer. A sender uses the multicast address instead of a single client address. Internet Group Management Protocol (IGMP) [15] is the enabler protocol for grouping the receivers and manage the multicast addressing. Multicast distribution is done by tree structure. Fig 2.1 shows a typical IP multicast structure. In the figure, the arrows denote the multicast tree. Although IP Multicast looks as a promising solution for multimedia streaming, it requires all the routers between the sender and the receivers to support multicast routing. Therefore, it is difficult to deploy IP multicast. Furthermore, routing tables grow dramatically with the use of IP multicast which makes it an unscalable solution.

IP Multimedia Subsystem (IMS) is another system proposed for delivering multimedia services over IP in next generation networks. It is designed by 3rd Generation Partnership Project (3GPP). In order to be competible with existing systems, IMS makes use of IETF protocols wherever it is possible. For example, Session Initiation Protocol (SIP) is used which is a

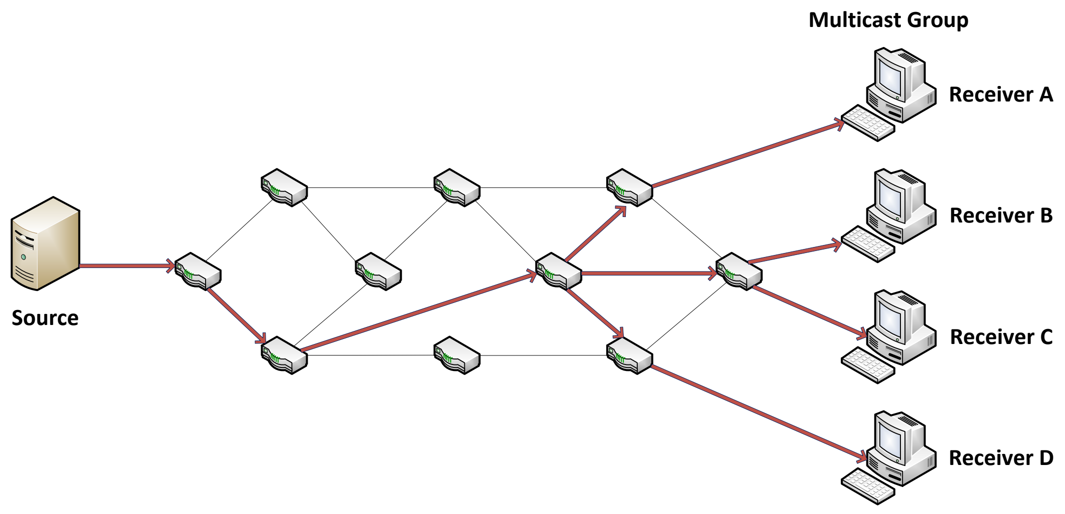

Figure 2.1: IP multicast tree structure

signalling protocol [16]. IMS is a platform that tries to form a fixed-mobile convergence for wired and wireless terminals. The problem with IMS is its complexity and traffic overhead due to control message. In [17], it is indicated that IMS based systems generate 50 to 75 percent more control signal messages than non-IMS systems. Therefore, scalability problems arise with the usage of IMS.

Table 2.1 is showing the advantages and disadvantages of RTP, IP multicast and IMS.

Table 2.1: Comparison of existing solutions for Internet Multimedia

	Advantages	Disadvantages
RTP	Network aware by RTCP Lighter Traffic	Not cachable in CDNs Firewall and NAT Traversal Problems Stateful Servers
IP Multicast	Avoids Redundant Traffic Higher QoS	Deployment Problem Larger Routing Tables
IMS	Aimed for NGN Fixed-Mobile Convergence	Complex Architecture High amount of Control Packets

2.2 Scalable Video Coding (SVC)

Beyond the networking protocols, important progresses are done in other video related fields. Video compression is an important piece of multimedia networking. It is the process of reducing the amount of original data by coding the information. It significantly decreases the traffic on the network. An HD video has a bitrate close to 1.3 Gb/s [18]. It is impossible for a network to carry such traffic. Therefore, it is necessary to compress these videos into bitrates such that it becomes suitable for multimedia networking.

A compression is based on eliminating the redundant information. The redundancy can be classified into two as *spatial redundancy* and *temporal redundancy*. The former denotes the similarity between neighbouring pixels and the latter measures the similarity between adjacent video frames. Compression is achieved by exploiting the spatial and temporal redundancy in the video.

Video compression techniques can be classified into groups according to the ability of conserving the original data. A compression technique is said to be *lossless* if no data is lost after decompressing the compressed video. *Lossy* compression is the opposite of lossless compression which loses some data during compression. MPEG4 is a lossless compression method whereas MPEG2 is a lossy compression which does not guarantee the full decoding of original data.

Another video compression classification can be done as constant bitrate (CBR) coding or variable bitrate (VBR) coding. CBR coding encodes the whole video with the same rate which may not guarantee a good quality because of video content. A frequent changing background of the video may cause a worse Peak Signal-to-Noise Ratio (PSNR). On the other hand, VBR coding keeps the quality constant by changing the encoding rate whenever it is necessary. However, it needs a more complex decoding and it may exceed the available network bandwidth during streaming.

H.264/AVC video coding is one of the most commonly used video compression techniques today. It is approved by ITU-T as Recommendation H.264 and by ISO/IEC as International Standard 14 496-10 Advanced Video Coding (AVC) [19]. It is created to supply a good video quality by lowering the resultant bitrates than previous standards such as MPEG1, MPEG2, H.263 etc.

H.264/SVC [13] is the extension of H.264/AVC which codes the videos by layers. In other words, a base layer is coded. Then by adding enhancements layers, the quality of the video is increased. When a server contains a video segment with resolution r_n, it automatically contains, r_1, r_2, ... r_{n-1} where r_1 is the lowest quality. We used this property in ALIVE which will be proposed in Chapter 4.

2.3 HTTP Adaptive Streaming

HTTP Adaptive Streaming (HAS) is a recent multimedia streaming technology that is aimed for high QoE against network fluctuations. There are some reasons for the popularity of HAS. First of all, it can be used by standard HTTP servers, caches and clients without changing the existing network infrastructure. The wide usage of HTTP ([20]) contributes to popularity of

HAS. Moreover, the underlying transport layer for HTTP is TCP which enables traversing firewalls and Network Address Translators as opposed to UDP which is mainly used as the transport layer protocol in standard video streaming applications such as RTP. In UDP based protocols, servers just push the content towards the clients, however it would generate problems if the clients are behind NAT because UDP is not capable to traverse across it. In HAS, the server offers different levels of quality for a given video and the client selects the most appropriate video quality resulting in a scalable receiver-driven approach. Hence, HAS is gaining popularity as a promising technology.

In HTTP Adaptive Streaming, servers encode the videos into multiple quality levels and divide these videos into small segments of equal lengths. The length of these segments is generally 1,2 or 10 seconds ([21]). A client first gets the Media Presentation Description file (MPD) which consists the URL,timing information, available resolutions etc. Fig 2.2 shows an example MPD file.

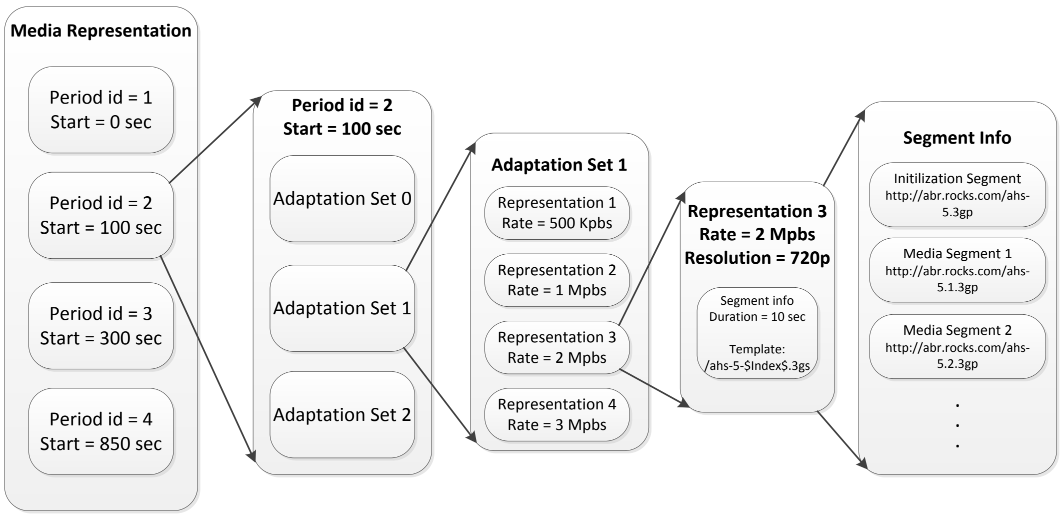

Figure 2.2: Illustration of Media Presentation Description File

After getting MPD, client sends a request to download the available segment to server following a *rate adaptation* scheme. This process is shown in Fig 2.3. In HAS, each client determines the video quality of the next segment to be requested according to its measurements such as download time or current buffer state. With this approach, HAS becomes robust against the scarcity and fluctuations in the network bandwidth.

One important feature of HAS is *idle waiting time*. A client sends the next request just after the end of previous download if its buffer contain less than a certain amount of data. However, if its buffer exceeds a threshold, then next request will be send for some time later. This time is called idle waiting time. Although this is necessary to avoid buffer overflows, in next sections it will be shown that idle waiting time will cause some unfairness issues.

HAS is different than traditional TCP streaming. In TCP streaming, the data that is down-

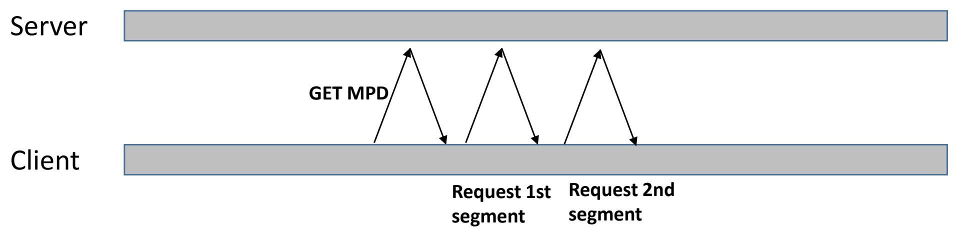

Figure 2.3: A client first downloads MPD and then downloads the successive segments

loaded after each round becomes ready to play immediately after that round finishes. A round starts with the transmission of W packets, where W is the current size of the TCP congestion window [22]. Whereas, in HAS, the data becomes ready to play after the segment will completely downloaded and delivered to application layer. Moreover, TCP streaming does not adapt the video bitrate, but HAS changes it dynamically.

MPEG-DASH [23] is the first international standard for HAS. MPD file format is defined which is an XML based schema that contains information about the video data. In [23], some additional features of MPEG-DASH are written as ad insertion, scalable video coding, compact manifest, multiple base URLs and segments with variable durations. MPEG DASH defines the quality metrics for reporting the session experience. The standard has a set of well-defined quality metrics for the client to measure and report back to a reporting server [24]. MPEG-DASH also supports Scalable Video Coding (SVC) and Multiview Video Coding (MVC). The MPD gives the information about the decoding dependencies between representations, which can be used for streaming any layered coded streams like SVC and MVC.

Many commercial tools such as Microsoft Smooth Streaming [5], Move Networks [4], Apple HTTP live Streaming [6] have adopted HAS with different rate adaptation implementations. They both offer Video on Demand and live tv services. Microsoft Smooth Streaming (MSS) is the only open source HAS player and many researchers compare their own work with MSS. We will also compare our work with MSS. Therefore, we will give the details of MSS in Section 2.3.1.

In the literature, there are some works about HTTP Adaptive Streaming. Many works deal with the rate adaptation algorithms. In [25], authors propose a rate adaptation algorithm that defines the video bitrate according to segment download time. This article is one of the earliest works in this field and many other papers compare their own work with the one in this article. We will also compare our proposed algorithms with this one, therefore we will explain the algorithm of this paper later in Section 2.3.2.

In [26], authors propose a rate adaptation algorithm that define states according to the fullness of the buffer. Then, combine these states with download times, then determine the resultant bitrates.

In some previous works such as [21] and [27], *unfairness* problem is observed for commercial HAS tools. This unfairness means the huge differences between the video qualities of HAS clients. It becomes unfair if one of the clients watch the videos in a high quality whereas another client watches it with the lowest quality. Authors in [28] propose an HAS solution to cope with unfairness, instability and low utilization. They add randomness to their rate adaptation to avoid these situations. Although their solution decreases the unfairness, it may not be useful when number of clients is high. In [29], a control theoretical rate adaptation algorithm is proposed, where the server determines the rates of the clients to achieve a fair service and efficient use of the available bandwidth. However, such server centric approach suffers from scalability.

In [27], authors examine the performance of MSS for the case of concurrent clients accessing videos in the same home. They suggest home gateway which works as a bandwidth manager and divides the bandwidth according to the target bitrates of the clients. This approach only ensures fair bandwidth division between the clients in the same home and it necessitates the deployment of these gateways in each home.

In [30], the advantages of IP multicast and HAS are combined. They divide the connection into two as UDP and HTTP part. In UDP part, content is sent with IP multicast. However, firewall traversal issue and IP multicast deployment issue makes UDP part less efficient.

Parallel HTTP connections are proposed in [31]. In the article, successive video segments are downloaded with more than one HTTP connections which increases the throughput and decreases the download delay. However, the system is only tested for single HAS client. In fact, it can be a disadvantage for multiple HAS clients due to high number of TCP and HTTP connections. Moreover, as the number of clients increases, the throughput cannot be further increased by increasing the number of connections as indicated in [32].

Authors investigate the effects of Scalable Video Coding on HAS in [33]. It is written that cache hit ratio increases with the usage of SVC. When cache contains a high quality version of a video, it automatically contains the lower quality version of the video because of layered coding. This ability of SVC will be used in ALIVE which will be proposed in Chapter 4 for live TV services.

Video bitrate adaptation is performed not only in HAS but also in some other systems. In the literature, there are some previous works that proposed adaptive streaming over other protocols rather than HTTP. The biggest difference between HAS and the existing works is that in HAS clients adapt the video bitrates whereas in other systems server is responsible from determining the video quality for each client. This feature makes HAS a much more scalable solution than existing solutions. For example, in [34], authors propose a rate adaptive video streaming solution which works with RTP over UDP. In this system, server determines the video bitrates. In [35], [36] and [37], authors propose sender-driven rate adaptation strategies based on layered encoded videos such as SVC. Although these systems consider the network's and clients' instant conditions, the fact that sender decides the video bitrates makes them infeasible solutions especially for high number of clients.

In Chapter 4, we propose ALIVE which can be thought as a combination of SVC, HAS and P2P systems. In the literature, adaptive streaming for P2P systems was proposed such as in [38]. In the article, authors propose a window based rate adaptation algorithm. However, ALIVE will be different from previous works because it will not be a pure P2P system and only some clients will supply videos to other clients. Moreover, ALIVE will use HTTP as the primary networking protocol different than previous works.

2.3.1 Microsoft Smooth Streaming

MSS is a commonly used open source HAS solution. In this section, we define the rate adaptation algorithm of MSS by referring to [39] which provides the operation details of MSS.

In most of the rate adaptation algorithms, clients measure the download time of each segment. Then, they calculate the available bandwidth by dividing the size of the downloaded video segment to the download time. These bandwidth measurements are averaged using a multi-sample sliding window to smooth the estimate in order not to react to momentarily bandwidth changes. In MSS, clients get the arithmetic mean of last 3 measurements. In the rest of this thesis, we will denote the measured bandwidth of client i with $b_i(t)$.

The maximum buffer level is indicated to be 20 seconds. Clients send the next request just after the last download finishes if the buffer contains less than 20 seconds of video data. If it is equal to 20 seconds, the client send the next request two seconds later which is the *idle waiting time* for MSS. Segment time of MSS is denoted to be 2 seconds.

In MSS, a client is either in Buffering State or Steady State. It starts with the lowest bitrate in buffering state. It increments the buffer as fast as possible while keeping the video bitrate high according to the measured bandwidth. It quits the buffering state when its buffer contains 14.5 seconds of video data. In steady state, it increases the video bitrate only if the buffer contains more than 17 seconds of data and the measured bandwidth is higher than the bitrate of next higher resolution. It starts decreasing the video bitrate when the buffer drops below 14 seconds. It re-enters the buffering state again when buffer contains less than 7 seconds of data and video bitrate is set to lowest. The aim of the rate adaptation algorithm is to put the buffer between 12 and 17 seconds while maximizing the video bitrate.

2.3.2 Rate Adaptation for Adaptive HTTP Streaming: Liu at al.

The article [25] is one of the earliest works in HAS. It proposes a rate adaptation algorithm according to the segment download time. A client starts with the lowest quality video. They define Segment Fetch Time (SFT) as the time between the time instant of sending GET request and receiving the last bit of the video segment. After each download, the clients calculate μ.

$$\mu = \frac{MSD}{SFT} \tag{2.1}$$

In Equation 2.1, MSD denotes the Media Segment Duration. According to the value of μ, next segment's video bitrate is calculated. If μ is greater than a threshold, then video bitrate is increased by one step.

$$\mu > 1 + \epsilon \tag{2.2}$$

If the above equation is satisfied, then next video's resolution is one higher than current resolution.

$$\epsilon = \max_{i=0}^{n-1} \frac{br_{i+1} - br_i}{br_i} \tag{2.3}$$

where br_i denotes the encoded media bitrate of representation i and n denotes the highest representation level. The value of ϵ becomes critical when there are few HAS clients. Sometimes, clients cannot increase their bitrates eventhough there is available bandwidth which will be shown in detail in Section 3.4.

The algorithm decreases the video bitrate if μ is less than another threshold.

$$\mu < \gamma_d \tag{2.4}$$

If the above condition is satisfied, then the client must choose a lower bitrate. In the simulations of the paper. γ_d is selected to be 0.67. A higher value of γ_d makes the a safer rate adaptation by decreasing the video bitrate even in smaller bandwidth reduction whereas a lower value lets the clients to decrease the video quality in much worse conditions only. In such situations client will choose the maximum br_i such that;

$$br_i < \mu * br_c \tag{2.5}$$

where br_c is the bitrate of current representation.

In the article, authors also include the *idle waiting time*. Following equation determines the idle waiting time.

$$t_s = t_m - t_{min} - \frac{br_c}{br_{min}} MSD \tag{2.6}$$

where t_s is idle waiting time, t_m is buffered media time, t_{min} is predefined buffered media time and br_{min} is the minimum bitrate representation. If t_s is lower than 0, then next request will be sent immediately after the last download finishes. If it is greater than 0, then the client will wait for t_s seconds before requesting the next video segment.

CHAPTER 3

FEAST: FEEDBACK BASED ADAPTIVE STREAMING OVER HTTP

3.1 Motivation for FEAST

HTTP runs over TCP which determines the transport rate of a connection by Additive Increase Multiple Decrease (AIMD) scheme. When multiple TCP connections share the same bottleneck link, the bandwidth is divided fairly between these connections due to nature of TCP. Therefore, it is reasonable to expect that when multiple HTTP Adaptive Streaming (HAS) users share a bottleneck link, they watch their videos in similar bitrates. However, this case is not true in some conditions. In [21], authors evaluate the scenario with multiple HAS users sharing the same link. In the article, it is shown that some users are watching the videos in much higher quality than other users. Same behavior is observed in [27]. The reason for this unfairness is investigated in [21] and the authors concluded that the main reason is ON-OFF behavior of HAS clients.

One of the important issues for HAS is *idle waiting time*. After filling its buffer to a certain level, a client has to wait for a certain period of time before its next request in order to prevent buffer overflow. For example MSS waits for two seconds before next segment's download if buffer contains 20 seconds of video data. This ON-OFF behavior affects the bandwidth estimation of the clients. Clients measure the segment download times and estimate the available bandwidth in the application layer. However, this estimation might be wrong due to overlapping download or idle waiting times with other HAS clients. Moreover, a user cannot estimate the bandwidth during the OFF period because it is not downloading. Therefore, it may not react to bandwidth fluctuations which occurs during OFF period. In other words, the bandwidth information may be stale because of some fluctuations in network conditions. Therefore, HTTP Adaptive Streaming is prone to unfairness problem although it is working on a fair transport layer protocol.

Fig. 3.1, 3.2 and 3.3 are showing three different cases for ON-OFF periods of HAS clients. Note that, normally download times are not slotted as in this example. It is just to show the root of unfairness problem. Assume that capacity of the shared link bandwidth is C. In normal conditions, if two TCP connections are sharing the same link, it is expected that both will use

$C/2$ portion of the bandwidth. If we analyze the first case, two clients are downloading and waiting synchronously. Therefore, they are measuring the network such that both of them use the half of the bandwidth. No unfairness will be encountered in this condition.

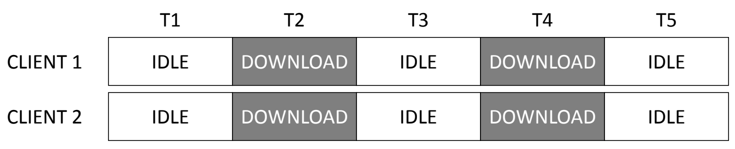

Figure 3.1: ON-OFF periods of two HAS clients: Case1

Fig. 3.2, is showing another case for two HAS clients. In this time chart, Client 2 is downloading a segment during T1, T2 and T3 whereas Client 1 only downloads in T2. First client will estimate the available bandwidth as $C/2$ because two clients are concurrently downloading in its active period. On the other hand, Client 2 will estimate the bandwidth as more than $C/2$ and most probably close to C because it is the only active client in most of its download time.

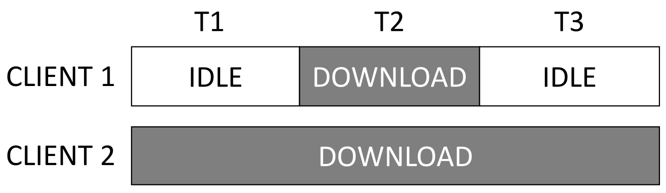

Figure 3.2: ON-OFF periods of two HAS clients: Case2

Finally, Fig. 3.2 is showing the case where another potential unfairness problem will occur. In this condition, only Client 1 is downloading in T2 whereas two clients are downloading in T1. Therefore, Client 1 will measure the available bandwidth as C and other clients will measure it as $C/2$. Here, we are facing an unfairness.

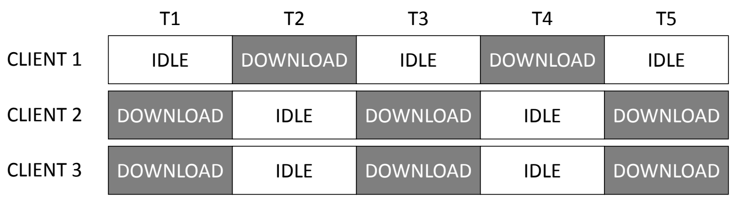

Figure 3.3: ON-OFF periods of three HAS clients

This problem can be avoided by removing *idle waiting times*. If clients always request the next segment just after the end of previous one's download, then unfairness problem will be solved. However, client buffers will contain enormous amount of data which consists of hundreds of seconds of video. These data will be wasted if user stops watching the video some time later. Moreover, it will be unsuitable for live tv applications in which video data is instantaneously prepared in the server. As a result, it is necessary for clients to wait for some time for successful operation of HAS.

Stability is also another important factor for HAS. If a client changes the video bitrate very frequently, user QoE will degrade seriously as indicated in [40]. Another circumstance that degrades QoE is sudden bitrate changes. In other words, users' satisfaction will decrease if video bitrate changes from a very high level to low level or vice versa. Authors discuss this problem in [41]. Therefore, less frequent and step-wise changes in video bitrate is necessary for HAS. Step-wise changes mean changing the video quality to one higher or one lower quality level.

3.2 Performance Metrics

In this Section, we define the performance metrics that measure the fairness and stability properties of HAS that we introduce previously. The *system* is defined as the video server and its clients that are downloading videos.

Fairness:
If one of the clients watches the video with a low quality whereas another client watches in a high quality, the bandwidth is shared unfairly between them. To this end, we first introduce the fairness index introduced by Jain [42] as follows:

$$JF(t) = \frac{(\sum_{i=1}^{u(t)} r_i(t))^2}{u(t)\sum_{i=1}^{u(t)}(r_i(t))^2} \tag{3.1}$$

In Eqn. (3.1), $u(t)$ is the number of connected users to the server and $r_i(t)$ is the video bitrate of user i at time t. Normally the value of this fairness index resides between $1/n$ and 1 where n is the number of users. Desired value of Jain's fairness value is 1 whereas it becomes unfair when it reaches to $1/n$. If it is equal to 0.2, then it is unfair for 80 percent of the users.

In this thesis, we use unfairness as $U(t) = 1 - JF(t)$ which means a system is perfectly fair when $U = 0$.

Switch Rate:
As denoted before, users are sensitive to frequent changes in the downloaded video rate. Accordingly we define the Switch Rate for the system SR in number of rate changes per second. It is computed for the sum of all clients' rate switches. A low value is desired for SR. In this thesis, we assume that average SR is computed over a time interval while it can be defined as an instantaneous measure as $SR(t)$.

Bandwidth efficiency and Video Quality:

A good HAS player should minimize $U(t)$ and SR while using the available bandwidth as efficient as possible. The bandwidth is used efficiently if the available bandwidth in the network is utilized by the clients for the transmission of videos with the highest possible quality.

We define *video quality* as the sum of instantaneous video bitrates of the clients. In [32], throughput/link capacity ratio of multiple parallel TCP connections on a shared link is found to approach to unity as the number of connection increase. The ratio is generally higher than 0.9 for 3 connections and higher than 0.95 for 6 or more connections. Therefore, for a good HAS application, the *video bitrate* should be close to the available link bandwidth.

The best way to achieve desired values for the above metrics is to equally divide the available bandwidth to all HAS users. However, it is generally not possible for users to exactly know the instantaneous available bandwidth.

To this end, we investigated the relation between the estimated bandwidth and video bitrates, with an experiment. We simulated MSS in NS2 with 9 HAS clients that share a common link with a bandwidth of 9 Mb/s. We calculated the average video bitrates and average estimated bandwidths of all users and plotted their values on the same graph with respect to time in Fig. 3.4. Average video bitrate $r_a(t)$ is the instantaneous video bitrate averate of 9 clients which is the average of $r_i(t)$ for i=1,2..,9. Average estimated bandwidth $b_a(t)$ is the instantaneous estimated bandwidth of 9 clients which is the average of $b_i(t)$ for i=1,2..,9. Only steady-state period is shown on the graph.

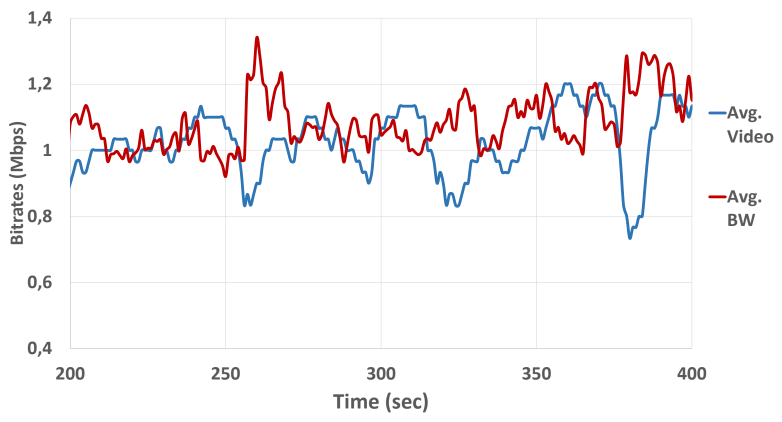

Figure 3.4: Average video bitrates and measured bandwidth of the clients

It is seen in Fig. 3.4 that average video bitrates and average estimated bandwidths continuously intersect. Accordingly, we deduce that, when users download more than the available bandwidth, download times get higher and the amount of data in their buffers decreases by time. Consequently, they decrease their video bitrates. After filling their buffers, they again increase their bitrates and exceed the available bandwidth. This cycle continues resulting in instability and unfairness. To avoid this problem, we propose a parameter $\rho(t)$ which is the ratio of average video bitrate of the users to average estimated bandwidth. Users will determine their video bitrates by considering this parameter.

3.3 Operation of FEAST

FEAST is proposed for achieving fairness and stability while using the available bandwidth as efficient as possible. In FEAST, the clients adapt their video rates based on the feedback information about the network state from the server and additional parameters that they can individually measure. We defined the segment lengths as 2 seconds which is the same for MSS. As the segment size increases (e.g. 10 sec), idle waiting time increases. Therefore unfairness increases and clients cannot react to bandwidth changes properly. In [43], it is writen that the additional overhead traffic increases when segment size decreases. Although it can be useful for live streaming to make the segment length 1 second, 31 Kbps traffic is introduced in every segment.

3.3.1 Client-Server Information Exchange and Server Side Computations

The request packets of the clients in our proposed FEAST contain the name of the requested video and the start time of the requested segment as in most HAS implementations. In addition, the request of each client i at time t includes the following information:

$r_i(t)$: Video bitrate of the currently requested segment

$pr_i(t)$: Video bitrate of the last requested segment

$b_i(t)$: Estimated bandwidth of client i before the current request.

$pb_i(t)$: Estimated bandwidth of client i before the last request.

Each client i measures its estimated bandwidth as in MSS as described in Section 2.3.1. A client sends 0 as $r_i(t)$ and $b_i(t)$ when it is leaving and sends 0 as $pr_i(t)$ and $pb_i(t)$ when it is connecting to the server first time.

The server maintains three system state variables. These are the average video bitrate of the clients, the average available bandwidth of the clients as measured on the client side and the number of connected clients which are denoted by $r_a(t)$, $b_a(t)$ and $u(t)$ respectively. For each request received from client i, the server updates these system state variables as shown in Algorithm 1.

Algorithm 1 Server's Equations

```
1: if new client is connecting then
2:     $r_a(t) = \frac{r_a(t)*u(t)+r_i(t)}{u(t)+1}$
3:     $b_a(t) = \frac{b_a(t)*u(t)+b_i(t)}{u(t)+1}$
4:     $u(t) = u(t) + 1$
5: else if a client is leaving then
6:     $r_a(t) = \frac{r_a(t)*u(t)-pr_i(t)}{u(t)-1}$
7:     $b_a(t) = \frac{b_a(t)*u(t)-pb_i(t)}{u(t)-1}$
8:     $u(t) = u(t) - 1$
9: else
10:    $r_a(t) = r_a(t) + \frac{r_i(t)-pr_i(t)}{u(t)}$
11:    $b_a(t) = b_a(t) + \frac{b_i(t)-pb_i(t)}{u(t)}$
12: end if
```

When server responds to a client request, it piggybacks $r_a(t)$, $b_a(t)$ and $u(t)$ with the requested video segment. Note that the server side computations are scalable as system state variables do not depend on the individual client. Server only tells the network condition to the clients. However, the rate adaptation process will be done individually at each client.

3.3.2 Rate Adaptation on the Client Side

Client i computes two state variables $C(t)$ and $F_i(t)$ which reflect the network congestion and fairness of video download according to the system state variables $r_a(t)$, $b_a(t)$ and $u(t)$ received from the server and $r_i(t)$. The FEAST rate adaptation of client i aims to decrease both SR and $U(t)$ according to the client state $S_i(t) = (C(t), F_i(t))$. It is important to note that the client buffer has to be full above a threshold in order to avoid video freezing and the client buffer size buf_i must be checked for each rate adaptation. The buffer size is measured in seconds as in [39]. Similar to MSS we choose the maximum buffer size as 20 seconds.

Initially, user tries to fill its buffer as much as possible. It starts with the lowest bitrate in order to achieve this. In this phase, buffer state is called to be *insufficient*. The user leaves this state and enters *enough* state when the buffer contains 12 seconds of video data. The user again enters the *insufficient* state when its buffer data drops below 8 seconds of data. The reason for choosing different values for entering and leaving the state is to avoid oscillation between states.

Accordingly client i selects $r_i(t)$ before each request to be sent at time t as shown in Algorithm 2 where $p_r^-(t)$ is the next lower and $p_r^+(t)$ is the next higher video bit rate with respect to the previous video bitrate $p_r(t)$.

We define decreasing the video bitrate as choosing the the next lower quality level whereas increasing the video bitrate means choosing the next higher quality level.

Algorithm 2 Rate Adaptation Algorithm

```
1: Initially: $r_i(t)$ is selected as the lowest bitrate
2: Initially: Buffer state $bs_i(t)$ is $insufficient$
3: For Each Request:
4: if $bs_i(t)$ is empty AND $buf_i(t) \geq 12$ then
5:     $bs_i(t) = enough$
6: else if $bs_i(t)$ is enough AND $buf_i(t) \leq 8$ then
7:     $bs_i(t) = insufficient$
8: end if
9: After computing $bs_i(t)$:
10: if $bs_i(t) = insufficient$ then
11:     $r_i(t) = pr_i^-(t)$
12: else
13:     Compute $C(t)$ and $F_i(t)$
14:     Increase of decrease $r_i(t)$ accordingly
15: end if
```

Computing $C(t)$:

As shown in Fig. 3.4, MSS clients increase their video bitrates when $b_a(t)$ is higher than $r_a(t)$. However, when $r_a(t)$ becomes greater, then video bitrates start falling down. This repeating behavior results in a large number of rate switches. In FEAST, the client takes into account the congestion state of the network to avoid this problem. To this end, we define $\rho(t) = \frac{r_a(t)}{b_a(t)}$ and introduce the network congestion parameter $C(t)$ that is computed at the client side as shown in Algorithm 3.

Algorithm 3 Network Congestion

```
1: if $\rho(t) < \alpha$ then
2:     $C(t) = 0$
3: else if $\alpha \leq \rho(t) \leq \beta$ then
4:     $C(t) = 1$
5: else if $\beta < \rho(t)$ then
6:     $C(t) = 2$
7: end if
```

If $C(t) = 0$, it means network is underutilized and video bitrates can be further increased. If $C(t) = 2$, then there exists a congestion in the network. $C(t)$=1 is the desired state in order to avoid both underutilization and congestion. α and β are the congestion threshold values. β can be close to unity whereas α can be a value between zero and one. We describe the selection of α and β later in this Section.

Computing $F_i(t)$:

$F_i(t)$ aims to bring $r_i(t)$ close to the average download rate $r_a(t)$ provided by the server as shown in Algorithm 4.

Algorithm 4 Fairness Variable

```
1: if $pr_i^+(t) < r_a(t)$ then
2:     $F_i(t) = 0$
3: else if $pr_i^-(t) \leq r_a(t) \leq pr_i^+(t)$ then
4:     $F_i(t) = 1$
5: else if $r_a(t) < pr_i^-(t)$ then
6:     $F_i(t) = 2$
7: end if
```

On the one hand, if $F_i(t) = 0$, then client i's video bitrate is lower according to the average and it should increase its bitrate. On the other hand, if $F_i(t)$=2 indicates that the client is unfairly downloading at a high bit rate possibly resulting in a decreased QoE for other clients. Hence, clients aim to keep $F_i(t) = 1$.

Rate Decision according to $S_i(t) = (C, F_i(t))$:

A user can be in nine different states according to the values of $C(t)$ and $F_i(t)$. We define different rate selection decisions for each state aiming to reach $S_i(t) = (1, 1)$ for all clients i as seen in Table 3.1. In this table, I means increase, D means decrease, NC means no change, PI means probabilistic increase and PD means probabilistic decrease.

Table 3.1: Rate Decision

Client State	$F_i(t) = 0$	$F_i(t) = 1$	$F_i(t) = 2$
$C(t) = 0$	I	PI	NC
$C(t) = 1$	I	NC	D
$C(t) = 2$	NC	PD	D

When $F_i(t)$ is zero for a client, it increases its video bitrate if there is no congestion in the network. It does not change its bitrate if congestion exists in the network and waits for other clients to decrease their video rates. A client with $F_i(t)$=1 does not change it video bitrate if $C(t)$=1 which is the desired condition. According to congestion status, it increases or decreases with p_i and p_d respectively. These probabilities may depend on the number of connected clients, its video bitrates or other parameters. In FEAST, p_i decreases with the increasing number of connected clients whereas p_d increases for the same situation. Equations 3.2 and 3.3 gives the equations for these two variables. With the contribution of $u(t)$, these variables also become time dependent.

$$p_i(t) = \frac{1}{u(t)} \tag{3.2}$$

$$p_d(t) = 1 - \frac{1}{u(t)} \tag{3.3}$$

If $F_i(t) = 2$ for a client, it decreases the video bitrate if $C(t) = 1$ or $C(t) = 2$. If the network is underutilized, it does not change the bitrate and waits for other clients to increase their video bitrates.

Selecting the congestion thresholds, α and β:

α and β are threshold values to determine $C(t)$ as described in Algorithm 3. In our experiments, we see that choosing β close to unity provides a reliable detection of congestion. Choosing α as a value between 0.7 and 0.8 works well when the following inequality is satisfied;

$$u(t) * r^{max} \geq bw_{avail} \tag{3.4}$$

where r^{max} is the maximum video bitrate and bw_{avail} is the actual available bandwidth. However, the inequality may not hold when the number of connected clients is low. In this condition, the clients suppose that the network bandwidth is fully utilized although an underutilization exists. This condition can be avoided by increasing the value of α with the low number of users. In other words, α should be a function of $u(t)$. For higher values of $u(t)$ (i.e $u(t) > 5$), α can be set to 0.65, otherwise we calculate it by the following formula;

$$\alpha(t) = 0.65 + 0.25 * exp(-3 * u(t)) \tag{3.5}$$

It is important to note that this is a heuristic formula. Exponential term and the exponent of -3 is found by experimentation. With a $u(t)$-dependent value of α, FEAST can work with any number of connected devices. After deciding its own video bitrate, a client sends a request to get the segment of 2 seconds of video.

3.4 Performance Evaluation of FEAST

We evaluate the performance of FEAST with respect to the performance metrics that are defined in Section 3.2. We used NS2 [12] as the simulation platform. We simulated the behavior of FEAST, MSS and the algorithm presented in Liu at al. [25] and compare their performances. We compared the results for different network conditions.

In [39], video bitrates of MSS are between 0.3 Mb/s and 2.436 Mb/s. Therefore, we will use similar video bitrates in our experiments. There will be eight different video bitrates between 0.3 and 2.4 Mb/s with an increase step size of 0.3 Mb/s. In FEAST, not all the users have to watch the same video. They may watch different videos, but we assume that coding rate levels of the videos are same. In other words, all the videos are coded into multiple bitrates from 0.3 Mbps to 2.4 Mbps.

In the experiments, we use the topology given in Fig. 3.5.

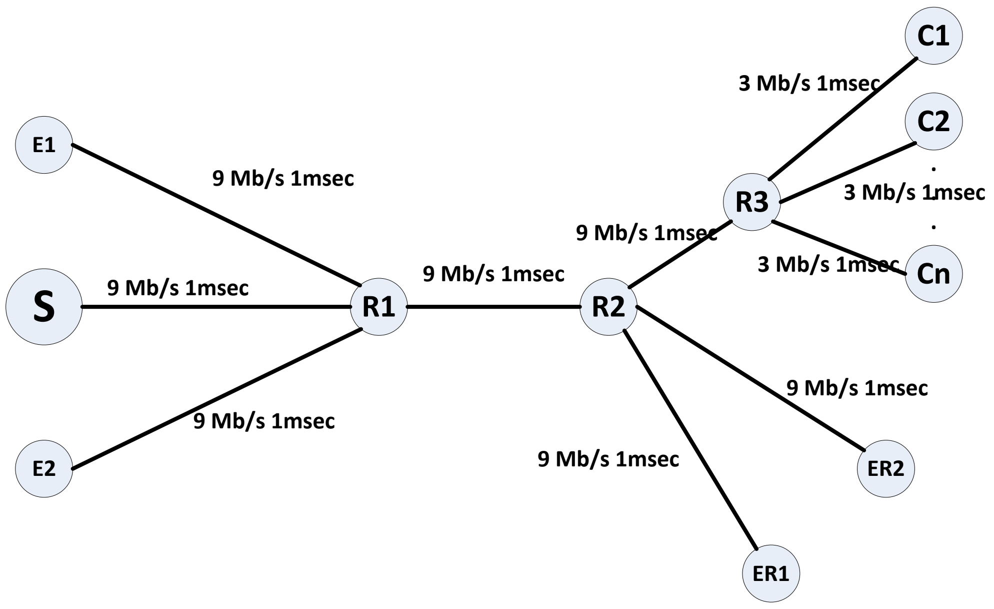

Figure 3.5: The network topology that is used in simulations

In Fig. 3.5, n clients (C1, C2 .., Cn) are downloading a video from the server S. In the experiment, we also generated background traffic. Nodes E1 and E2 are generating exponential traffic towards ER1 and ER2 respectively. The links from clients to router R3 have a bandwidth of 3 Mb/s each and all other links have 9 Mb/s bandwidth in the network. In the simulations, all the clients start downloading at a random time instant between t=0 and 20 sec. They quit watching at a random time between t=480 and 500 sec.

We compare our results in terms of video bitrates, unfairness and the rate of video rate switches. The same start and quit times are used for FEAST, MSS and Liu at al..

The first experiment is conducted with 9 clients and without background traffic. Video segment size is set to 2 seconds in all of the experiments. Unfairness, rate switching and average video bitrates of 9 clients are measured. Fig. 3.6 is showing a sample of the instantaneous unfairness of 9 clients in the steady-state period.

It is clearly seen from Fig. 3.6 that FEAST provides significantly more fair service. We also investigated the average video bitrates of the users. In [32], it is shown that when more than 6 TCP connections share the same link, total TCP throughput will approach to the capacity of the link. Therefore, it is reasonable to expect that TCP throughput of our clients will be close to 1 Mb/s. There will be also some overheads between transport layer and application layer, so HTTP throughput will be less than but close to 1 Mb/s. Fig. 3.7 is showing the average

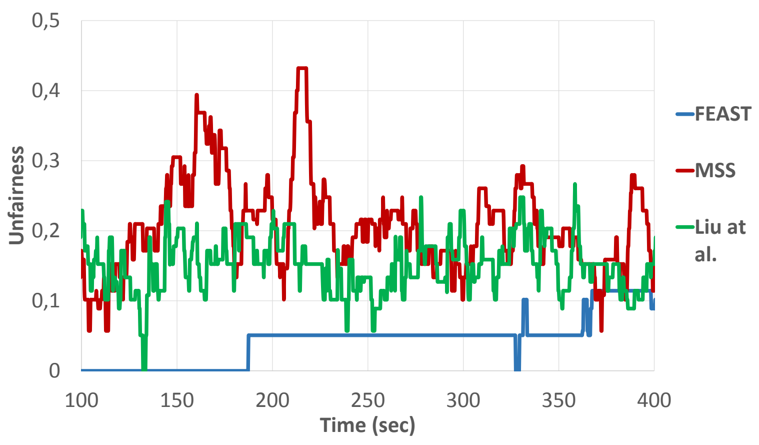

Figure 3.6: Unfairness comparison of HAS algorithms

video bitrates again in steady state. We display only data between 200 and 400th seconds in the graph to have a more clear view. In the graph, we see that all the algorithms are arranging the bitrates such that average video bitrates are generally close to 0.95 Mbps. FEAST is performing much more stable than other algorithms. Average video bitrate of FEAST does not change frequently while the bit rates for the others are oscillating. Switching rate is also much lower than other algorithms. Total switching rate of all clients is 0.18 switch/second in FEAST whereas it is 1.01 for MSS and 0.92 for Liu at al..

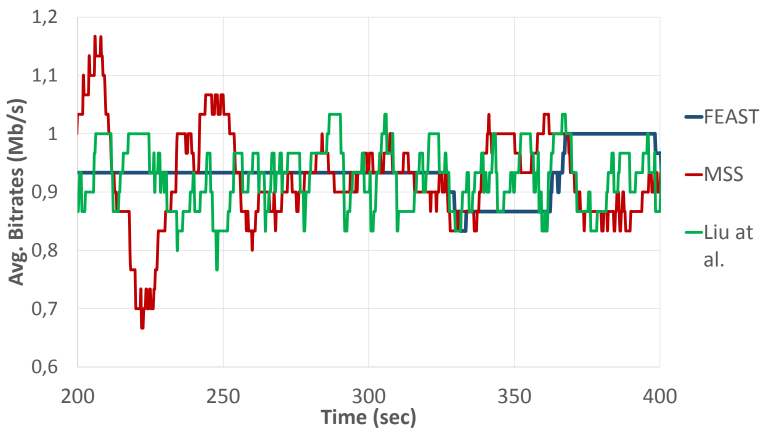

Figure 3.7: Average video bitrate comparison of HAS algorithms

In next step, we added some background traffic to see the reaction of FEAST to other traffics. In this step, two sources are continuously producing exponential UDP traffic at a total rate of 3 Mb/s. In other words, sum of rates of two exponential traffic sources are 3 Mb/s. In Fig. 3.5, E1 and E2 are sources and ER1 and ER2 are receivers for this background traffic.

Fig. 3.8 is showing the unfairness of three algorithms for this scenario. Although unfairness of

FEAST increased when it is compared to no background traffic scenario, it still outperforms other two algorithms. It does not only performs a more fair but also more stable service to users by avoiding unnecessary bitrate fluctuations. Total switching rate of all clients is computed to be 0.21 switch/second in FEAST whereas it is 1.09 for MSS and 0.99 for Liu at al. It means 0.21 user is changing its video bitrate each second for FEAST.

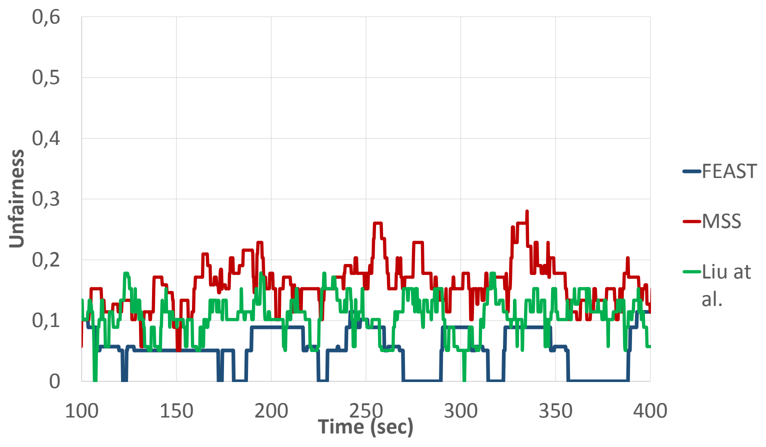

Figure 3.8: Unfairness comparison of HAS algorithms in the existance of background traffic

Fig. 3.9 is showing the average video bitrates for the same scenario. It is seen that average video bitrates of the clients are generally close for FEAST, MSS and Liu at al., but FEAST is again performing the most stable rate adaptation. It is an expected situation for the average bitrates to decrease when it is compared to the case without background traffic because background exponential traffic uses some portion of the bandwidth.

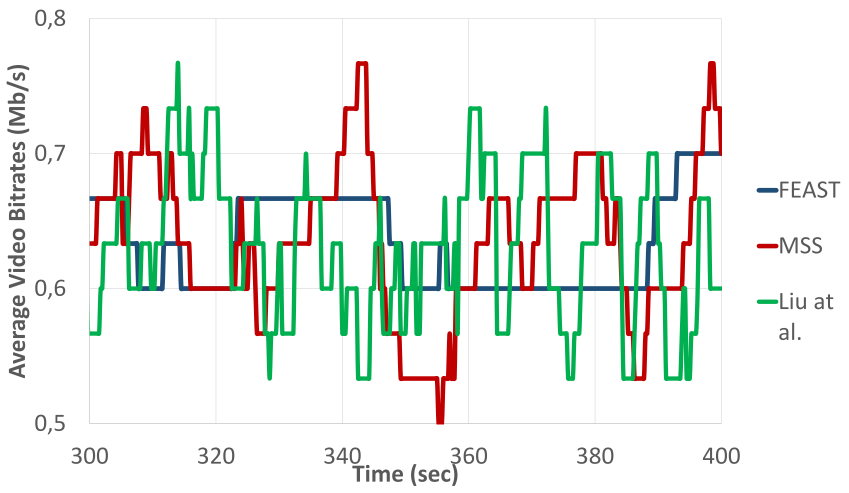

Figure 3.9: Average video bitrate comparison of HAS algorithms in the existance of background traffic

In the next experiment, we examined how unfairness, stability and utilization change according to the amount of background traffic. We used the same topology here. We changed the background traffic rate from 0 to 5 Mb/s with step size of 0.5 Mb/s and measure the network performance.

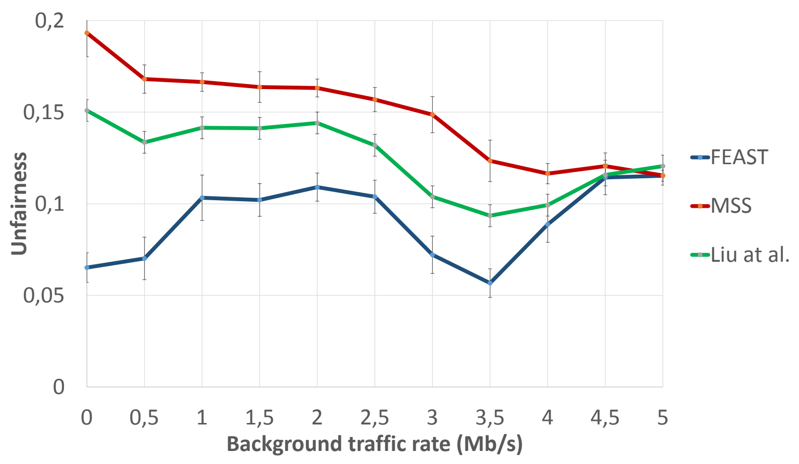

Figure 3.10: Average unfairness comparison of HAS algorithms according to background traffic rate

In this part, we define the average unfairness U_a which shows the time average for $U(t)$ over time interval of 0-500 seconds. Figure 3.10 is showing the variation of average unfairness according to background traffic rate. It is clearly seen that FEAST is outperforming against other HAS algorithms. It is interesting to see that when background traffic exceeds a point, all algorithms show a similar performance because the available bandwidth becomes enough only for the lowest video bitrates for the clients. Note that the error bars on the graph denotes the max-min difference of the results.

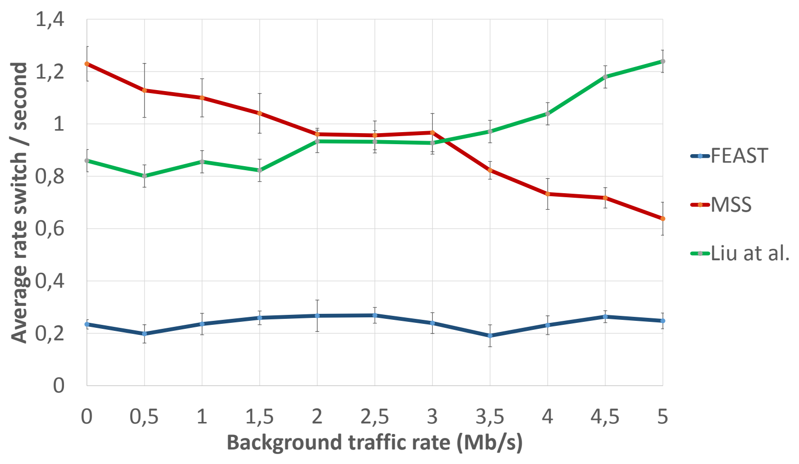

Figure 3.11: Rate switching comparison of HAS algorithms according to background traffic rate

Fig. 3.11 is showing the average rate switches of all the users. FEAST clients changes their video bitrates much less frequently than the clients using other two HAS algorithms. It is interesting to see that for the algorithm in Liu at al., switching rate increases with increasing amount of background traffic whereas it is decreasing for MSS algorithm. This most probably because Liu at al. only considers the download time of the last segment whereas FEAST and MSS takes the average of last three downloads. Therefore, a smoother estimation is done and systems react better to instantaneous changes in the background traffic.

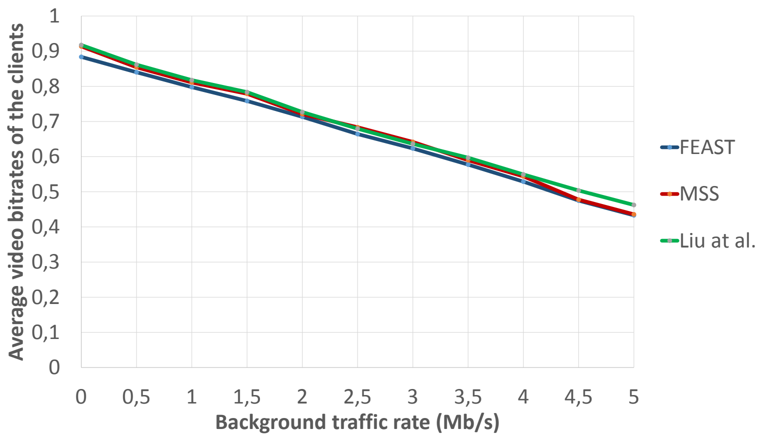

Figure 3.12: Average video bitrate comparison of HAS algorithms according to background traffic rate

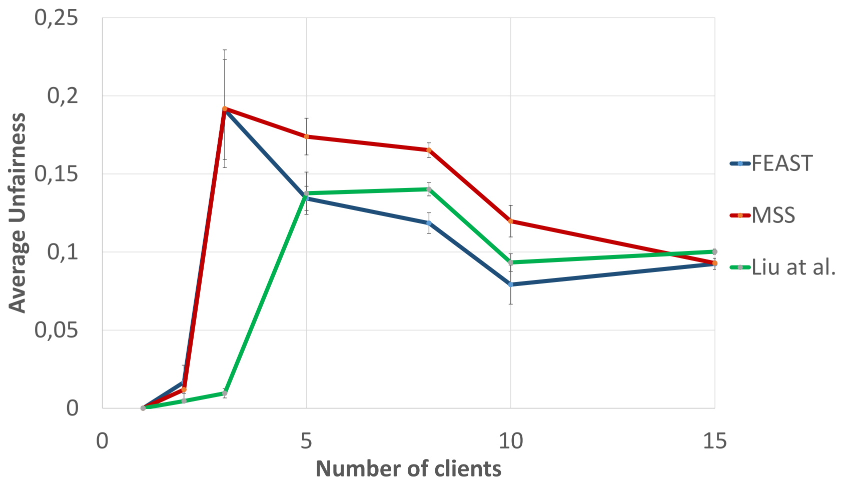

Figure 3.13: Average unfairness comparison of HAS algorithms according to number of clients

We also took the long term average of *average video bitrates of the users* and compare them in Fig. 3.12. It is obvious that average bitrates decrease with higher background traffic rate. FEAST, MSS and Liu at al. provide very close average bitrates. The negligibly small difference between FEAST and other algorithms is due to a more stable operation and not attempting unnecessary video bitrate increments.

Next experiment is conducted to find out the relation between HAS performance and number of clients. Same topology is used in this experiment. Background traffic is set to 3 Mb/s. Number of clients are changed from 1 to 15 to investigate the effects on unfairness, video bitrates and switching rates.

Fig. 3.13 is showing the unfairness of FEAST, MSS and Liu at al. according to varying number of clients. When number of clients is higher than 5, FEAST shows a better performance in terms of unfairness. When number of clients exceeds a point, all the algorithms perform close to each other because all the clients should get the lowest quality videos in such conditions.

It is interesting to see that Liu at al. seems to perform better when number of clients is lower than 5. However, it is understood that it is not the case when avg. video bitrate graph is examined.

In Fig. 3.14, it is seen that Liu at al. cannot increase the video bitrates although there is available bandwidth when there are a few number of clients. All the clients watch the videos in lower qualities. Therefore, less unfairness is observed in that region. However, due to low quality videos Liu at al. performs much worse than FEAST and MSS. On the other hand, then number of clients increase, all the algorithms start to perform close to each other in terms of average video bitrates. The reason for the low quality videos when number of clients is fewer than 5 is due to the rate switch up algorithm of Liu at al. The algorithm increases the bitrate if the available bandwidth is $1 + \epsilon$ times higher than the current video bitrate which is defined in Section 2.3.2. For the given video bitrates, ϵ is calculated to be 1 which means the available bandwidth must be 2 times greater than the video bitrate of the client. In this topology, when number of the clients is low, the bottleneck bandwidth for the clients is the local links which is 3 Mb/s. The clients cannot exceed 1.5 Mb/s due to the value of ϵ. Therefore, performance becomes much worse than MSS and FEAST for such conditions.

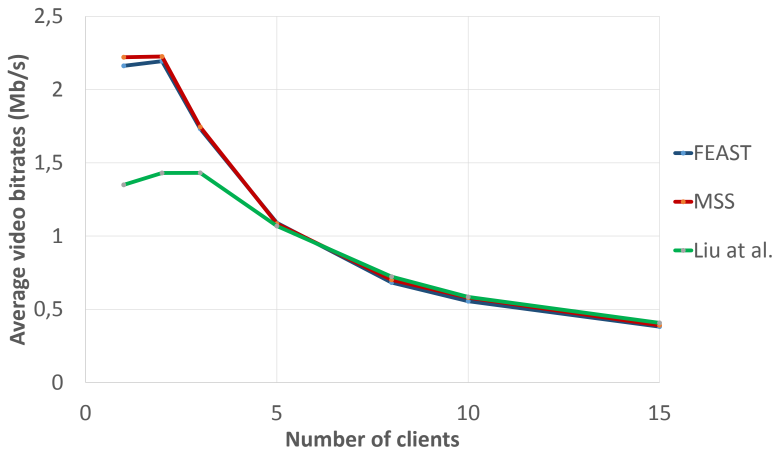

Figure 3.14: Average video bitrate comparison of HAS algorithms according to number of clients

The last observation of this experiment is rate switching. Fig. 3.15 shows that when number of clients is more than 5, FEAST performs the best. However, again Liu at al. seems to perform better here with low number of nodes because it cannot increase its bitrate, so rate switching is less. However, it enormously increase the switching rate when number of clints increase.

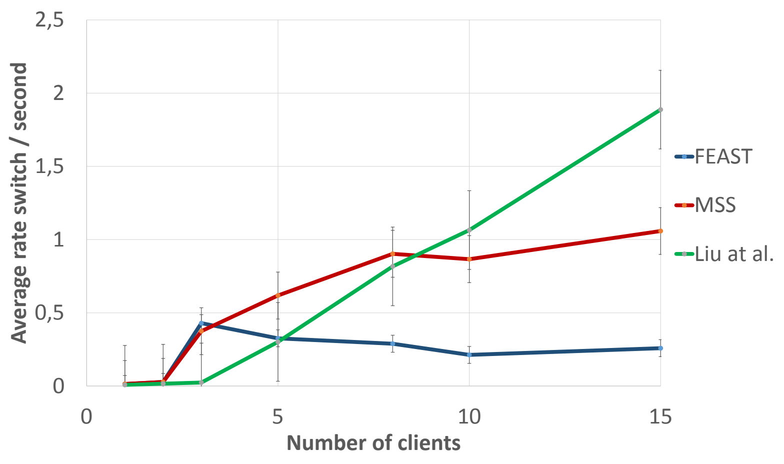

Figure 3.15: Average rate switch comparison of HAS algorithms according to number of clients

We conducted a final experiment to see the effects of encoding rates. HTTP servers encode the videos into multiple bitrates and clients request the most appropriate video resolution. Assume maximum video quality is 2.4 Mb/s. Until now, in our simulations, server encoded the video into 8 different resolutions with the lowest of 0.3 Mb/s and highest of 2.4 Mb/s. There is no standard for the number of different resolutions defined for HAS. What would happen if the server encoded it into 4 different resolutions of 0.6, 1.2, 1.8 and 2.4 Mb/s? In this part we simulated both conditions and compared them.

The most important result of the final experiment is the supported number of clients. When the lowest encoding rate of the video is 0.3 Mb/s, more users can watch the video in the worst network conditions. When it becomes 0.6 Mb/s, this number of users decreases almost by half.

We used the same topology and same background traffic. In the experiment, when number of clients is less than 8, average video bitrates of the clients is almost the same for two encoding scheme. When video is encoded into 8 different resolutions, unfairness decreases because users will have more choice of video bitrates. On the other hand, less switching occurs when video is encoded into 4 different bitrates.

When we increased the number of clients from 8 to 10, *freezing* starts to occur for 4 different resolution cases because the lowest bitrate is 0.6 Mb/s and users cannot decrease further to fill their buffers. Freezing occurs with an average of 34 times for each client in each simulation. When number of clients becomes 15, it becomes impossible for the users to watch a video due to frequent freezing. On the other hand, this number is only 0.1 when it is encoded into 8 different bitrates. Tables 3.2, 3.3 and 3.4 are showing the average unfairness, switching rates and video bitrates for these conditions respectively. In the tables n_r denotes the number of video resolutions. When the tables are examined, it is seen that most of the resultant values are almost same for the two cases. Switching rate seems lower for $n_r = 4$ case because it lets the clients to increase or decrease their video bitrates faster. However, it cannot support as many number of clients as $n_r = 8$ case

Table 3.2: Unfairness

$nodes$	$n_r = 8$	$n_r = 4$
1	0	0
2	0.013	0.006
3	0.18	0.30
5	0.13	0.13
8	0.11	0,13
10	0,08	0,01

Table 3.3: Switching Rates

$nodes$	$n_r = 8$	$n_r = 4$
1	0.01	0.01
2	0.03	0.01
3	0.43	0.43
5	0.33	0.21
8	0.32	0.16
10	0.24	0.04

Table 3.4: Avg. Video Rates

$nodes$	$n_r = 8$	$n_r = 4$
1	2.18	2.18
2	2.20	2.21
3	1.75	1.74
5	1.08	1.11
8	0.69	0.70
10	0.55	0.58

In this chapter, we introduced FEAST which provides a fair, stable and efficient HTTP Adaptive Streaming service. However, it is based on pure client-server architecture. When we apply it to a live streaming case, available server bandwidth may not be enough due to very high number of users. Moreoever, we use a buffer length of 20 seconds of video data. Therefore, FEAST is best suitable for Video on Demand (VoD) services in which the contents are ready before users watch it. For live streaming, we propose ALIVE in the next chapter.

CHAPTER 4

ALIVE: ADAPTIVE LIVE STREAMING OVER HTTP

4.1 Motivation for ALIVE

In Video on Demand services, the possibility of multiple clients' watching the same video is quite low. However, in live TV applications, it is the opposite case. It is almost certain that more than one clients will watch the same TV channel. By using this property of live streaming, we developed *Adaptive LIVE Streaming over HTTP* (ALIVE) which is based on downloading the video segments from a nearby client instead of the server if possible. ALIVE can be considered as a combination of HAS, SVC and P2P systems.

In recent years, people started watching television over the Internet instead of cable television networks. The frequently employed best-effort services are not sufficient for the television over Internet. Hence, some new techniques must be developed to give a satisfactory experience to the clients. HTTP Adaptive Streaming has become a popular solution also for live streaming. Microsoft Smooth Streaming [5], Move Networks [4] and Apple HTTP Live Streaming [6] are some examples of this fashion.

Live streaming differs from video on demand streaming in a number of points. First of all, content is dynamically prepared. It means that server cannot prepare the contents of next few minutes as it does in recorded video streaming. A client cannot cache the next 20 seconds of the video in its buffer as it is done in MSS or FEAST. Moreover, a client has to wait for the next requested segment until it becomes ready. Buffering amount must be limited such that the delay between the actual TV flow and the one on client's screen is small enough. In [43], the live TV delay for HTTP Adaptive streaming is analyzed. They concluded that segment size is the most critical element in this delay. When the segment size is greater, the delay becomes larger. For some TV events, this delay may become critical. For a football game audience, it is the worst nightmare to hear a goal from the neighbours before they watch it on the screen. Buffering delay, transmission delay and segment preparation delay are the main contributers to this delay. Buffering delay is necessary for robustness. Therefore, buffer management becomes much more critical. A detailed investigation of this analysis will be given in Section 4.2.

A second difference between live and recorded video streaming is the download popularity of the videos as it is indicated in the first paragraph of this chapter. When many clients are downloading exactly same data from the server, redundant data circulates in the network. This lowers the quality and generates a delay for IP packets. To avoid this redundancy and manage the dynamically prepared video contents, a different approach is needed for live streaming.

In HTTP Adaptive Streaming solutions, all the clients download the contents directly from the server. In the ideal case, the shared bandwidth is fairly divided among TCP connections and all the clients will get C/N portion of the bandwidth when C is the bandwidth of the link and N is the number of TCP connections. In this part of the thesis, we propose *Adaptive LIVE Streaming over HTTP* which enables the clients to download the contents from nearby clients instead of the server whenever it is possible. It is actually a kind of Peer-to-Peer system with HAS and SVC. ALIVE is a novel architecture because it is the first work combining HAS, SVC and P2P systems. With this solution, the load of the server significantly decreases and more clients are supported with more QoE. ALIVE clients can watch the video with a quality more than C/N video bitrate.

4.2 Operation of ALIVE

The main aim of ALIVE is to decrease the load of the server and give a better service to the clients. If the number of clients who download the contents from the server decreases, then the quality of the video can be increased. Furthermore if a client downloads from a nearby client rather than from a server that is many hops away, both the traffic load distribution in the network and Qos/QoE perceived by the downloading client will improve. We call the client that provides the video to the other clients *seed*.

When a client connects to ALIVE, it first connects to the server and downloads a number of initial segments from the server to have initial buffering of the video. During these downloads, it learns whether there is any seed that is watching its channel or not. Then the client decides to connect to a seed according to the available bandwidth, number of hops to the seed certain criteria by combining the available data, it gives the decision of connecting to a seed. An additional feature of ALIVE is enabling the client to download the video from the seed at different rates by making use of Scalable Video Coding [13].

When a neighbour contains a video segment with resolution r_n, it automatically contains, r_1, r_2, ... r_{n-1} where r_1 is the lowest quality. Therefore, the client can choose the most appropriate rate by downloading the necessary layers. Although the maximum video bitrate seems to be limited when it is downloaded from the neighbour, it will be seen in Section 4.2.1 that a client never connects to a neighbour whose condition is worse than the client itself. Those conditions will also be explained in the next section.

Two important components of ALIVE operation are the selection of the seed and the rate adaptation for downloading.

4.2.1 Seed Selection for ALIVE

A client selects a seed whenever it connects to ALIVE and whenever it changes the channel. Now, the elements of seed selection for ALIVE will be defined.

Network Address Translators (NATs)

First of all, a client can download the content from another client if the other client is not behind NAT because it will not be reachable otherwise. Although some NAT traversal methods exist in the literature as explained in [44], they are complex and the advantage of HTTP Adaptive Streaming due to NAT traversal will be lost with the usage of them. In order to make ALIVE simpler, a client can become a seed if it is not behind NAT.

Available Bandwidth

In ALIVE, a client measures the available bandwidth in application layer between itself and the server. This measure helps the client for seed selection. Therefore, bandwidth measurement will have a critical role. In the literature, many bandwidth measurement tools are proposed. Some of them are Pathload [45], Pathchirp [46], Assolo [47], Spruce [48], IGI [49] and Abing [50]. These tools measure or estimate the available bandwidth by introducing some additional traffic to the network and observing the results of that packets. In [51], [52] and [53], performance of existing bandwidth measurement tools are compared with respect to their estimation time, additional traffic and accuracy. It is seen that all of the tools introduce a significant traffic to the network which is an undesirable case. More importantly, in [52] the time to estimate the available bandwidth for these methods is between 5 and 30 seconds which is extremely high for ALIVE. In the evaluations, it is seen that more accurate estimations are done when measurement times and additional traffics are higher. For example, Abing measures the bandwidth in a very small time with great inaccuracy. Beyond all the analysis, in [54], the relation between the available bandwidth and TCP throughput is analyzed and it is seen that in most cases they are not very relevant to each other. With all the information about bandwidth measurement tools, we concluded that instead of all the complex bandwidth measurement or estimation tools, we will simply use the download information of HAS clients which contains the download time of a segment and the bitrate of the downloaded video segment. That result gives us the available bandwidth at application layer, not at the network layer which is the case for the above tools and may give misleading information about TCP throughput. Moreover, by taking the average of last three downloads, a smoother bandwidth estimation is achieved.

Hop Count

The topology of the network plays a critical role in determining the connection between clients and seeds. For example, consider the simple network given in Fig. 4.1.

Assume that Client 1 is a seed and downloading from the server. When Client 2 connects to ALIVE, it may select Client 1 as its seed or download the segments from the server. The better

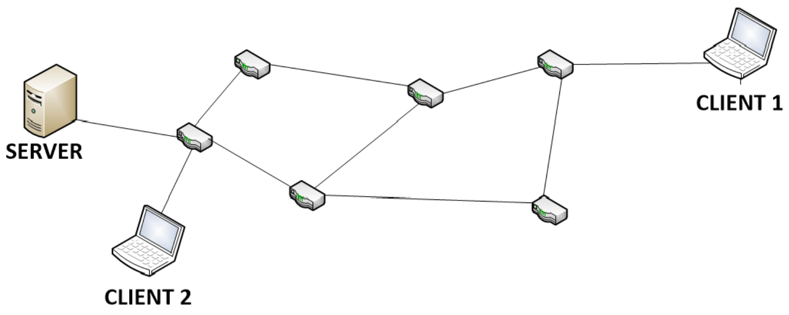

Figure 4.1: The topology that denotes the importance of hop count

choice here is to connect to the server because it is closer. Otherwise, if Client 1 connects to Client 2 it will download the videos in a worse quality and it will increase the network traffic. Therefore, the hop count between the nodes will be a second parameter for this decision after available bandwidth in the application layer.

Reachability

When a client selects another client as its seed, it must check whether the seed is reachable or not. After learning the seeds from the server, the clients will send UDP messages to those seeds to both learn their bandwidth between the seed to the server, hop information from the seed to the server and test their reachability. The client also sends *ping* messages to the seeds to again test the reachability. With ping operation, the client can also find the number of hops to the seed.

Seed selection functions

After obtaining the available bandwidth from the seeds to server and number of hops between the seeds and the server, clients made some computations in order to find the best seed. Let i be the client and j ($0 < j <= J$ where J is the number of seeds) be the seeds. $F(j)$ is called the *choice function* which is used in determining whether to connect to a seed or download the videos from the server.

$$F(j) = b_j(t) - \alpha * hop_j(server) \tag{4.1}$$

In equation 4.1, $b_j(t)$ is the available bandwidth from seed j to the server at time t and $hop_j(server)$ is the number of hops between seed j and the server. $F(j)$ is defined as the difference between the available bandwidth and the weighted number of hops between seed j and server. A client calculates $F(j)$; $j = 1, ..J$ where J is the number of seeds. It also calculates $F(i)$ which is the result of choice function of the client itself.

$$F(i) = b_i(t) - \alpha * hop_i(server) \tag{4.2}$$

Bandwidth information will be used in this formula by Mb/s. α will be the coefficient of the hops and will be 0.1 . To make it clear, consider the following example. If the measured bandwidth between seed j and the server is 2 Mb/s and number of hops between seed j and the server is 6, then $F(j)$ will be 1.4 .

If the client decides that it should connect to a seed, it calculates $L(j)$; $j = 1, ..J$ where $L(j)$ is a weighted version of $F(j)$ which introduces the number of hops between the client and each seed.

$$L(j) = \frac{F(j)}{hop_j(client)} \tag{4.3}$$

In equation 4.3, $hop_j(client)$ denotes number of hops between the client and seed j. These functions will be used in determining the target for downloading the video contents.

Catching the real TV flow

In live streaming, it is very important to keep the delay between the TV events and the time client watch that event on its screen minimum. In [43], this delay is investigated. First component of the delay is the segment creation time in the server. Assume that the segments are t_s seconds long. A server can prepare the segment of time 0-t_s only after t_s'th second because it needs the whole t_s second video to encode it. A simple time diagram of content preparation is shown in Fig. 4.2. The server can start encoding the T1 segment after T1 finishes. It can encode it with a small encoding delay which we will neglect in our calculations.

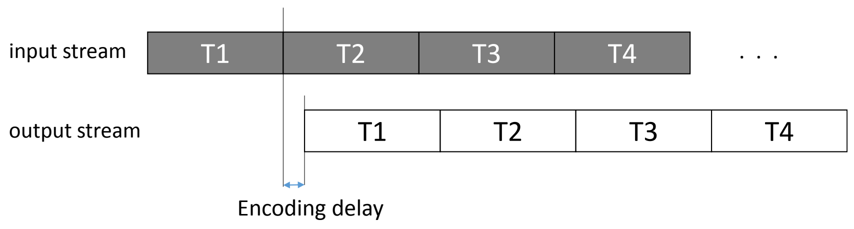

Figure 4.2: Content preparation in the server

Other delays are asynchronous fetch, download delay and buffering delay. Asynchronous fetch denotes the time that when a client sends a request to the server, it will get the latest available segment. For example if segment time is 1 seconds, when a client request a segment at 3.99th second, it will get the segment which becomes ready at t=3. Therefore, a maximum delay of t_s is also introduced here.

In [43], it is indicated that a client should have one or two full segments ready in its buffer to cope with sudden network condition changes. Therefore additional delays are introduced with this buffering scheme.

In ALIVE, we propose a new buffering scheme which aims to minimize live TV delay. To this end, we choose the length of video segments as 1 second which is shorter than FEAST as written in [43]. When a client first connects to ALIVE, it should download four segments before playout to make ALIVE robust against network conditions. In each download, client requests the *latest available segment*. Figures 4.3, 4.4 and 4.5 are showing three cases of first four downloads.

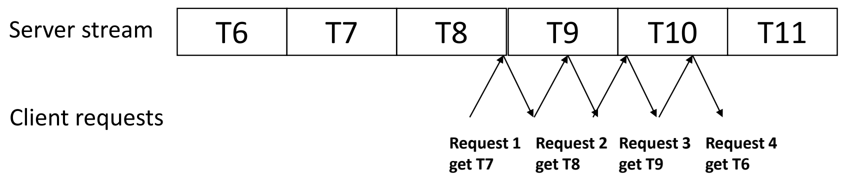

Figure 4.3: Client downloads the first four segments: Case1

In Fig 4.3, in the first request, client downloads T7 because it is the latest available segment. In the second download, T8 becomes ready, therefore T8 is downloaded. In the third download, T9 becomes available and it downloads T9. In the fourth download, T9 is still the latest available segment, however the client downloaded it before as well as T8 and T7. Therefore, it downloads T6 in the fourth request. The client can start playing from its buffer just after the download of T6 finishes. It will start by playing T6.

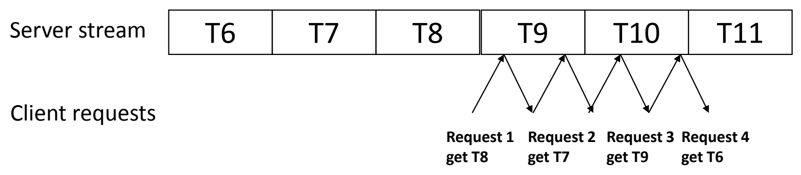

Figure 4.4: Client downloads the first four segments: Case2

In Fig 4.4, client first downloads T8 because it is the latest available segment. In the second request, T8 is still the latest available segment, however the client downloads T7 because it already has T8. When it sends the third request, T9 becomes available and it is downloaded. In fourth download, T9 is still the latest available segment, however the client downloads T6 because it downloads T9, T8 and T7 before. After the last download, client can start playing by T6. This is almost the same with case 1 however, the order of the first 4 downloaded segments is different.

In the third case, which is shown in Fig 4.5, client first downloads T8. In the second download T8 is still the latest available segment and the client downloads T7. When the client sends the request for third segment, T9 is the latest available segment and client downloads it. In the

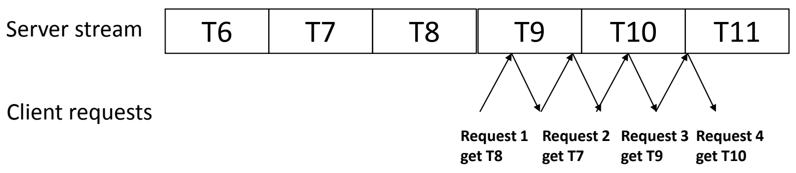

Figure 4.5: Client downloads the first four segments: Case3

fourth request, T10 becomes ready and it is downloaded In this case the delay between the events and their playouts becomes the lowest. Note that for all cases, after fourth download, client downloads the segments in sequence.

Algorithm 5 is showing the decision algorithm when a client connects to ALIVE. The algorithm explains the way for client *i* to find the most appropriate target for downloading the segments. A client first connects to the server. In first four downloads from the server, the client fills its buffer and during these downloads it learns the seeds of the channel. If no seeds exist, then it will continue to download from the server. Moreover, if it is not behind NAT, then that client declares itself as a seed for that channel. If it is behind NAT, then it downloads from the server without being the seed. It will check if there are other seeds, periodically in this case.

If there are more than one seeds for that channel, the client chooses the most appropriate seed for itself. First of all, it checks if it will be in a better condition, which will be explained in detail in next paragraphs, than the seeds. After learning about the seeds of the channels, it will send UDP messages to all seeds and learns the bandwidth information and hop count to the server of the seeds. The reasons of UDP usage are avoiding unnecessary TCP connections which will waste bandwidth and time, and observing the reachability of the seeds. By bandwidth information and hop count to the server of the seeds, the client calculates $F(j)$ where j is jth seed for each seed and $F(i)$ of itself. $F(j)$ and $F(i)$ are the formulas given in 4.1 and 4.2 respectively.

After calculation of $F(j)$ for each seed and the client itself, the client chooses the highest of all. If the client's $F(i)$ is the highest and the client is not behind NAT, it will become a seed. Otherwise, it will update its $F(i)$ by multiplying it with 0.8, which is a heuristic formula, and compare it the $F(j)$ of the seeds. If the client's $F(i)$ is still greater than seeds' $F(j)$, the client will continue to download from the server and periodically checks the status of the seeds. Note that $F(j)$ and $F(i)$ formulas are used only for determining whether the client should connect to a seed or download from the server.

After that update, if $F(i)$ of the client is not greatest, then it will calculate $L(j)$ for each seed where $L(j)$ is defined in equation 4.3. $L(j)$ is actually the weighted version of $F(j)$ by considering the number of hops between the seed and the client. It avoids a client to connect

Algorithm 5 Seed Selection in ALIVE

```
1: Client downloads the first four segments from the server
2: Client calculates its available bandwidth and learn the seeds
3: if No seeds exist for that channel then
4:     if Client is not behind NAT then
5:         Client becomes a seed
6:     else if Client is behind NAT then
7:         Client downloads from the server
8:         It periodically checks the situation of seeds
9:     end if
10: else if There is(are) seed(s) for that channel then
11:     Let the number of seeds be J
12:     $F(j)$ is calculated for each seed $j$=1,2,..,J and $F(i)$ for client itself
13:     Highest of $F(j)$ and $F(i)$ is selected
14:     if $F(i)$ is highest then
15:         if Client is not behind NAT then
16:             Client becomes a seed
17:         else if Client is behind NAT then
18:             Update $F(i)$ as 0.8 * $F(i)$
19:             if $F(i)$ is highest then
20:                 Client downloads from the server
21:                 It periodically checks the situation of seeds
22:             else
23:                 Calculate $L(j)$ for each seed
24:                 Choose $j$ as seed with highest $L(j)$
25:             end if
26:         end if
27:     else
28:         Calculate $L(j)$ for each seed
29:         Choose $j$ as seed with highest $L(j)$
30:     end if
31: end if
32: ALL THE VIDEO DATA TRANSFERS ARE DONE VIA HTTP ADAPTIVE
    STREAMING
```

to a seed that is far away from itself. Therefore, unnecessary delays and network traffic is also avoided. The client will connect to the seed with the highest $L(j)$.

To sum up the Algorithm 5, the client tries to connect to a seed with high bandwidth to server, low number of hops to the server and the client itself. If no seed exists, the client will become a seed unless it is behind NAT. If it is behind NAT, it will download the contents from the server. When there are one or more seeds, the client chooses the most appropriate seed. If the situation of the client is better than other seeds, the client will be a seed if it is not behind NAT. If the situation of the client is *much* better than the seeds and the client is behind NAT, it will download the contents from the server. If there is a seed in a better situation, then the client will download the contents from that client.

The clients download all the contents via HTTP and they adapt their video bitrates before sending GET requests each time. The rate adaptation algorithm of ALIVE will be shown in Section 4.2.2

In lines 8 and 21 of Algorithm 5, it is written that the client periodically checks the situation of the seeds. This period is defined to be 10 segment downloads for ALIVE. In other words, after downloading 10 video segments from the server, the client again learns the seeds, learns their bandwidth and hop information, calculate $F(j)$ and evaluate the situation again.

In ALIVE system, when a client connects to a seed, it will download the segments from the seed unless some *unwanted* conditions occur. Now, we will define those conditions.

Seed exits or changes channel

Clients that are connected to that seed is informed by the seed about that situation as soon as the seed changes the channel or exits from ALIVE. In this situation, clients perform the steps in Algorithm 5, then either finds a new seed, becomes a seed or continues downloading from the server.

Insufficient download bandwidth of the seed

This condition occurs when the download bandwidth of the client becomes insufficient. There are two cases for this condition. The first case happens when buffer of the seed contains less than 1.5 seconds of video data. Second case happens when the seed cannot download the video segments in high quality. In other words, if the seed cannot exceed a certain video quality for a certain time, then it means the download bandwidth of the seed is not enough. In ALIVE, we defined this condition as not being able to download the video segments with a quality of higher than 1.2 Mb/s for 10 successive segment downloads. In such a condition, the seed stop being a seed and apply the algorithm in 5. In this case, it cannot be a seed again even if it satisfies the conditions. It can connect to the second most available seed or download the segments from the server.

If a seed, to which a client is connected, becomes insufficient, that client disconnects from the seed and find a better seed by the same algorithm.

Insufficient upload bandwidth of the seed

This conditions occurs when the download bandwidth of a seed is enough but, it cannot upload the videos to other clients sufficiently. We defined this condition for two cases. If the buffer of a client which is connected to a seed contains less than 1 second of data, then we can say that upload bandwidth of the seed is not enough. Second condition occurs if the client's video bitrate is much less than the video bitrate of the seed's video bitrate for a certain time. In ALIVE, we defined this condition as not being able to download the video segments in a quality higher than 60 percent of quality of the seed for 10 successive segment downloads.

When it is realized that the upload bandwidth of the seed is not enough, the client disconnects from that seed and applies Algorithm 5. After the algorithm, it can be a seed or download the segments from the server without being a seed or connect to a seed which is not the previous one.

Empty Buffer

When the buffer of a client or a seed becomes empty, the video freezes which is an annoying experience for clients. In this conditions, if a client is connected to a seed, it disconnects from that seed and applies the same algorithm. Note that it cannot connect to its previous seed here. If a seed's buffer becomes empty, then it quits being a seed and applies the same algorithm. It is notable that when the buffer of a client or a seed starts decreasing, it is considered as insufficient upload bandwidth or download bandwidth of the seed respectively. Therefore, it decreases the chance of empty buffer by taking precautions beforehand.

When the buffer of the client depletes, there are two choices. The client can either skip the segments that it cannot watch due to video freeze and download the next available video segment, or it can download the last segment that it cannot watch and increase the live tv delay. In ALIVE, we choose the former case and the clients will download the next segment by skipping the previous ones.

4.2.2 Rate Adaptation for ALIVE Algorithm

In this section, we will define the rate adaptation algorithm for ALIVE. Rate adaptation is one of the most important elements of HTTP Adaptive Streaming. Unlike FEAST or MSS, live streaming requires less buffering in order to minimize the delay between the events and their playout on the screen. Rate adaptation becomes much more critical when buffer contains very small amount of data because buffer may be depleted in a badly defined rate adaptation algorithm.

In ALIVE, we will use a similar version of the algorithm in Liu at al [25]. However, we will consider the buffer size in the adaptation scheme which is not used in Liu at al.

Algorithm 6 defines the rate adaptation algorithm for ALIVE. Note that, when the client first

Algorithm 6 Rate Adaptation Algorithm for ALIVE

```
1:  Initially: $r_i(t)$ is selected as the lowest bitrate
2:  For Each Request:
3:  if $buf_i(t) < 1$ then
4:      $r_i(t) = 1$
5:  else
6:      $\mu^0 = MSD/SFT$
7:      $\mu = (mu^0 + mu^{-1} + mu^{-2})/3$
8:      if $\mu > 1.2$ then
9:          $r_i = pr_i^+$
10:     else if $\mu < 0.8$ then
11:         $r_i = \max(r_c) : r_c < \mu * pr_i$
12:     else
13:         $r_i = pr_i$
14:     end if
15: end if
```

connects to ALIVE or changes the channel, it first connects to the server and downloads the first four segments. This four segments are downloaded with the lowest video bitrate. Rate adaptation is done after this period.

Aim of the rate adaptation algorithm is to keep the video bitrate maximum while avoiding the depletion of the buffer. Therefore, we performed aggressive rate switching down. Whenever the buffer contains less than one seconds of video data, the rate is set to the lowest for precaution. When the buffer size is higher, the client computes μ which is the ratio between media segment duration to segment fetch time. A higher value of μ (higher than 1.2) denotes a good network situation and it lets the client increase the video bitrate one step. If μ is low (less than 0.8), than the client will choose the maximum bitrate r_c which satisfies

$$r_c < \mu * pr_i \tag{4.4}$$

Note that pr_i is the previous segment's bitrate whereas r_i is the next segment's video bitrate. Different than Liu at al., we calculate μ as the average of last three μ values to make a smoother network prediction.

In ALIVE, both the clients and the seeds perform the same rate adaptation algorithm. However, the time that they send the GET request will be different for them. The seed or a client who downloads from the server sends a request whenever the requested segments becomes ready. A client who downloads from a seed sends the request 1 second after the content becomes ready because the seed will download the content at that time. If the download of that content by the seed does not finish at that time, the seed sends the data to that client whenever it finishes downloading.

4.3 Performance Evaluation of ALIVE

In this Section, we compare the performances of ALIVE with Liu at al. which is proposed in [25] and is a pure client server architecture. The reason that we do not compare the performance of ALIVE with MSS is that live TV algorithm of MSS is not available. Actually, many of the HAS algorithms adapt the video rates according to high buffering values which is not appropriate for live TV case. We performed several experiments on NS2 to measure the server load, the average bitrates and supported number of clients according to number of clients, percentage of clients behind NAT, number of channels.

In the experiments, we used the backbone given in Fig 4.6. In the figure, server is connected to one of the core routers. The core routers form a backbone network. Then, the distribution routers are connected to these backbone routers. Finally, clients are connected to those distribution routers. An example complete network with 20 clients is shown in Fig 4.7.

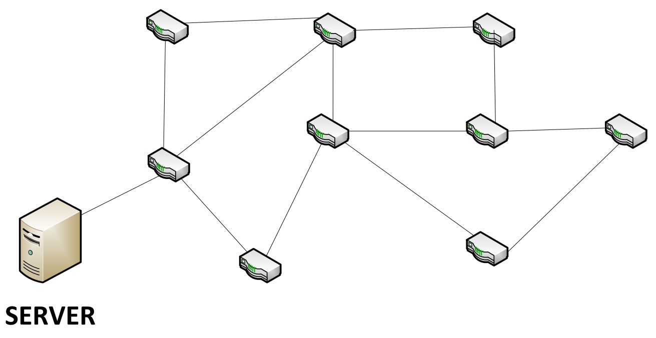

Figure 4.6: The backbone network that is used in ALIVE simulations

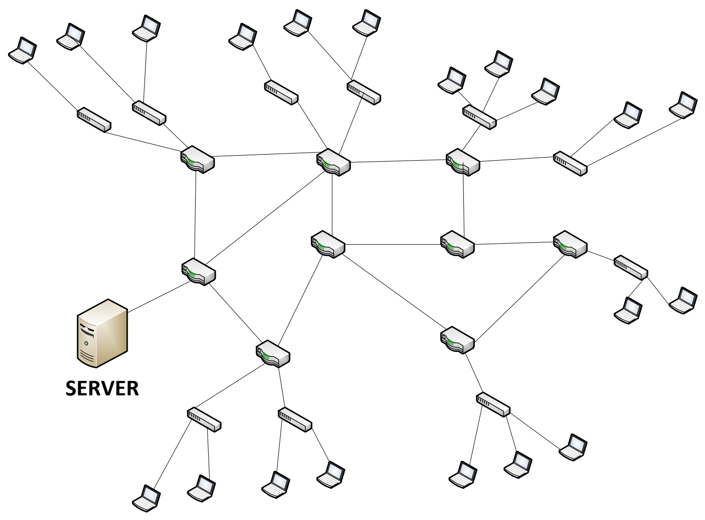

Figure 4.7: An example network with 20 clients that is used in ALIVE simulations

In the network, there are 4 types of links. The *server link* is the link between the server and the corresponding core router. The *core links* are the links between the core routers. The *distribution links* are between the core routers and distribution routers. Finally the *local links* are the links between the distribution routers and clients.

It is important to note that some additional nodes create background traffic. That nodes are also connected to distribution routers. There are both TCP and UDP background traffic generated by those nodes.

Some clients are behind NAT whereas some others are not. The percentage of clients behind NAT is selected to be 30 [55]. However, we will change that rate in some part of the experiment to see the effect of that rate on the performance of ALIVE.

Scalable Video Coding extension of H.264/AVC will be the coding standard of ALIVE. We use the same bitrates that are used in FEAST. There are 8 different encoding rates which are 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1 and 2.4 Mb/s. In [13] and [56], it is stated that there is a 10 percent encoding overhead in each layer of scalable video coding. Therefore, the applied encoding rates will increase and they are given in Table 4.1.

Table 4.1: Applied Encoding Rates

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
AVC (Mb/s)	0.3	0.6	0.9	1.2	1.5	1.8	2.1	2.4
SVC (Mb/s)	0.3	0.63	0.96	1.29	1.62	1.95	2.28	2.61

In our experiments, clients connect to ALIVE and each client selects a channel to watch. They switch the channel after some time. In [57], [58] and [59], it is shown that channel popularity can be modelled as Zipf distribution. Probability Mass Function of Zipf Distribution is;

$$f(k, s, N) = \frac{\frac{1}{k^s}}{\sum_{i=1}^{N} \frac{1}{i^s}} \tag{4.5}$$

where N is the number of channels, k is the rank and s is the value of the exponent characterizing the distribution [60]. The rank of the most watched channel is 1 whereas the rank of the least watched channel is N. The value of s is critical in determining the channel selection probabilities. It changes between 0 and 1. When it is equal to 0, all channels are equally probable to be watched by a client. As it is increased, the probability of watching more popular channels increases whereas the probability of watching less popular channels decreases. Fig 4.8 is showing the probability mass distribution of Zipf distribution according to different s values for 10 channels. Note that graph is plotted in log scale.

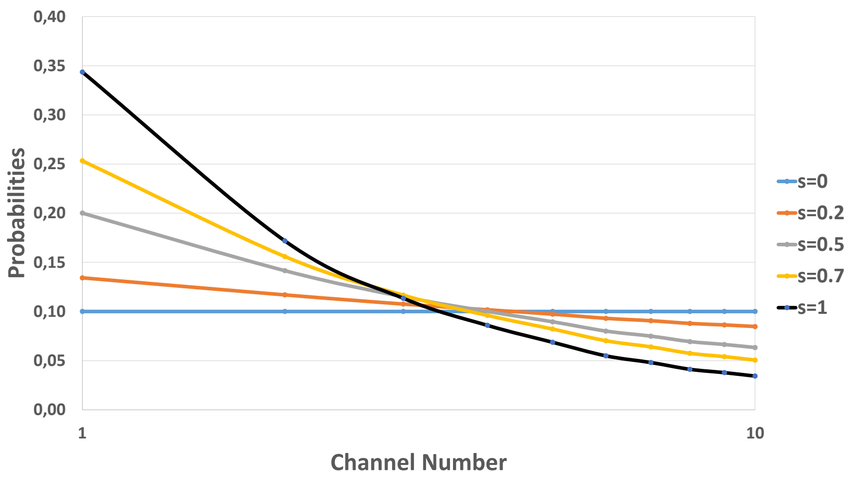

Figure 4.8: Effect of s parameter on the probability mass function of Zipf distribution

From the figure, it is clearly seen that the difference of probabilities between watching the most and least popular channels increases as s approaches to 1 whereas it is exactly 0 when s is zero. We will use s as 0.5 in our experiments.

In [57] and [61], the clients' TV watching behavior over Internet is modeled. According to these articles, the duration of watching each channel follows an exponential distribution. In other words, the time that is passed between two channel changes of the client is an exponential random variable. In our simulations, we used exponential distribution with a mean of 100 seconds as channel switch time. Our simulations last for 500 seconds, therefore each client changes the channel with an average of 5 times during a simulation.

We first present a very simple experiment in order to show the preliminary results of ALIVE. In this experiment, 30 clients are randomly connected to the distribution routers. UDP and TCP background traffic is applied. The bandwidth of the link between server and the core router is set to 30 Mb/s. The links in the core network is also set to 30 Mb/s. The links between the core routers and distribution routers have a bandwidth of 7.5 Mb/s and the local links have a bandwidth of 3 Mb/s. All links have the propagation delay of 1 ms. All the clients enter ALIVE in a random time between 0 and 20th seconds and quit ALIVE in a random time between 450 and 500th seconds. There are 10 TV channels. Fig 4.9 is showing the average of video bitrates of the clients with respect to time for a sample simulation. Only steady state period is shown in the graph.

It is clearly seen from Fig 4.9 that average video bitrate is higher for ALIVE. This is mostly because the number of clients that downloads the contents from the server is lowered. Fig 4.10 is showing the number of clients that are downloading the contents from the server.

From the figure, it is clearly seen that, fewer number of clients download the content from the server. This lowers the server's load and increases the video quality that is downloaded from

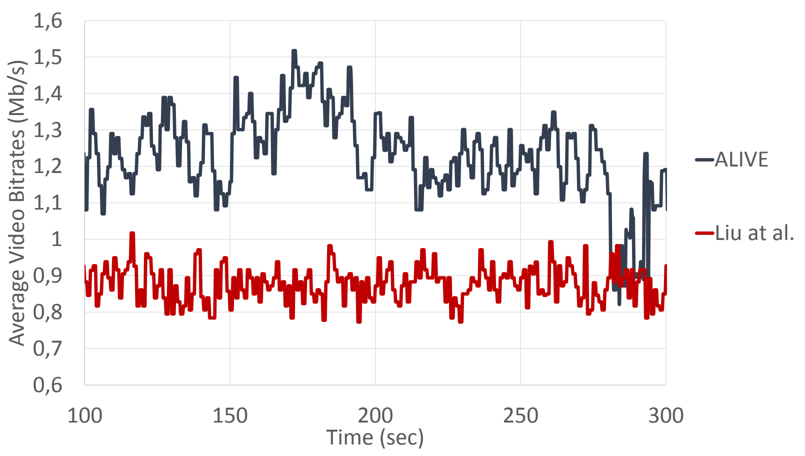

Figure 4.9: A sample Average video bitrates vs time graph for ALIVE and Liu at al.

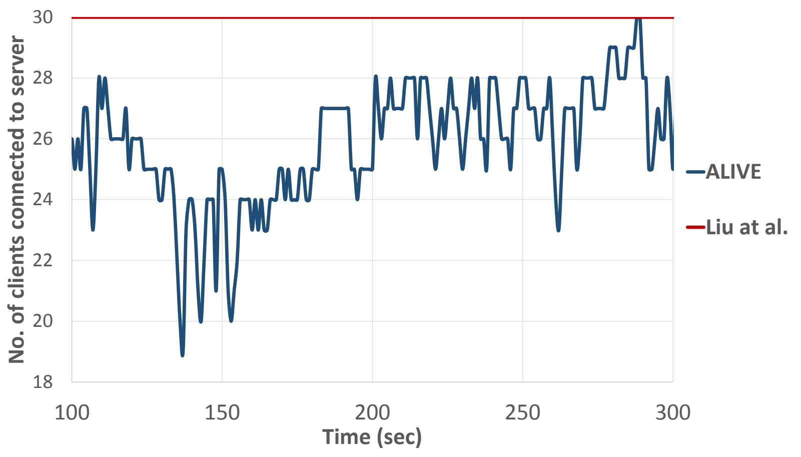

Figure 4.10: Comparison of number of clients that download the content from the server

the server.

After making it clear that ALIVE increases the video quality, we investigate the effects of various parameters on ALIVE performance.

In the second experiment, the number of clients is set to 50. The bandwidth of the server link and core links are set to 50 Mb/s. Distribution and local links have the same bandwidth with the previous experiment. We are first interested in the effect of percentage of clients behind NAT. If all of the clients are behind NAT, then it is expected that ALIVE and Liu at al gives similar performances because all the clients will download from the server in ALIVE. However, as the number of clients behind NAT decreases, the number of seeds increases and therefore number of clients that downloads from the server decreases. Fig 4.11 is showing the result of this experiment. As it is expected, average video quality decreases with a high number of clients behind NAT. Actually, it performs almost the same with Liu at al. when all the clients are behind NAT. Note that number of clients behind NAT does not affect the performance of Liu at al. because clients always download the contents from the server.

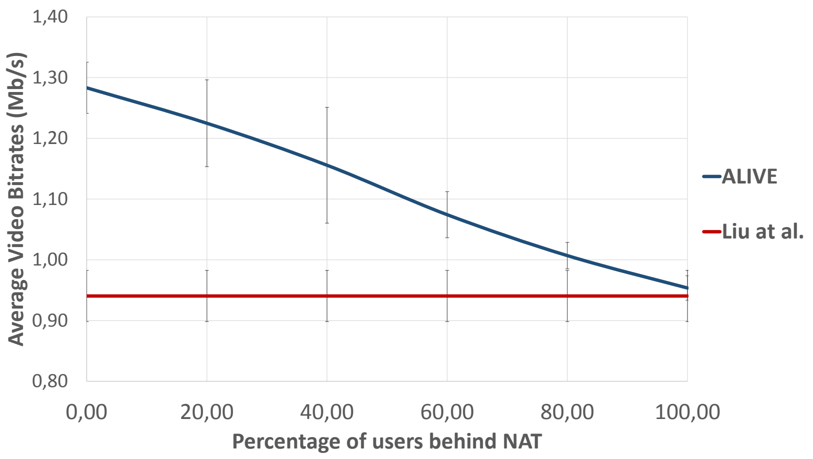

Figure 4.11: Average video bitrates of the clients according to different percentage of clients behind NAT

Now we investigate the effect of number of clients on the performance of ALIVE. Fig 4.12 is showing the average video bitrates of the clients according to different number of clients.

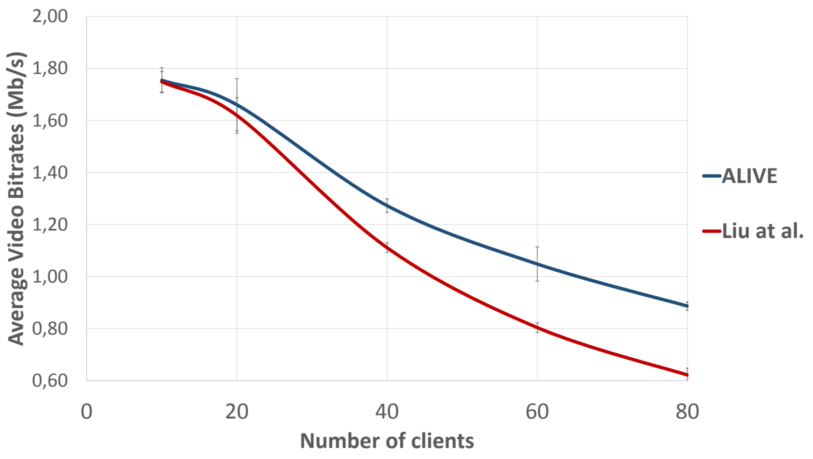

Figure 4.12: Average video bitrates of the clients according to different number of clients

It is seen from Fig 4.12 that average video bitrate decreases with the higher number of clients. It is expected because as the number of clients increase, the competition for the bandwidth increases. However, the decrease for Liu at al is greater than ALIVE. When the number of clients is low such as 5, they perform almost the same because number of seeds is low and the probability of more than one clients' watching the same channel is also low. However, as the number of clients increase, the number of seeds is most likely to increase. Therefore, the decrease in the average video bitrate for ALIVE is less than Liu at al. We also performed the same experiment with 100 clients. In that experiment it is seen that 100 client cannot be supported by Liu at al. because the server bandwidth will not be enough even for the lowest video bitrate for 100 clients. Buffer of the clients becomes free so frequent that it becomes impossible for the clients to watch the videos. On the other hand, ALIVE can support 100 clients with average video bitrate of 0.39 Mb/s and much less buffer depletion ratio than Liu at al. This experiment proves that more clients can be supported by ALIVE. We performed another experiment to see how the number of channels effect the performance of ALIVE. We tested 5, 10, 15, 20 and 50 channels and the results are given in Table 4.2. In this step, channel popularities are again calculated with Zipf distribution and channel change times are the same with the previous experiments.

Table 4.2: Avg. Video bitrate according to number of channels

No. of channels	5	10	15	20	50
ALIVE	1.25	1.18	1.14	1.09	0.95
Liu at al.	0.94	0.94	0.93	0.93	0.93

When the number of channels is low, the probability of more than one clients' watching the same channel increases. Therefore, more seeds exist in such conditions and more clients connect to those seeds. This causes a decrease in server's load which increases the downloaded video quality. However, as the number of channels increase, the number of clients that watch the same channel decreases. Therefore, more clients download the contents from the server and performance of ALIVE becomes closer to Liu at al. Note that performance of Liu at al. again does not change with the number of channels.

We also computed the channel change delays which is the time between the client changes the time and beginning of playout. Note that this number is still not enough to meet the 0.43 seconds which is defined to be the acceptable delay in [62]. However, ALIVE performs better than Liu at al with respect to channel change delay because of server link's lower utilization. Average delay is 1.24 seconds for Liu at al. whereas this number is 1.04 seconds for ALIVE when the number of clients is 50 and number of channels is 10. A client does not start playout until 4 segments are downloaded. 4 segments with lowest quality of 0.3 Mb/s means 1.2 Mb of data. Due to the fact that less than 50 clients use the server's link directly, ALIVE clients can download them faster than Liu at al. Note that the main reason for this high delay is the necessity to download the 4 segments. If the client is allowed to start playout after downloading the first (or second, third) segment, this delay reduces significantly and it most probably become less than 0.43 seconds. However, the system becomes prone to video freezes due to network fluctuations and the buffer most probably deplete with such a solution. Therefore, there is a tradeoff between robustness and channel change delay where robustness is necessary during the whole TV experience whereas channel change delay is experienced only after changing the channel.

CHAPTER 5

CONCLUSION

In this thesis, we examined HTTP Adaptive Streaming (HAS) and proposed new solutions for weaknesses of HAS. First, the reasons for these weaknesses were investigated. Then we proposed FEAST and ALIVE architectures for Video on Demand and Live TV services respectively.

FEAST is proposed for avoiding unfair, instable and inefficient HAS service by getting certain feedback information from the server. In FEAST, the server keeps the minimal information about the clients which are number of clients, average video bitrates and average bandwidth measurements. Here we note that, the state variables are selected such that they are not client specific and can be easily computed to achieve a scalable system. We evaluated the performance of FEAST by simulations via NS2. We analyzed the performance of FEAST for different number of clients, different amount of background traffic and different number of video qualities. It is seen that clients can decide their video bitrates in much more fair and efficient manner only by considering those values and their buffer status when it is compared to MSS and Liu at al. Moreover, switching rate is much less than other two HAS implementations so that client QoE is not degraded with frequent quality changes. Furthermore FEAST utilizes the available link bandwidth upto 95 percent. In our future work, we will apply FEAST to multi-server environments where the content is encoded in more than one servers. Another issue to be considered is local link failures which will limit the rate adaptation capabilities of the clients. If the local link of a client cannot supply the necessary bandwidth, the client cannot increase its rate even if other clients decrease their rates. Clients should not drop their video bitrates due to a local link failure of another client. We will also incorporate the support for videos that are encoded at different rate levels with a weighted approach to maintain the fairness.

We proposed ALIVE for live TV services. ALIVE is based on caching the video data in clients which are called seeds and give the other clients the opportunity to download the videos from those seeds instead of the server. Scalable Video Coding is used for rate adaptation of the clients who download the contents from other clients but not from the server. We simulated ALIVE and it is seen that load of the server decreases with this solution. Therefore, the system can serve to more clients than the normal case in which clients always download from the server. Moreover, clients can get a better QoE because of downloading from a closer neighbour instead of a far server. We analyzed the performance of ALIVE for different number of clients, number of channels and number of clients behind NAT. In our future work, we will extend ALIVE to the case where clients behind NAT can also be seeds by NAT traversal solutions. Moreover, we will work on to combining FEAST and ALIVE to give a fair and stable service with higher video qualities for live TV streaming.

REFERENCES

[1] C. Systems, “Cisco visual networking index: Forecast and methodology, 2012-2017,” May 2013.

[2] “Youtube:,” http://www.youtube.com, [last accessed on 20/08/2013].

[3] “Hulu:,” http://www.hulu.com, [last accessed on 18/08/2013].

[4] “Move Networks:,” http://www.movenetworks.com, [last accessed on 18/08/2013].

[5] “Microsoft Smooth Streaming:,” http://go.microsoft.com/?linkid=9682896, [last accessed on 18/08/2013].

[6] “Apple HTTP live Streaming:,” http://tools.ietf.org/id/draft-pantos-http-live-streaming-04.txt, [last accessed on 18/08/2013].

[7] J. Postel, “Internet Protocol,” RFC 791 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1349, 2474, 6864.

[8] A.-V. T. W. Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications,” RFC 1889 (Proposed Standard), Internet Engineering Task Force, Jan. 1996, obsoleted by RFC 3550.

[9] “ Technical Specification Group Services and System Aspects (2006), IP Multimedia Subsystem (IMS), Stage 2, TS 23.228, 3rd Generation Partnership Project.”

[10] S. Deering, “Host extensions for IP multicasting,” RFC 1112 (INTERNET STANDARD), Internet Engineering Task Force, Aug. 1989, updated by RFC 2236.

[11] A. C. Begen, T. Akgul, and M. Baugher, “Watching video over the web part 1: Streaming protocols,” *Internet Computing, IEEE*, vol. 15, no. 2, pp. 54–63, March-April 2011.

[12] “The Network Simulator NS-2,” http://www.isi.edu/nsnam/ns/, [last accessed on 20/08/2013].

[13] H. Schwarz, D. Marpe, and T. Wiegand, “Overview of the scalable video coding extension of H.264/AVC standard,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 9, pp. 1103–1120, 2007.

[14] C. Burmeister, R. Hakenberg, A. Miyazaki, J. Ott, N. Sato, and S. Fukunaga, “Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback: Results of the Timing Rule Simulations,” RFC 4586 (Informational), Internet Engineering Task Force, Jul. 2006.

[15] W. Fenner, “Internet Group Management Protocol, Version 2,” RFC 2236 (Proposed Standard), Internet Engineering Task Force, Nov. 1997, updated by RFC 3376.

[16] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session Initiation Protocol," RFC 2543 (Proposed Standard), Internet Engineering Task Force, Mar. 1999, obsoleted by RFCs 3261, 3262, 3263, 3264, 3265.

[17] I. Baronak and L. Kockovic, "Alternatives of providing IPTV using IMS," *International Journal of Computers and Technology*, vol. 3, no. 2, 2012.

[18] "Video Bitrate Calculator," http://web.forret.com/tools/videofps.asp, [last accessed on 12/08/2013].

[19] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, 2003.

[20] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun. 1999, updated by RFCs 2817, 5785, 6266, 6585.

[21] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen, "What happens when HTTP adaptive streaming players compete for bandwidth?" in *Proc. ACM Int. Wksp. Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2012.

[22] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia streaming via TCP: An analytic performance study," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, no. 2, pp. 16:1–16:22, May 2008.

[23] I. Sodagar, "The MPEG-DASH standard for multimedia streaming over the internet," *MultiMedia, IEEE*, vol. 18, no. 4, pp. 62–67, 2011.

[24] "Overview of MPEG-DASH Standard:," http://dashif.org/mpeg-dash/, [last accessed on 25/08/2013].

[25] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive HTTP streaming," in *Proceedings of the second annual ACM conference on Multimedia systems*, ser. MMSys '11, 2011, pp. 169–174.

[26] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz, "Adaptation algorithm for adaptive streaming over HTTP," in *Packet Video Workshop (PV), 2012 19th International*, May, pp. 173–178.

[27] R. Houdaille and S. Gouache, "Shaping HTTP adaptive streams for a better user experience," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12, 2012, pp. 1–9.

[28] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '12, 2012, pp. 97–108.

[29] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback control for adaptive live video streaming," in *Proceedings of the second annual ACM conference on Multimedia systems*, ser. MMSys '11, 2011, pp. 145–156.

[30] N. Bouten, S. Latre, W. V. de Meerssche, K. D. Schepper, B. D. Vleeschauwer, W. V. Leekwijck, and F. D. Turck, "An autonomic delivery framework for HTTP adaptive streaming in multicast-enabled multimedia access networks." in *NOMS*. IEEE, 2012, pp. 1248–1253.

[31] C. Liu, I. Bouazizi, and M. Gabbouj, "Parallel adaptive HTTP media streaming," in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, 2011, pp. 1–6.

[32] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic, "Parallel TCP sockets: Simple model, throughput and validation," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April, pp. 1–12.

[33] Y. Sanchez, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. D. Vleeschauwer, W. V. Leekwijck, and Y. L. Louedec, "Efficient HTTP-based streaming using scalable video coding," *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 329 – 342, 2012.

[34] A. Kantarci, N. Ozbek, and T. Tunali, "Rate adaptive video streaming under lossy network conditions," *Signal Processing: Image Communication*, vol. 19, no. 6, pp. 479 – 497, 2004.

[35] B. Gorkemli and A. M. Tekalp, "Adaptation strategies for MGS scalable video streaming," *Signal Processing: Image Communication*, vol. 27, no. 6, pp. 595 – 611, 2012.

[36] P. Papadimitriou and V. Tsaoussidis, "A quality adaptation scheme for internet video streams," in *Wired/Wireless Internet Communications*, ser. Lecture Notes in Computer Science, F. Boavida, E. Monteiro, S. Mascolo, and Y. Koucheryavy, Eds. Springer Berlin Heidelberg, 2007, vol. 4517, pp. 165–176.

[37] R. Rejaie, M. Handley, and D. Estrin, "Layered quality adaptation for internet video streaming," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 12, pp. 2530–2543, 2000.

[38] S. S. Savas, C. G. Gurler, A. M. Tekalp, E. Ekmekcioglu, S. Worrall, and A. Kondoz, "Adaptive streaming of multi-view video over P2P networks," *Signal Processing: Image Communication*, vol. 27, no. 5, pp. 522 – 531, 2012.

[39] S. Benno, J. O. Esteban, and I. Rimac, "Adaptive streaming: The network HAS to help," *Bell Lab. Tech. J.*, vol. 16, no. 2, pp. 101–114, Sep. 2011.

[40] N. Cranley, P. Perry, and L. Murphy, "User perception of adapting video quality," *Int. J. Hum.-Comput. Stud.*, vol. 64, no. 8, pp. 637–647, Aug. 2006.

[41] R. K. Mok, E. W. Chan, X. Luo, and R. K. Chang, "Inferring the QoE of HTTP video streaming from user-viewing activities," in *Proceedings of the first ACM SIGCOMM workshop on Measurements up the stack*, ser. W-MUST '11, 2011, pp. 31–36.

[42] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems," DEC-TR-301, Digital Equipment Corporation, Tech. Rep., Sep. 1984.

[43] T. Lohmar, T. Einarsson, P. Frojdh, F. Gabin, and M. Kampmann, "Dynamic adaptive HTTP streaming of live content," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, 2011, pp. 1–8.

[44] A. Muller, G. Carle, and A. Klenk, “Behavior and classification of NAT devices and implications for NAT traversal,” *Network, IEEE*, vol. 22, no. 5, pp. 14–19, 2008.

[45] M. Jain and C. Dovrolis, “Pathload: A measurement tool for end-to-end available bandwidth,” in *In Proceedings of Passive and Active Measurements (PAM) Workshop*, 2002, pp. 14–25.

[46] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk Jiri Navratil, and L. Cottrell, “pathChirp: Efficient available bandwidth estimation for network paths,” in *PAM 2003, 4th Passive and Active Measurement Workshop*, R. Ritke, T. McGregor, and J. Micheel, Eds., NLANR/MNA. San Diego, CA, USA: UCSD, Apr. 2002.

[47] E. Goldoni, G. Rossi, and A. Torelli, “Assolo, a new method for available bandwidth estimation,” in *Internet Monitoring and Protection, 2009. ICIMP ’09. Fourth International Conference on*, 2009, pp. 130–136.

[48] J. Strauss, D. Katabi, and F. Kaashoek, “A measurement study of available bandwidth estimation tools,” in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, ser. IMC ’03. New York, NY, USA: ACM, 2003, pp. 39–44.

[49] N. Hu, S. Member, P. Steenkiste, and S. Member, “Evaluation and characterization of available bandwidth probing techniques,” *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 879–894, 2003.

[50] J. Navratil and R. L. Cottrell, “ABwE:a practical approach to available bandwidth estimation,” in *4th Passive and Active Measurements Workshop San Diego, CA, USA*, 2003.

[51] A. Botta, A. Davy, B. Meskill, and G. Aceto, “Active techniques for available bandwidth estimation: Comparison and application,” in *Data Traffic Monitoring and Analysis*, ser. Lecture Notes in Computer Science, E. Biersack, C. Callegari, and M. Matijasevic, Eds. Springer Berlin Heidelberg, vol. 7754, pp. 28–43.

[52] C. D. Guerrero and M. A. Labrador, “On the applicability of available bandwidth estimation techniques and tools,” *Comput. Commun.*, vol. 33, no. 1, pp. 11–22, Jan. 2010.

[53] E. Goldoni and M. Schivi, “End-to-end available bandwidth estimation tools, an experimental comparison,” in *Proceedings of the Second international conference on Traffic Monitoring and Analysis*, ser. TMA’10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 171–182.

[54] M. Jain and C. Dovrolis, “End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput,” *Networking, IEEE/ACM Transactions on*, vol. 11, no. 4, pp. 537–549, 2003.

[55] L. D’Acunto, J. Pouwelse, and H. Sips, “A measurement of NAT and Firewall characteristics in peer to peer systems,” in *Proceedings of 15th ASCI Conference*, 2009, pp. 1–5.

[56] R. Huysegems, B. De Vleeschauwer, T. Wu, and W. Van Leekwijck, “SVC-based HTTP adaptive streaming,” *Bell Labs Technical Journal*, vol. 16, no. 4, pp. 25–41, 2012.

[57] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain, “Watching television over an ip network,” in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, ser. IMC ’08. New York, NY, USA: ACM, 2008, pp. 71–84.

[58] T. Qiu, Z. Ge, S. Lee, J. Wang, J. Xu, and Q. Zhao, “Modeling user activities in a large iptv system,” in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ser. IMC ’09. New York, NY, USA: ACM, 2009, pp. 430–441.

[59] U. Oh, S. Lim, and H. Bahn, “Channel reordering and prefetching schemes for efficient iptv channel navigation,” *Consumer Electronics, IEEE Transactions on*, vol. 56, no. 2, pp. 483–487, 2010.

[60] “Zipf’s law,” http://en.wikipedia.org/wiki/Zipf’s_law, [last accessed on 20/08/2013].

[61] L. Vu, I. Gupta, K. Nahrstedt, and J. Liang, “Understanding overlay characteristics of a large-scale peer-to-peer iptv system,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 6, no. 4, pp. 31:1–31:24, Nov. 2010.

[62] R. Kooij, K. Ahmed, and K. Brunnstrom, “Perceived quality of channel zapping,” in *Fifth IAESTED Intern Conf on Communication Systems and Networks (CSN 2006), Aug 28-30, 2006, Palma de Mallorca, Spain; Proc. Pp*, 2006, pp. 155–158.