EFFICIENT AND FAIR ADAPTIVE STREAMING: ALGORITHM, IMPLEMENTATION AND EVALUATION

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

BY

AHMET ÖGE

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

JUNE 2017

Approval of the thesis:

EFFICIENT AND FAIR ADAPTIVE STREAMING: ALGORITHM, IMPLEMENTATION AND EVALUATION

submitted by **AHMET ÖGE** 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. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Tolga Çiloğlu
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Şenan Ece Güran Schmidt
Supervisor, **Electrical and Electronics Eng. Dept., METU** _____

Examining Committee Members:

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. Şenan Ece Güran Schmidt
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Electrical and Electronics Engineering Department, METU _____

Prof. Dr. Ahmet Coşar
Computer Engineering Department, METU _____

Assist. Prof. Dr. Ulaş Beldek
Mechatronic Engineering Department, Çankaya University _____

Date: **June 19, 2017**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: AHMET ÖGE

Signature :

ABSTRACT

EFFICIENT AND FAIR ADAPTIVE STREAMING: ALGORITHM, IMPLEMENTATION AND EVALUATION

Öge, Ahmet
M.S., Department of Electrical and Electronics Engineering
Supervisor : Assoc. Prof. Dr. Şenan Ece Güran Schmidt

June 2017, 68 pages

HTTP Adaptive Streaming (HAS) is a popular video streaming method where the client downloads video segments over standard HTTP protocol. In HAS, the server stores the video segments that are encoded in different qualities which determine the video bit rates. To this end, the client first downloads a file which describes the video segments. Then, using a rate adaptation algorithm, the client decides on the most appropriate video bit rate for the next segment to download and sends an HTTP request for that segment. The rate adaptation algorithm utilizes measurements of the network bandwidth by dividing the previously downloaded segments' sizes by their download times. HAS exploits that HTTP is an ubiquitous application layer protocol which can easily pass any network device, firewall and Network Address Translation.

Video streaming performance is measured by the user's perception that is quantified by Quality of Experience (QoE). Accordingly, video freezes must be avoided as they decrease QoE significantly. The client aims for downloading at the highest quality utilizing the available bandwidth as much as possible. However, if the requested bit

rate is increased too much, delays and packet loss events drive the client to decrease the bit rate subsequently. Such frequent rate switches decrease the QoE. Furthermore, it is desired that fairness among the clients is preserved where the clients that stream over a common bottleneck link share the bandwidth fairly.

In this thesis, we provide an Efficient and Fair Adaptive STreaming (EFAST) architecture to improve the performance of HAS according to the performance metrics that are defined above. In this architecture, clients rate adaptation is implemented by using a Fuzzy Logic Controller. The inputs of EFAST Fuzzy Logic Controller are the receiver buffer size and the estimated bandwidth. After fuzzy control steps, it selects a proper video bit rate of next segment. An analytical model of rate adaptation algorithm is defined to show that EFAST achieves the desired bit rate and buffer occupancy. We implement EFAST in both simulation environment and in real life network. We then perform experiments that evaluate the performance of EFAST in comprehensive network scenarios. Furthermore, we compare EFAST to other well-known HAS rate adaptation algorithms. Our results show that EFAST has more fairly bandwidth allocation among clients who share bottleneck, low switch rate changes, and high bottleneck efficiency with no buffer depletion.

Keywords: HTTP based Adaptive Streaming, Quality of Experience , Stability, Fairness, Efficiency

ÖZ

VERİMLİ VE ADALETLİ UYARLANABİLİR VİDEO AKIŞI: ALGORİTMA, GERÇEKLEŞTİRİM VE DEĞERLENDİRME

Öge, Ahmet
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Şenan Ece Güran Schmidt

Haziran 2017 , 68 sayfa

HTTP Uyarlanabilen Akış (HAS) İstemcinin video segmentleri standart HTTP protokol üzerinden indiren popüler bir video akış yöntemidir. HAS'da sunucu videonun bit hızını belirleyen farklı kalitede kodlanmış video segmentlerini depolar. Bu amaçla, istemci ilk olarak video segmentleri tanımlayan dosyayı indirir. Ardından, bir hız uyarlama algoritması kullanarak, istemci bir sonraki segmenti indirmek için en uygun video bit hızına karar verir ve o segment için HTTP isteği gönderir. Hız uyarlama algoritması daha önce indirilen segmentlerin boyutlarını indirme zamanına bölen ağ bant genişliği ölçümleri kullanır. HAS HTTP'nin herhangi bir ağ aygıtı, güvenlik duvarı ve Ağ Adres Çevirici'sini kolayca geçirebilen yaygın bir uygulama katman özelliğinden faydalanır.

Video akış performansı Deneyim Kalitesi (QoE) tarafından belirtilen kullanıcının algılamasıyla ölçülür. Buna göre, Deneyim Kalitesini önemli derecede azaltan video donmalarından kaçınılmalıdır. İstemci mevcut bant genişliği mümkün olduğunca kullanarak en yüksek kalitede indirmeyi hedeflemektedir. Ancak, eğer ki istenen video

bit hızı çok fazla artıp ardından hemen düşerse deneyim kalitesini azaltan sık oran değişikliği meydana gelir. Dahası, ortak darboğaz linki paylaşan istemciler arasında adaletin korunması istenmektedir.

Bu tezde yukarda belirtilen HAS'ın performans metriklerini yükseltmek için Verimli ve Adil Uyarlanabilen Akış (EFAST) mimarisi sağlıyoruz. Bu yapıda, istemcilerin hız uyarlama algoritması Bulanık Mantık Kontrolcusu (FLC) kullanarak gerçeklenmektedir. Bulanık Mantık Kontrolcusu alıcı arabellek boyutu ve bant genişliği tahmini olmak üzere iki girdi alır. Bulanık kontrol adımlardan sonra, bir sonraki segmentinin uygun video bit hızını seçer. Hız uyarlama algoritmasının analitik modeli performans gelişimini gösterecek şekilde tanımlanmıştır. EFAST'ın değerlendirtme sonuçları, diğer HTTP tabanlı Uyarlanabilir Akış çözümleri ile karşılaştırılması sunulmuştur. Ayrıca, simülasyon ortamında ve gerçek ağ ağında EFAST'ın performans değerlendirilmesini inceliyoruz. Deneyler gösteriyor ki EFAST hiçbir video donması yaşanmadan aynı darboğaz kanalı paylaşan istemciler arasında adaletli, az video bit oranı değişikliği ve yüksek kanal verimi elde etmiştir.

Anahtar Kelimeler: HTTP tabanlı Uyarlanabilen Akış, Deneyim Kalitesini, Kararlılık, Adillik, Verim

To My Love

ACKNOWLEDGMENTS

I would like to express my great appreciations to my supervisor Assoc. Prof. Dr. Şenan Ece Güran Schmidt for her support and guidance throughout this thesis. I am thankful for her guidance, which was very helpful in my research and writing of the thesis.

I would also like to thank TÜBİTAK-BİDEB for their financial support during my graduate education. In addition, I wish to thank ASELSAN A.Ş. for giving me the opportunity of continuing my education. I wish to thank my colleagues and seniors in the software design department.

I would like to express my special appreciation to Ahmet Emrah Demircan for his contributions to improve my engineering skills. I would like to special thanks to my family. Even if they could not be with me all the time, I can feel their best-wishes. Finally, I would like to express my appreciation to my friends Erham Mergen, Sena Alpaslan and Murat İlter. During thesis work, they helped me in every matter and they didn‘t leave me alone.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

IP	Internet Protocol
UDP	User Datagram Protocol
RTP	Real-time Transport Protocol
RTCP	Real-time Transport Control Protocol
NAT	Network Address Translator
HAS	HTTP Adaptive Streaming
CDN	Content Distribution Network
TCP	Transmission Control Protocol
HTTP	Hyper-Text Transfer Protocol
MSS	Microsoft Smooth Streaming
MPEG	Moving Pictures Experts Group
DASH	Dynamic Adaptive Streaming over HTTP
MPD	Media Presentation Description
URL	Uniform Resource Locator
AIMD	Additive Increase Multiple Decrease
QoS	Quality of Service
QoE	Quality of Experienc
NS2	Network Simulator 2
BD	Buffer Depletion
JF	Jain Fairness
UF	Unfairness Index
SC	Switch Change
FLC	Fuzzy Logic Controller
ELASTIC	fEedback Linearization Adaptive STreamIng Controller
FEAST	Feedback Based Adaptive Streaming over HTTP
PANDA	Probe AND Adapt
EFAST	Efficient and Fair Adaptive STreaming

CHAPTER 1

INTRODUCTION

In the fast developing communications of the 21^{st} century, intensity of network traffic has been dramatically increasing where video traffic has the highest increase rate. [5] states that annual global IP traffic is about 1.1 ZB per year. In addition, IP video traffic is 70 percent of entire IP traffic. For increasing video traffic, changing network substructure is inefficient and expensive. Thus, people work on miscellaneous solutions about possibility of storage and transfer of video data on network.

One proposal is Video IP multicast [14]. In this approach, video data is dispatched in multicast mode for increased bandwidth efficiency. However, it leads to some other problems like addressing, dynamic registration, multicast routing and forwarding. In order to solve multimedia video delivery problem, various protocols have been studied. Real-Time Transport Protocol [11] and Real-Time Transport Control Protocol [10] are the most popular protocols among them. RTP which is running on top of the UDP is developed for end to end audio and video transfer. RTCP which is working with hand in hand with the RTP, provides security by sending several controlling packages. Even though these two protocols are used commonly, for real-time service, they do not guarantee quality of service. Also because of using UDP they cannot pass barriers like firewall or NAT.

Recently, the most popular multimedia streaming solution is HTTP based Adaptive Streaming (HAS) [28]. In the application layer, it uses HTTP. Since HTTP is commonly used, it can pass firewall and NAT easily. It benefits from infrastructure of existing content delivery networking (CDN) proxy and cache due to using standard HTTP servers. Standard HTTP servers do not track any state and all states are re-

alized by client. Servers store video segments in different quality levels. Quality of a video segment is defined with the encoding bit rate. Clients can adjust quality of video segments and time delay of requesting new segments as they wish according to computation and adaptation based on network load. To this end, the clients implement some rate adaptation algorithm which makes use of clients' measurements of the network bandwidth. Such measurements are based on the download time and the size of the downloaded segments. Therefore, the rate adaptation algorithm runs at client side independently of server. Lately, companies like Apple, Adobe and Microsoft pay attention to HAS studies. Apple HTTP Live Streaming [2], Adobe HTTP Dynamic Streaming [1] and Microsoft Smooth Streaming [21] are some examples of commercial video tools. However, there is no standard between these HAS player. Segment formats and manifest format are not same; therefore there is no interoperability between devices and servers of various vendors. To solve this interoperability, MPEG defines standard file format for HAS so that any standard-based client can download video segments from any standard HTTP server. Thus, MPEG-DASH was developed [22] which is one of the international standard for HAS. MPEG determines format of MPD file which describes components such as type of codecs, encoded video bit rate and manifest file.

Since HTTP works on TCP, it is under the influence of TCP's feedback based data rate adjustment. Therefore, HAS clients may not perceive the network load correctly and may not select proper quality of the video segments and the time of requesting new segments. The video played by a HAS client should not freeze so buffer should not be empty anytime. Additionally, it should not download too much video data, therefore receiver buffer should not be overloaded. Quality of the downloaded video segments should switch as infrequently as possible. To this end, we identify three performance metrics to investigate in this thesis for HAS service. The first one is efficiency which implies downloading video segments at the possible highest bit rate (High available bandwidth Efficiency). The second is stability which implies that the bit rate of the downloaded segments switch as infrequently as possible. The third is fairness where all clients that download over a common bottleneck link share the link bandwidth as fair as possible. Here we note that these targets contradict with each other.

There are many studies about HTTP based Adaptive Streaming which propose different rate adaptation algorithms. Most of these studies follow heuristic methods. In addition, they evaluate their performance under basic network topologies only.

In this thesis study, we develop a systematic method that we call Efficient Fair Adaptive STreaming (EFAST). EFAST is a client side rate adaptation algorithm which adheres to all MPEG-DASH standards. EFAST try to improve the performance metrics that we define above. To this end, EFAST implements a fuzzy-control based rate adaptation algorithm. Different than, heuristic methods in the literature, we show that EFAST achieves the desired system state under specific conditions. We evaluate EFAST with network simulator 2 [23] and dummynet emulation [3] under a number of different network scenarios. Our experimental results show that EFAST provides high bandwidth efficiency, infrequent video rate switches and fair bandwidth allocation among clients.

The remainder of the thesis is outlined as follows, In Chapter 2, we provide background information about HAS service and Fuzzy control. In Chapter 3, we give literature research of HAS and previous work on HAS. In Chapter 4, we present EFAST architecture in detail. In Chapter 5, we give experimental research of our solution and compare other HAS players. In Chapter 6, we summarize the thesis and explain future work.

CHAPTER 2

BACKGROUND

2.1 HTTP Adaptive Streaming

HTTP based Adaptive Streaming (HAS) enables high quality streaming of media content over the Internet delivered from conventional HTTP web server. In HAS, the standard HTTP server stores small video segments of same length in different video resolutions (bit rates). The client first downloads a description file about the stored video segments. Then, the client selects the resolution of each segment and sends a standard HTTP GET request to download it.

MPEG-DASH [27] is a standard that describes the format of the description file that is called Media Presentation Description (MPD) file. Furthermore it provides media related functions such as segment formats, ad insertion and synchronization. [22] lists main aims of HAS services such as efficient delivery of MPEG media over HTTP, support of live streaming of multimedia content, ease of use of existing content distribution infrastructure components such as caches, proxies, CDNs, NATs and firewalls, support of integrated services with multiple components, support for signaling, delivery, utilization of multiple content protection and support for efficient content forwarding.

Segment sizes vary between 2 to 10 seconds in general. Each segment is playable independently. Therefore, clients can play segments with different qualities. MPD consists of more than one periods. Each of those periods contains timing information and media components such as codec, resolution, audio components for various languages, subtitles etc. This information is arranged in adaptation sets. Each period can

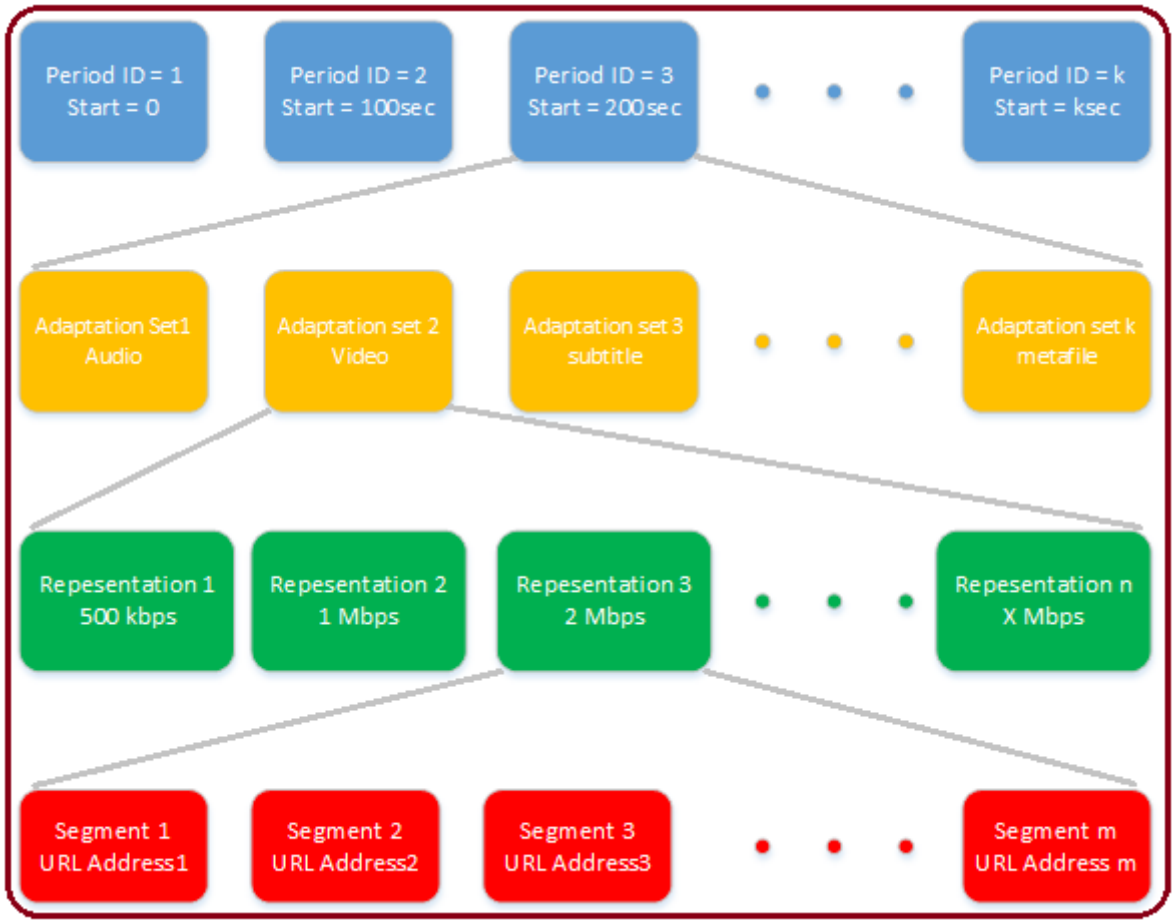

Figure 2.1: Example of Media Presentation Description File Organization

support one or more adaptation sets which group media components. Adaptation sets are divided into representations according to video quality. Representations consist of the URLs and byte range for each accessible segment which is requested by clients. In Figure 2.1, example of MPD file is given.

After downloading a segment, a client estimates available bandwidth by dividing segment size into downloading time and chooses the next video bit rate according to estimated bandwidth. When amount of data in the client's receiver buffer is low, the client sends the request for a new segment immediately. However, when the buffer is full, the client sends new segment request after some seconds which is called *idle waiting time*. Idle waiting time causes ON-OFF traffic of HAS applications. Clients determine next requested video quality according to the *rate adaptation algorithm*

to improve Quality of Experience (QoE) which is the user's perception of the media experience. To this end, watching the video without any freezes, achieving the maximum possible rate and maintaining the stability of the video streaming with minimum number of changes in the rate (rate switches) is important. In addition the network bandwidth should be used as efficiently as possible with a fair allocation among the streaming users on the bottleneck link. Here we note that MPEG [27] does not provide a rate adaptation algorithm. We will discuss several rate adaptation algorithms in Chapter 3.

2.1.1 HTTP Adaptive Streaming Issues

We identify the following three issues that motivate this thesis in accordance with the previous literature on HAS.

The first issue is unfair bandwidth allocation among clients which share the same bottleneck link. One would expect that HAS clients share bottleneck link fairly since HTTP uses TCP. TCP connections share bottleneck bandwidth fairly thanks to the property of additive increase multiple decrease (AIMD). However, because of ON-OFF traffic behavior of HAS services, this fair share of clients is not realized. In the ON period, clients download video segments from the server. However, if the receiver buffer is full or exceeds certain level, clients stop downloading segments. If one of the clients is in the OFF period, the other clients increase usage of bottleneck link which is shared among other clients. When the client switches to ON period, it starts to download video segment from server. In this case, clients do not share equally bottleneck link. The duration of OFF period is changed according to different rate adaptation algorithm. This OFF period is called idle waiting time. End of this section, we will give a scenario and investigate how the idle waiting time affects quality of experience in HAS services.

The second issue is the inaccuracy of the estimated bandwidth calculation. In HAS services, estimated bandwidth is calculated by dividing downloaded segment size into downloading time in general. Therefore, clients learn available bandwidth when each video segment is downloaded. Even if it appears to be working, the available bandwidth is not calculated correctly with this method. There are two reasons for this

problem to emerge. The first reason is that the segment duration is too long compared to network variability. In general, segment size is approximately between 2 to 10 seconds, thus clients calculate the estimated bandwidth or learn the available bandwidth in each 2 to 10 seconds intervals. This time duration is not enough to adapt to the network variability. If clients download video segments very fast compared to the segment duration, receiver buffer increases continuously which causes buffer overflow or ON-OFF traffics. The second reason is that while clients are in OFF period, they do not get any feedback information about network conditions. Once clients go to OFF period, they stop downloading video segments during idle waiting time. After idle waiting time duration finishes, they are starting to send HTTP Get requests to download a new segment. Since clients did not learn the available bandwidth during OFF period, they send new requests according to the past information. In addition, when one client is in OFF period, the other clients increase their bandwidth usage of bottleneck link. Once bandwidth usage increases, they increase video bit rate. However if a client that is in OFF period switches its state to ON period, the total download rate exceeds the channel capacity. Therefore, clients decrease requested video bit rate since downloading time of each segment increases. This situation causes frequent video quality changes which decreases Quality of Experience (QoE). In addition, it causes estimated bandwidth and instantaneous downloading rate fluctuations. When we consider average estimated bandwidth and instantaneous download bit rate, they do not converge to any level. They fluctuate around ideal average level.

We demonstrate the fluctuating behavior of instantaneous download bit rate and estimated bandwidth with an example in Figure 2.2. To this end, we implement Microsoft Smooth Streaming [21] in Network Simulator 2 [23]. In this example scenario, there are 8 HAS (Microsoft Smooth Streaming) clients sharing a link with the bandwidth of 8Mb/s. Figure 2.2 shows that the average instantaneous video bit rates and the average estimated bandwidths over all clients with respect to time. There is a continuous intersection between the curve of averaged estimation of bandwidths and the curve of average download bit rate. Download bit rate follows estimated bandwidth curve. When instantaneous download bit rate is lower than estimated bandwidth, clients increase their video bit rates. Once total instantaneous download rate exceeds available bottleneck link capacity, downloading time of each segment increases as packet loss

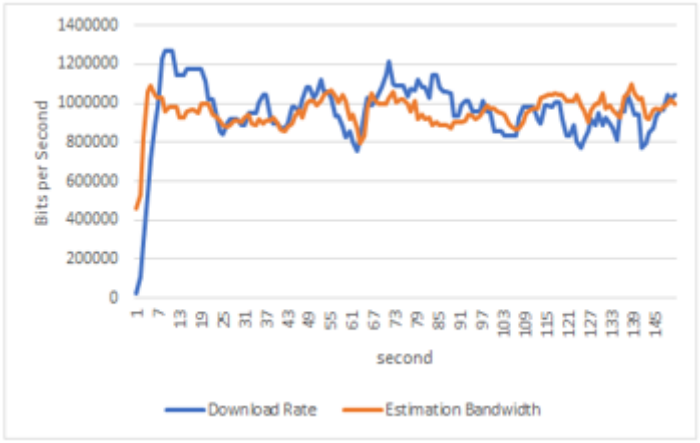

Figure 2.2: Download rate and estimated bandwidth of 8 clients

and packet delay occur. Since downloading time increases, estimated bandwidth decreases. Here we note that estimated bandwidth is calculated as dividing the downloading segment size into downloading time. When estimated bandwidth decreases, clients decrease their video bit rate. If total instantaneous downloading rate is under the available bottleneck capacity, estimated bandwidth increases. Therefore, fluctuations between instantaneous download bit rate and estimated bandwidth constantly occur.

Considering first and second issues mentioned above together, we demonstrate how ON-OFF traffic affects performance of HAS services in scenario. The table 2.1 shows ON-OFF period of three HAS clients which share C bps bottleneck link. Download periods are not slotted in general. However, In order to understand effect of ON-OFF traffic, we consider simple scenario. In T1, all three clients are in ON period. Thus, all clients download segments with $C/3$ bps because of TCP fairness. In T2, client 1 goes to OFF period so the other clients download segments with $C/2$ bps. In T3, only client 3 downloads segments with C bps. At the beginning of T4 period, both client 1 and client 2 were starting to download a segment with the previous estimated bandwidth such as $C/3$ bps for client 1, $C/2$ bps for client 2. Moreover client 3 continues to download segment with C bps. Although bottleneck link capacity is C bps, total requested video rate is about $11C/6$ bps. Since total download rate exceeds available bottleneck capacity, packet loss and delay of segments increase. This situation causes estimated bandwidth and instantaneous download rate fluctuations. In T5, all clients

Table2.1: ON-OFF senario

	T1	T2	T3	T4	T5	T6
Client 1	ON	OFF	OFF	ON	OFF	ON
Client 2	ON	ON	OFF	ON	OFF	ON
Client 3	ON	ON	ON	ON	OFF	OFF

are OFF which means they stop downloading segments. Since bottleneck efficiency is zero in this time interval, average throughput over session decreases. Consider entire scenario, it is obvious that clients do not share bottleneck link fairly.

The third issue is selecting next requested video bit rate in an entirely heuristic way rather than designing rate adaptation algorithm with a systematic approach. We discuss various rate adaptation algorithms in Chapter 3. These works obtain high performance in specific scenarios. If network topology or any other parameters changes, performances of that solutions are affected significantly. For each network topology, optimizations of algorithms are needed. When designing HAS solution in systematic way, it keeps Quality of Experience under different network topology.

2.1.2 Constraints and Performance Metrics for HAS

In this part, we define the constraints and the performance metrics of the HAS, HTTP based adaptive streaming. The main constraint is that the buffer of the receiver must not be empty (buffer depletion). When the buffer is empty, video freezes constantly during the play that decreases the QoE, quality of experience. In HAS, clients select the download bit rate of the video from server. Downloading at highest rate possible resulting in best video quality, however increases the segment size in bits and the download time which might lead to buffer depletion under low bottleneck bandwidth. If there is a quality difference between clients, we face unfairness problem among clients sharing the same bottleneck. Furthermore frequent switches of the downloaded segment bit rates also decreases QoE.

To this end, a good HAS scheme should avoid buffer depletions, minimize the unfairness and rate switches, meanwhile maximizing the video download rate.

We next define these performance metrics:

Buffer Depletion:

Each HAS client has a buffer implemented to store downloaded video segments. Underflowing this buffer leads to total stop or occasional freezing during video play, which is a serious impact on a client's QoE. We define BD, buffer depletion, for the system as the number of buffer depletions per client over the session.

Bottleneck Bandwidth Efficiency:

The instantaneous download rate averaged over all connected clients is defined as:

$$d_a(t) = \frac{\sum_{i=1}^{u(t)} d_i(t)}{u(t)} \tag{2.1}$$

In equation (2.1), $u(t)$ defines the number of total clients which are downloading video segments and $d_i(t)$ defines current download rate of i^{th}client. To get high efficiency or high throughput, $u(t) \times d_a(t)$ should be close to capacity of bottleneck link between client and server $C(t)$. Accordingly, bottleneck bandwidth efficiency defined as:

$$\eta(t) = \frac{u(t) \times d_a(t)}{C(t)} \tag{2.2}$$

Rate Switch:

Clients are sensitive to frequent video download rate changes. If frequent rate changes occur in a short amount of time, quality of server decreases significantly. The rate switch is defined as the number of video bit rate variations over the session.

Fair bandwidth allocation among the clients:

If there is a quality difference between clients, bandwidth is shared unevenly. JF(t) [12], instantaneous fairness, is termed as:

$$JF(t) = \frac{(\sum_{i=1}^{u(t)} r_i(t))^2}{u(t) \times \sum_{i=1}^{u(t)} (r_i(t))^2} \tag{2.3}$$

$u(t)$ is number of total clients online to the server and $r_i(t)$ is current video bit rate

of the client i. We define unfairness index derived from Jain's fairness index [12] as follows:

$$UF(t) = 1 - JF(t) \tag{2.4}$$

Unfairness index $UF(t)$ can vary between zero and one. When clients share available bandwidth fairly, $UF(t)$ close to zero. That also means, when the clients share the available bandwidth unfairly, $UF(t)$ becomes closer to one.

2.2 Fuzzy Logic Basics

2.2.1 Fuzzy Logic

Fuzzy Logic, was developed by Lütfü Aliasker Zade, is a form of logic which is different from the classical logic approach [30]. Fuzzy logic is based on fuzzy set and subset. In the classical approach, an entity is an element of a set or not. When it is defined mathematically, if an entity is element of the set, it takes value of '1' and if the entity is not element of the set, it takes value of '0'. Fuzzy logic is extension of classical set representation. In fuzzy set, all entities have a degree of membership. Entity's degree of membership can be any value between (0,1) and it is shown with membership function.

Fuzzy sets are generally used for definition of social concepts which are unclear and do not have certain and definitive values because it is impossible to categorize many magnitudes and expressions with definite boundaries. For instance, if there is a rule such as people under 30 years of age are young, in classical logic, both a 29 year old person and a 20 year old person are in young category without any difference. However, in fuzzy logic, they have different degrees in young category.

Fuzzy logic is used commonly in systems not having digital input similar the way humans think. It is also used in systems which do not have a mathematical models and it is difficult to create a mathematical model. For example, it gives concrete results in nonlinear systems and solution of undefined problems.

2.2.2 Fuzzy Logic Controller

Fuzzy Logic Controllers, unlike classical and modern control theories, do not need certain and definite mathematical models. The fundamental point of Fuzzy Logic Controllers is generating information, experience, intuition and control strategy of an expert system operator as a database in controller design. Control activities are realized according to verbal term rules of information and experience. For instance, if an expert defines control attitudes which are necessary for system as terms like 'small', 'fast' and 'slow', rules, composed of (IF-THEN) commands, will be derived by using verbal term concepts.

Fuzzy control system is composed of four basic units which are Fuzzification, Knowledge base, Inference engine and Defuzzification. Fuzzification element comes into play as a first unit of fuzzy control system. Data, in form of precise and feedback results, is fuzzificated in this unit by changing its scale [15]. Data, came in Inference engine unit, are combined with data of knowledge base like if-then-else rules and fuzzificated data. In here, mentioned logical proposition according to structure of problem can also be built with numerical values. In the last step, results, which are derived by using logical decision propositions which are suitable for structure of problem, are sent to Defuzzification unit. Each fuzzy data is converted to real number by again changing scale in relations of fuzzy set which are sent to defuzzification unit.

CHAPTER 3

PREVIOUS WORK ON HAS

One of the oldest study of HAS is proposed in the article [18] by Liu at al. They provide a novel rate adaptation algorithm which measures available bandwidth based on segment fetch time. In this algorithm, it uses the smoothed HTTP throughput to select video bit rate. In addition, it aims to quickly adapt to match the available bandwidth and video bit rates by using a set-wise increase and aggressive decrease method. This algorithm calculates network capacity by dividing media segment duration into segment fetch time instead of using any transport layer information such as round trip time and packet loss rate. According to network capacity, it increments video bit rate in step-wisely and decrements video bit rate aggressively. Moreover, it calculates idle waiting time to prevent buffer depletion. Although, simulation result shows that their rate adaptation algorithm quickly adapts video bit rate to available bandwidth, they simulate only one client. Thus, fairness issue is not considered.

In [7], ELASTIC proposes feedback linearization to select next video bit rate by avoiding buffer overflow and underflow. ELASTIC is designed based on feedback control theory. It keeps size of buffer in certain level to prevent buffer depletion. In addition, it improves fairness by avoiding on-off traffic generation. Although experiment results show that ELASTIC gets high performance such as fairness and channel utilization compared to PANDA and FESTIVE, magnitude of video quality change is high. Therefore, range of encoded video bit rates increase, there is no guarantee of ELASTIC's high performance.

Jiang et al. [13] provide FESTIVE algorithm which consists of harmonic bandwidth estimator, delayed bit rate update mechanism and randomized scheduler. FESTIVE

calculates available bandwidth reliably by using harmonic mean of previous 20 successful segment downloads. According to available bandwidth, it changes video quality gradually which implies that video player changes its video quality to the next higher or lower level. In addition, it schedules the timing of next HTTP request randomly to solve unfairness problem of clients which are sharing the same bottleneck link. Experimental results show that FESTIVE performs higher performance such as fairness, efficiency and stability compared to other video players such as Microsoft smooth streaming (MSS), Netflix and Akamai. Although delayed bit rate update and harmonic bandwidth estimator is more reliable bandwidth estimation method providing stability of video quality, it cannot adapt to the dynamic network conditions. Therefore, when available bandwidth changes frequently, performance of FESTIVE significantly decreases.

In [16], Li et al. presents PANDA (Probe AND Adapt) which is client-side rate adaptation algorithm. It uses similar idea of TCP congestion control in application layer. PANDA calculates available download rate by probing video data. After that, it selects video bit rate and time delay until next segment request sends such that available bandwidth and average data downloading times are equal. Experimental results show that PANDA obtains high performance such as stability and fairness. However, channel efficiency is not maximized. Since there is a time delay between two consecutive segment requests, bottleneck efficiency decreases.

The paper [24] proposes feedback based Adaptive Streaming over HTTP (FEAST). FEAST enables clients to determine more precise video bit rates. In addition, it provides fair usage of bandwidth among the clients, avoiding frequent video rate changes and maintaining a high video quality by enabling clients to utilize the available bandwidth as much as possible. Clients determine the requested video bit rates according to feedback information from server side. In FEAST, there are 6 algorithms which are improved in heuristic way. Although these algorithms are not based on mathematical theory, the simulation results show that FEAST fully utilizes the available bandwidth, therefore, provides a fairer bandwidth allocation among the clients, which results in less switch rate changes and buffer depletion compared to the other HAS architectures such as MSS [21] and the algorithm of Liu et al [18]. Although FEAST improves the Quality of Experience such as efficiency, fairness and stability, there

are two problems. Firstly, FEAST uses heuristic way to improve their performance. It chooses some parameters which are used in FEAST algorithms according to their network topology and encoded video bit rates. Therefore, these parameters depend on network topology and encoded video bit rate. If network topology which is used in FEAST simulation part is changed, the performance of FEAST could decrease. Second problem is that FEAST algorithms are based on feedback information from the HTTP server. Thus, since a server does not work as a standard HTTP server anymore, the profit which standard HTTP protocol provides becomes meaningless.

Huang et al. [9] provides buffer-based rate adaptation algorithm which chooses next video bit rate based on buffer level and capacity estimation if it is necessary. In this model, there are two operation phases. In steady-state phase, clients choose next video rate by considering only playback buffer level. In startup state, clients select next video rate by using the capacity estimation. Although, it gets high performance in constant bandwidth scenario, the performance significantly decreases when available bandwidth between server and client varies frequently.

Miller et al. [20] propose a rate adaptation algorithm that chooses next video rate based on network conditions. While the algorithm avoids buffer underflow over session, it maximizes average and minimum video bit rate. In addition, it minimizes startup delay and number of video bit rate changes. when It takes two inputs such as amount of data in buffer and available bandwidth, it creates output as next requested video bit rate and time delay for sending new HTTP request. They evaluate adaptation algorithm in real network environment. The evaluation results show that the adaptation algorithm perform similar performances in various scenarios. However, they do not consider fairness issue over clients which share bottleneck link. Therefore, the performance of adaptation algorithm decreases in concurrent clients scenario compared to single client scenario.

Cicco et al. [8] states the model of the automatic stream-switching controller which is the mathematical model of video streaming service. It describes dynamic behavior of the control system which takes two inputs such as estimated bandwidth and buffer level. It tries to select video bit rate as an available bandwidth which changes dynamically. In this model, there are two controllers such as buffer controller and

stream-switching controller. Buffer controller sends control information about player buffer length from client to server. In addition, Stream-switching controller sends control information combining amount of player buffer and measured bandwidth to server. When server receives these control information, it selects suitable video quality. The results of experiments show that players immediately adapt video rate to bottleneck link capacity. Although various scenarios are considered, there is only one player is simulated. When one or more players are simulated, unfairness problem occurs. Despite the mathematical model is clearly indicated, the HTTP server is not standard server since the server receives and processes control information. Thus, it loses the benefit of standard HTTP properties.

Liu et al. [18] provides a novel rate adaptation algorithm which measures available bandwidth based on segment fetch time. In this algorithm, it uses the smoothed HTTP throughput to select video bit rate. In addition, it aims to quickly adapt to match the available bandwidth and video bit rates by using a set-wise increase and aggressive decrease method. This algorithm calculates network capacity by dividing media segment duration into segment fetch time instead of using any transport layer information such as round trip time and packet loss rate. According to network capacity, it increments video bit rate in step-wisely and decrements video bit rate aggressively. Moreover, it calculates idle waiting time to prevent buffer depletion. Although, simulation result shows that their rate adaptation algorithm quickly adapts video bit rate to available bandwidth, they simulate only one client. Thus, fairness issue is not considered.

Vergados et al. [29] provides a fuzzy logic based rate adaptation over the MPEG-DASH standard. They design fuzzy logic controller which is one of the most important application of fuzzy logic. In this controller, it takes two inputs such as buffering time and differential of the buffering time then creates output as available channel throughput. By using output of fuzzy logic controller, the rate adaptation algorithm chooses next video bit rate such that it tries to keep buffering time to certain target level in order to avoid buffer depletion. Moreover, in order to avoid video bit rate fluctuations, it does not change video bit rate when the buffer level exceeds or eludes target buffer level for 60 seconds. Although, the idea of combining fuzzy logic controller and rate adaptation of HTTP based adaptive streaming was quitted in 2014,

it needs more evaluations in various scenarios to decide that FDAST obtains high performances such as stability and efficiency. In addition, the authors state that the simulation result shows that FDASH obtains fairness among clients however there is no performance metrics about fairness issue.

Another fuzzy logic based rate adaptation algorithm is stated in [26]. Sobhani et al. suggests fuzzy logic controller which provides minimum ON-OFF traffic and maximum network utilization. In this model, it takes two inputs such as throughput and amount of buffer. Since it considers both network characteristic and buffer level, simulation result shows that it gets higher performance compared to FDASH algorithm [29]. In addition, they measured estimated bandwidth by using exponentially weighted moving average for avoiding unnecessary video bit rate fluctuations. Moreover, it adjusts idle waiting time according to network characteristic and buffer level. Thanks to fuzzy logic controller, it minimizes negative effects of ON-OFF behavior and obtains high performance such as high bandwidth efficiency and low switch rate. However, because of ON-OFF traffic, there is still unfairness problem among clients which share the same bottleneck link.

In Table 3.1 we summarize solution of HAS according to name of algorithm, approach and target performance.

Table3.1: Summary of HAS Solutions

Algorithm, year	Approach	Target performance metrics	Comments
[18],2011	Heuristic, Buffer based	Efficiency, Stability	Poor evaluation
ELASTIC [7], 2013	Feedback Control theory, Buffer based	Fairness, Efficiency, Stability	
[9], 2015	Heuristic, Buffer based	Buffer depletion, Stability	
[20],2012	Heuristic, Buffer based, Bandwidth based	Stability, Efficiency	
[8],2014	Mathematical model, Buffer based, Bandwidth based	Fairness, Efficiency, Stability	Server modified
FDASH[29],2014	Fuzzy logic, Buffer based	Stability	Poor evaluation
[26], 2015	Fuzzy logic, Buffer based, Bandwidth based	Efficiency, Stability	Poor evaluation
FEAST [24],2014	Heuristic, Buffer based, Bandwidth based	Fairness, Efficiency, Stability	Server modified
FESTIVE [13], 2012	Heuristic,Bandwidth based, Buffer based,	Fairness, Efficiency, Stability	
PANDA [16], 2014	Heuristic,Bandwidth based, Buffer based,	Fairness, Efficiency, Stability	
EFAST, 2017	Fuzzylogic,Bandwidth based, Buffer based,	Fairness, Efficiency, Stability	

CHAPTER 4

PROPOSED EFAST (EFFICIENT AND FAIR ADAPTIVE STREAMING) ALGORITHM

In this section, we will provide a solution of HTTP adaptive streaming with a systematic approach meanwhile using standard HTTP protocol that we call EFAST (Efficient and Fair Adaptive STreaming). EFAST provides a client side rate adaptation algorithm that conform MPEG-DASH standards. EFAST improves the performance and quality of experience of HTTP based adaptive streaming by providing high bandwidth efficiency, low buffer depletion, infrequent rate switch and fairness. Network state such as available bandwidth, traffic load, routing paths and packet delay vary all the time, thus modeling HTTP adaptive streaming process becomes complex and difficult. We design a fuzzy logic based rate adaptation controller since it has three advantages compared to conventional control method. To begin with, numerous fuzzy rules are necessary to implement a fuzzy logic controller, it is the exact opposite of conventional control system which has a single control strategy to determine the action. Therefore fuzzy logic controllers are more suitable to depict systems which may be complex, nonlinear or both. Furthermore, control becomes more robust, since, it has multiple strategies resulting in more resilience to single errors. Lastly, modeling of the control strategy is done considering linguistic terms resulting in an easier representation of human knowledge. To sum up, we propose EFAST, fair and efficient adaptive streaming built on fuzzy logic controller.

4.1 Model and Operation

HTTP Adaptive Streaming is one of the client-server multimedia streaming applications that the client downloads video from standard HTTP server which contains small video segments of same lengths in different video resolutions. In general length of the segment varies between 2 to 10 seconds. Client starts to download the segment with the lowest video bit rate. By using download time and segment size, client estimates available bandwidth. Then client requests video bit rate according to rate adaptation algorithm which directly affects QoE. Thus, we design rate adaptation mechanism by considering the system requirements and quality of experience together. In our design, we completely eliminate the idle waiting time for two reasons which are obtaining high bottleneck bandwidth efficiency and fairness among clients similar to the work [7]. Firstly, since during idle waiting time clients do not download any segments, it causes a decrease in the throughput of session. In other words, clients do not use network resources during idle waiting time which leads to a decrease in bandwidth efficiency. Secondly, we have discussed that idle waiting time leads to unfairness bandwidth allocation among clients. Thus, removing idle waiting time provides fairness bandwidth allocation among clients thanks to TCP. Although removing the idle waiting time improves performance, buffer management becomes more difficult. Since the buffer is getting closer to getting full, there is no any other way to decrease buffer size except for increasing the time to download the segment. Thus, we consider buffer level to determine the next requested video bit rate. According to the buffer level, client adjusts video bit rate in order to avoid buffer depletion such as buffer overflow and buffer underflow. If client wants to decrease buffer level, it increases downloading segment time via increasing video bit rate. Naturally, if client wants to increase the buffer level, it decreases downloading segment time via decreasing video bit rate. Thus, the video bit rate jumps to a lower level which impacts quality of experience negatively. In addition, it also causes buffer underflow in the worst case. When client downloads the video under available bandwidth, it decreases throughput of session. Therefore, selecting next video bit rate according to buffer level is not enough, EFAST also considers available bandwidth information together so that client can play video with the maximum possible quality.

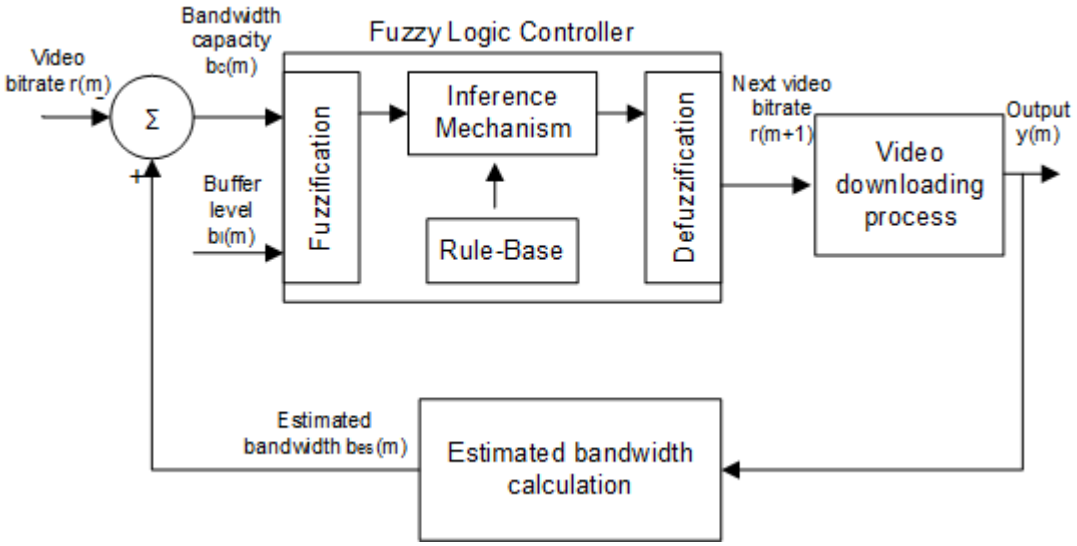

Figure 4.1: Model of EFAST

In our model, the client downloads the first segment with the lowest video bit rate and then goes on determining the next requested video bit rate by using fuzzy logic controller. To this end, we first define $S_{size}(m)$ and $t_{down}(m)$ as the size and download time of the last downloaded segment respectively. Accordingly, the instantaneous download rate for segment of (m) is $S_{size}(m)/t_{down}(m)$.

In Figure 4.1, we show the model of EFAST. Let $r(m)$, $r_{next(m)}$, $y(m)$, $b_{es}(m)$, $b_c(m)$ and $b_l(m)$ denote the system variables defined as follows:

$r(m)$: Downloading video bit rate for the last downloaded segment m (in bps)
$r(m+1)$: Next Requested video bit rate (in bps)
$y(m)$: Output of video downloading process; size of the selected and downloaded segment (in bits). This becomes the input for the estimated bandwidth calculation.
$b_{es}(m)$: Estimated bandwidth after downloading segment m (in bps)
$b_c(m) = b_{es}(m) - r(m)$: Bandwidth capacity (in bps). This parameter can take a negative value.
$b_l(m)$: Buffer level after segment m downloaded which shows the amount of data in the receiver buffer (in seconds)
$q(m)$ Output of fuzzy logic controller

Estimated bandwidth is calculated by using the output of video streaming process

over last downloaded w segments as follows:

$$b_{es}(m) = \frac{1}{w} \cdot \sum_{k=0}^{w} \frac{S_{size}(m-k)}{t_{down}(m-k)} \tag{4.1}$$

We use $w = 3$ for the rest of the thesis. Here we note that a larger w would decrease the rate switches however the achieved efficiency will be less if the bottleneck use is frequently changing.

Our fuzzy logic controller consists of four components, rule-base, interface mechanism, fuzzification and defuzzification:

- A rule-base (a set of If-Then rules), which contains a fuzzy logic quantification of the expert‘s linguistic description of how to achieve good control
- An inference mechanism (also called an "inference engine" or "fuzzy inference" module), which emulates the expert‘s decision making in interpreting and applying knowledge about how best to control the plant.
- A fuzzification, which converts controller inputs into information that the inference mechanism can easily use to activate and apply rules.
- A defuzzification, which converts the conclusions of the inference mechanism into actual output for the process.

In this fuzzy model, $r(m+1)$ is the output of fuzzy model which denotes next requested video bit rate. $b_c(m)$ and $b_l(m)$ are the fuzzy inputs. EFAST aims for maximizing bottleneck bandwidth efficiency and fairness among the multimedia streams which share the same bottleneck link while minimizing the occurrences of buffer depletion and rate switches.

In our model, we need to provide a description of how to best control the plant in some natural language. Firstly, we define a linguistic variable that describes varying fuzzy control inputs and output each time. Secondly, we recognize linguistic values of each input and output. Thirdly, we define membership functions to quantify meaning of the linguistic values. Fourthly, we explain fuzzy rules which determine the relation between linguistic variables of inputs and linguistic variables of output. Finally, we

recognize how to convert fuzzy output to next requested video bit rate. Let us define linguistic variables of our Fuzzy Logic Controller (FLC):

"Bandwidth Capacity" describes $b_c(m)$
"Buffer Level" describes $b_l(m)$
"Change Ratio" describes $q(m)$

Now, we describe linguistic values for inputs and output. Let us define $L(x)$ to represent the linguistic values of linguistic variable x. In this model, there are three linguistic variables and each variable map to five linguistic values.

$L(b_c(m))$ = { "Negative-Large", "Negative-Small", "Zero", "Positive-Small", "Positive-Large" }
$L(b_l(m))$ = { "Empty", "Low", "Medium", "High", "Full" }
$L(q(m))$ = { "Decrease-Large", "Decrease-Small", "No-Change", "Increase-Small", "Increase-Large" }

Now let us define membership functions which quantify the meaning of linguistic values. These functions are clearly defined within the numerical range of fuzzy inputs and outputs. Membership functions are represented by $\mu(x)$ which takes a value between zero and one. Therefore, all input values are mapped to linguistic values which have values between zero and one. In Figure 4.2, it shows the membership functions for bandwidth capacity. Encoded video bit rate at server is denoted by r_i where $1 <= i <= n$. while r_1 represents the lowest video bit rate, r_n represents maximum video bit rate. In addition, r_k denotes current video bit rate where $r(m) = r_k$ and $1 <= k <= n$. R denotes maximum difference between consecutive encoded video bit rate at server. We define design variables $n1, n2, p1, p2$:

$$R = \{max((r_{k+1}) - r_k) | n > k \geq 1\}$$
$$p1 = \{(r_{k+1}) - r_k | n > k, R | n = k\}$$
$$p2 = \{(r_{k+2}) - r_k | (n-1) > k, 2 \cdot R | (n-1) \leq k\}$$
$$n1 = \{(r_{k-1}) - r_k | 1 < k, -R | k = 1\}$$
$$n2 = \{(r_{k-2}) - r_k | 2 < k, -2 \cdot R | k \leq 2\}$$

Figure 4.3 shows the membership function of buffer level. Let us define t_{max} as the maximum size of buffer in client and design variables t_1, t_2, t_3, t_4, t_5 and t_6 are 50%,

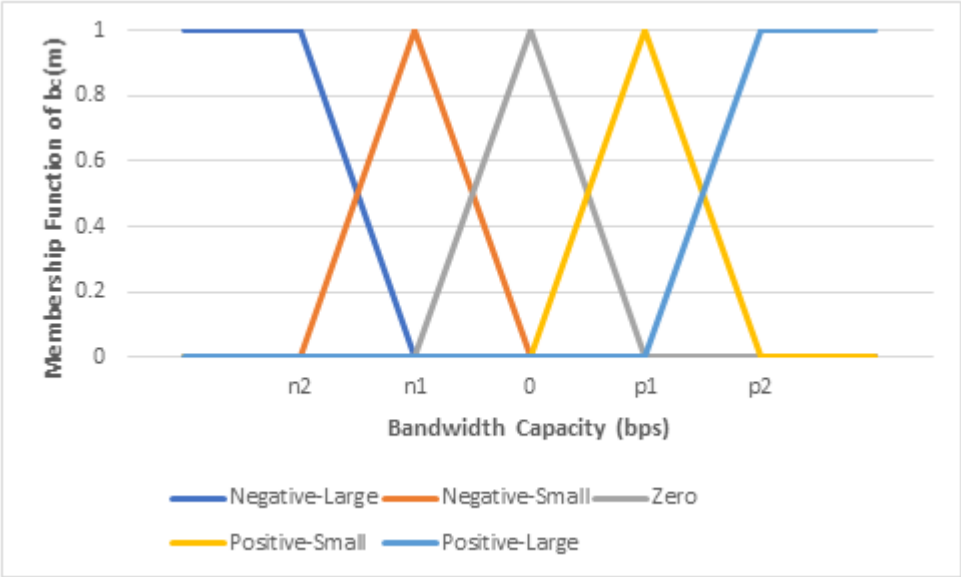

Figure 4.2: Membership Function of Bandwidth Capacity

60%,70%,80%,90% and 100% of t_{max} respectively. Figure 4.4 shows membership function of change ratio $q(m)$.

Now we need to specify a set of rules. (a rule-base) that captures the expert's knowledge about how to control the plant. In order to determine proper rules, the relation between fuzzy inputs and outputs are investigated. For simplicity, we analyze fuzzy inputs separately. The first fuzzy input is bandwidth capacity. When bandwidth capacity is either "Positive-Small" or "Positive-Large", the estimated bandwidth is larger than current video bit rate. Thus, client increases video bit rate in order to watch video as high quality as possible and maximize available bandwidth efficiency. When bandwidth capacity is either "Negative-Large" or "Negative-Small", the estimated bandwidth is lower than current video bit rate. In this case, since available bandwidth is overloaded, packet loss and delay significantly increases. Thus, downloading time of segment increases, amount of buffer decreases. In a worse case, buffer underflow occurs which significantly decreases quality of experience. For avoiding buffer underflow, when estimated bandwidth is lower than current download bit rate, client should decrease video bit rate. When bandwidth capacity is "Zero", estimated bandwidth and current video bit rate is almost equal. Therefore, client does not change the requested video bit rate for avoiding frequent rate switches. Second fuzzy input is buffer level

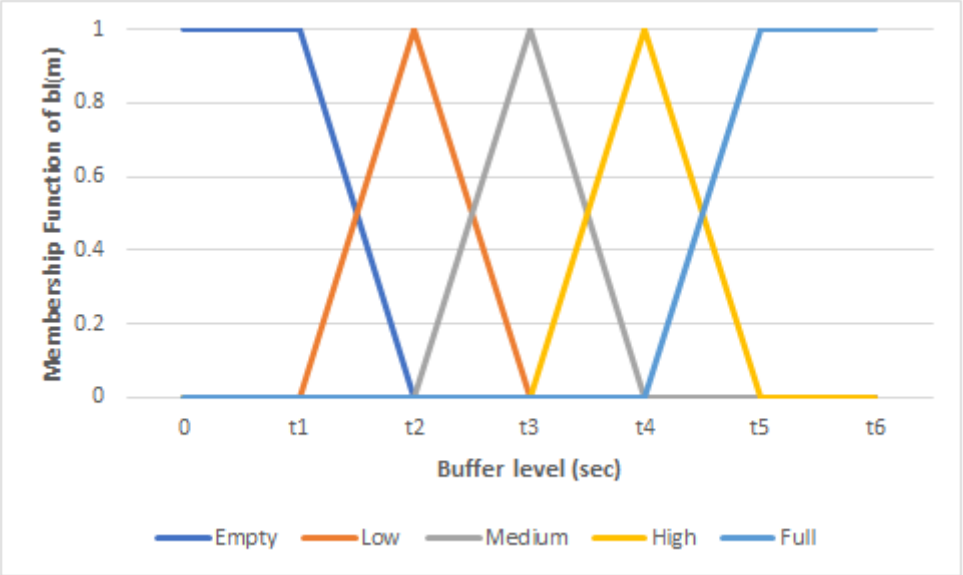

Figure 4.3: Membership function of buffer level

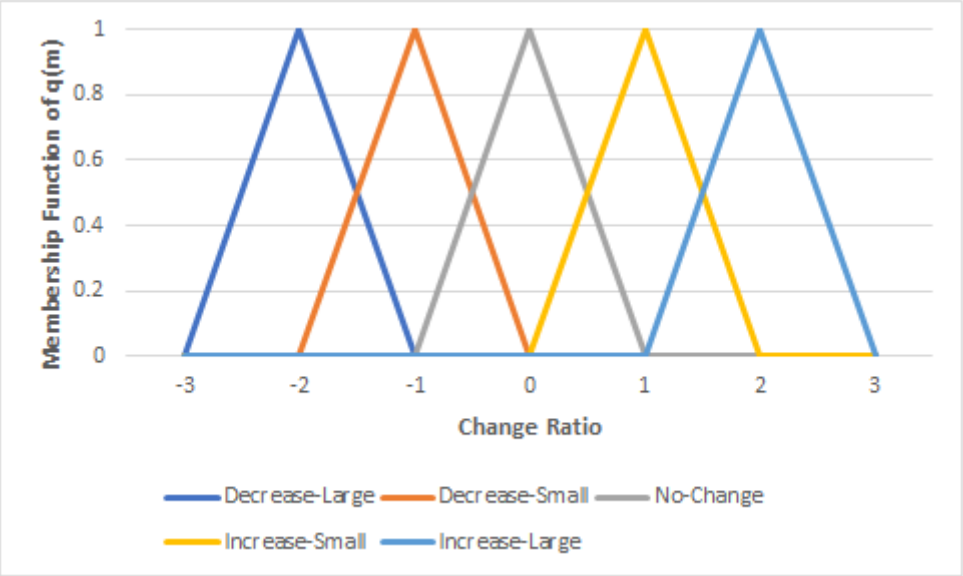

Figure 4.4: Membership function of change ratio

which represents amount of data in receiver buffer. We divide buffer level according the amount of data in receiver buffer such as "Empty", "Low", "Medium", "High" and "Full". When buffer level is either "Empty" or "Low", client should decrease video bit rate in order to avoid buffer underflow since downloading time is proportional to video bit rate. When buffer level is either "Full" or "High", client should increase the video bit rate for avoiding buffer overflow. Since we completely remove idle waiting time in our model, client continuously downloads video segments. To this end, the fairness is improved and we avoid buffer depletions by selecting video bit rate according to the buffer level. Therefore, client decreases its buffer level via increasing video bit rate. In this model, video bit rate is increased or decreased by 2 levels in maximum so that client avoids frequent rate switches and immediately adapts to dynamic network conditions. Since we only specify a finite number of linguistic variables and linguistic values, finite numbers of possible rules are also defined. In this fuzzy logic controller, with two inputs and five linguistic values for each of these, there are at most $5^2 = 25$ possible rules. We define 25 rules considering both bandwidth capacity and buffer level conditions. We summarize all rules in Table 4.1:

1. If buffer level is “Empty“ and bandwidth capacity is “Negative-Large“ than change ratio is “Decrease-Large“.
2. If buffer level is “Empty” and bandwidth capacity is “Negative-Small” than change ratio is “Decrease-Large”.
3. If buffer level is “Empty” and bandwidth capacity is “Zero” than change ratio is “Decrease-Large”.
4. If buffer level is “Empty” and bandwidth capacity is “Positive-Small” than change ratio is “Decrease-Small”.
5. If buffer level is “Empty” and bandwidth capacity is “Positive-Large” than change ratio is “No-Change”.
6. If buffer level is “Low” and bandwidth capacity is “Negative-Large” than change ratio is “Decrease-Large”.
7. If buffer level is “Low” and bandwidth capacity is “Negative-Small” than change ratio is “Decrease-Large”.

8. If buffer level is “Low” and bandwidth capacity is “Zero” than change ratio is “Decrease-Small”.
9. If buffer level is “Low” and bandwidth capacity is “Positive-Small” than change ratio is “No-Change”.
10. If buffer level is “Low” and bandwidth capacity is “Positive-Large” than change ratio is “Increase-Small”.
11. If buffer level is “Medium” and bandwidth capacity is “Negative-Large” than change ratio is “Decrease-Large”.
12. If buffer level is “Medium” and bandwidth capacity is “Negative-Small” than change ratio is “Decrease-Small”.
13. If buffer level is “Medium” and bandwidth capacity is “Zero” than change ratio is “No-Change”.
14. If buffer level is “Medium” and bandwidth capacity is “Positive-Small” than change ratio is “Increase-Small”.
15. If buffer level is “Medium” and bandwidth capacity is “Positive-Large” than change ratio is “Increase-Large”.
16. If buffer level is “High” and bandwidth capacity is “Negative-Large” than change ratio is “Decrease-Small”.
17. If buffer level is “High” and bandwidth capacity is “Negative-Small” than change ratio is “No-Change”.
18. If buffer level is “High” and bandwidth capacity is “Zero” than change ratio is “Increase-Small”.
19. If buffer level is “High” and bandwidth capacity is “Positive-Small” than change ratio is “Increase- Large”.
20. If buffer level is “High” and bandwidth capacity is “Positive-Large” than change ratio is “Increase-Large”.
21. If buffer level is “Full” and bandwidth capacity is “Negative-Large” than change ratio is “No-Change”.

Table4.1: Rule Table for the EFAST

Change Ratio q(m)		Bandwidth Capacity				
		Neg-Large	Neg-Small	Zero	Pos-Small	Pos-Large
Buffer Level	Empty	Dec-Large	Dec-Largee	Dec-Large	Dec-Small	No-Change
	Low	Dec-Large	Dec-Large	Dec-Small	No-Change	Inc-Small
	Medium	Dec-Large	Dec-Small	No-Change	Inc-Small	Inc-Large
	High	Dec-Small	No-Change	Inc-Small	Inc-Large	Inc-Large
	Full	No-Change	Inc-Small	Inc-Large	Inc-Large	Inc-Large

22. If buffer level is "Full" and bandwidth capacity is "Negative-Small" than change ratio is "Increase-Small".

23. If buffer level is "Full" and bandwidth capacity is "Zero" than change ratio is "Increase-Large".

24. If buffer level is "Full" and bandwidth capacity is "Positive-Small" than change ratio is "Increase-Large".

25. If buffer level is "Full" and bandwidth capacity is "Positive-Large" than change ratio is "Increase-Large".

Now we summarize how fuzzy logic controller works. In the fuzzification step, non-linear inputs bandwidth capacity and buffer level are mapped to linguistic values with values between 0 and 1 by using membership functions. Inference mechanism decides the current rules to apply by comparing controller inputs with the all the rules. This mechanism checks for every rule. Then, all the decisions are used to create a single conclusion by inference mechanism. After that, it combines all the recommendations from all the rules to determine the fuzzy output. In the defuzzification step, the latest controller output is provided by inference mechanism which is using implied fuzzy sets and combination of their effects. In our system, we use center of gravity defuzzification method to determine final output. In this method output of the fuzzy logic controller is calculated as:

$$q(m) = \frac{\sum_i \int_i b_i \mu(i)}{\sum_i \int_i \mu(i)} \tag{4.2}$$

Where b_i denotes the center of the membership function of output and $\mu(i)$ is the

implied fuzzy set. Since we use center of gravity method [25] for defuzzification, the result of $q(m)$ should be in range between -2 to 2. This output value does not describe directly the next requested video bit rate. Thus, we make decision based on fuzzy output by means of the following equation where encoded video bit rate at server are defines as r_1,r_2, r_{k-1}, r_k, $r_{k+1} \cdots r_n$ and current video bit rate is $r(m) = r_k$.

$$r(m+1) = \begin{cases} r_{k+2} & (q > 1.5)\&(n > k+1) \\ r_{k+1} & (q > 1.5)\&(n = k+1), \\ r_{k+1} & (1.5 \geq q > 0.5)\&(n > k), \\ r_k & (0.5 \geq q \geq -0.5)\&(n > k > 1), \\ r_k & (q > 0.5)\&(k = n), \\ r_k & (-0.5 > q)\&(k = 1), \\ r_{k-1} & (-0.5 > q \geq -1.5)\&(k > 1), \\ r_{k-1} & (-1.5 > q)\&(k = 2), \\ r_{k-2} & (-1.5 > q)\&(k > 2) \end{cases}$$

4.2 Convergence of EFAST

In this part, we show that, for a single client on the bottleneck link with bandwidth of C bps, EFAST converges to the desired video bit rate of $r(m) = C$ under some predetermined conditions. Encoded video bit rates at the server are represented by r_i where $1 \leq i \leq n$ such that $r_i < r_{i+1}$ when $1 \leq i < n$ and $r_n < 2C$. Maximum size of the receiver buffer is $20 \cdot S_d$ where S_d represents segment duration. Here 20 factor exists to prevent the buffer level jump two level.

Current video bit rate is $r(m) = r_k$ where $1 \leq k \leq n$. Let us assume that there exists a video bit rate r_m where $1 \leq m \leq n$ such that $r_m = C$. In addition, process delay, propagation delay and queuing delay at routers are negligible, since they very small compared to transmission delay of video packets. Size of HTTP GET request is also very small compared to size of video packet, so it is negligible too. If these assumptions are satisfied, the requested video bit rate converges to $r_m = C$ and buffer level converges to an interval between 60% and 80% of the maximum buffer level.

EFAST fuzzy logic controller has two inputs, which are bandwidth capacity and buffer level. Bandwidth capacity is defined as difference between estimated bandwidth $b_{es}(m)$ and current video bit rate $r(m) = r_k$. Client measures the instantaneous available bandwidth by dividing video segment size $S_s(m)$to the downloading time $t_{down}(m)$ which is the sum of propagation delay, processing delay, queuing delay and transmission delay. Since propagation, processing and queuing delay are negligible compared to the transmission delay, the downloading time is:

$$t_{down}(m) = \frac{S_s(m)}{C} \tag{4.3}$$

Estimated bandwidth 4.1 becomes $bes(m) = C$. Then we state bandwidth capacity as :

$$b_c(m) = C - r(m) \tag{4.4}$$

Since client plays a segment while downloading a segment, buffer level $b_l(m)$ (amount of data in receiver buffer in seconds) continuously changes. When a segment is downloaded, buffer level increases segment duration S_d. However, buffer level reduces by the time client sends HTTP GET request and downloads video segment. Since HTTP request message is negligible small compare to video segment size, we ignore time delay due to the sending HTTP request message. Moreover, propagation, process and queuing delay are also negligible compared to transmission delay of video segment. We calculate buffer level $b_l(m)$ when each segment is downloaded as following equation:

$$b_l(m) = b_l(m-1) + S_d - t_{down}(m) \tag{4.5}$$

Downloading time of segment is stated as equation in 4.3 when segment size is calculated:

$$S_s(m) = S_d \cdot r_k \tag{4.6}$$

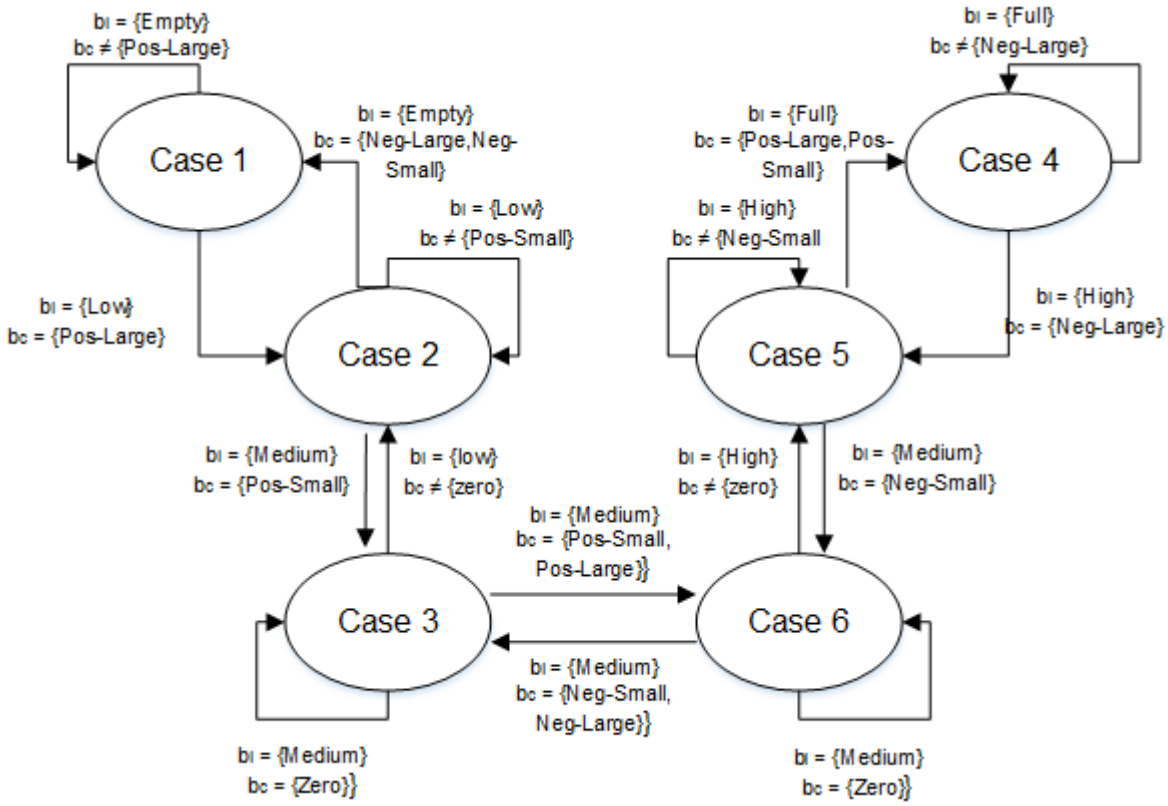

Figure 4.5: Transition Diagram of Cases

Consider equations 4.3, 4.5 and 4.6 to calculate buffer level as following equation:

$$b_l(m) = b_l(m-1) + S_d \cdot (1 - \frac{r_k}{C}) \tag{4.7}$$

4.7 implies that if $r(m) = r_k > C$, buffer level b_l decreases. Constantly, if current video bit rate $r(m) = r_k < C$, buffer level b_l increases. Since $0 < r_i < 2C$ where $1 \leq i \leq n$, the magnitude of buffer change is less than segment duration S_d for each segment downloaded.

In order to prove that the video bit rate converges to $r_m = C$, we need to analyze all the input combinations. We identify 6 different cases according to the buffer level: where t_1, t_2, t_3, t_4, t_5 and t_6 are 50%, 60%,70%,80%,90% and 100% of t_{max} respectively. Figure 4.5 shows transition diagram of cases.

Case1: $0 \leq b_l < t_1$

In this case, buffer is always "empty". When bandwidth capacity is "negative large", "negative small", "zero" or "positive small", the change ratio $q(m) < 0$ which indicates that video bit rate continuously decreases (see Table 4.1). Since video bit rate r_k decreases, bandwidth capacity converges to "positive large". When bandwidth capacity is "positive large", video bit rate becomes stable at r_i where $i > (k + 2)$ since since change ratio $q(m)$ is "no change" . However, since bandwidth capacity is "positive large" $b_c(m) > 0$ which indicates $C > r(m)$. Thus, from equation 4.7 amount of data in receiver buffer increases whenever new segment is downloaded. Buffer level becomes $t_1 \leq b_l < t_2$ (goes to case 2) . Although video bit rate is not changed, the amount of data in receiver buffer increases. Therefore, there is no stable state with respect to both video bit rate and buffer size. Note that before bandwidth capacity converges to "positive large", buffer level goes to case 2. In any case, case 1 always goes to case 2.

Case2: $t_1 \leq b_l < t_2$

In this case, buffer level is either "empty" or "low". When bandwidth capacity is either "negative large", "negative small" or "zero", change ratio $q(m) < 0$ which indicates that video bit rate continuously decreases (see Table 4.1) similar to case 1. In this case, since video bit rate decreases, bandwidth capacity shifts to either "positive small" or "positive large" according to buffer level. When amount of data in receiver buffer is close to point t_1, bandwidth capacity converges to "positive large" since dominated buffer level is "empty". Although video bit rate does not change, amount of data in the buffer increases since $r(m) < C$ bps (see equation 4.7) Therefore, buffer level shifts from t_1 to t_2. When buffer level is close to t_2, the buffer level whose linguistic dominated buffer level has a larger membership value becomes low. Bandwidth capacity goes from "positive large" to "positive small". Once bandwidth capacity converges to "positive small", video bit rate becomes stable since change ratio $q(m)$ is "no change". However, since amount of buffer are continuously decreases, buffer level shifts to case 3 which is in interval $t_2 \leq b_l < t_3$. In addition, there is a possibility that before bandwidth capacity converges to "positive small", buffer level shifts to case 3.

Case3: $t_2 \leq b_l < t_3$

In this case, buffer level is either "low" or "medium". When bandwidth capacity is either "negative large" or "negative small", change ratio $q(m) < 0$ which indicates that video bit rate continuously decreases (see Table 4.1). In this case, since video bit rate decreases, bandwidth capacity shift to either "zero" or "positive small" according to buffer level. When amount of data in buffer gets closer to point t_2, bandwidth capacity converges to "positive small" since dominated buffer level is "low". Although video bit rate does not change, amount of data in buffer increases since $r(m) < C$ (see equation 4.7). Therefore, buffer level shifts to point t_3 from t_2. When buffer level gets closer to t_3, dominated buffer level becomes "Medium". Bandwidth capacity goes to zero from "positive small". When bandwidth capacity is zero, following inequality should be satisfied from figure 4.3:

$$n_1 < b_c < p_1 \tag{4.8}$$

We know that n_1, p_1 and b_c from equation 4.5:

$$(r_{k-1} - r_k) < C - r_k < (r_{k+1} - r_k) \tag{4.9}$$

Then we get:

$$r_{k-1} < C < r_{k+1} \tag{4.10}$$

This condition is satisfied only when $r_k = r_m$ since we assume that there exists a video bit rate r_m such that $r_m = C$.

From equation 4.5, when current video bit rate equal to C bps, amount of data in receiver buffer does not change. Buffer level is medium indicates that $t_2 < b_l < t_4$ Since video bit rate and amount of data does not changed, the system becomes stable which is video bit rate converges to r_m and buffer level converges to interval between 60% and 80%of the maximum buffer level.

Case4: $t_5 < b_l \leq t_6$

In this case, buffer level is always full. When bandwidth capacity is, "negative small",

"zero", "positive small" or "positive large", the change ratio $q(m) > 0$ which indicates that video bit rate continuously increases (see Table 4.1). Since video bit rate $r(m)$ increases, bandwidth capacity converges to "negative large". When bandwidth capacity is "negative large", video bit rate becomes stable which is not changed since change ratio $q(m)$ is "no change" . However, since $b_c(m) < 0$ which indicates $C < r(m)$. Thus, from equation 4.7 amount of data in receiver buffer decreases whenever new segment is downloaded. Buffer level goes to case 5 which is in interval $t_4 \leq b_l < t_5$. Although video bit rate is not changed, amount of data in receiver buffer decreases. Therefore, there is no stable state with respect to both video bit rate and buffer size. Note that before bandwidth capacity converges to "negative large", buffer level goes to case 5. In any case, case 4 always goes to case 5.

Case5: $t_4 < b_l \leq t_5$

In this case, buffer level is either "high" or full. When bandwidth capacity is either "positive large", "positive small" or "zero", change ratio $q(m) > 0$ which indicates that video bit rate continuously increases (see Table 4.1) similar to case 4. In this case, since video bit rate increases, bandwidth capacity shifts to either "negative small" or "negative large" according to buffer level. When amount of data in receiver buffer is close to t_5, bandwidth capacity converges to "negative large" since dominated buffer level is full. Although video bit rate does not change, amount of data in buffer increases since $r(m) > C$ bps (see equation 4.7). Therefore, buffer level shifts from t_5 to t_4. When buffer level is close to t_4, dominated buffer level becomes "high". Bandwidth capacity goes to from "negative large" to "negative small". Once bandwidth capacity converges to "negative small", video bit rate becomes stable. However, since amount of buffer are continuously increases, buffer level shifts to case 4 which is in interval $t_3 \leq b_l < t_4$. In addition, it is possible that before bandwidth capacity converges to "negative small", buffer level shift to case 4.

Case6: $t_3 < b_l \leq t_4$

In this case, buffer level is either "high" or "medium". When bandwidth capacity is either "positive large" or "positive small", change ratio $q(m) > 0$ which indicates that video bit rate continuously increases (see Table 4.1). In this case, since video bit rate increases, bandwidth capacity shifts to either "zero" or "negative small" according

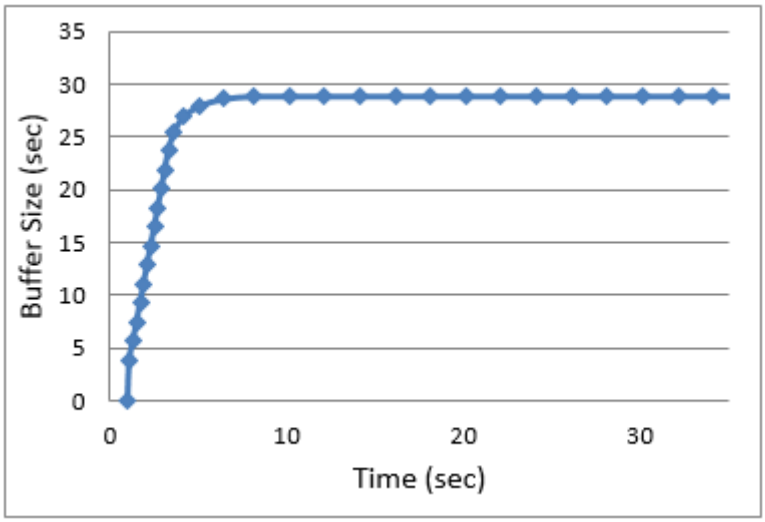

Figure 4.6: Buffer Size of Client

to buffer level. When amount of data in buffer is close to t_4, bandwidth capacity converges to "negative small" since dominated buffer level is "high". Although video bit rate does not change, amount of data in buffer decreases since $r(m) > C$ bps (see equation 4.7). Therefore, buffer level shifts from t_4 to t_3. When buffer level is close to t_3, dominated buffer level becomes Medium. Bandwidth capacity goes from negative small to zero. When $bc = 0$, the system becomes stable which video bit rate converge to r_m and buffer level converges to interval between $t_3 = 60\%$ and $t_4 = 80\%$ of maximum buffer level like case 3

We demonstrate the convergence of EFAST with an example simulation experiment in NS2 [23]. In this setup, there is one video streaming server and one client that is connected directly over a bottleneck link with a constant $C = 900$ Kbps. The receiver buffer size is 40 seconds, segment duration is 2 seconds and encoded video bit rate r_1 to r_{21} are 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900 and 2000 kbps. Client starts downloading the video at $t = 1$ second. We investigate buffer size and video bit rate. Figure 4.6 and Figure 4.7 show that buffer size and video bit rate of client respectively. Initially, client starts downloading the video segment with the lowest video bit rate. Since buffer level is empty and bandwidth capacity is positive large, video bit rate stays constant (case1). During this interval, buffer level increases since video bit rate

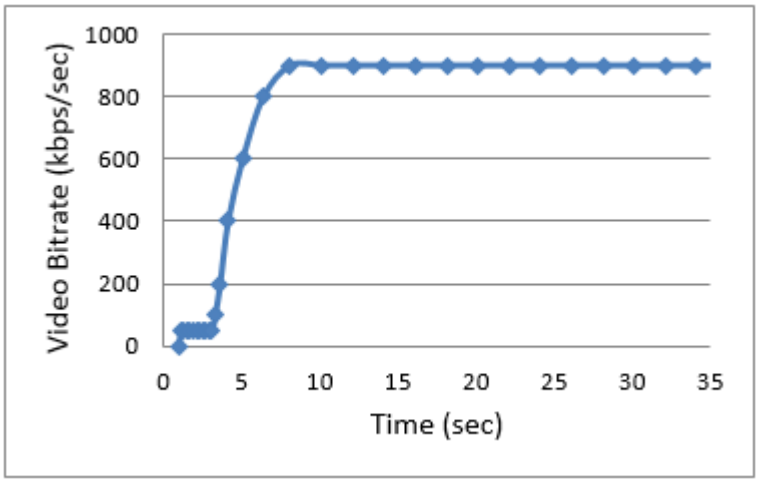

Figure 4.7: Video Bit rate of Client

$r_k < 900kps$. When buffer level goes to low, video bit rate starts to increase (case2). Since there exists video bit rate as equal to bottleneck link and receiver buffer size is larger than $20 \cdot S_d$, the video bit rate converges to 900 kbps and buffer level converges to 28 seconds which is a value between 60% and 80% of the maximum size of receiver buffer (case 3). Although, video bit rate and buffer level converge to the point that we expected, we know that it does not converge to stable state in real life. Therefore in Chapter 5, we will analyze performance of EFAST in detail.

CHAPTER 5

EVALUATION OF EFAST ARCHITECTURE

5.1 Experiments and Results

In this chapter, we evaluate the performance of EFAST in various scenarios which includes both simulation and real network environment. Firstly EFAST will be analyzed with a constant bottleneck bandwidth C and a constant number of clients N. Note that because of TCP fairness, each client that shares the bottleneck link gets C/N bandwidth which is a main factor in the performance of EFAST. After that, N EFAST clients will be simulated in different C/N ratios. We will investigate how C/N ratio affects performance such as fairness, efficiency and rate switch frequency. Thirdly, EFAST will be evaluated in real network environment with constant bottleneck link. Fourthly, we will evaluate varying bottleneck link scenario in simulation environment. We will investigate how EFAST adjusts video bit rate under dynamic network conditions and how it affects performance in detail. Finally, we will compare performances of EFAST to other HAS solutions such as, ELASTIC, FESTIVE and PANDA in constant bottleneck scenario, in different number of client scenario, and in TCP backgrounds flow scenario.

5.1.1 Experiment 1: Performance of FEAST Under Constant Bottleneck with 2 Clients

In this experiment, we will investigate 2 clients which are sharing constant bottleneck link $C = 2$Mbps scenario in NS2 [23] network simulator. Each client has 40 seconds of receiver buffer. Encoded video bit rates are selected in real MPEG

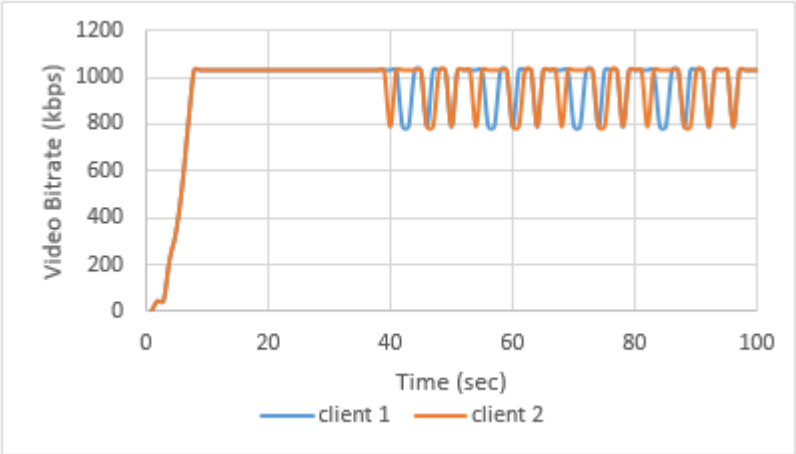

Figure 5.1: Video bit rate of client 1 and client 2

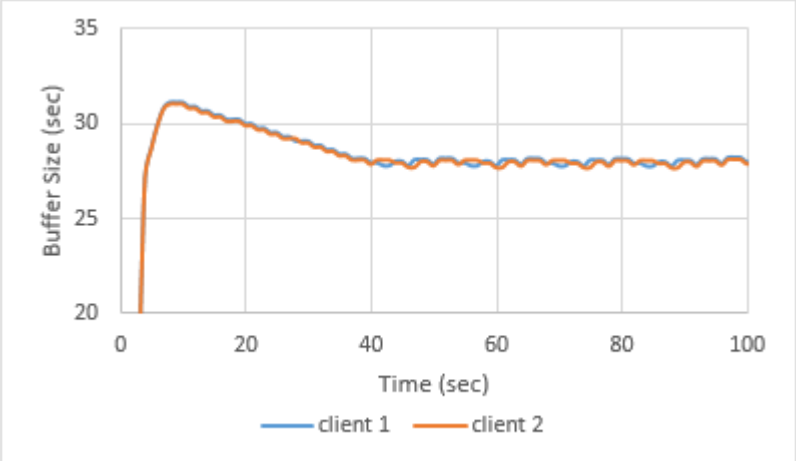

Figure 5.2: Buffer size of client 1 and client 2

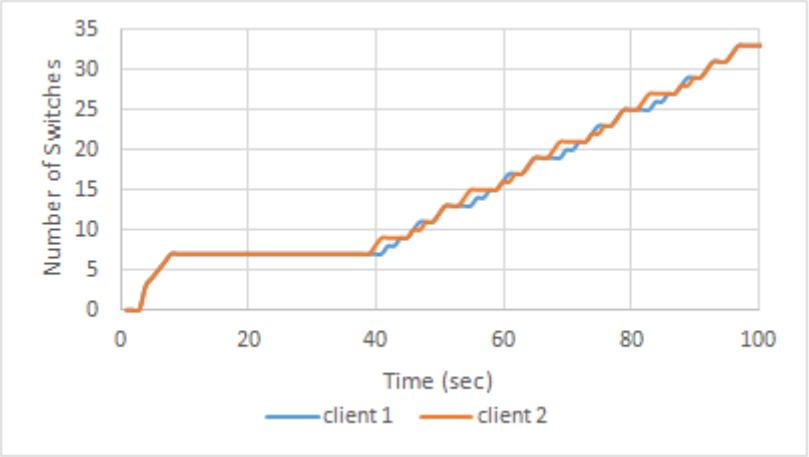

Figure 5.3: Number of switches of client 1 and client 2

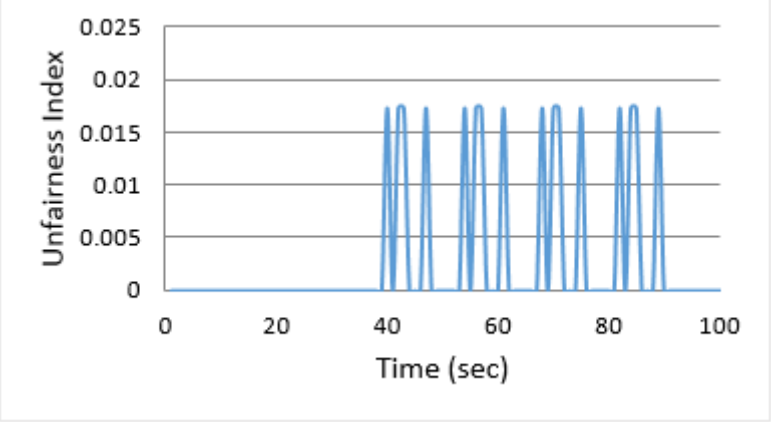

Figure 5.4: Unfairness Index of two clients

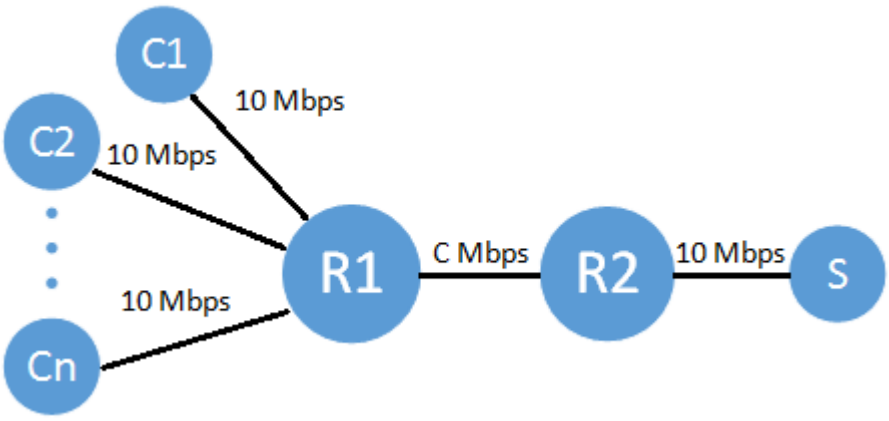

Figure 5.5: Network topology of simulation

DASH server. The site[6] provides several videos with different segment duration in MPEG-DASH standard. We choose segment duration is 2 seconds and encoded video bit rates are 45652, 89283, 131087, 178351, 221600, 262537, 334349, 396126, 522286, 595491, 791182, 1032682, 1244778, 1546902, 2133691, 2484135, 3078587, 3526922, 3840360, 4219897 bps. Network topology which is used in simulation is given Figure 5.5.

We simulate two clients for 300 seconds. In this experiment, we will compare two clients under same conditions. Both client 1 and client 2 start downloading segment from server at $t = 0$. When time interval between $t = 0$ and $t = 10$, both clients increase buffer level and video bit rate (see Figures 5.1 and 5.2). When time between $t = 10$ and $t = 40$, although video bit rate of client 1 and client 2 converge to about 1100 kbps, buffer level decreases slowly since total download rate exceed bottleneck link capacity. In this interval, behavior of clients is exactly same so unfairness index is zero (see Figure 5.4). After $t > 40$, when video bit rate oscillates between 791 kbps and 1032 kbps, buffer level keeps around between 26 and 28 seconds. Figure 5.3 shows number of video quality switches. Since video bit rates oscillate, number of video quality switches also increases. However, magnitude of quality switches is small. We summarize average performance over session (300sec) such as average bottleneck efficiency as 0.974, average buffer level is 27.09, average unfairness index is 0.0034412 and average video quality switches is 0.11. We expect these results since EFAST try to maximize bottleneck efficiency and minimize unfairness index with no

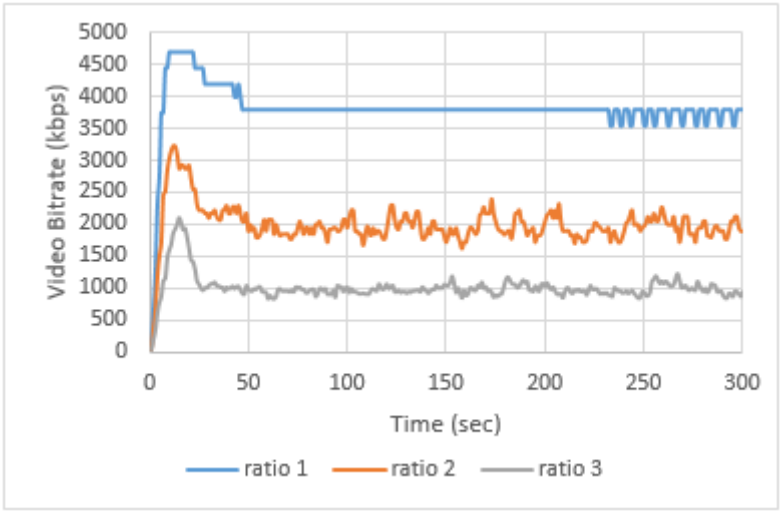

Figure 5.6: Video bit rate of different C/N ratios

buffer depletion.

5.1.2 Experiment 2: Performance of FEAST Under Different C/N Ratios

In this experiment, we investigate how $R = C/N$ ratio affects performance of EFAST when C represents bottleneck link capacity and N denotes number of clients which share same bottleneck link. In this setup, bottleneck link capacity C is constant which capacity is 8Mbps and number of clients are $N_1 = 2$, $N_2 = 4$ and $N_3 = 8$. We simulate 300 second in NS2 [23] with three different C/N ratios; $R_1 = 8/2$ Mbps, $R_2 = 8/4$ Mbps and $R_3 = 8/8$ Mbps. Other simulation parameters such as segment duration, receiver buffer size, network topology and encoded video bit rates are same as Experiment 5.1.1.

Let us analyze performance of EFAST such as buffer size, video bit rate, bottleneck efficiency, unfairness and switch change respectively with different C/N ratio. In Figure 5.6, it shows that average video bit rate over clients for different R_1, R_2 and R_3. In three cases, average video bit rate increases initially. Video bit rates of R_1, R_2 and R_3 converge to around 4Mbps, 2Mbps and 1Mbps respectively since number of clients N ($N_1 = 2$, $N_2 = 4$, $N_3 = 8$) share bottleneck link 8 Mbps respectively. In Figure 5.7, it shows the average buffer level over clients. Initially, buffer increases

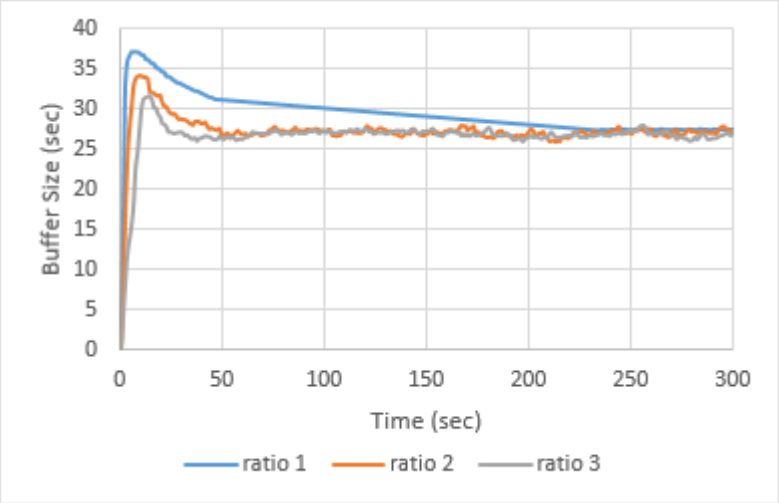

Figure 5.7: Buffer size of different C/N ratios

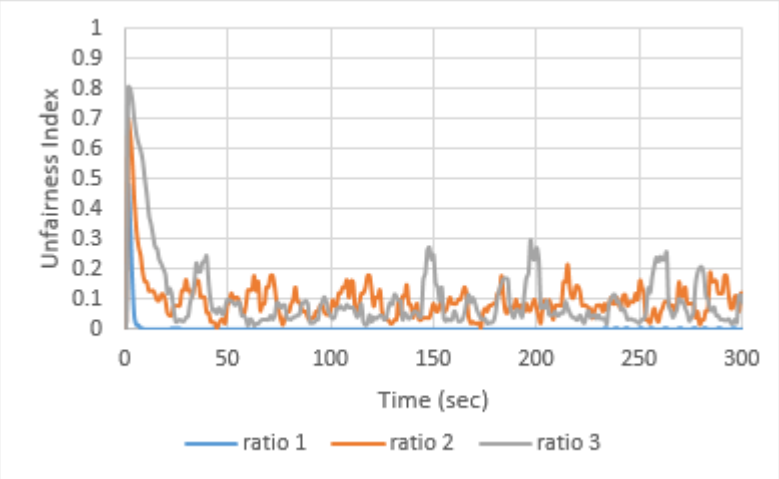

Figure 5.8: Unfairness Index of different C/N ratios

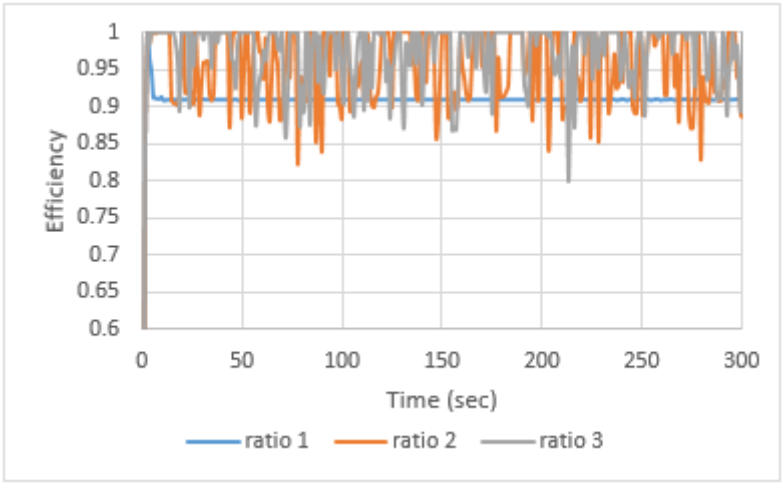

Figure 5.9: Bandwidth efficiency of different C/N ratios

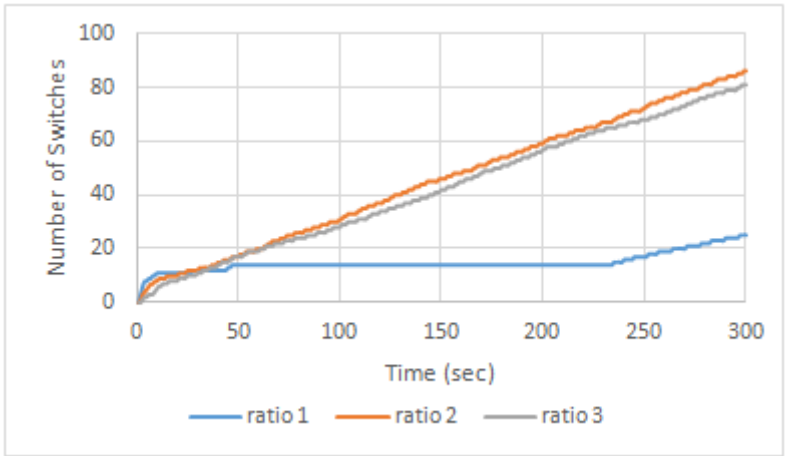

Figure 5.10: Number of switches of different C/N ratios

Table5.1: Performance comparison of different ratios

	Efficiency	Unfairness	Buffer Size	Video Switches
R_1	0,954	0,0039	29,37	0,083
R_2	0,978	0,0967	27,18	0,26
R_3	0,996	0,104	26,52	0,28

very fast since sum of video bit rates is less than bottleneck link C bps. Maximum point of buffer level are different for three different cases since case of R_1 has more step to reach average video bit rate compare to cases of R_2 and R_3. In any case, buffer levels converge to between 26 and 28 seconds. In Figure 5.8, it shows that Unfairness index of case R_1, R_2 and R_3. When number of clients increases, average video bit rate decreases. In general, differences between two consecutive video bit rates are not equal to each other. When video bit rates are low, difference between the current video bit rate and consecutive video bit rates are also small. Thus, clients change video bit rate more frequently when current video bit rate is small. Therefore, number of video quality switches of case R_3 and R_2 are higher than case of R_1 (see Figure 5.10). In addition, unfairness of case R_3 is worse than cases of R_1 and R_2. Although case of R_3 provides lower performance according to video quality switches and unfairness, efficiency is higher compare to other cases. (See Figure 5.9)

Now, we compare average efficiency, unfairness, buffer level and video quality switches over each session (300 seconds). For case of R_1, average efficiency is 0.954, average unfairness is 0.0039, average buffer is 29.37 and average video switches is 0.083. For case R_2, average efficiency is 0.978, average unfairness is 0.0967, average buffer is 27.18 and average video switches is 0.26. For case R_3, average efficiency is 0.996, average unfairness is 0.104, average buffer is 26.52 and average video switches is 0.28 (see table 5.1). In all three cases, video bit rate converges around C/N and buffer level becomes stable at level 26 seconds. In addition, there is no buffer depletion in all three cases. As a result, simulation results show that when C/N ratio decreases, bottleneck bandwidth efficiency becomes better. However unfairness and video switches become worse.

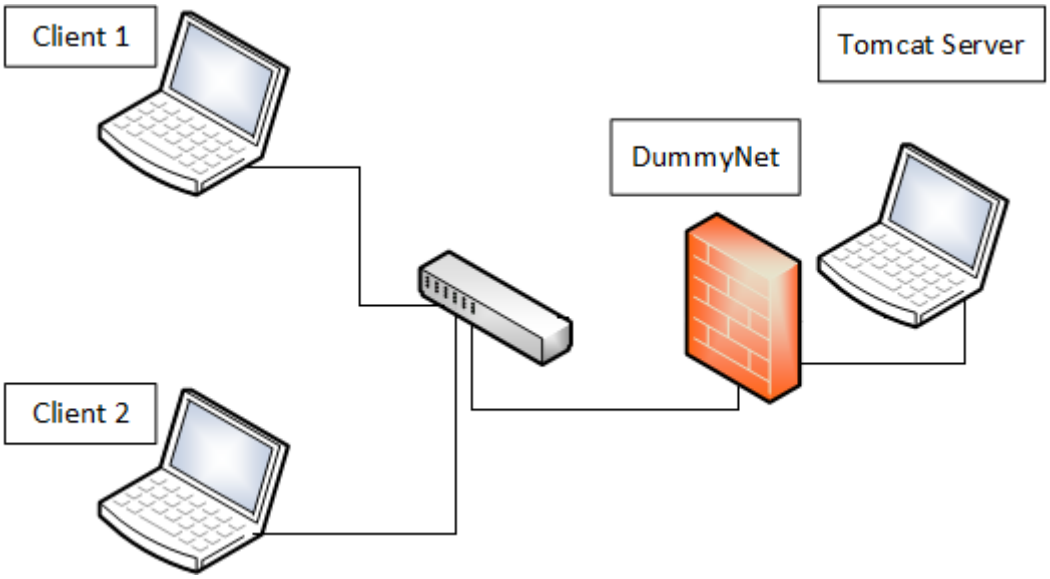

Figure 5.11: Network topology of real-network

5.1.3 Experiment 3: Performance Comparison Between Simulation and Real-Network

In this experiment, we investigate EFAST in real network environment. We show that EFAST gets high performance in constant bottleneck link scenario in Experiments 5.1.1 and 5.1.2. We analyze how performances of EFAST change in real network environment. Therefore, we implement EFAST by using libdash library [17]. After we evaluate EFAST, we compare simulation results to real network results. network topology is given in Figure 5.11. In this setup, clients connect to real MPEG DASH server [6]. Dummynet [3] is a link emulator which is used in user-configrable network environments. Configration of Dummynet is done running ipfw programs. By help of ipfw command, bandwidth of communication link is fixed and propagation delay is adjustable easly. Previous works [26], [20] and [13] use Dummynet in performance evaluations. We use Dummynet for limiting the bandwidth so that we analyzed EFAST in constant bottleneck link. Video segment duration is 2 seconds and receiver buffer size is 40 seconds. Encoded video bit rates are the same as Experiment 5.1.1.

In order to compare performances such as buffer level, video bit rate, unfairness index, video switches between simulation and real network environment, we show normal-

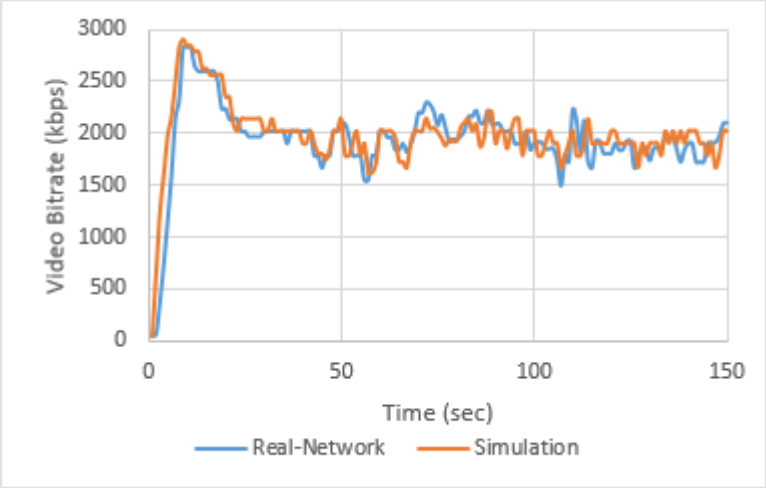

Figure 5.12: Video bit rate of real-network and simulation

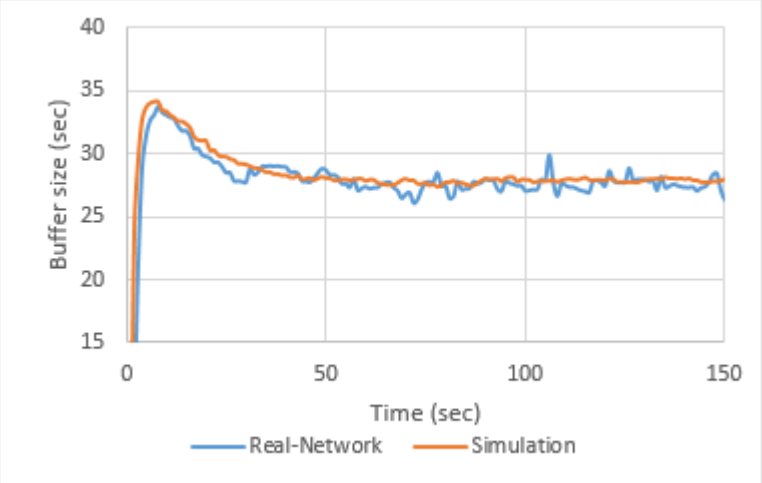

Figure 5.13: Buffer size of real-network and simulation

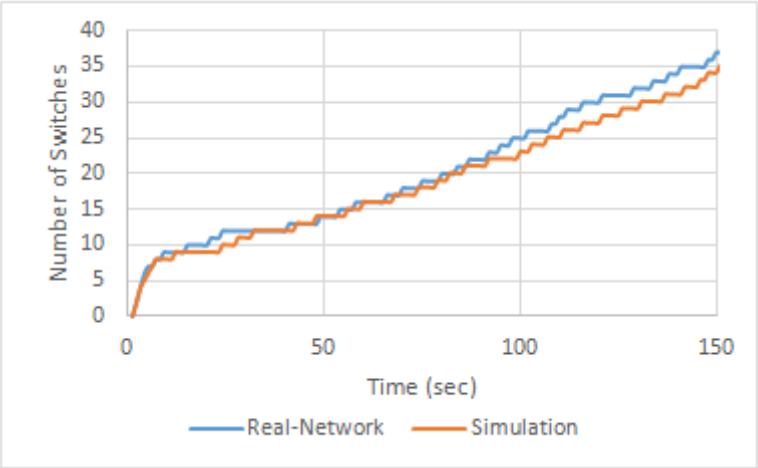

Figure 5.14: Number of switches of real-network and simulation

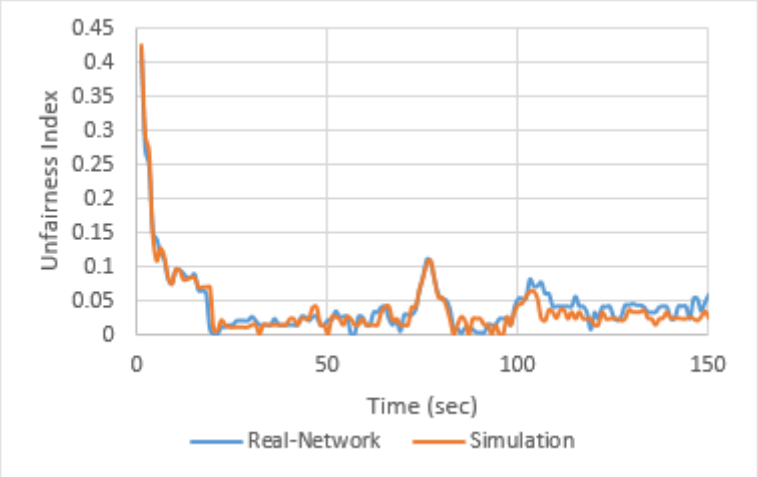

Figure 5.15: Unfairness Index of real-network and simulation

ized cross correlation and session average y_n within $+\Delta\%$ of the true mean μ with a probability of P%. Let's show how we calculate y_n, Δ and μ:

We calculate session average y_n when d_i denotes sample and n represents total number of samples.

$$y_n = \frac{\sum_{i=1}^{n} d_i}{n} \tag{5.1}$$

Now we calculate the standard deviation s_n as:

$$s_n = \sqrt{\frac{\sum_{i=1}^{n} (d_i - y_n)^2}{n}} \tag{5.2}$$

We define $\Delta = S_n * t_g/\sqrt{n}$ where $t_g = 1.96$ if $P = 95$ and/or $t_g = 2.58$ if $P = 99$

Therefore, we compare two performances with average value over session within $\Delta\%$ of the true mean μ with probability of 95% or 99%.

When session average values of two series are similar, it indicates that they are on the same horizontal axis. However, average value is not enough to show similarity of two series. Thus, we also compare normalized cross correlation of two series to show similarity of experimental results between simulation and real network environment. One-dimensional normalized cross correlation [19] between two discrete series with delay d is calculate following equation where m_x and m_y are means values of the corresponding series:

$$NCC(x, y, d) = \frac{\sum_{i=1} (x[i] - m_x)(y[i - d] - m_y)}{\sqrt{\sum_{i=1} (x[i] - m_x)^2} \sqrt{\sum_{i=1} (y[i - d] - m_y)^2}} \tag{5.3}$$

Normalized cross correlation is used to show similarity of two series, signals, images etc. [31] states that normalized cross correlation is the simplest but effective method as a similarity measure. The result of normalized cross correlation varies between -1 to 1. [4] states that value of normalized cross correlation is larger than 0.7 or 0.8 indicates a pretty good match.

Figures 5.12, 5.13, 5.14 and 5.15 show video bit rate, buffer level, number of video

Table5.2: Results of comparison between real-network and simulation

		Simulation	Real-Network
Video Bit rate	Average over Session(Mbps)	1,976	1,962
	$\Delta\%$ with P=95%	2,546	2,63
	$\Delta\%$ with P=99%	3,352	3,561
	Cross-Correlation	0,851	
Buffer Size	Average over Session(Mbps)	28,53	28,62
	$\Delta\%$ with P=95%	0,91	1,13
	$\Delta\%$ with P=99%	1,21	1,35
	Cross-Correlation	0,889	
Unfairness	Average over Session(Mbps)	0,034	0,032
	$\Delta\%$ with P=95%	10,05	11,50
	$\Delta\%$ with P=99%	14,75	15,14
	Cross-Correlation	0,962	

switches, unfairness index respectively. Although fluctuation of video bit rate and buffer size are larger in real network environment compare to in simulation, the results of simulation and real network are almost same. Cross correlation of video bit rates, buffer level, unfairness index and video changes are 0.851, 0.889, 0.962 and 0.9962 respectively. We summarize average values over session with Δ in table 5.2. The values indicated in table 5.2 are the results in simulation and in real network is close thus our simulation is reliable.

5.1.4 Experiment 4: Performance of EFAST Under Variable Bottleneck Link

In this experiment, we investigate EFAST in variable bottleneck link condition. We show high performance of EFAST under constant bottleneck link in Experiments 5.1.1 and 5.1.2. We analyze how performance of EFAST is affected when bottleneck link capacity is changed. During simulation, we change the number of clients N and we apply background UDP traffic. Network topology is given in Figure 5.16. In this scenario, the client 1 starts downloading segment with bottleneck link $C = 4$ Mbps. When $t = 100$, client 2 starts to download segment which shares bottleneck link with first client. After time $t = 200$, we apply UDP background traffic from node U_s to node U_d. Size of UDP packets are 100 bytes with generation rate is 20 Kbps. Other parameters such as segment size, receiver buffer size and encoded video bit rates are same as Experiment 5.1.1.

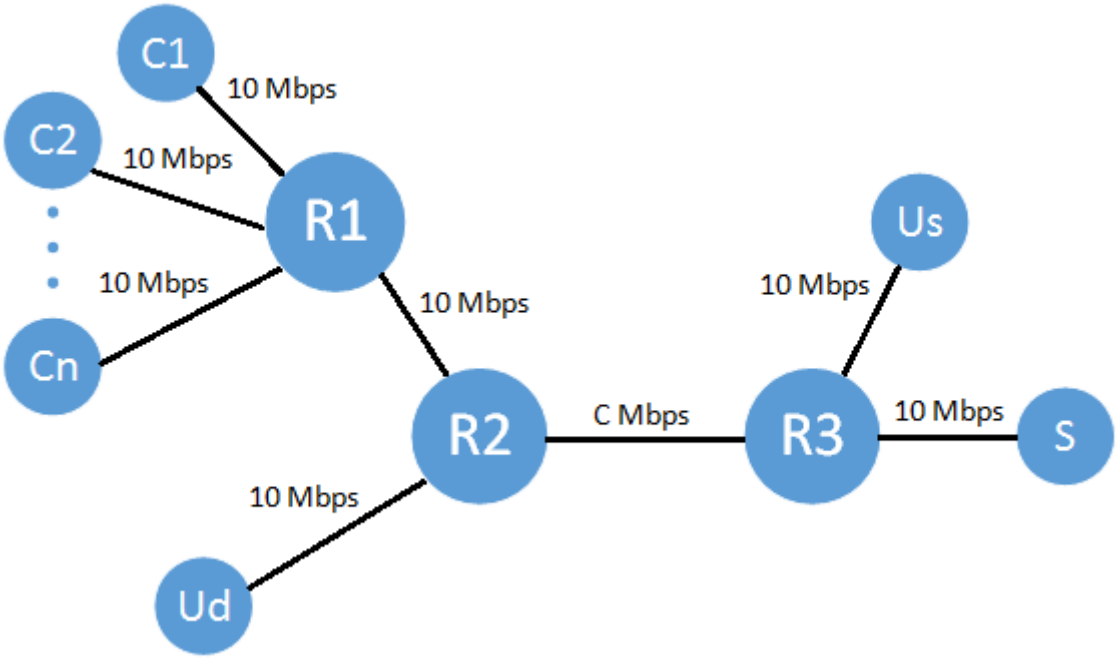

Figure 5.16: Network topology of Experiment 4

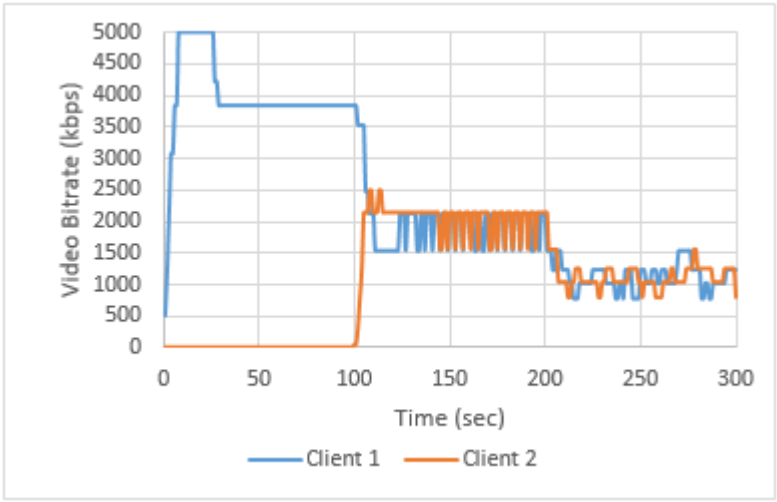

Figure 5.17: Video bit rate of clients under variable bottleneck link

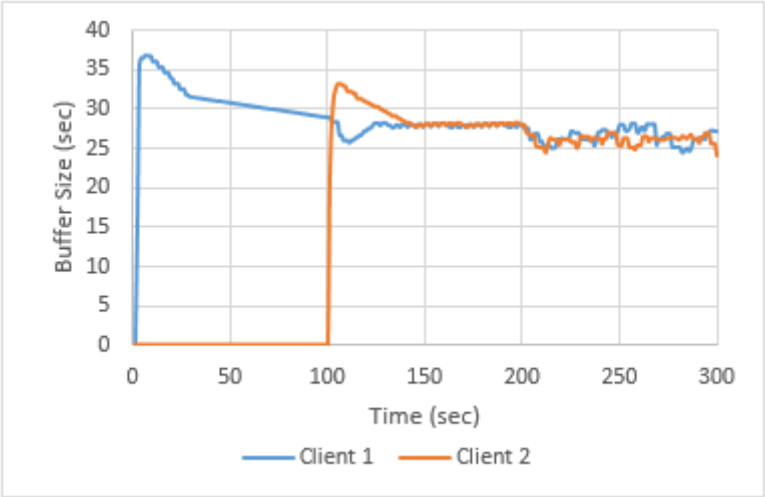

Figure 5.18: Buffer size of clients under variable bottleneck link

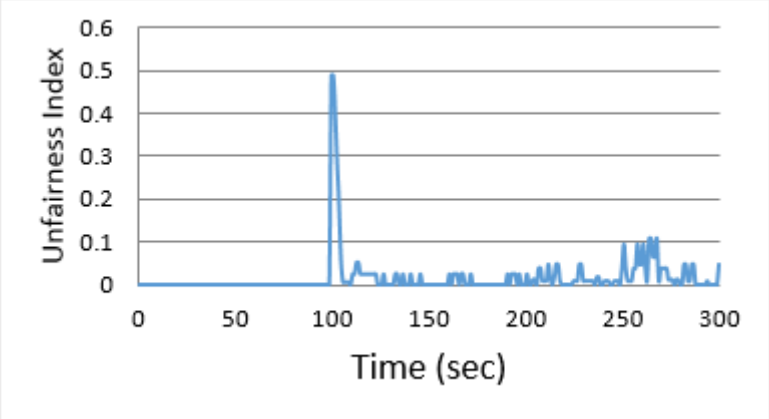

Figure 5.19: Unfairness Index of clients under variable bottleneck link

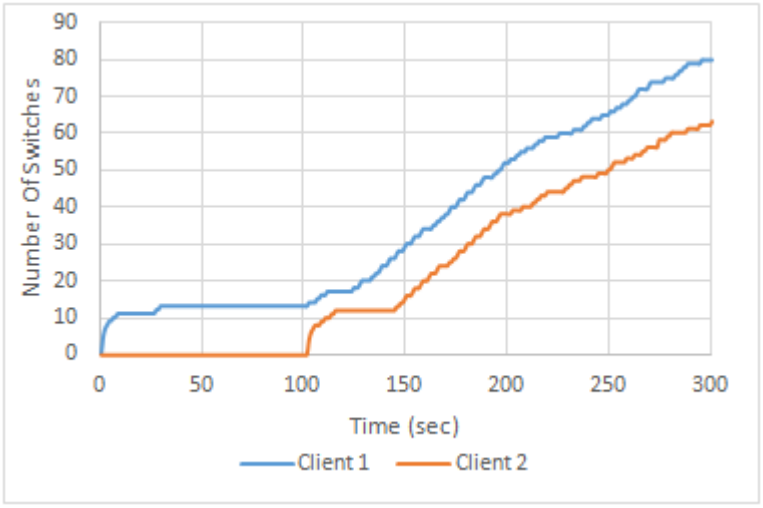

Figure 5.20: Number of switches of clients under variable bottleneck link

In Figure 5.17, it shows that video bit rate of client 1 and client 2 with respect to time. When time between $t = 0$ and $t = 100$, client 1 increases its video bit rate around 4 Mbps. When time between $t = 100$, client 2 starts to download video bit rate. Client 1 and client 2 immediately adjust their video bit rate around 2 Mbps. When time $t = 200$, UDP traffic is applied. Since UDP traffic occupies half of bottleneck link capacity, average video bit rates of client 1 and client 2 decrease to 1 Mbps. Although bottleneck link capacity is changed during simulation, clients adjust their video bit rate to available bandwidth level.

In Figure 5.18, it shows the buffer size of client 1 and client 2 with respect to time. Clients succeed to keep buffer level between 24 and 38 seconds despite of variable bottleneck link. There is no buffer depletion and overflow during simulation. Vibration of buffer level increases after $t = 200$ because of UDP background traffic. Figure 5.19 shows unfairness index of clients. During simulation, unfairness is less than 0.12 expect to time when client 2 starts downloading. Number of video quality switch increases after $t = 100$ because small UDP packets make it more difficult to estimate bandwidth calculation. Although number of video quality switches increases, magnitude of switches is small. (See Figure 5.17)

5.2 Discussion and Comparison with ELASTIC, PANDA and FESTIVE

In Experiments 5.1.1, 5.1.2, 5.1.3 and 5.1.4, we observe performance of EFAST itself in various scenarios. In this part, we investigate EFAST with different solutions of HAS. We chose three popular and well-known HTTP adaptive streaming solutions such as ELASTIC, PANDA and FESTIVE. We show advantages and disadvantages of EFAST compared to other solutions. Firstly, we analyze video level and instantaneous download rate of two clients which share 4 Mbps bottleneck link. Secondly, we investigate how fairness and channel utilization changes when number of client increases. Thirdly, we show average video bit rate and number of switches with respect to varying number of video clients and TCP connections. We change various N_{tcp}/N ratio from 0 to 0.75 when Ndenotes total number of connections and N_{tcp} represents number of TCP connections. Experiment environment is determined in article [7]. We use tomcat server instead of Debian Linux server. We implement EFAST algorithm using libdash library in Windows 7. We use dummynet as the network shaper. At server, "Elephant's Dream" video is encoded with five different quality such as 300, 700, 1500, 2500 and 3500 kbps.

5.2.1 Experiment 5: Performance Comparison Under Constant Bottleneck Link

In this experiment, we compare the performance of two clients sharing a 4Mbps constant bottleneck link. We investigate instantaneous downloading rate and corresponding video bit rate of each client. In Figure 5.21, results of 4 algorithms, EFAST, PANDA, FESTIVE and ELASTIC, can be seen. Instantaneous download rate can be seen at the top, and instantaneous video quality is shown at bottom. Resolutions of the video levels are in Table 5.3. Since 4Mbps channel is shared by 2 clients, each of the clients has 2 Mbps bandwidth capacity under ideal conditions. Firstly when the fairness among clients is compared, PANDA gave the best performance. It generated the same video quality with the least amount of video switches. ELASTIC and EFAST algorithms showed similar results when they were considered for fairness. FESTIVE was the worst one. Even if the ELASTIC had less video switches, it jumped from video quality 2 to video quality 4 directly, moreover this direct jumps affects quality

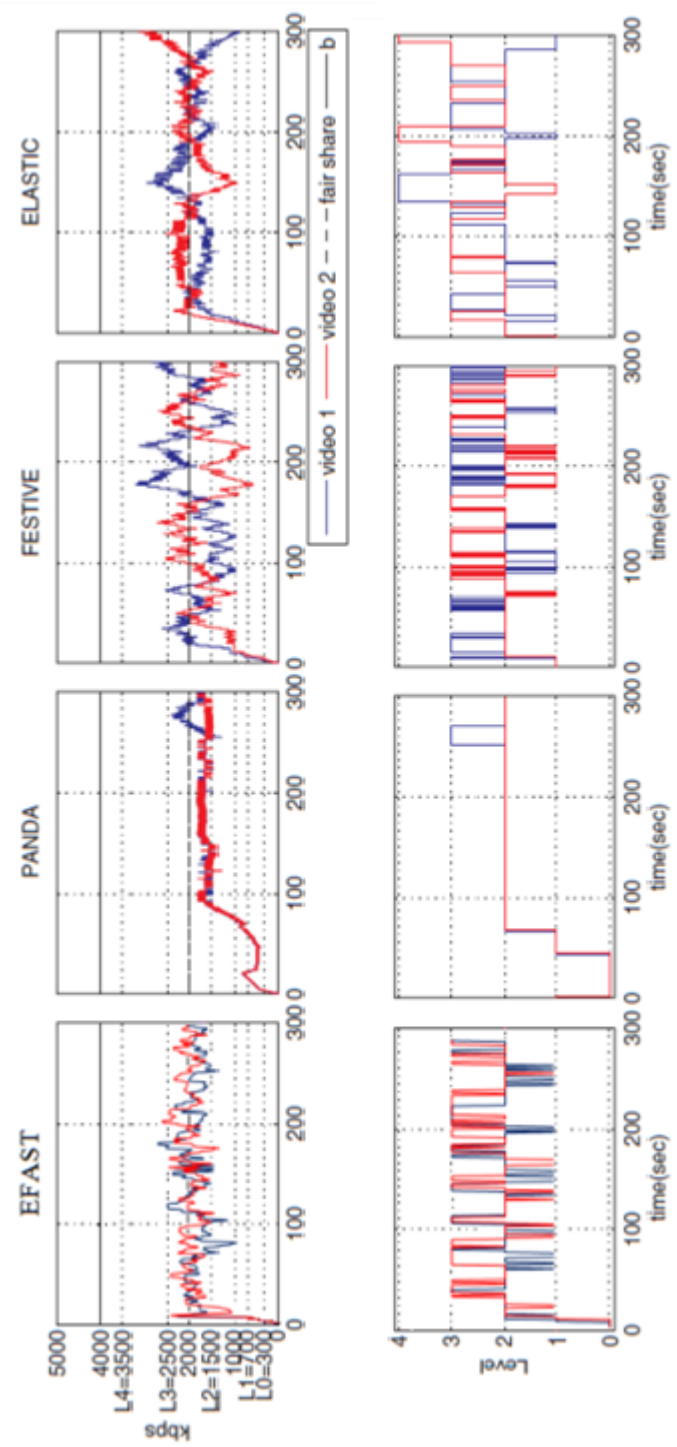

Figure 5.21: Video bit rate of EFAST, PANDA, FESTIVE and ELASTIC

Table5.3: Video level of Elephant‘s Dream

Video level	Resolution	Bit rate (bps)
L_0	320x180	300000
L_1	640x360	700000
L_2	640x360	1500000
L_3	1280x720	2500000
L_4	1280x720	3500000

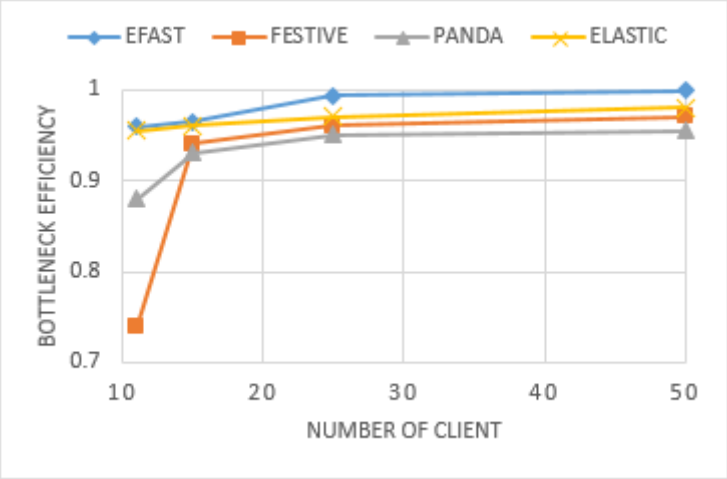

Figure 5.22: Bottleneck efficiency

of experience drastically. PANDA was the best one when considered about fairness and video quality. However, average download rate is below 2 Mbps, which results in 75% bottleneck efficiency. All the other algorithms except PANDA kept the average download rate at 2 Mbps with the channel efficiency of 90%.

5.2.2 Experiment 6: Performance Comparison with Various Numbers of Clients

In the experiment, we have analyzed the bottleneck efficiency and fairness with respect to number of clients. We increased the bottleneck bandwidth capacity to 40 Mbps, and changed number of clients from 11, 15, 25, and 50 respectively. We have seen the bottleneck efficiency, and number of client relation at Figure 5.23. ELASTIC and EFAST is better at efficiency, then the other 2 algorithms. They kept the bottle-

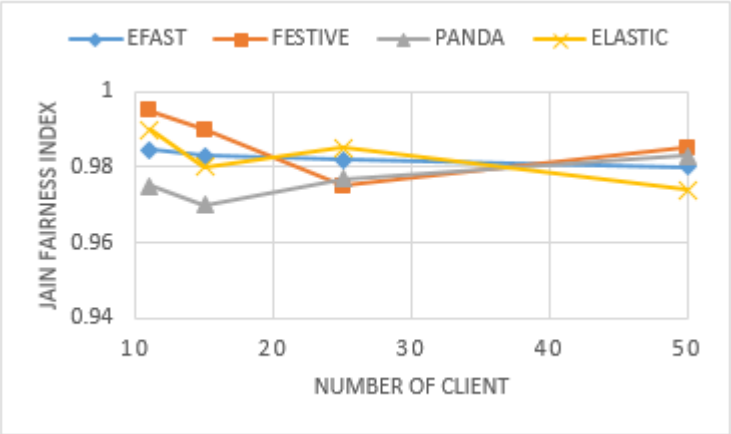

Figure 5.23: Jain Fairness Index

neck efficiency above 0.95 meanwhile number of clients (N) kept between 11 and 50. Reason of that is removing the idle waiting time, and requesting for video segment immediately after video segment download finishes. Efficiency is increasing for all algorithms if number of client increases. Jin fairness index is shown at Figure 5.23. Even if number of clients increase, it stayed above 0.96. When EFAST, ELASTIC and PANDA are not affected by number of client changes, Performance of FESTIVE is affected negatively.

5.2.3 Experiment 7: Performance Comparison with Different N_{tcp}/N Ratios

In this experiment, we compare the performance of the video client with background TCP connections. Constant bottleneck will be kept at 40 Mbps again. Total number of video clients are N_c, total number of background TCP connections are N_{tcp} and, total number of flow of N are $N_c + N_{tcp}$. In this experiment, we observe the performance dependency on the different values of N and N_{tcp}/N. In Table 5.4, N_c and N_{tcp} can be seen when total number of flow are 11 15 25 and 50.

In figure 5.24, average video bit rate change depending on the N_{tcp}/N can be seen when total number of flow N kept 11, 15, 25, 50. Independent from the number of clients, EFAST always gave the highest average video rate. Also average video rate stayed at ideal fair ($40\text{Mbps}/N$) not affected by changes in N. After EFAST,

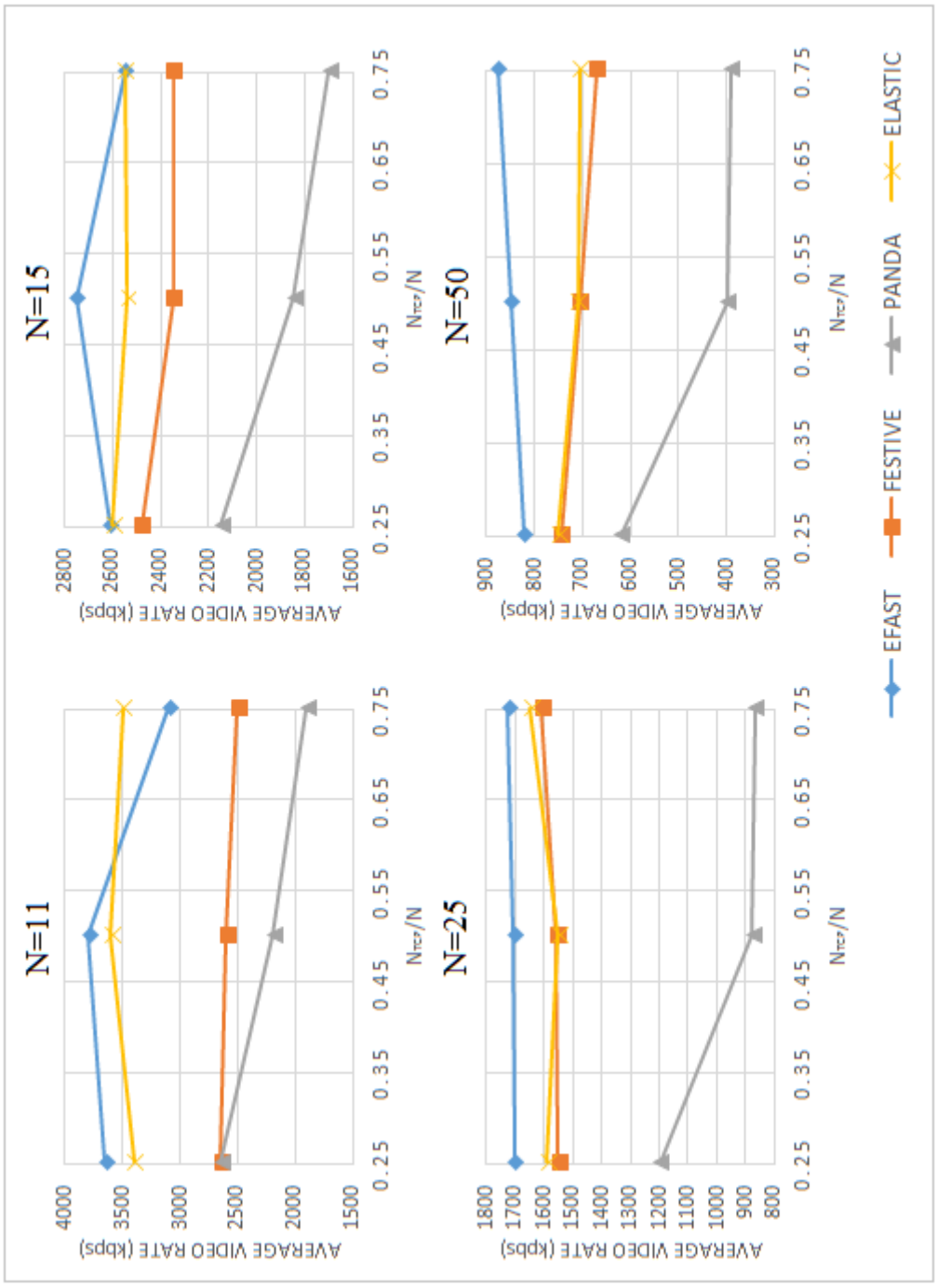

Figure 5.24: Average video bit rate of EFAST, PANDA, FESTIVE and ELASTIC

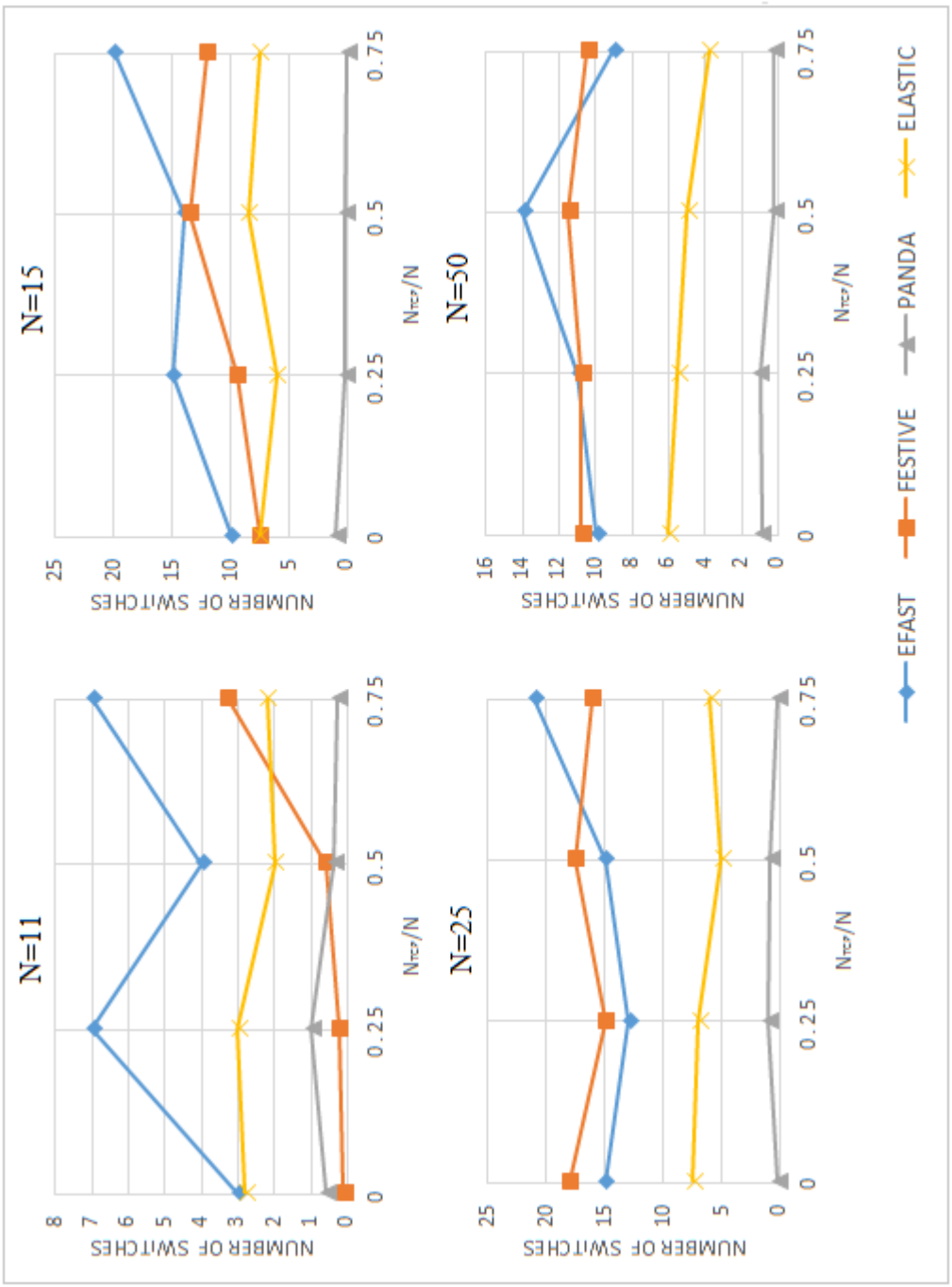

Figure 5.25: Number of switches of EFAST, PANDA, FESTIVE and ELASTIC

Table5.4: Number of N_c and N_{tcp} with different N_{tcp}/N

		N_{tcp}/N = 0	N_{tcp}/N = 0,25	N_{tcp}/N = 0,5	N_{tcp}/N = 0,75
N=11	N_c	11	8	5	3
	N_{tcp}	0	3	6	8
N=15	N_c	15	11	7	4
	N_{tcp}	0	4	8	11
N=25	N_c	25	18	12	6
	N_{tcp}	0	7	13	9
N=50	N_c	50	38	25	12
	N_{tcp}	0	12	25	38

ELASTIC FESTIVE and PANDA followed performance wise. Increase in N_{tcp}/N affected PANDA negatively, meanwhile EFAST and ELASTIC is not affected drastically. Reason is that EFAST and ELASTIC algorithms don't wait between two consecutive segment requests. Therefore they shared the bandwidth fairly with the other TCP connections.

Figure 5.25, shows the switch rate with respect to N_{tcp}/N meanwhile total number of flows N is kept 11, 15, 25, and 50. PANDA had the least video switches. PANDA managed to keep the average video rate between 0 and 2, which is not dependent upon number of flows. Main reason of that is rate adaptation algorithm of PANDA is very similar to TCP congestion control. Performance wise ELASTIC followed the PANDA. ELASTIC is not affected by N_{tcp}/N that much, but average video switches stayed between 3 and 7. FESTIVE had very good performance at $N = 11$, however, performance decreased as N increased. Average video rate switches stayed between 10 and 20 in EFAST. The worst performance when compared with the numerical value seems to be EFAST, but it is due to the narrowness of the encoded video bit rate range. For this scenario, the encoded video rate has only 5 levels and the bit rate difference between them is quite high. EFAST constantly changes the video rate in such a case to increase the average video bit rate and keep the channel utilization high.

In Experiments 5.1.1, 5.1.2, 5.1.3 and 5.1.4 we examined the EFAST algorithm through various scenarios. If we evaluate the test results in general, EFAST responds rapidly to changing network conditions and brings video bit rate to the available bandwidth level. This allows the channel to keep the utilization at a high level and provide the

best quality video viewing. While trying to keep video quality at its highest level, it tries to keep the buffer level between 60 and 80 percent. This prevents buffer depletion and at the same time prevents the buffer level from rising continuously with time. When there are multiple video clients, it is observed that they share a fair channel among themselves.

In Experiment 5.2.1, 5.2.2, and 5.2.3, performance comparison with ELASTIC, PANDA and FESTIVE was made with EFAST algorithm. Although the number of video bit rate switches seems to be more than the other algorithms, the average video rate is higher than the other algorithms. EFAST has made it even more successful than other 3 algorithms such as ELASTIC, PANDA and FESTIVE to share a channel fairly even among other TCP connections, not just between their clients. EFAST can show high performance on various network conditions without being affected by the number of clients.

CHAPTER 6

CONCLUSION

In this thesis, we focus on HTTP based adaptive streaming (HAS) and propose a fuzzy control based rate adaptation algorithm for the HAS client. Our proposed client side HAS rate adaptation method that we call EFAST (Effcient and Fair HTTP Adaptive STreaming) only requires implementation on the client side without any modification on the server. EFAST conforms to the all MPEG-DASH standards. EFAST aims for downloading at the segments at the highest possible rate avoiding the buffer depletions. Furthermore the bit rates of the downloaded videos infrequently switch and the clients that share a bottleneck bandwidth fairly.

Different than previous heuristic rate adaptation algorithms EFAST utilizes a systematic approach. To this end a fuzzy logic controller is designed which selects next video bit rate by using the amount of data in receiver buffer and calculation of available bandwidth.

The fuzzy logic control of EFAST allows showing that EFAST achieves the target video bit rate. The work in this thesis shows the design of the fuzzy logic controller. We show that for a single client EFAST rate adaptation converges to the bottleneck link bandwidth as the video rate together with the possible state transitions. As the system with more clients is very complex to enumerate the state space, we perform a detailed evaluation of EFAST using NS2 simulator and emulated experiments under different network parameters. To this end, we evaluate EFAST under constant and variable bottleneck link conditions. We investigate how number of the clients and bottleneck link capacity affects performance of EFAST in detail. In addition, we simulate and analyze EFAST with background UDP traffic. EFAST is evaluated in

real network by using dummynet tool for bandwidth limiting. Simulation results show that EFAST improved QoE such as efficiency, fairness and stability without buffer depletion. Finally, we compare the performance of EFAST with other well known MPEG-DASH solutions such as Elastic, Festive and Panda by implementing all these works as described in the respective publications. Experimental results show that EFAST performs more efficiently and has more fair bandwidth share among the connections compared to other solutions.

In the current work, we observe that performance of EFAST depends on available bandwidth calculation. When clients estimate available bandwidth more correctly, the performance metrics improve more. Therefore, we will develop a more specific and reliable bandwidth calculation method. In addition, although simulation results show that EFAST performs with high efficiency and fair bandwidth allocation among clients which share same bottleneck link, number of video bit rate switches increases in some scenarios. Thus, we will work on the optimization of video bit rate switches.

REFERENCES

[1] Adobe HTTP Dynamic Streaming (HDS) Technology Center. http://www.adobe.com/devnet/hds.html.

[2] HTTP Live Streaming. https://developer.apple.com/streaming/.

[3] M. Carbone and L. Rizzo. Dummynet revisited. *ACM SIGCOMM Computer Communication Review*, 40(2):12–20, 2010.

[4] Cross-Correlation. https://www.ocean.washington.edu/courses/ess522/lectures/08_xcorr.pdf.

[5] The Zettabyte Era — Trends and Analysis – Cisco. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html.

[6] Dynamic Adaptive Streaming over HTTP. http://www-itec.uni-klu.ac.at/dash/?page_id=207.

[7] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. Elastic: a client-side controller for dynamic adaptive streaming over http (dash). In *Packet Video Workshop (PV), 2013 20th International*, pages 1–8. IEEE, 2013.

[8] L. De Cicco and S. Mascolo. An adaptive video streaming control system: Modeling, validation, and performance evaluation. *IEEE/ACM Transactions on Networking (TON)*, 22(2):526–539, 2014.

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

[10] C. Huitema. Real time control protocol (rtcp) attribute in session description protocol (sdp). 2003.

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

[12] R. Jain, A. Durresi, and G. Babic. Throughput fairness index: An explanation. Technical report, Tech. rep., Department of CIS, The Ohio State University, 1999.

[13] 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*, pages 97–108. ACM, 2012.

[14] A. Kostuch, K. Gierłowski, and J. Wozniak. Performance analysis of multicast video streaming in ieee 802.11 b/g/n testbed environment. In *Wireless and Mobile Networking*, pages 92–105. Springer, 2009.

[15] C.-C. Lee. Fuzzy logic in control systems: fuzzy logic controller. i. *IEEE Transactions on systems, man, and cybernetics*, 20(2):404–418, 1990.

[16] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, 2014.

[17] bitmovin/libdash. `https://github.com/bitmovin/libdash`.

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

[19] D. Lyon. The discrete fourier transform, part 6: Cross-correlation. *Journal of object technology*, 9(2):17–22, 2010.

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

[21] Smooth Streaming. `https://www.iis.net/downloads/microsoft/smooth-streamingl`.

[22] Dynamic Adaptive Streaming over HTTP (DASH): Past, Present, and Future. `http://www.streamingmediaglobal.com/Articles/Editorial/Featured-Articles/Dynamic-Adaptive-Streaming-over-HTTP-(DASH)-Past-Present-and-Future-93275.aspx`.

[23] The Network Simulator-ns-2. `https://www.isi.edu/nsnam/ns/l`.

[24] Y. OZCAN. *HTTP ADAPTIVE STREAMING ARCHITECTURES FOR VIDEO ON DEMAND AND LIVE TV SERVICES*. PhD thesis, MIDDLE EAST TECHNICAL UNIVERSITY, 2013.

[25] K. M. Passino, S. Yurkovich, and M. Reinfrank. *Fuzzy control*, volume 2725. Addison-wesley Reading, MA, 1998.

[26] A. Sobhani, A. Yassine, and S. Shirmohammadi. A fuzzy-based rate adaptation controller for dash. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 31–36. ACM, 2015.

[27] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

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

[29] D. J. Vergados, A. Michalas, A. Sgora, and D. D. Vergados. A fuzzy controller for rate adaptation in mpeg-dash clients. In *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*, pages 2008–2012. IEEE, 2014.

[30] L. A. Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

[31] F. Zhao, Q. Huang, and W. Gao. Image matching by normalized cross-correlation. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 2, pages II–II. IEEE, 2006.