QoE Enhancement in HTTP based Adaptive Video Streaming

Author

Shahid Nabi

171088

Supervisor

Dr. Muhammad Umar Farooq

Co-Supervisor:

Dr. Farhan Hussain

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

APRIL, 2019

QoE Enhancement in HTTP based Adaptive Video Streaming

Author

Shahid Nabi

171088

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Computer Engineering

Thesis Supervisor:

Dr. Muhammad Umar Farooq

Thesis Supervisor's Signature:____

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

April, 2019

DECLARATION

I certify that this research work titled "*QoE Enhancement in HTTP based Adaptive Video Streaming*" is own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student
SHAHID NABI
2016-NUST-MS-CE-00000171088

LANGUAGE CORRECTNESS CERTIFICATE

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. The work is original contribution of the author and does not contain any plagiarism. Moreover, Thesis is also according to the format given by the university.

Signature of Student

SHAHID NABI

2016-NUST-MS-CE-00000171088

Signature of Supervisor

DR. MUHAMMAD UMAR FAROOQ

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I am thankful to my Creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which You setup in my mind to improve it. Indeed I could have done nothing without Your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was Your will, so indeed none be worthy of praise but You.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of my life.

I would also like to express special thanks to my supervisor Dr. Muhammad Umar Farooq for his help throughout my thesis and also for Selected Topics in Computer Networks courses which he has taught me. I can safely say that I haven't learned any other engineering subject in such depth than the ones which he has taught.

I would also like to pay special thanks to Dr. Farhan Hussain for his tremendous support and cooperation. Each time I got stuck in something, he came up with the solution. Without his help I wouldn’t have been able to complete my thesis. I appreciate his patience and guidance throughout the whole thesis.

I would also like to thank Dr. Arslan Shaukat and Dr. Ali Hassan for being on my thesis guidance and evaluation committee and express my special Thanks to these faculty members for his help

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Dedicated to my exceptional parents and adored siblings whose tremendous support and cooperation led me to this wonderful accomplishment

ABSTRACT

In the last decade, there has been an exponential increase in the video traffic over the internet. Social Medias are becoming one of the main source of live and on-demand video streaming content. With ever-increasing popularity of online different video streaming services on heterogeneous platforms, new research challenges are arising day by day. Few of the main challenges that online video streaming services face are high latency of the video, instability of the video, unfairness among the clients, inefficiency of the algorithm to adapt to the changes in the network and the start-up delay of the video. Most of the existing algorithms fail to maintain a balance between stability and efficiency of the algorithm in unstable network conditions. We have proposed SHANZ rate adaptation algorithm for which address these challenges. We have developed two versions of the algorithm. SHANZ-I algorithm works on HTTP1.1 protocol. It is a dynamic rate adaptation algorithm with feedback control mechanism and adaptive step up function, which acts as an explicit knob to maintain a balance between stability and efficiency of the algorithm, even in drastic network conditions. Moreover, it introduces randomized download delay for the clients to overcome bandwidth overestimation problem occurred in multiple clients. The second version we have proposed is SHANZ-II rate adaptation algorithm, which is based on HTTP/2 protocol. It utilizes HTTP/2 features like server-push, streams multiplexing and header compression for the enhancement of quality of experience. It minimizes the latency and start-up delay of the video, which are the main challenges for live video streaming. The algorithm defines an intelligent control mechanism for server-push, which maximizes the utility function. We have simulated our algorithm using ns-3 and compared our results with FESTIVE, PANDA and AAASH algorithms by using multiple test cases. The results demonstrate that our proposed algorithm outperforms other algorithms by addressing the key issues and by achieving higher Quality of Experience.

Key Words: *HTTP streaming, QoE, DASH, Dynamic Adaptive Streaming over HTTP, Rate Adaptation Algorithm, Adaptive Bitrate Streaming, ABR*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AAC	Advanced Audio Coding
ABR	Adaptive Bit Rate
AVC	Advanced Video Coding
CDN	Content Distribution Network
DNS	Domain Name Service
CSP	Content Service Provider
DASH	Dynamic Adaptive Streaming over HTTP
HAS	HTTP Adaptive Streaming
HTTP	Hypertext Transfer Protocol
MPEG	Moving Picture Expert Group
MP4	MPEG-4 File Format
MPD	Media Presentation Description
NAT	Network Address Translation
NC	Network Controller
QoE	Quality of Experience
RTP	Real-time Transport Protocol
SVC	Scalable Video Coding
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WAN	Wide Area Network

CHAPTER 1: INTRODUCTION

Due to advancement of the network infrastructure in the last decades, there has been an exponential increase in the video traffic over the Internet. Video content on social media websites like Facebook, are one of the main causes of the rise in video traffic. According to CISCO's Visual Networking Index [1], video content on the Internet in 2022 will constitute 82 percent of total IP traffic. There has been a rapid convergence of many multimedia services like video conferencing, live and on-demand video content distribution services, Traditional TV service over the Internet and IPTV. In 2015, YouTube and Netflix accounted for 50% of the total network traffic in North America [2].

1.1 Background, Scope and Motivation

In the past, either traditional UDP based protocol like Real time Transport Protocol (RTP) [3] or TCP based Real-time Messaging Protocol (RTMP) was used for video streaming over the internet. In the traditional Media Servers, multiple protocols were used in combination for multimedia streaming over the Internet. Real-time Streaming Protocol (RTSP) was used to set-up a streaming session and saving the information of state of server for the session between client and server. RTP was responsible for the transfer of media from server to client; where as RTP Control Protocol was responsible for sending the status of the client to the server, which can help server to perform rate adaptation. This resulted in complex and computationally expensive media servers. One of the main drawbacks of using traditional media servers was their inability to traverse through NAT and Firewall. Additional protocols were required along with RTP to perform this functionality [4]. Moreover, due to differences in implementation or some optional features in place, these media servers behaved differently and had scalability issues, despite having same fundamental protocols in the baseline. These issues caused the failure of server or other glitches during the video streaming session.

The traditional process of video streaming over the Internet is known as progressive download. In progressive download, source video file, which is usually of MP4 format, is transmitted from source server to the destination clients. The source video file remains the same regardless of network state or device capabilities of the client. As a result, clients face two major issues in progressive download. If the file size or bitrate is too high, the client will start facing video stalling issue either due to lower network bandwidth or due to low processing power of the client. On the other hand, if the file size or bitrate is low, the client will face pixelation issue, as the video will not fit to the screen size and will create a blurred image due to stretching. Adaptive video streaming provides a solution to the above-mentioned problem, by providing a dynamic mechanism. Adaptive streaming ensures that client experiences the video on maximum possible video quality according to its device capabilities. Secondly, the quality of the video adjusts along with drastic network conditions.

1.2 What is Adaptive Streaming?

In 2005, adaptive video streaming method was introduced by Move Networks, which got famous instantly due to its simple design with better features. Many leading content providers quickly adopted it due to its cheap deployment cost. In 2012, MPEG declared DASH as a standard protocol for video streaming over the internet. HTTP based adaptive streaming (HAS) protocol uses HTTP as its application layer protocol and TCP as its transport layer protocol for video streaming. Implementing HAS on top of TCP helps the service providers to use the existing infrastructure of HTTP like stateless HTTP webservers, HTTP cache and CDNs. YouTube used server based streaming strategy for video streaming in the past but now it has shifted to HAS.

HAS media severs have several advantages over traditional media servers. Firstly, it has multiple bit rates of the same video, which enables to deliver the video content to clients according to their operations. Secondly, it provides flexible service models to the clients, which means that clients can be charged according to its subscription. Finally, it enables the client to adapt the current video bit rate according to changing network bandwidth and conditions, so the client can enjoy seamless video streaming without any interruption.

In the last decade, there has been a significant increase in the popularity of HTTP based Adaptive Video Streaming. Currently majority of multimedia servers use HAS based architecture to provide their services. Majority of leading commercial players like Microsoft's Smooth Streaming [5], Apple's HTTP Live Streaming (HLS) [6] and Adobe's HTTP Dynamic Streaming (HDS) [7] are all based on DASH based architecture.

1.3 Problem Statement

One of the main challenges for online video streaming services is inability of the algorithm to cope with unstable network condition. Most of the existing algorithms fail to maintain a balance between stability and efficiency of the algorithm in unstable network conditions. We have proposed a dynamic rate adaptation algorithm with feedback control mechanism and adaptive step up function, which acts as an explicit knob to maintain a balance between stability and efficiency of the algorithm, even in drastic network conditions.

1.4 DASH Architecture

HAS contains multiple copies of the same video also called representations, each encoded at a different bit rate and resolution. Moreover, each video representation is divided into equally sized time fragment usually ranging from 1 to 10 seconds known as segments. Segment length is the shortest duration of the video after which a quality shift can occur. To ensure seamless streaming, video quality switches only occur at the end of a video segment and video segment index of all the representations are perfectly time aligned and are of exactly same time interval so that client can switch up or switch down its quality level without interruption in the video.

In HAS the client operates in the two phases during the video streaming process. The first phase is buffer-filling phase. In this phase, the client continuously downloads the video segments until it reaches a certain threshold level of the buffer. After the buffer reaches that threshold value, the client enters a steady phase, in which the client operates in ONN-OFF pattern. The client keeps on downloading the video segments until it reaches the upper threshold value, after that it stops, and the client is only playing the video, so its buffer level starts decreasing. The client remains in OFF state unless its buffer reaches a lower threshold value, after which it starts downloading video segments again.

First segment of the video does not contain any actual metadata rather it contains decoding information about the video for client's decoder. Each video segment must contain at least one Stream Access Point (SAP), from where client's decoder starts the decoding of a video segment. Each segment of the video also contains indexes and its explicit and implicit start time along with its duration. Index of the segments can be used for downloading a segment in parts from multiple source servers by using a signaling sub-segments method, as mentioned in the specification [8]. A client can request sub-segment from multiple servers simultaneously by using Partial HTTP GET requests.

A simple HTTP based Adaptive Video Streaming scenario is shown in Figure 1.1. The process contains two phases, in the first phase MPD file of the video is transported which contains all the attributes of the video. After Successful transmission of MPD file, Client starts requesting the video segment from one of the available representations of the video. HTTP 1.1 protocol is used for video streaming over the internet.

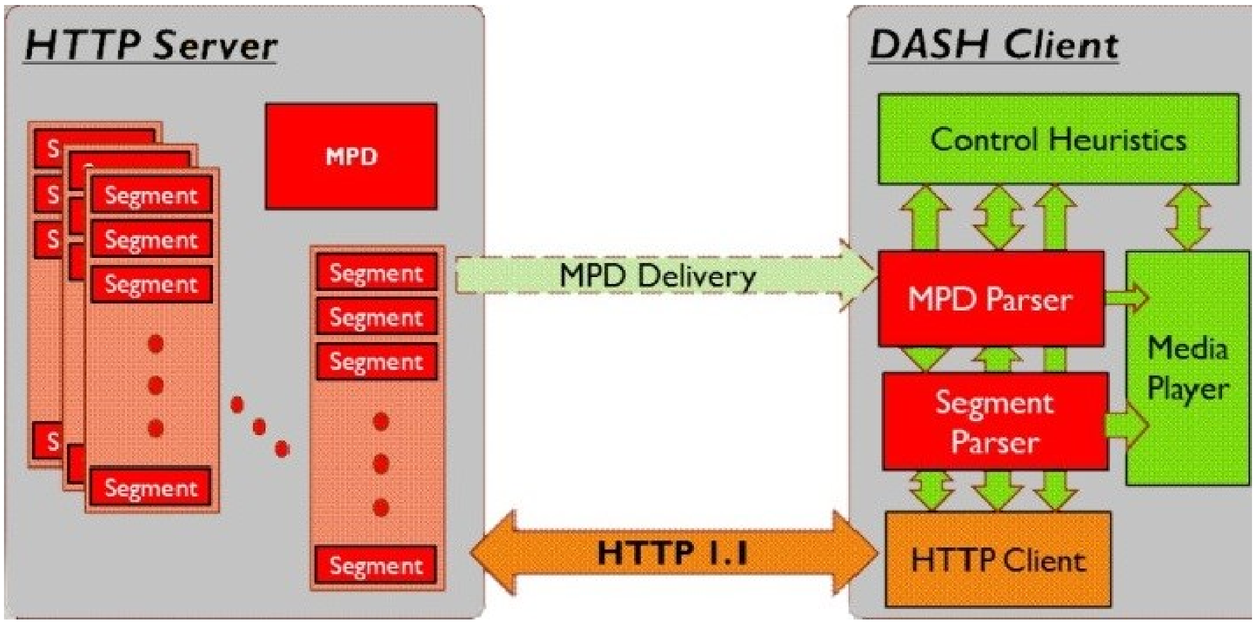


Fig. 1.1: DASH Architecture [9]

1.4.1 MPD File

The first task a server needs to perform to start video streaming session with the client is to send Media Presentation Description (MPD) file to the client. It can be transported using any transport medium like HTTP, broadcast, etc. MPD file in presented in XML file format, which contains information of segment length, segment URLs, representation levels and other attributes of a particular video. Each file can consist of one or more periods. A period contains information about audio and video codec of the video and its corresponding attributes like maximum and minimum available bitrate of the video, frame rate, audio channel etc. Each period can consist of one or more adaptation sets. An Adaptation set contains multiple representations for a same video. It also contains one or more media components of the same video. For instance, one adaptation set can contain different representations of video components of a specific multimedia, while another adaptation set can contain different bitrates of audio components of that video.

A representation of the video contains different encoded copy of a same video. These representations may differ in video resolution, bitrate in which the video is encoded, total number of channels used for encoding the video, or they may differ in any other similar characteristic of the video. Each representation of the video is divided into multiple equally sized media streams called as segments. A segment can vary from 1 to 10 seconds in length. Each video segment can be accessed through its unique URL. Figure 1.2 explains the hierarchical data model of media presentation description file of HAS.

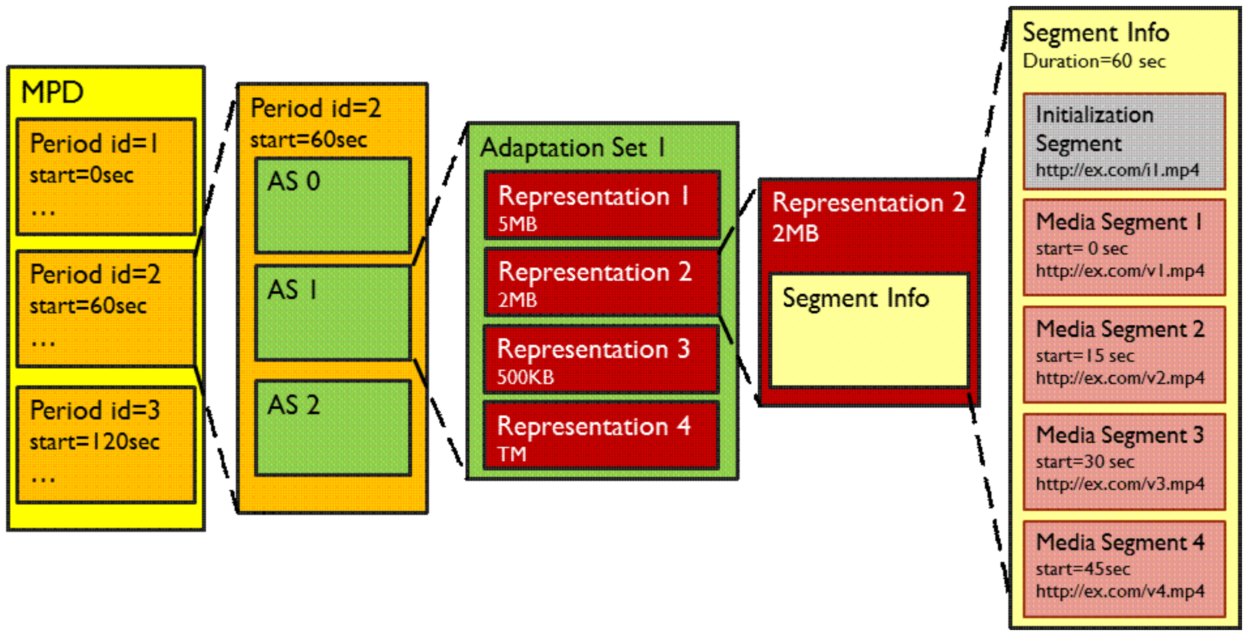


Fig. 1.2: Hierarchical Data Model of MPD File [9]

All of this hierarchical information of the video is stored in its MPD file. A client downloads the MPD file of the video from the server and parses the XML file to check the list of available attributes of the video. A client then chooses suitable adaptation set of the video from one or more available periods according to its codec and other attributes. After the selection of an adaptation set, client formally starts requesting video segments from the server in a sequential order.

Adaptation algorithm is run on the client, which drives the playback of the video. Adaptation algorithm selects the representation of a video and requests video segments from that representation in such a way that is maximizes the overall Quality of Experience of the user. There are various parameters, which can affect the overall QoE metrics of the client. These parameters include measured throughput of the link, buffer status of the client, device specifications like screen resolution etc. Fig 1 below describes the overview of a DASH Architecture.

Client drives the video streaming process by performing some of the most important decisions like selection and scheduling of next video segment to download, and maintaining the buffer level On the other hand, server regulates the video streaming by preparing the video content by deciding the suitable audio and video encoder. H.264/AVC is the most widely used video encoder used for HAS [10]. Server also decides the optimum number of representation levels for a certain video, and how many of them should be available to a certain client. Server also decides the best suitable segment size for a video.

When multiple users in the same network stream videos simultaneously, it creates a bottleneck link in the network, which results in instability and unfairness among the clients [11]. Currently networks have very less capability to overcome the instability and unfairness issues of multiple DASH clients. So its responsibility of the video service providers to ensure fairness and stability among the clients by implementing intelligent rate adaptation algorithms.

1.5 Challenges in HAS

Although HAS media servers have a clear advantage over traditional media servers but there are still numerous challenges that are faced by HAS systems which degrade their performance. One of the main issues faced by HAS systems is instability of the video, which means frequent oscillations in the bit rate of the video, which might be due to fluctuating bandwidth or any other factor. Instability is one of the main reason of performance degradation in HAS based videos and service providers try to minimize its effect.

Second challenge faced by HAS based systems is unfairness among their clients. When multiple clients are streaming video in the same network, they create a bottleneck link. Sometimes when multiple clients are competing for the same resource, they do not get fair share of the network, rather the client is consuming higher bandwidth keeps on getting greater share in the network and bandwidth of other clients drop, which results in unfairness in the clients. Unfairness is one of the main challenges, which content service providers try to overcome.

Third most concerned challenge of HAS based system is inefficiency of the algorithm. Mostly, when developers try to overcome first two challenges of the algorithm i.e. instability and unfairness, starts facing the third challenge i.e. inefficiency. The algorithm starts to underperform; as a result, client is forced to watch the video at lower bit rate even though bandwidth is available.

These are three major challenges of HAS. All the HAS based systems develop their algorithms to overcome these challenges by using different strategies. Adaptation Algorithm is not the part of DASH standard, so every commercial HAS based media player has its own adaptation algorithm but the basic principle of all the algorithms is same.

1.6 QoE parameters

The basic goal of adaption algorithm is to select the video segments in such a way that it maximizes the Quality of Experience (QoE) of the user. QoE is a subjective criterion for analysis of user satisfaction of overall video quality. It consists of many attributes like average video quality, start-up delay, number of video quality switches, number of video freezing events etc.

There are multiple factors, which can affect user's Quality of Experience like, network channel, network signal strength, network congestion events, bandwidth fluctuation, number of clients in a network, etc. One algorithm can outperform other algorithms in one particular scenario but its performance may drop when the network conditions change. Adaptation algorithm should be robust as well as adaptive to perform efficiently in fluctuating network conditions.

There are variety of parameters, which collectively contribute to overall user's Quality of Experience (QoE). Some of the parameters have a positive effect on QoE while others have a negative effect. The main goal of adaptation algorithm is to maximize the parameters having a positive effect on QoE and to minimize the parameters that have a negative effect on QoE. In the corresponding paragraphs, we will briefly discuss each of the QoE parameter and its effect on overall experience of the user.

Average Bit Rate: This is one of the most important parameters of QoE metrics. It describes average video quality perceived by the client. Average Bit Rate is computed by adding the bit rate of each video segment divided by total number of video segments. This parameter gives direct impression of overall user experience. The goal of adaptation algorithm is to keep average bit rate of the client as high as possible according to the available bandwidth.

Initial Delay: This parameter is also crucial for the analysis of overall experience of the user. Initial delay is the total time from when the user requests the video until the client plays first video segment. Higher initial delay has very negative impact of client's user experience. Initial delay depends upon numerous factors like, distance of the source media server/ CDN from the client, popularity of the video etc. The goal of adaptation algorithm is to keep the initial delay of the video as minimum as possible.

Stalling Frequency: Stalling Frequency is defined as total number of video freezing event during the playback. Video stalling is one of the most irritating event in the playback. It has a very negative impact on the quality of experience of the user. One of the most possible cause of the stalling event is buffer underrun or buffer depletion.

Stalling Duration: Stalling duration is another important parameter, which have adverse effect on the user's quality of experience. It is defined as total time in seconds for which video freezing event has occurred during the playback. Adaptation Algorithms are developed in such a way that it avoids any stalling event in the playback. One possible way to avoid the stalling event is to decrease the bit rate of the video.

Switching Frequency: Switching Frequency is the total number of video quality switches during the playback. Greater the switching frequency, lower is the overall quality of experience of the user. Some of the possible reasons of high switching frequency could be drastic changes in the bandwidth of the network or aggressive approach of the algorithm. Therefore, the algorithm aims to keep up the efficiency of the algorithm with minimum number of video quality oscillations.

Switching Amplitude: This parameter is defined as the difference of bitrates before and after the video quality switch. Higher amplitude of video quality switch is perceived as a negative indicator of performance of adaptation algorithm. Therefore, the adaptation algorithm aims to switch the bit rate of the video with lower switching amplitude.

Segment Size: Segment size is another parameter, which contributes towards quality of experience of the user. Smaller segment size means that the client can quickly adapt to the changes in the network, on the other hand, smaller video segments increases encoding overhead of the video. So, optimum size of video segment is required to get the best Quality of Experience for the user.

Video Codec: The performance of the algorithm also depends upon the video codec used for encoding the video. There are two most popular codecs for encoding the video in HAS format. One of them is Advance Video Encoder (AVC). In AVC, each representation of the video is encoded separately. The second type of codec is Scalable Video Coding (SVC). In SVC, the lowest representation of the video acts as a base layer, and all the upper layers are added in the base layer to upgrade the bit rate of the video.

Average Buffer: Average buffer is one of the most important parameters of Quality of Experience metrics. The buffer status drives the overall process of video streaming. The aim of the algorithm is to find out, what is the optimum buffer level for the client and to keep the buffer level closer to the optimum buffer level. If the buffer level drops too low, it will cause buffer underrun and create stalling events. If the buffer level gets to large, it will waste the bandwidth and can create unfairness among the clients. So optimum value of buffer level is to be maintained by the adaptation algorithm to achieve high quality of experience.

Multiple Clients: Total number of clients in a network also effect the user's Quality of Experience. If there are multiple users in the same network, all the clients compete for bandwidth resource and it creates a bottleneck link in the network. The aim of adaptation algorithm should be to maintain fairness among the clients and to ensure that every client gets a fair share of the network. Otherwise, one user may suppress the performance of other clients by using higher slice of the bandwidth of a network by greedy approach.

1.7 MPEG DASH

In April 2012, Moving Pictures Expert Group (MPEG) developed the standard protocol for HTTP based video streaming. The official name given to the protocol was Dynamic Adaptive Streaming over HTTP (DASH). The protocol was revised in 2014 as an ISO standard. DASH enables the streaming of high quality media content by using existing HTTP infrastructure of the network. There are few other popular technologies for HTTP based adaptive streaming like Adobe HDS, Apple HLS, and MSS but they are vendor dependent and have limited support for other streaming servers. MPEG DASH has been declared as an International standard and it is vendor independent.

1.7.1 Additional Attributes of DASH

MPEG DASH is a standard protocol for adaptive video streaming over the Internet. It has a support for live video content, 360-degree videos and Video on Demand. Majority of commercial video players has a support for MPEG DASH. There are several key features of DASH, which makes it the most popular protocol for adaptive video streaming. Some important characteristics of DASH are as follows.

Multiple parallel Streams: MPEG DASH has capability of adding multiple streams, which are switchable and adaptive. All the streams are synchronous to each other, which ensures seamless switching between the streams. For example, audio streams are available in different languages and client can switch to any language in the middle of the playback. Similarly subtitles for each audio stream can be added, which are switched automatically along with the audio stream. Multiple camera angles can be added, so that the client can get better viewing experience by switching the streams.

Target Ad insertion: MPEG DASH gives the capability to insert the advertisements at precise locations in live and on-demand video content. An ad can be inserted at segment level either after one segment ends or at period level, at the end of one period. For example, in a live cricket match, ads can be added manually at the end of each over.

Manifest File Compression: This feature enables DASH to reduce the size of the media presentation description file. It is performed by using template scheme to signal the URL of the video segments. One of the main advantages of this feature is that it reduces the video start-up delay for the clients, as less time is required to download the MPD file.

Manifest File Fragmentation: This is another key feature of MPEG DASH, which reduces the start-up delay for the playback. One large manifest file is partitioned into multiple smaller sunsets, so that only essential attributes of the file are sent at the start of the playback and rest of the parts can be sent periodically during the playback.

Variable Duration segments: This is one of the most powerful features of MPEG DASH. This feature allows variable length of the video segments. This feature is very useful for live video streaming because in that case, content generation, video encoding and video streaming are all performed simultaneously. In live video streaming, there might be delay in the content generation process, which might increase the latency. Therefore variable segment duration allows smoothening the processes and delivering the content as soon as it reaches, without waiting for a complete length of the segment. The length of the next video segment can be notified to the client while sending the current video segment.

Alternate base URLs: With the help of this feature, the same video content can be distributed to multiple servers or CDNs. These servers might be present at different geographical locations to globalize the distribution of the content. So that the client can switch to any one of the available servers which provides maximum throughput with minimum latency. Secondly, this feature can also help to shift the client on its prescribed CDN according to its subscription. It will allow in differentiating regular customers from premium customers, which can get more services.

Support Multiple Encoders: MPEG DASH supports multiple encoders. The most widely used encoder for DASH is Advance Video Encoder (AVC, in which each representation layer is encoded separately. Apart from this, DASH also supports Scalable Video Coder (SVC), in which the lowest representation level is encoded as a base layer and all the successive layers are added to the base layer for the enhancement of video quality. DASH also has a support for Multiview Video Coding (MVC).

Metrics for QoE: DASH has a well-defined set of Quality of Experience metrics for the analysis of experience of the user. These parameters may include start-up delay, average video quality, video quality transitions, current buffer status, stalling events, etc. The client records these parameters and reports them to the server. These parameters help the server to analyze the viewing experience of the client during the playback. The adaptation algorithm adapts the video in order to maximize the viewing quality experience of the client.

1.7.2 MPEG DASH Frameworks

MPEG DASH has introduced three important frameworks. These variants include DASH with HTTP 1.1, DASH with Server Push and Web Sockets, and finally Server and Network Assisted DASH (SAND). The standard covers the technical details and working flow model of the video streaming process, however the adaptation algorithm is not part of the standard. Each of the above mentioned variants of DASH protocol are designed but are not limited for a specific scenario. In the coming paragraphs we will briefly describe technical aspects and applications of each of the above-mentioned variations of DASH protocol.

1.7.2.1 DASH with HTTP1.1

This is the first framework of MPEG DASH standard protocol, which was introduced in 2012. This is the most popular and widely used protocol. This framework is based on HTTP 1.1, and uses persistent HTTP connections. There are multiple features, which make this a powerful framework. Multiple HTTP connections can be opened between the server and client to fetch the video segments in parallel. Secondly, it also supports pipelining of multiple video segments, which means that client can sequentially request multiple video segments, without waiting for the delivery of the previous video segments. This framework works best for on-demand video content.

Although it is, still the most widely used protocol of DASH but it has certain limitations, which degrades the overall quality of experience of the user. Firstly, the complexity of the algorithm increases due to opening of multiple persistent HTTP connections for parallel fetching of video segments. It also consumes a lot of resources of the clients, which degrades the overall efficiency of the client. The second issue of this framework is that it does not takes any feedback from the network or the server load, as it is a client driven approach, so client is unaware of the current network conditions. It results in either unfairness among multiple clients in the network or it causes congestion in the network. Due to lack of support in the protocol, these issues must be tackled in the adaptation algorithm, which increases complexity of the algorithm.

1.7.2.2 DASH with Server Push and Web Sockets

This is the second variant of MPEG DASH protocol. It is based on HTTP 2.0 web connection. It uses HTTP 2.0 features like Server Push and Web Sockets for enhancement of the performance of DASH. As mentioned before that, on HTTP 1.1 connection, the complexity of the algorithm increased. It also had more latency for live video streaming because the content is generated, encoded and streamed at the same time. Therefore, the previous version of DASH was not very efficient for live video streaming.

HTTP/2 can be used to boost the performance of DASH for live video streaming by using its feature named as Server Push. In server Push, once the client establishes the connection with the server and, the server starts sending the resources to the client, which the client needs in the future, without waiting for its request. Therefore, those resources reach the client quicker and are stored in its local cache, so the client can fetch them when needed. This feature is used in DASH, the server starts sending future video segments to the client, which it needs in the future. It will decrease the start-up delay as well as latency, especially for live videos.

The client needs to enable server push, in order to utilize this feature in DASH. There are three different push strategies for server push. The first one is NO-PUSH, in which the client does not allow server push, and only receives those resources that are requested by the client. Second is ALL-PUSH strategy in which the client allows to push all the video segments as soon as they are available. The last one is K-PUSH, in this strategy; the client allows next K number of segments to be pushed by the server without waiting for the request from the client. Therefore, this feature reduces the overall live latency of the video.

1.7.2.3 Server and Network Assisted DASH (SAND)

The last protocol used server capability to improve its performance for live video streaming, but it was still unaware of the current network conditions. In December 2016, MPEG introduced a variant of DASH named as Server and Network Assisted DASH (SAND), in which the client will be aware of its current network conditions, the state of the server and about also about other clients in the network. This protocol defines standard message format and exchange protocol, which can be exchanged between the servers, network operators and the clients to in order to enhance the overall performance of the video streaming process.

The SAND allows better coordination between all the entities of the network, which can help in the enhancement of performance in video streaming by intelligent caching based on the feedback from the DASH clients. It will also help in optimizing the performance of the servers. The clients will be able to get better user experience based on the network information and it will improve fairness in the network.

There are four differ different types of network entities in the network, as defined in SAND. The first one is DASH Clients, which are streaming the video. The second is Regular Network Elements (RNE), which are unaware of the DASH traffic. Thirdly, DASH Aware Network Elements (DANE), which have the intelligence to read the DASH message and may prioritize, parse or modify the message. Lastly, Metrics Server, which are also aware of DASH messages and are responsible of receiving metrics from the clients.

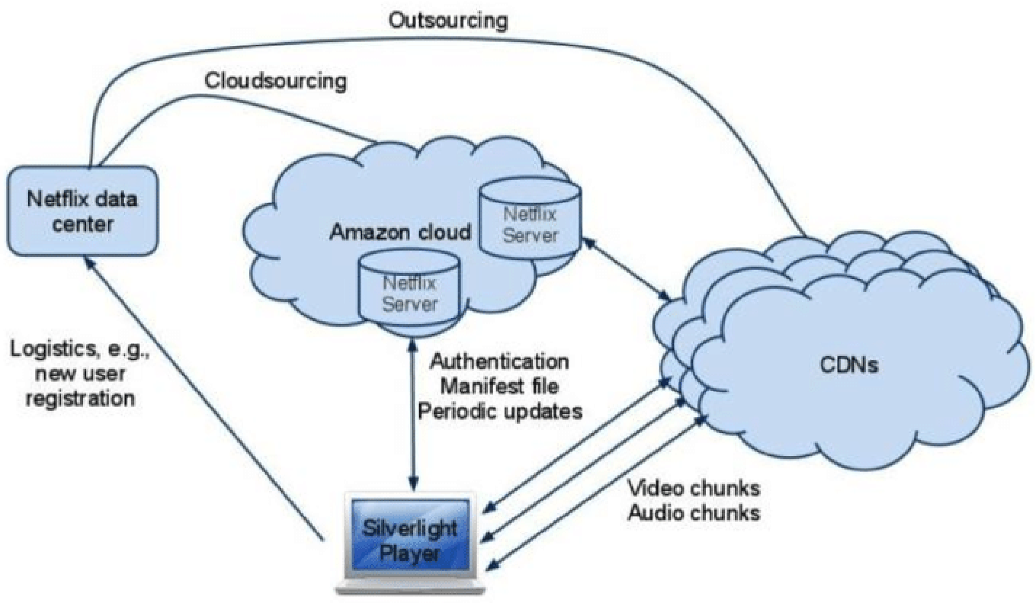
There are four categories of defined messages that are exchanged in the SAND. And in every category there is a set of messages which can be used to exchange the information in the SAND. This defines the overall mechanism and workflow process that is used in SAND, to make it server assisted and network aware DASH architecture. This is the not fully implemented yet, as it depends upon multiple network entities and requires interaction from each stage in the network.

1.8 Leading Commercial DASH Players

Netflix and Hulu are most popular commercial DASH players [12]. Both of them provide subscription-based streaming services. They stream music, movies and other exclusive content to the users by streaming the content over the Internet. Netflix uses DASH protocol, whereas Hulu uses RTMP protocol for multimedia streaming over the Internet. However, each one of them have a different rate adaptation algorithm, which drives the video streaming process. Netflix and Hulu both rely on the third party infrastructure for video streaming. For the distribution of the multimedia content to their users all over the world, Netflix and Hulu utilize the services of Limelight, Level3 and Akamai CDNs. However, the management policies and CDN selection strategies of Netflix are quite different from that of Hulu. In the subsequent subsections, we will discuss the basic architecture and workflow process of Netflix and Hulu, and we will highlight some key points, which distinguish them from other common service providers.

1.8.1 Netflix Architecture

In 2014, the total number of worldwide subscribers of Netflix were more than 48 million [13]. Netflix is mainly dependent on Amazon cloud for majority of its services. There are four key components of Netflix architecture, namely, Netflix Data Centers, Amazon cloud, CDNs and multimedia players. Figure 1.3 describes the basic architecture of Netflix.



Outsourcing
Cloudsourcing
Netflix data center
Amazon cloud
Netflix Server
Netflix Server
CDNs
Logistics, e.g., new user registration
Authentication Manifest file Periodic updates
Video chunks Audio chunks
Silverlight Player

Fig. 1.3: Netflix Architecture [12]

The first and the most important component of the Netflix architecture is Netflix data centers. The basic responsibility of the Netflix data centers in to register the new users and to collect their payment information. After that process is completed, the user is redirected to one of the amazon cloud servers, which hosts the movies. Apart from, the main server, which is responsible for sign up process, all other servers are hosted by amazon. Each amazon server has a different set of responsibilities, which includes, CDN routing, logging, log analysis, DRM, mobile support, etc.

The third component of Netflix infrastructure is CDNs. Netflix have engaged three CDNs for the distribution of their content across the world. These servers include Akamai, Level-3 and Limelight. The basic function of these CDNs is to deliver high quality multimedia content to the end users. The same content is stored in all the CDNs, with same quality.

The last component of the architecture is multimedia player. Netflix have employed Silverlight player for online multimedia streaming to the end users. These players download the content, decode it and then it is played for the user. These players are responsible for playing the content on Desktop PCs, laptops, tablets, mobile phones and on other devices like Roku, Wii, etc.

Now we will describe the detailed and sequential flow of the video streaming process on Netflix. The first step to start the video streaming over the Internet is to open the Netflix website in the web browser. After that, Microsoft Silverlight application plug-in is downloaded on the client's web browser, and the user authentication process is initiated. After the verification of the user, the user is redirected to one of the Amazon cloud server, which hosts the movies.

Once the client clicks on its desired movie icon to play that movie, the player makes a request to the host server to fetch the manifest file for that movie. Netflix generates client-specific manifest files according to the user capabilities and subscriptions. These files store the metadata, which control and drive the video streaming process for a specific client. Netflix uses SSL connection to deliver the manifest file to the client. Manifest file includes the information about available CDNs, preferred CDN, video chunk size, audio/video segment URLs, available representation levels, trickplay information etc.

Netflix uses a chunk size of 4 seconds. Initially, the player downloads the video segments frequently to fill up the buffer and then the periodic download goes on along with the playback. The video player keeps on sending the periodic logs and feedback messages to the control server throughout the playback phase. Netflix uses trickplay interval of 4 seconds and supports the functions like pause, play, rewind, forward and random. Thumbnails of the video are downloaded and added as periodic snapshots for the trickplay. The specifications of these thumbnails are mentioned in the manifest file.

The user specific manifest file of a particular video enlists the available CDNs as well as preferred CDN for that user. Preferred CDN for a particular user remains the same, irrespective of the type of video played, or the geographical location of the user. Netflix assigns a particular CDN to the user and it is kept as its preferred CDN for several days for all type of videos. A user keeps getting video from its preferred CDN and does not switch to the next CDN when the bandwidth decreases. When the bandwidth is decreased, it starts downgrading the bitrate of the video and shifts to lower representation levels. When the bandwidth is dropped up to a certain point that, the playback stops, then Netflix switches to the next CDN in the list. When the client requests a manifest file from the server, it also sends its device capabilities like the screen resolution and the file format supported, to the server. The server generated a manifest file for the user according to its device capabilities. Netflix offers HD videos in 14 different representation levels and SD videos in 12 different representation levels.

1.8.2 Hulu Architecture

Hulu is another popular OTT service provider. Unlike Netflix, Hulu provides subscription as well as free services. Its subscription-based services include HD videos, and the support for video streaming on multiple devices, whereas the free services offer SD videos for desktop PCs. Hulu utilizes Akamai services for the execution of their operations. One of the key features of Hulu is advertisement. A short video ad is played before the main video. Figure 1.4 describes the basic architecture of Hulu.

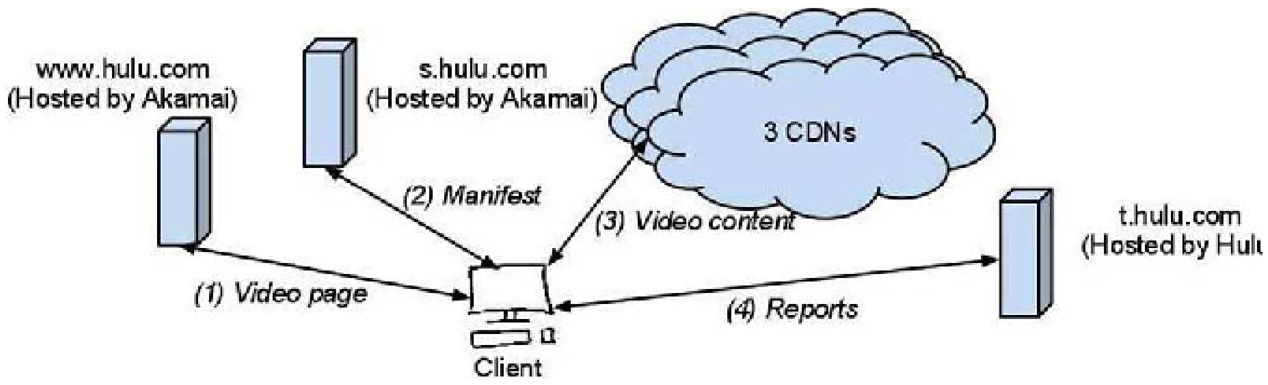


Fig. 1.4: Hulu Architecture [12]

When a client clicks on an icon to play a particular video, firstly, HTML page of that video is opened, and then the client connects to the server s.hulu.com to download the manifest file of that video. The manifest file includes the details like available CDNs, available bitrates etc. Hulu has also employed the same three CDNs as of Netflix. The client checks the hostname of its preferred CDN from the manifest file and then connects to its particular CDN by using DNS hostname resolution, and starts the video streaming. The client also sends periodic feedback reports to the control server. These logs include status of the client's device, current bitrate of the video, current amount of memory utilized, total number of frames dropped, current bandwidth, number of buffer underrun events occurred, etc. This information is sent to one central control server in the US.

Either Hulu uses Real-time Messaging Protocol (RTMP) protocol or RTMP tunneled over HTTP (RTMPT) for video streaming over the Internet. Akamai and Limelight CDNs use RTMPT protocol whereas Level-3 CDN uses RTMP protocol for video streaming over the internet. Huluplus, which is a subscription-based service of Hulu, uses adaptive streaming over HTTP for video streaming on mobile phone devices. Hulu uses one CDN for a particular video and usually switches to another CDN for the next video. The CDN selection strategies of Hulu and Netflix are almost similar. Hulu also stays on the same CDN as long as the video stalling event occurs and then switches to the next CDN.

Hulu sends the manifest files to the client in encrypted format. Manifest file includes metadata and corresponding information of a video for a specific client, like available CDNs, preferred CDN, total number of bitrates available etc. The preferred CDN for a client can be different each time the client requests a video. Hulu randomly assigns a preferred CDN to a client irrespective of the type of video, client ID, time or current network conditions. The stats show that level3 is the most preferred CDN of Hulu.

1.9 Thesis Contributions

We have thoroughly studied the HTTP based streaming strategies, commercial DASH players and research contributions of the authors in this area. We have investigated the research challenges and key issues that are the clients face in different network conditions. Finally, we have developed a rate adaptation algorithm, which provides the solution to most of the investigated problems.

We have developed two variants of our proposed algorithm, named as SHANZ Adaptation Algorithm. The first one is SHANZ-I Algorithm, based on HTTP 1.1 protocol. This algorithm maintains the balance between stability and efficiency of the algorithm by using a feedback control mechanism. We have also used a randomized delay to avoid the bandwidth overestimation problem in multiple clients' scenario. Some of the important attributes are listed below.

- Higher Efficiency of the algorithm at lower Bitrates
- Better stability of the algorithm at higher Bitrates
- Balance between stability and efficiency of the algorithm
- Stability ensured even in drastic network conditions.
- Bandwidth overestimation problem resolved
- Better Fairness among the clients

We have also proposed HTTP 2 based SHAN-II Algorithm, which is mainly aimed for live video streaming where the latency of the video is the major challenge. The key features of the algorithm are highlighted below.

- Minimum latency
- Minimum startup delay
- Computationally efficient
- Less Memory consumption
- Maintains the balance between efficiency and stability of the algorithm.
- Minimal Video Stalling events

Our proposed algorithms are efficient even in drastic network conditions, and maintains the higher quality of experience. The algorithms have achieved better results when compared with other algorithms. Detailed Explanation of our proposed algorithm is given in chapter 3.

CHAPTER 2: LITERATURE REVIEW

There is a long road of evolution from UDP based video streaming algorithms to existing HTTP based dynamic adaptive streaming (HAS) algorithms. Adaptation algorithms can be classified into four broad categories, namely Client-based adaptation, Server-based adaptation, Network-assisted adaptation and Hybrid adaptation schemes. A comprehensive survey of adaptive video streaming techniques is performed in [14], [15]. Adaptation Algorithms and service architectures of most of the commercial content service providers is unknown. There are many factors, which contribute the overall performance and QoE of these CSPs. [16] analyses service architectures of Netflix and Hulu. Author stated that both of them are dependent on third party infrastructure to deliver the video content.

The performance of HAS depends upon numerous factors. Some of the parameters contribute to enhance the performance of HAS system while others contribute in downgrading the overall performance. It is important to have a thorough understanding of, how each parameter effects the HAS system. Multiple contributions have been made in this area of research. [17] Claimed that frequent switches in video quality downgrades the overall Quality of Experience of the user. They suggested that there should be least number of video quality switches in the playback. They also claimed that smooth adaptation has a better user experience than quick adaptation, which means that incremental downgrade of video quality should be preferred rather than one switch of a higher magnitude. However, they claimed that abrupt switch in upgrade of video quality can perceive higher QoE rather than a smooth upgrade. Finally, they deduced that that higher average quality at the end of video is also perceived as higher QoE for the user.

There are numerous work carried out for the comparison of performance of existing HAS based solutions. In [18] authors used vehicular environment to compare DASH, HLS, MSS and HDS. After the comparison of results, they found that MSS outperformed other solution in vehicular environment, with having higher average video quality with minimum number of video quality oscillations.

In [19] authors compared the adaptive and non-adaptive video streaming in vehicular mobile devices and claimed that when the bandwidth of the networks fall, with the help of adaptation stalling events can be reduced to 80% in comparison to non-adaptive streaming. They also analyzed the effect of change on the size of threshold buffer size and video segment size, on the stalling events. They deduced that in vehicular devices, the optimum value of threshold buffer size is of 6 seconds, and further increase of buffer size will give rise to initial delay. They also found that large target buffer would also consume more memory, which is a vital resource in vehicular mobility. After the analysis effect of change in video segment size, they concluded that smaller segment sizes are efficient for quick adaptation of video quality but they also increase the overhead for encoding the video. On the other hand, using larger video segments decrease the capability of the client for quick adaptation, in the case of change in the bandwidth of the network, so it can lead to more stalling events. They suggested balancing the tradeoff by using longer buffer threshold value, which means more video segments would be stored in the buffer, before the playback of the video starts.

Adaptation Algorithms can be classified into 4 broad categories, based on their working mechanism. These categories include Server-based Adaptation Algorithms, Client-based Adaptation Algorithms, Network-Assisted Adaptation Algorithms and Finally Hybrid Algorithms. Client-based algorithms are those algorithms where the adaptation module is run on the client node. On the other hand, in Server-based adaptation algorithms, the adaptation module is run on the server. In Network-Assisted Adaptation algorithms, networks send the information about the current state of the network, which is used for the adaptation of video quality of clients. Finally, Hybrid algorithms are those adaptation algorithms, which work by using two or more above-mentioned approaches. Each class of the above mentioned algorithms have its advantages and disadvantages. However, the most dominant category is of client-based adaptation algorithm, as the majority of the research has been conducted in this area. In the preceding subsections, we will discuss the research contributions made in each of the mentioned categories of adaptation algorithms.

2.1 Server-based Adaptation Algorithms

Although majority of the work for adaptation of video quality in HAS, is done on the client side, but, still there is some work on control and optimization of server for the better adaptation of video quality. In [20] authors revealed that "Block Sending Algorithm" is used by YouTube as an application control flow mechanism.

[21] proposed a mechanism for limiting the rate of video traffic on the server with the help of TCP congestion widow. They resolved the problem of network congestion and packet loss. They proposed a mechanism on server to limit the upper bound of TCP congestion window, by using phenomena called Trickle, which is a function of RTT and the rate of streaming. They evaluated their work on YouTube data centers and found that their algorithm reduced the average rate of packet loss of TCP by 43%, and RTT by 28%.

Authors [22] implemented Server-side adaptation algorithm, in which they propose a bitrate shaper to be deployed at the gateway of the network which controls the bitrate switching to increase stability and fairness. In server-side techniques, adaptation algorithm is implemented on the server, which increases its overhead and complexity as server saves the state of every client.

In [23] the authors proposed an algorithm, which controls the flow of packets to the clients by phenomena called 'Zippy Pacing'. They regulate the segments in such a way that it reaches the client just when they need. It helps the server in load balancing. The segments are transmitted immediately initially until the buffer of the client is filled up to a certain limit, after that point the server starts delaying the packets. It also avoids the wastage of bandwidth in case of the client stops or switches the playback without finishing the buffered video. The algorithm improved the overall performance of the server and helped the clients to utilize the network bandwidth efficiently.

Another area of research, which in getting popular in the research community of adaptive video streaming industry is the use of Multipath TCP (MPTCP) [24]. Although this is an emerging area of research and very less work is done in this field, still there are few people who have contributed in using MPTCP for HAS. [25-27] are some of the major contributions in using MPTCP for HAS, in these works they can have analyzed the performance of the clients by adding the cross layer scheduler, which prioritize the video streams of different channels according to the requirements of the user. [27] developed the analytical framework which finds the best access channel for video streaming in heterogeneous networks. They used Forward Error Correction coding scheme to minimize the end-to-end delay and the rate packet loss.

2.2 Network-Assisted Adaptation Algorithms

Author [28] evaluated Network-assisted adaptation algorithms, which require active interaction between client and the network. They compared three categories of network-assisted strategies. Firstly, Bandwidth Reservation approach, which assigns a dedicated slice of the bandwidth to a bunch of video. Secondly, Bitrate Guidance approach in which Network Controller (NC) computes the optimal bitrate for the client and then client requests the video at that bitrate and finally, Bitrate Guidance with Bandwidth Reservation approach is the combination of the two approaches in which NC computes the bitrate for the client and assigns a bandwidth slice. They examined that bitrate guidance provides better fairness whereas bitrate allocation provide better average video quality.

In [29] authors proposed a traffic shaping mechanism in the network by adding a bitrate manager on the default gateway of the home network. They resolved the problem of video instability and unfairness among clients, in the case when multiple video streams are flowing through the network. They used Microsoft Smooth Streaming Player for the validation of their work. They claimed that the stability of the videos increased and there were lesser video quality oscillations. The authors also claimed that the fairness amongst the clients was improved, when they share a bottleneck link of the network.

Authors proposed a DASH system for bandwidth measurement named as QDASH [30]. One proxy node was in the network just before the server, which tells the maximum video quality level a client can support, in the current network conditions. Available bandwidth was computed by using RTTs. Another proxy node was installed on the client side, which tells the client the most suitable quality level under current network conditions. They deduced that the clients prefer gradual adaptation of video quality rather than a sudden change, which effects negatively on user’s quality of experience.

In [31] authors proposed a network based framework for the video traffic in the cellular networks. The framework distinguishes the DASH video traffic from other traffic and it allocates the resources to the DASH clients according to the user requirements. The resource-allocation framework maximizes the stability of the video. It also distributes the network resources among DASH clients in such a way that it maintains the level of fairness between them.

2.3 Client-based Adaptation Algorithms

The major contribution of research in HAS is done in client-side adaptation algorithms. Client-side algorithms are more efficient because they do not require any modification on the server or network as all the adaptation logic is run on the client. The client-based video streaming algorithms drive the streaming session by adjusting the video quality based on current or past network conditions and some supporting parameters. Current available throughput from the source server is considered as the most important criteria for predicting the future behavior of the network. Client-based algorithms use three different approaches for the adaptation of video streaming. These approaches include bandwidth-based adaptation, buffer-based adaptation and finally time-based adaptation techniques. Some of the algorithms may overlap in two different techniques by using a hybrid approach, but majority of them use one of the above approach as their main driving force.

2.3.1 Bandwidth based Adaptation Algorithms

These algorithms use current or past throughput of the network to predict the future behavior of the network, and adapt the video quality accordingly. It is the most commonly used approach. These algorithms select the next video segment with representation whose bit rate is best matches with the predicted future throughput of the network [32]. The most common way to compute the bandwidth is by using TCP congestion window. Some of the algorithms use this instantaneous bandwidth to predict the future video segments. There is high fluctuation in the instantaneous bandwidth of the network, which will cause unwanted oscillations in the video quality. So most of the algorithms use some averaging technique to minimize the effect of outliers in the prediction.

PANDA algorithm [33] reduces the stability issues in the bottleneck link by implementing probe and adapt principle for the adaptation of the bitrate based on average data rate available. This principle is similar to congestion control mechanism of TCP but it works at the Application Layer.

FESTIVE algorithm [34] was proposed to improve stability, efficiency and fairness of the video, among multiple DASH clients sharing a bottleneck link. Bandwidth estimation randomized scheduler and delayed update mechanisms were used in this algorithm to achieve the desired goal. They developed the algorithm as open source media framework to test it on multiple commercial players and identified several root causes on performance degradation.

The authors in [35] proposed an adaptation algorithm which used smoothed measurement of throughput by AIMD, the algorithm used step-wise up and aggressive down switching approach for the adaptation of video quality. The algorithm worked without requiring any prior information from the transport layer like RTT etc. The stepwise approach was used to probe the spare capacity of the network and if found, the algorithm adapted to the higher representation of the video. They claimed that the probing method was efficient to use spare capacity of the network and for avoiding congestion in the network.

Author in [36] have developed a control system for adaptive video streaming. The system consists of two controllers. One of the controllers is at the client side and the other one is at the server side. The controller selects the representation level of the next video segment based on the bandwidth estimation. The authors claimed that their system improves the QoE of the user.

In [35] authors introduced a rate adaptation algorithm which uses segment fetch time (SFT) to detect the changes in the bandwidth of the network and to detect the congestion in the network. They used SFT to decide the bit rate of the next video segment to download. The authors also extended their work in [37] to compare the sequential and parallel segment fetching mechanisms. They compared expected segment fetch time with the actual segment fetch time, to detect the current state of the network, and based on these parameters, they computed the bitrate of the next video chunks to download.

2.3.2 Buffer-based Adaptation Algorithm

These algorithms use current buffer occupancy level and rate of change in buffer level of the client to predict the representation level of the next video segment. Either the rate map of buffer is directly used as a decision factor [38, 39] or threshold values of the buffer level are used to make bounded regions [40, 41].

In [42] authors proposed buffer-based adaptation algorithm called BIEB algorithm for scalable video codec (SVC). In BIEB algorithm, the client fills the buffer with the base layer to stabilize the video playback. Before it downloads the enhancement layers. The algorithm maximizes the video quality and reduces video quality switches and playback interruptions. However, the algorithm does not considers stalling events or video quality oscillations during the peak hours, when the cross traffic exists in the network.

AMBA+ algorithm [43] uses precomputed buffer maps to find buffering probability for each representation of the segment and selects the highest possible representation having buffering probability less than a certain threshold value. The authors claimed that algorithm proved computationally efficient and stable, preventing video quality oscillations.

Author [44] Proposed an online buffer based control algorithm named as BOLA, by using utility maximization problem. Utility increases with increase in average bitrate and decreases when rebuffing events occur. It also provides the opportunity to set the weights to each of the key metrics, average video quality and rebuffing event by setting an explicit knob. Author claimed that BOLA achieved higher utility than other offline algorithms like ELASTIC and PANDA.

Author [39] proposed a rate adaptation algorithm, which selects the bitrate for the video segments based on the current buffer status of the client. The algorithm only takes one parameter as an input, i.e. buffer status of the client. The algorithm does not take into account other factor like current bandwidth or status of the network to optimize the QoE of the user.

In [45] authors proposed a buffer based mechanism to improve the QoE of the user by minimizing the video quality switches and playback interruptions. They have also proposed an online buffer-based controller by using control theory. The controller computes the bitrate of each video segment to be requested, based on the buffer status of the client. The authors claimed that their algorithm has showed the better results as compared to other similar works.

2.3.3 Time-based Adaptation Algorithm

This category of algorithms use Segment Download Time (SDT) as a parameter to find the best representation for next video segment. SDT is defined, as the time consumed when the request was made to download a certain segment, until the moment it was completely downloaded in the buffer of the client. SDT depends upon two main factors; one of the factors is available throughput and second is size of the segment to be downloaded. These algorithms aim to download those video segments whose SDT synchronize with their Segment Playback Time to ensure that the segment is completely downloaded before their playback time arise.

SARA Algorithm [46] takes segment size into account for deciding the next bitrate. They modified the MPD file to add segment size of each segment. Weights are assigned to each segment, which are proportional to its segment size. Client decides the next segment to download by considering segment weights, buffer status and available bandwidth.

Authors [35] use the ratio of Segment Playout Time and Segment Download Time for a segment k, for the adaptation of the video quality. They proposed that if this ratio is higher than a certain threshold value, then a higher representation will be selected for that video segment, and if the ratio is lower than a certain threshold value, then the lower representation will be selected. Since this algorithm depends upon instantaneous values of SDT and its payback time, it suffers from unwanted video quality oscillations.

In another approach, the authors [47] used a normal distribution of segment download time (SDT) and introduced analytical model of the play-out buffer. The model calculated rebuffering probabilities for each representation and the representation having the minimum probability, was selected for next video segment. This normal distribution depends upon multiple segments, so it was smooth and more resistive to the outliers.

2.4 Hybrid Adaptation Algorithms

ELASTIC algorithm [48] was proposed to avoid unfairness in the bandwidth utilization due to ON-OFF patterns in the steady state in multiple clients scenario, which avoids ON-OFF pattern by using Feedback Control Theory. Authors claimed that ELASTIC achieved higher fairness then PANDA when competing with greedy TCP flow.

The authors proposed SVAA algorithm [40], which used feedback control mechanism to select the representation of the video segment having bitrate which matches the estimated throughput multiplied with buffer-based factor F_k. This factor is a product of three sub-factors, which are the size of buffer, rate of change in buffer and its chunk size. SVAA drops its video quality, if its buffer level drops half of its target value. This algorithm uses hybrid approach as it uses available bandwidth as well as its buffer occupancy level to make its decision of adaptation of the video.

The authors in [49] presented a client side rate adaptation algorithm based on the control theory. The proposed MPC algorithm that decides the appropriate bitrate of the next video chunk by using the current buffer status of the client as well as available throughput. The authors claimed that their algorithm has outperformed other similar algorithms.

In [50], the authors presented an algorithm that performs stateful prediction of the throughput. They analyzed the large dataset, provided by a commercial provider in China and claimed that the sessions having common attributes like its geographical location or network channel, have a similar network layer throughput. However, at client level the throughput might not be similar because of its dependency on many other factors as well. The algorithm also accurately computed the initial startup delay of the video. The algorithm managed to give 40% better results of prediction of bandwidth than other similar algorithms.

The authors proposed SQUAD algorithm [51] which used which enhanced the average video quality and minimize the oscillation of video quality by using throughput as well as buffer level. They used a perimeter of spectrum in their quality of experience metrics, which was defined as deviation of current video quality from average video quality. When the algorithm was compared with others, it showed better QoE with lesser video quality oscillations.

Authors in [52] proposed SDN based QFF framework to improve the fairness among multiple clients sharing the same network. QFF enhances the fairness based on two parameters, i.e. current condition of the network and screen resolution of the client's device. Clients are connected to the SDN controller, which is responsible for the enhancement of QoE and fairness among the clients. However, the framework does not consider any other information, like the buffer status of the clients, which might cause buffer underruns. Moreover, it only works on limited number of clients and increases the latency of the video, which may negatively affect the performance of the clients.

CHAPTER 3: PROPOSED METHODOLOGY

In this chapter, we will explain the methodology of our proposed adaptation algorithms named as SHANZ-I Algorithm and SHANZ-II Algorithm. SHANZ-I is based on HTTP 1.1 protocol, whereas SHANZ-II algorithm utilizes HTTP 2 protocol, and it is mainly aimed for live video streaming with minimum latency and startup delay. Following subsections explain both the variants of our proposed algorithm in detail.

3.1 SHANZ-I Algorithm

The basic principle of SHANZ-I Algorithm is to improve the Quality of Experience of the user by using feedback control mechanism and dynamic step up function, which act as an explicit knob to optimize the efficiency and stability of the algorithm. Secondly, we have also resolved bandwidth overestimation issue occurred in multiple clients' scenario by using randomized download delay, which has also enhanced the fairness of the algorithm among multiple clients.

Whenever client starts online video streaming session, it establishes a TCP connection with the server and requests to download the Media Presentation Description File (MPD) file of a particular video from the server, which contains all the information about that video like total number of available Representations levels, video segment duration etc. The client starts requesting video segments from the server according to available bandwidth. Client runs the adaptation algorithm to compute the next video representation index for every segment and requests the server accordingly.

3.1.1 Estimated Weighted Throughput

To calculate the bitrate for the next representation index, adaptation algorithm need to compute estimated throughput (τ_n^{est}) of each segment but due to sharp changes in the bandwidth, instantaneous throughput is not a better option, as it will increase instability of the video. So Weighted Estimated Throughput ($\tau_{n+1}^{weighted}$) is used instead of instantaneous throughput. Weighted Estimated Throughput is computed by taking weighted average of the estimated throughput of the last ten segments. Highest weight is assigned to the most recent segment. The symbol (w) is the weight assigned to a particular video segment, as described in Equation 3.1 below.

$$\tau_{n+1}^{weighted} = \frac{1}{\sum_{k=1}^{k=10} kw} w\,\tau_{n-9}^{est} + 2w\,\tau_{n-8}^{est} + 3w\,\tau_{n-7}^{est} + . \;\; . \;\; . + 9w\,\tau_{n-1}^{est} + 10w\,\tau_{n}^{est} \tag{3.1}$$

3.1.2 Stability Function

The next step is to calculate the stability of the video, which depends upon video quality oscillations. Stability *(δ)* defines the function, where η defines the number of video quality switches in the last 30 seconds. α is the regulating factor for smoothness, having an empirical value of 0.15. Equation 2 describes the stability function.

$$stability\ (\delta) = \frac{1}{e^{\alpha\eta}} \tag{3.2}$$

Stability function drops exponentially, with the increase of number of video quality switches in a unit time. Figure 3.1 describes the behavior of stability function with respect to video quality switches.

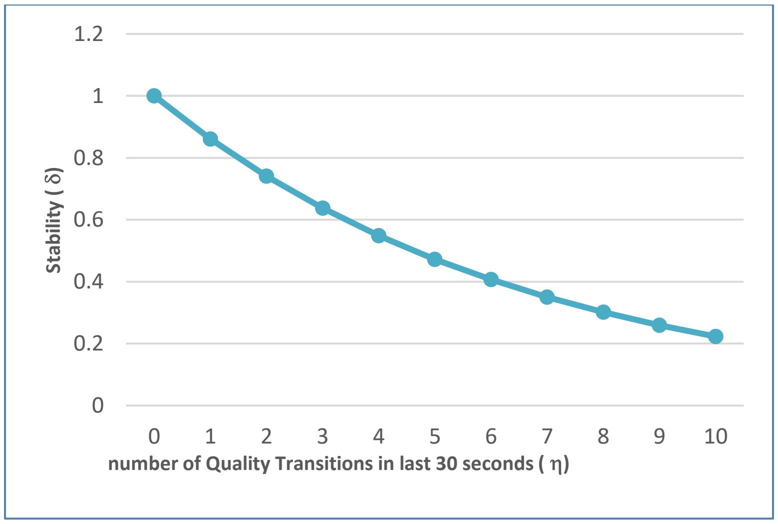


Fig. 3.1: Stability function graph

3.1.3 SHANZ-I Algorithm Pseudo Code

The algorithm takes current segment index (n), current buffer index (β_{cur}), current Representation index ($\Re_n^i$), and Estimated Throughput at current segment index (τ_n^{est}) as input parameters, and performs the computation to return the Next Representation Index ($\Re_{n+1}^i$), and Next Segment Delay (β_{delay}) as an output. Table 3.1 describes the pseudo code of our proposed algorithm named as SHANZ-I adaptation algorithm.

TABLE 3.1: SHANZ-I ALGORITHM PSEUDO CODE

```
SHANZ-I Adaptation Algorithm
Data:
n: Video segment index
$\Re_n^i$: $i$th Representation level for nth segment
$\beta_{cur}$: Current buffer level
$\beta_{min}$: Minimum buffer level
$\beta_{max}$: Maximum buffer level
$\beta_{opt}$: Optimum buffer level
$\beta_{rand}$: Random buffer level
$\delta$: Stability function
$\Omega$ Dynamic Step-up Function
$\tau_{n+1}^{weighted}$: Weighted Throughput of last 20 seconds
$B_{avg}^i$: Average Bitrate at Representation index i
$\eta$: Video quality switches in last 30 seconds
$\Delta$: Throughput threshold
$\alpha$: Regulating factor
Input:
$n, \Re_n^i, \tau_{weighted}, B_n^{avg}, \beta_{cur}$
Constant:
$\beta_{opt} = (\beta_{min} + \beta_{max})/2;$
Initialization:
$\Omega = \max(\Re_n^i, \eta);$
$\delta = 1 / e^{\alpha\eta};$
```

```
if( $0 \le n \le \beta_{min}$ )
    fastStart = true;
else
    fastStart = false;
if( $(\Re_n^i > \Re_n^{min})$ && $((B_{avg}^i > \Delta . \tau_{weighted})$ || ( (!fastStart) && $(\beta_{cur} < \beta_{min})$ ))
    $\Re_{n+1}^i = \Re_n^i - 1$;
    $\eta++$;
else if ( $(B_{avg}^{i+1} < \delta . \tau_{weighted})$ && $(\Re_n^i < \Re_n^{max})$ && (fastStart || $\beta_{cur} > \beta_{min}$) && $(\delta > 0.5)$ )
    if(counter $\ge \Omega$)
        $\Re_{n+1}^i = \Re_n^i + 1$;
        $\eta++$;
        counter = 0;
    else
        counter++;
else if( $\delta < 0.5$ )
    $\Re_{n+1}^i = \Re_n^i$ ;
else if ( $\beta_{cur} > \beta_{max}$ )
    $\beta_{rand} = rand(\beta_{opt}, \beta_{max})$;
    $\beta_{delay} = \beta_{curr} - \beta_{rand}$ ;
Output:
$\Re_{n+1}^i, \beta_{delay}$
```

3.1.4 Fast start phase

The algorithm starts in the Fast Start Phase, and remains in that phase for first 10 segments, so that the buffer can quickly fill up to the level of β_{min}. When the algorithm is in this phase, it can also increase its video quality, if the bandwidth is available even though its buffer level is below the value of β_{min}. It helps the client to minimize the startup delay of the video.

3.1.5 Video quality Quality Increment Function

For better efficiency and stability of the video, algorithm passes through multiple checkpoints before increasing the representation index of the video. First condition to upgrade the video quality is that its Current Representation Index ($\Re_n^i$) should be less than highest Representation Index of the video ($\Re_n^{max}$). The client can get the information of available representations by downloading the MPD file of the video from the server.

The second condition to fulfill for upgradation of video quality is that the current value of stability function should be greater than 0.5, which means that the algorithm will not upgrade the video quality if more than 4 video quality switches have already occurred within last 30 seconds. This factor avoids the unnecessary video quality oscillations when the bandwidth is fluctuating.

The third criteria to fulfill in order for the upgradation of representation index is that either the algorithm should be in Fast Start phase or the current buffer level of the algorithm should be greater than β_{min}. This factor avoids the upgrade if the buffer level of the client is less than a certain threshold value.

Fourth condition to be fulfilled is that, average bitrate of the next representation index B_{avg}^{i+1} should be less than the factor, $\delta.\, \tau_{weighted}$, which is the current stability value, multiplied by weighted throughput for last 20 seconds. This threshold keeps on getting aggressive as the value of stability function decrease So as the stability decrease, higher value of estimated throughput will be required in order to upgrade the video quality, which means algorithm will further reduce unwanted video quality oscillations.

3.1.6 Dynamic Step-up Function

Each time algorithms fulfills all four of the above conditions, it will increment the value of a counter. The algorithm will finally update the Representation Index o when the value of counter becomes equal to the value of a Dynamic Step-up Function (Ω). Dynamic Step-up function is defined as maximum of the value of Current Representation Index ($\Re_n^i$) and total number of video quality switches in last 30 seconds (η), as described in Equation 3.3.

$$\textit{Dynamic Step-up Function } (\Omega) = \max\, (\Re_n^i\, , \eta) \tag{3.3}$$

There are two main advantages of using a dynamic step-up function. Firstly, step-up function acts as dynamic knob to control the efficiency and stability of the algorithm. It linearly increases the time delay for updating the quality level with the increase of Representation Index. At lower quality levels the efficiency is the main priority of the algorithm because there is no use of delaying the update, when the bandwidth is available, it quickly increments the quality level. However, as the Representation Index increase algorithm keeps on increasing the delay to increment the video quality and increases the stability of the algorithm at higher quality levels.

Secondly, Step-up function also maximize the stability of the algorithm even at lower network bandwidth and in the state of network congestion. If the available bandwidth is low and there are frequent switches even at lower bitrate, step-up function will take the maximum of the value of Representation level ($\Re_n^i$) and number of video quality switches per unit time (η), and will use that value as time delay to increase the quality level.

Finally, the case when Stability of the algorithm is less than a threshold value of 0.5, and the algorithm is unable to increase its Representation Index, it will keep the Current Representation Index of the video.

3.1.7 Video Quality Decrement Function

The foremost condition to decrement the video quality is that Current Representation Index ($\Re_n^i$) should be greater than Lowest Representation Index of the video ($\Re_n^{min}$). If the above condition is true, the algorithm will decrement the video quality if either one of the next two conditions is true.

Firstly, if the algorithm is not in fast start phase, its Representation Index will be decremented if its current buffer level (β_{cur}) is less than the threshold value of β_{min}. This condition will prevent the client from buffer underrun or stalling events. Secondly, the algorithm will decrement the video quality when Average bitrate at Current Representation Index (B_{avg}^i) is greater than a certain threshold value (Δ) of Weighted Estimated Throughput ($\tau_{weighted}$). This means that available throughput is not enough to stream the video at current representation index so the algorithm will downgrade the video quality to avoid playback interruption.

3.1.8 Randomized Download Delay

When the current buffer level of the client (β_{cur}), becomes greater than the maximum buffer level β_{max}, the algorithm will stop the download of subsequent segments for a random time period until the buffer level reaches a threshold value of (β_{rand}), which is calculated by random function from a minimum range of β_{opt} and a maximum range of β_{max}, as described in Equation 4.

$$\textit{Random Buffer Level}\ (\beta_{rand}) = rand\ (\beta_{opt}\ , \beta_{max}) \tag{3.4}$$

Randomized download delay also resolves the bandwidth overestimation problem in multiple clients. Bandwidth overestimation occurs due to non-overlapping of on-off patterns of the clients. So due to randomized download delay, there can be partial overlapping in the on-off pattern of the clients. Therefore, it will increase the fairness of the algorithm by eliminating any chances of biasness among the clients.

3.1.9 Finalization

The adaptation algorithm resides on the client and its function is to drive the process of video streaming in such a way that it maximizes the overall quality of experience of the user. For every next video segment, the adaptation algorithm is run on the client and it passes through the each of the above-mentioned stages. The next video segment is selected according to the conditions mentioned in the algorithm. After that process, the adaptation algorithm returns the next representation index and time delay to the client and the client makes that request to the server accordingly. The process keeps on repeating until the playback is finished.

3.2 SHANZ-II Algorithm

The second variant of our proposed algorithm is SHANZ-II, which is based on HTTP/2 protocol. This algorithm utilizes the key features of HTTP/2 to address the key issues of HTTP 1.1 based algorithms. The algorithm is mainly aimed for live streaming videos, where the main challenge is to cope with the latency of the video. The algorithm ensures the seamless live video streaming, with minimum startup delay and latency. Some of the key differences between HTTP 1.1 and HTTP are highlighted in the next subsection. After the comparison of two protocols, our proposed algorithm is explained in the subsequent subsections.

3.2.1 Comparison of HTTP 1.1 and HTTP/2

Some of the main challenges in HTTP 1.1 were higher latency, higher startup delay and higher memory consumptions. Thanks to HTTP/2 protocol, which has addressed these issues. Three main key features of HTTP/2 are dedicated header compression scheme (HPACK), server push and HTTP streams. HTTP/2 has resolved some of the biggest issues of HTTP 1.1, i.e. higher latency and higher memory consumption.

In HTTP 1.1, only supported unidirectional flow of data at a certain time, in one connection. So, to speed up the process one of the solutions was pipelining, but it resulted in head of the line blocking. Another solution was to open multiple connection between server and client, but this solution results is large consumption of resources. HTTP/2 offers the solution by introducing multiplexing of multiple streams in one HTTP connection. Each stream is a bi-directional and independent sequence of frames that are exchanged between the server and client, in a single http2 connection. Each stream has its unique Stream ID. Each stream has a priority number, which defines its significance. So the streams with higher priority are sent first. If one stream is dependent on another stream, it can be signaled by using a priority tree. Priority of the streams can be changed at the run time to give higher priority to the resources, which the client requires urgently.

HTTP is a stateless protocol, which means that each http request a lot of prior information so that the server can process the request. It includes a lot of redundant metadata, cookies, headers and prior request information. The number of objects on a webpage are increasing exponentially, so a large number of http requests are made to load these objects. Due to redundant information sent in each http request, size of the message becomes so large that it may exceed the initial TCP congestion window. This slows down the download of resources. HTTP2 provides a secure header compression mechanism called HPACK. HPACK compress the headers by using a static dictionary, which includes the list of common headers, a dynamic dictionary and static Huffman coding. HPACK is a secure header compression mechanism and it avoids the breaching of data.

The third and the most powerful feature of http2 is Server Push. When the client establishes a connection with the server and starts requesting the resources, there are certain resources, which the client will definitely request in the future. Therefore, instead of waiting for the client to make a request, the server speeds up the process by sending the resources to the client in advance. This mechanism is known as server push. The pushed resources are added to cache of the client, from where they can be retrieved by the client. The server can only push the resources when a client explicitly allows the server push. The server can notify the client about the resources to be pushed in the future by a PUSH_PROMISE message. A client can terminate a pushed stream by sending RST_STREAM message to the server.

3.2.2 HTTP/2 for Adaptive Video Streaming

HTTP2 was released in the mid of 2015. It is relatively a new protocol, so there is very less contribution in use of http2 for adaptive video streaming. Majority of the commercial players are still running on http1.1. One of the reason might be that http2 is not compatible with http1.1, as http2 is a binary protocol unlike http1.1. Although http2 is not fully used for adaptive video streaming, but it can be a future for efficient video streaming in the future.

HTTP2 can be used to resolve some of the key issues of adaptive video streaming. HTTP2 features like multiple streams, header compression and server push can be used to decrease the overall latency and startup delay of the videos. Live latency is one of the biggest issues in live video streaming. Live latency is defined as the amount of time delay occurred between the actual happening of live event and the moment its video reach the client and is ready to be played. One of the reasons for high latency is because of the number of RTTs required for downloading each video segment. Firstly, client makes the request to the server for a video segment, then the server responds by sending the video segment, and finally, the client acknowledges the server about the received video segment. A lot of time and bandwidth is lost during this process.

One of the solutions to decrease the latency could be to increase the size of the segment, so that lesser number of requests are generated per unit time of video streaming, so lesser number of RTTs and less latency. However, this solution has two major disadvantages. Firstly, it will increase the size of the packet, which means one video segment will be transmitted by splitting it into multiple packets, which might further increase the latency of the video. Secondly, by using larger segment size, the efficiency of the algorithm will be decreased, because it will take more time to respond to a change in the network. Therefore, a better solution is required to address the problem of latency. As mentioned earlier, http2 is the solution for many challenges that were faced in adaptive video streaming due to lack of efficiency of http1.1.

3.2.2 Goals and Challenges of SHANZ-II Algorithm

One of the most important features of http2 is server push. This feature can play a vital role in the enhancement of the performance of the algorithm by reducing the latency of the video. An optimal push strategy is required to get the maximum benefit from this feature. Push strategy defines which video segments should be pushed, what is the criteria to push the video segments, How many segments should be pushed at a time. These parameters will decide the overall performance of the algorithm.

When the server will start pushing the video segments, it will only push the segments of a certain representation level, so the client cannot adapt to the changes in the bandwidth while the server is pushing the segments. The client will either accept the pushed segments and will wait until all the segments are received or it will cancel the stream that is pushing the video segments, by sending a RST_STREAM message to the server. So, a dynamic mechanism is required which can decide what is the best time to allow the server push and what are the ideal number of segments to be pushed at a time. Server push feature should be used in such a way that it does not effect on the efficiency of the algorithm, and the algorithm should still be quick to respond to any change in the network.

3.2.3 Proposed Algorithm

We have proposed SHANZ-II algorithm, which utilizes HTTP/2 features to boost its performance. The algorithm minimizes the start-up delay and latency of live streaming and on-demand videos. SHANZ-II algorithm is an advancement of SHANZ-I rate adaptation algorithm, it extends the basic logic of the prior algorithm. Dynamic step-up function, stability function and Randomized download function of SHANZ-II algorithm are similar to that of SHANZ-I algorithm. Table 3.2 below describes the pseudo code of the proposed algorithm, named as SHANZ-II Rate Adaptation Algorithm.

TABLE 3.2: SHANZ-II ALGORITHM PSEUDO CODE

SHANZ-II Adaptation Algorithm

Data:

n: Video segment index	$\mathfrak{R}_n^i$: ith Representation level for **n**th segment
δ: Stability function	**Ω:**Dynamic Step-up Function
η: Video quality switches in last 30 seconds	**Δ:** Throughput threshold
B_{avg}^i: Average Bitrate at Representation index **i**	**α:** Regulating factor

$\tau_{n+1}^{weighted}$: Weighted Throughput of last 20 seconds

$PushSegments(k)$: Allow the server to push next **k** number of segments.

$\{\beta_{cur}, \beta_{min}, \beta_{max}, \beta_{opt}, \beta_{rand}\}$: Current, Min, Max, Optimum, Random buffer level

Input:

$n, \mathfrak{R}_n^i, \tau_{weighted}, B_n^{avg}, \beta_{cur}$

Constant:

$\beta_{opt} = (\beta_{min} + \beta_{max})/2;$

Initialization:

```
$\Omega = \max(\mathfrak{R}_n^i, \eta);$
$\delta = 1 / e^{\alpha\eta};$
if( $0 \le n \le \beta_{min}$ )
    fastStart = true;
else
    fastStart = false;

if( ($\mathfrak{R}_n^i > \mathfrak{R}_n^{min}$) && (!fastStart) && ($\beta_{cur} < \beta_{min}$) )
    if($B_{avg}^i > \Delta . \tau_{weighted}$)
        $\mathfrak{R}_{n+1}^i = \mathfrak{R}_n^i - 1;$
        $\eta++;$
    else
        serverPush = true;
        pushSegments(k);
```

```
else if ( ($\Re_n^i < \Re_n^{max}$) && (( $\beta_{cur} > \beta_{min}$) || (($fastStart$) ))
    $if(counter < \Omega)$
        $\Re_{n+1}^i = \Re_n^i;$
        $serverPush = true;$
        $pushSegments(\Omega - counter);$
        $counter + +;$
    $else\ if((counter \geq \Omega))$
        $if\ ((B_{avg}^{i+1} > \delta\,.\,\tau_{weighted})\ ||\ (\delta < 0.5)\ )$
            $\Re_{n+1}^i = \Re_n^i;$
            $serverPush = true;$
            $pushSegments(k);$
        $else\ if\ (B_{avg}^{i+1} < \delta\,.\,\tau_{weighted})$
            $if(\delta > 0.5)$
                $\Re_{n+1}^i = \Re_n^i + 1;$
                $\eta + +;$
                $counter = 0;$

    $else\ if\ (\beta_{cur} > \beta_{max})$
        $\beta_{rand} = rand(\beta_{opt}, \beta_{max});$
        $\beta_{delay} = \beta_{curr} - \beta_{rand};$
    Output:
    $\Re_{n+1}^i, \beta_{delay}$
```

3.2.4 Server Push

SHANZ-II algorithm uses server push feature of HTTP/2 to minimize the start-up delay and latency of the video. With server-push feature, the server sends the future resources to the clients even before the client requests. The server sends those resources, which are most likely to be requested by the client in the future. This phenomenon can be used in video streaming, and the client can allow the server to push the future video segments in advance. However, server-push can have one drawback. Once the server is pushing the video segments, the client can no longer adapt to the changes in the network, because the server will only push video segments of one representation level. So, efficiency of the algorithm may decrease. Therefore, a mechanism is required in order to use this feature in such a way that latency and start-up delay of the video is minimized without compromising on the efficiency of the rate adaptation algorithm.

We have developed an intelligent and efficient control mechanism in our proposed algorithm to use server-push feature. The client will only allow the server to push limited number of video segments under certain conditions. There are 3 moments, when the client will allow the server to push the video segments. We will explain each of them in the preceding paragraphs.

Firstly, when the buffer level of the client drops less than β_{min}, if the available throughput is less than a certain threshold value $\Delta\,.\,\tau_{weighted}$, the algorithm will decrease its quality level. However, if the buffer level is less than β_{min}, and the available throughput is higher than a certain threshold value $\Delta\,.\,\tau_{weighted}$, the client will enable server push feature for next *k* number of segments. The value of *k* can be set according to the video segment length and minimum buffer threshold value. By allowing server push feature, the buffer vale will quickly increase, so the chances of video stalling are minimized. The server will push the segments from the current representation level of the client.

The second and third point to allow server push occur when the client is either in fast start phase or when the buffer value of the client is greater than β_{min}. The algorithm uses the same dynamic step-up function as in SHANZ-I. On each representation level, the client waits for certain number of segments Ω, before upgrading its video quality. This number of segments are defined by the dynamic step-up function. The client upgrades its representation level when the value of $counter$ is greater than Ω. But, if the value of *counter* is less than Ω, it means the client will not upgrade the video quality for a certain number of segments, and the available throughput is greater than average throughput required for current representation level. The client will allow server push for next $\Omega - counter$ number of video segments. On this condition, it is obvious that the client will stay on its current representation level; unless the value of *counter* increases the value of Ω, therefore the server push feature is enabled by the client to reduce the latency of the video.

Finally, the client will also allow server push when the value of *counter* is also greater than Ω, but either the value of stability δ is less than 0.5 or the average throughput required for the next representation level is less than the available throughput. At this point, the client will stay at its current representation level and it will enable server push for next *k* number of video segments. The server will push next *k* video segments from the current representation level.

The sever push feature reduces the latency of the video, which is one of the main challenges in the live video streaming. This feature will smoothen the video streaming process and enhance the overall quality of experience of the user by minimizing the start-up delay, latency and stalling events of the video.

3.2.5 Header Compression

HTTP2 provides a secure header compression mechanism called HPACK. HPACK compress the headers by using a static dictionary, which includes the list of common headers, a dynamic dictionary and static Huffman coding. HPACK is a secure header compression mechanism and it avoids the breaching of data. This feature also reduces the latency of the video, because it reduces the overhead, which is cause due to redundant information sent over the network. With the help of header compression, the bandwidth consumption is also decreased because it saves the extra number of bytes sent over the network.

3.2.5 Multiplexing of Streams

HTTP/2 allows multiplexing of multiple streams in one HTTP connection. Each stream is a bi-directional and independent sequence of frames that are exchanged between the server and client, in a single http2 connection. Each stream has its unique Stream ID. Each stream has a priority number, which defines its significance. So the streams with higher priority are sent first. If one stream is dependent on another stream, it can be signaled by using a priority tree. Priority of the streams can be changed at the run time to give higher priority to the resources, which the client requires urgently. With the help of this feature, multiple streams can be sent on single HTTP connection, which will enhance the performance of the algorithm.

3.2.6 Alternate Service

HTTP/2 has the alternate service capability, which can be offered by the servers to the clients. Server can utilize this service to redirect the client to another server that might be at a different geographical location. Servers can offer Alternate service to the clients for multiple reasons like load balancing or they can redirect their clients to a server, which is at a closer distance from the client and has lesser latency. Alternate service can be used by the server for the segmentation of the clients into multiple groups, and offering services according to their geographical locations, capabilities or subscriptions.

The server offered via alternate service may have a different protocol configuration and may have security issues. The client needs to authenticate the alternate server offered before switching to that server. Alternate service also has a freshness lifetime, after which the alternate service is invalidated and origin server revalidates the alternate service with a new freshness time. Alternate service is a transparent service. The application layer of the client is unaware of the server switching mechanism. The client can discover the alternate service that is associated with the origin server by multiple ways are many mechanisms by which the client can discover the alternate service.

Alternate service is an efficient HTTP/2 feature, which can be utilized for the enhancement of QoE. There might be the case that the client in Pakistan is using a global DNS server address, which resides in US instead of using a local DNS server. In this case, DNS may redirect the client to its closest CDN in the US. So there will be a high latency for that CDN because of its large geographical distance from the client. The server can diagnose this problem and can offer an alternate service to the client. In HTTP/2 header, alternate service message is sent to the client along with the address of alternate servers. The client can use the alternate service for switching to the server, which offers maximum throughput with minimum latency.

Alternate service can boost the performance of the video streaming when used effectively. However, it is an additional feature of HTTP/2 and it is not mandatory for the server or client to use this feature. This feature might have some security issues, so it is up to the client to make sure that the alternate server is under the association of original server. Some additional security check mechanism should be created to ensure the authenticity of the alternate server.

CHAPTER 4: EXPERIMENTAL SETUP AND RESULTS

4.1 Experimental Setup

For implementation and experimental verification of our proposed algorithm, we have used Advanced Network Simulator tool ns-3.[53] This tool helps us to test our algorithm by using state of the art protocols and network infrastructure. To test our algorithm in a realistic network environment, we have used building model of ns-3 to create a building of 10 rooms in a single floor. All the rooms are of same dimensions and adjacent to each other arranged in two rows in the order of 5x2. Each room has width and height of 10 meters and a length of 20 meters. The rooms are in residential building model having external walls of concrete with windows.

For evaluation of our algorithm, we have used an open-source movie Big Buck Bunny [54] as a test video. The video is encoded in MPEG DASH format, containing 10 representation levels. The video is available in 6 different segment sizes but for the experimental setup, we have used video with segment size of 2 seconds. Total duration of the video is 9 minutes and 56 seconds, and contains 298 segments in total. The Table 4.1 below shows the remaining attributes of the dataset.

TABLE 4.1. DATASET ATTRIBUTES

Representation Level	Bitrate		Resolution
0	89283 bps	89 Kbps	320x240
1	221600 bps	221 Kbps	480x360
2	396126 bps	396 Kbps	480x360
3	595491 bps	595 Kbps	854x480
4	1032682 bps	1.0 Mbps	1280x720
5	1546902 bps	1.5 Mbps	1280x720
6	2133691 bps	2.1 Mbps	1920x1080
7	3078587 bps	3.0 Mbps	1920x1080
8	3526922 bps	3.5 Mbps	1920x1080
9	4219897 bps	4.2 Mbps	1920x1080

The server was positioned in the first room at coordinates (5, 5, 5) and Access point was placed at the coordinates (25, 20, 5). Wi-Fi protocol IEEE 802.11n was used to connect clients with the server via access point. The clients positioned were randomly inside the building. The remaining parameters for the configuration setup were set as mentioned in [55].

Bandwidth of the link between server and access point is set to 10 Mbps. Number of clients can be specified before the start of simulation. Each client connects with the server one after another, after a constant time interval of 5 seconds and starts downloading the video segments, according to its rate adaptation algorithm. The simulation stops once all the clients have completed the playback of the video.

4.2 Experimental Results

For evaluation of our proposed algorithm, we have compared our algorithm with three state of the art algorithms namely, FESTIVE [34], PANDA [33], and TOBASCO/AAASH [41]. Every experiment was carried out 10 times for each test case and the average results were taken.

For the evaluation of our proposed algorithm (SHANZ), the value of minimum buffer level (β_{min}) was set to 10, maximum buffer level (β_{max}) was set to 40, for computing weighted average throughput the value of weight (w) was set as 0.1, and finally, the value of Throughput threshold (Δ) was set to 0.85.

For every experiment, we have evaluated the performance of the algorithm by comparing its QoE metrics with other algorithms. QoE metrics consists of 6 parameters namely, Total number of playback interruptions or video stalling events, average playback video quality, total number of video quality transitions, average playback buffer size, and finally, stability of the video.

4.2.1 Single Client Scenario

In the first scenario, algorithms were tested on a single client with no background traffic and the link speed was set to 10 Mbps. In this case, a client was located at a random location inside the building. Once the client connects with the server, it starts downloading the video segments according to its adaptation algorithm. The simulation stops once the client has completely played the video.

All the algorithms show similar performance in single client scenario except AAASH algorithm, which has the lowest average video quality. Maximum value of Average Video Quality of 9 was shown by our proposed algorithm (SHANZ-I) with average buffer level of 31 seconds. None of the algorithms showed stalling event during the simulation. QoE metrics for each of the five adaptation algorithms is shown in Table 4.2.

TABLE 4.2. QOE METRICS FOR SINGLE CLIENT SCENARIO

QoE Metrics	**FESTIVE**	**PANDA**	**AAASH**	**SHANZ-I**	**SHANZ-II**
Playback Interruptions	0	0	0	0	0
Avg. Video Quality	7	8	4	8	8
Video Quality Transitions	9	5	6	9	9
Avg. Buffer Size	28	28	32	31	33

Figure 4.1 shows the graph of average playback video quality during the simulation in single client scenario for five adaptation algorithms. As the graph shows, SHANZ-I and SHANZ-II algorithms were most responsive to adaptation especially at lower bitrates.

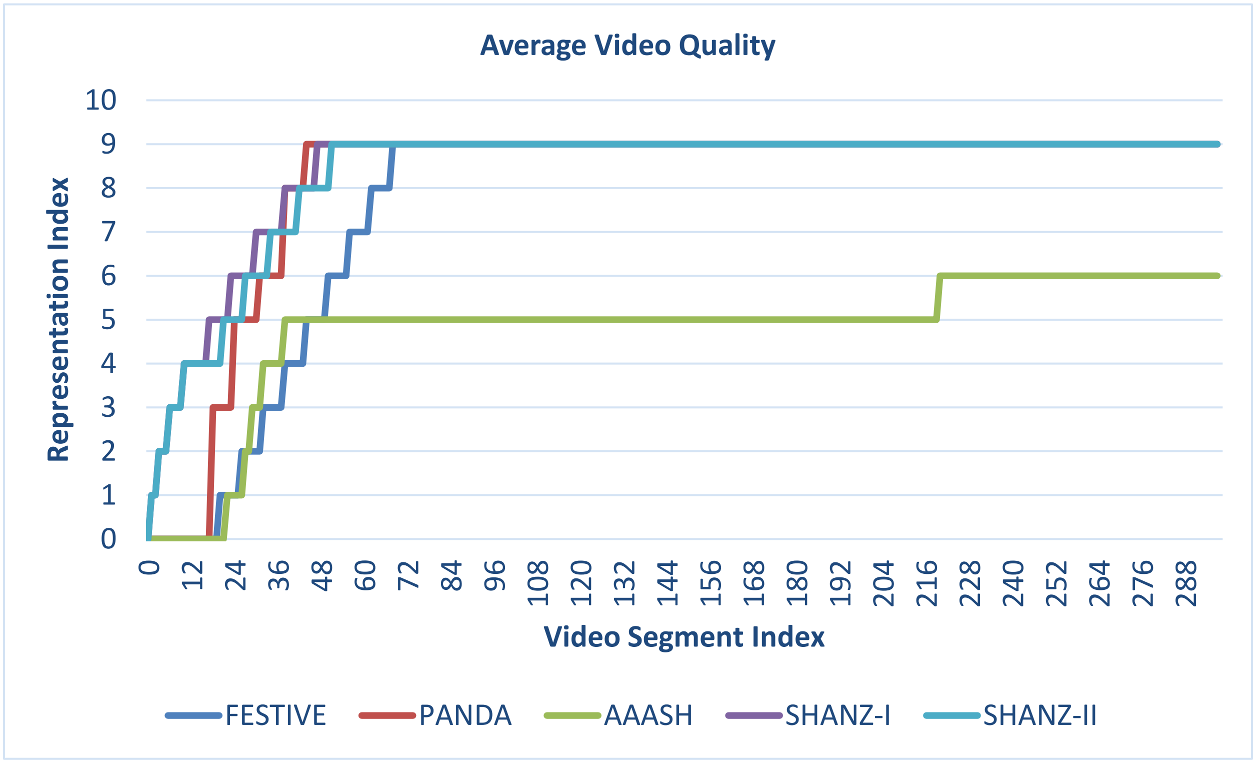


Fig. 4.1: Average Video Quality for single client scenario

Another perimeter to evaluate the performance of adaptation algorithms is stability. We have defined the stability function (δ) earlier, which depends upon number of video quality switches in a unit time. Figure 4.2 shows stability index of the algorithms. All the algorithms show similar stability graph for single client scenario. SHANZ-I and SHANZ-II algorithm showed the minimum stability value of 0.55 and reached back to its maximum value.

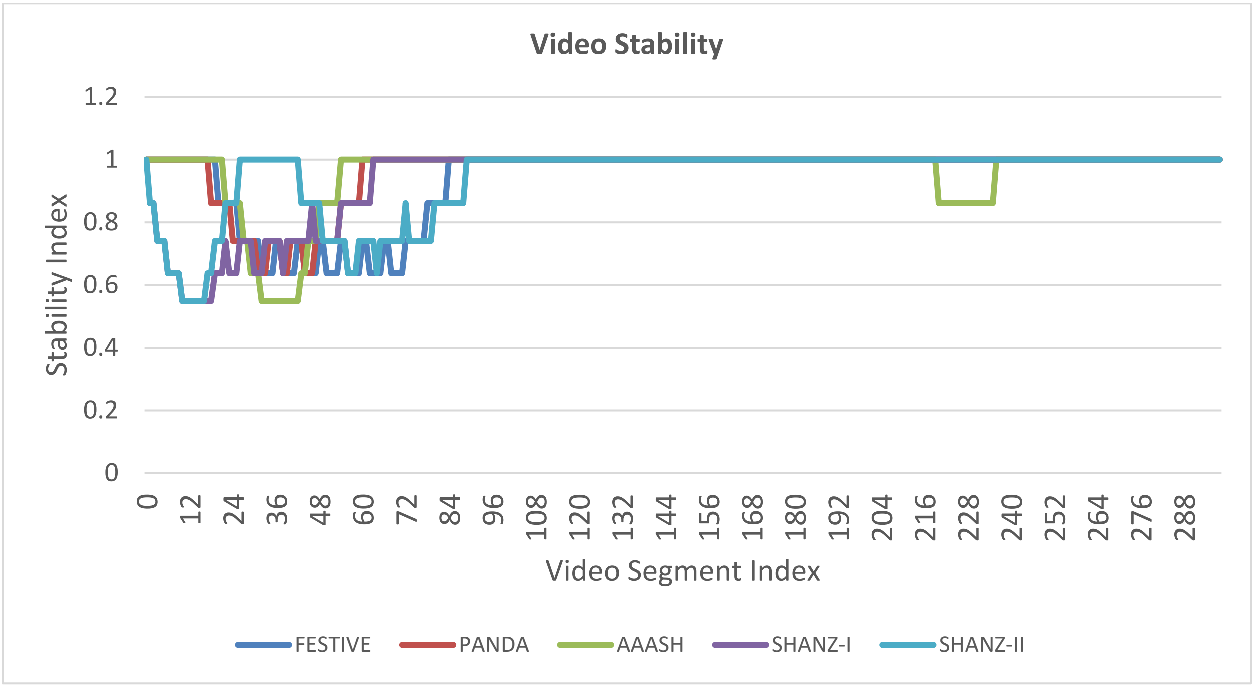


Fig. 4.2: Stability Index for single client scenario

4.2.2 Multiple Clients Scenario

In the second phase of experiment, we tested our algorithms with 5 clients. In this case, the link speed between server and access point was set to of 10 Mbps. Each client connected to the server to start the playback, after a constant time interval of 5 seconds.

Table 4.3 shows QoE metrics of FESTIVE algorithm, with 5 clients. The results show that clients 1 and 5 have the lowest average video quality along with multiple playback interruptions. While other three clients have comparatively higher average quality level, but experienced huge number of video quality switches.

TABLE 4.3. QOE METRICS OF FESTIVE ALGORITHM WITH 5 CLIENTS

FESTIVE QoE Metrics	**Client 1**	**Client 2**	**Client 3**	**Client 4**	**Client 5**
Playback Interruptions	13	0	0	0	8
Avg. Video Quality	1	5	6	6	1
Video Quality Transitions	36	55	64	62	40
Avg. Buffer Size	23	27	26	27	23

Results of QoE metrics of PANDA adaptation algorithm are shown in Table 4.4. There are comparatively lesser number of video quality transitions than FESTIVE but the shifts are large and sudden, not step wise. Secondly, there are fewer stalling events as compared to previous algorithm. Average buffer size of the clients of PANDA is almost similar to clients of FESTIVE algorithm.

TABLE 4.4. QOE METRICS OF PANDA ALGORITHM WITH 5 CLIENTS

PANDA QoE Metrics	**Client 1**	**Client 2**	**Client 3**	**Client 4**	**Client 5**
Playback Interruptions	0	4	0	1	0
Avg. Video Quality	5	1	6	2	7
Video Quality Transitions	11	24	19	19	17
Avg. Buffer Size	27	22	27	25	26

Table 4.5 shows the results of AAASH algorithm for 5 clients, the clients of AAASH algorithm has underperformed with least average video quality level size amongst all five algorithms, but has also shown least number of video quality transitions and without playback interruption.

TABLE 4.5. QOE METRICS OF AAASH ALGORITHM WITH 5 CLIENTS

AAASH QoE Metrics	**Client 1**	**Client 2**	**Client 3**	**Client 4**	**Client 5**
Playback Interruptions	0	0	0	0	0
Avg. Video Quality	4	3	3	3	3
Video Quality Transitions	5	4	4	4	4
Avg. Buffer Size	30	32	31	32	31

Table 4.6 describes the results of SHANZ-I algorithm. It has shown better fairness amongst its clients, with higher average playback quality as compared to other algorithms. All

the clients played the video without any playback interruption. Finally, it has also shown lesser number of video quality transitions as compared to FESTIVE and PANDA.

TABLE 4.6. QOE METRICS OF SHANZ-I ALGORITHM WITH 5 CLIENTS

SHANZ QoE Metrics	**Client 1**	**Client 2**	**Client 3**	**Client 4**	**Client 5**
Playback Interruptions	0	0	0	0	0
Avg. Video Quality	4	6	5	6	4
Video Quality Transitions	9	10	13	11	10
Avg. Buffer Size	27	32	31	33	26

Table 4.7 describes the results of SHANZ-II algorithm. The algorithm achieved better fairness amongst its clients, with higher average playback quality as compared to other algorithms. It has also shown lesser number of video quality transitions as compared to FESTIVE and PANDA.

TABLE 4.7:QOE METRICS OF SHANZ-II ALGORITHM WITH 5 CLIENTS

SHANZ QoE Metrics	**Client 1**	**Client 2**	**Client 3**	**Client 4**	**Client 5**
Playback Interruptions	0	0	0	0	0
Avg. Video Quality	5	6	5	5	6
Video Quality Transitions	8	9	10	8	7
Avg. Buffer Size	33	30	29	32	30

The graphs below from Figure 4.3 to Figure 4.7 show, the Average playback video quality of five algorithms in 5 clients scenario**.** AAASH and PANDA has lower average video quality overall, with PANDA having steepest video quality fluctuations. FESTIVE has the largest number of video quality transitions. SHANZ-I and SHANZ-II algorithm showed higher average representation index with better stability as compared to other three algorithms.

Figure 4.3 shows the average playback video quality of FESTIVE algorithm, as the graph shows FESTIVE has highest video quality oscillations with least stability. It also showed unfairness amongst its clients.

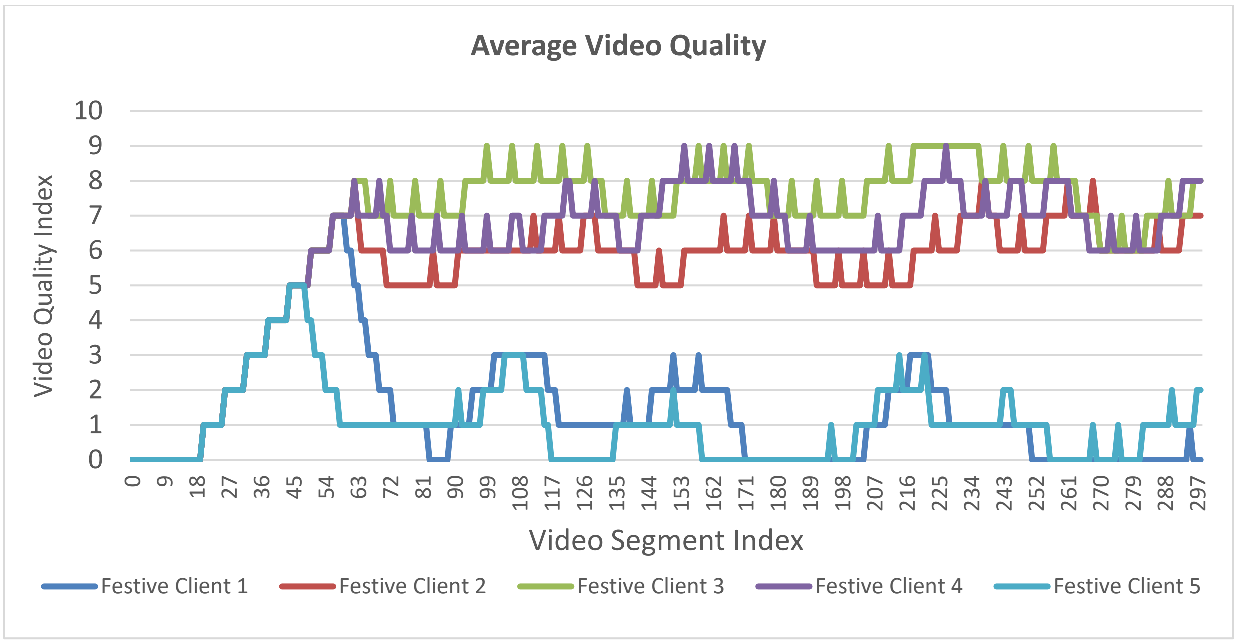


Fig. 4.3: Average video quality of FESTIVE for 5 clients scenario

Figure 4.4 shows the average playback video quality of PANDA algorithm. PANDA showed the sharp oscillations in the video quality, with minimum stability. It also showed unfairness amongst its clients.

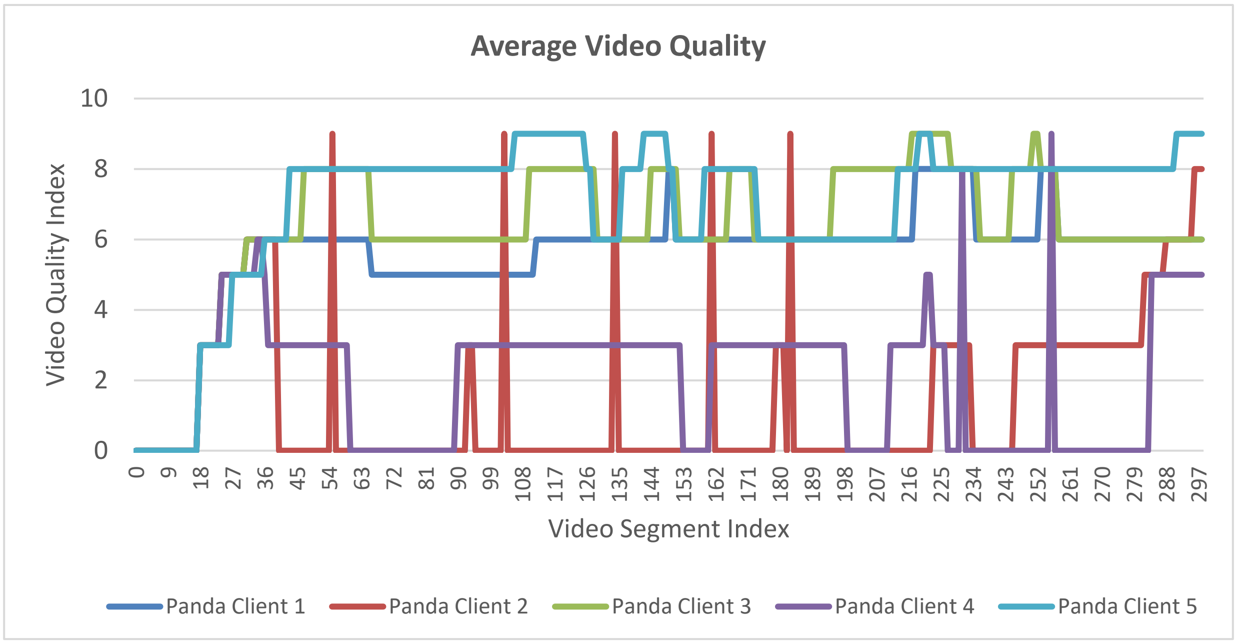


Fig. 4.4: Average video quality of PANDA for 5 clients scenario

Figure 4.5 shows the average playback video quality of TOBASCO algorithm, as the graph shows that it has achieved better stability and fairness among the clients but has the least average playback quality level. TOBASCO algorithm has underutilized the bandwidth link.

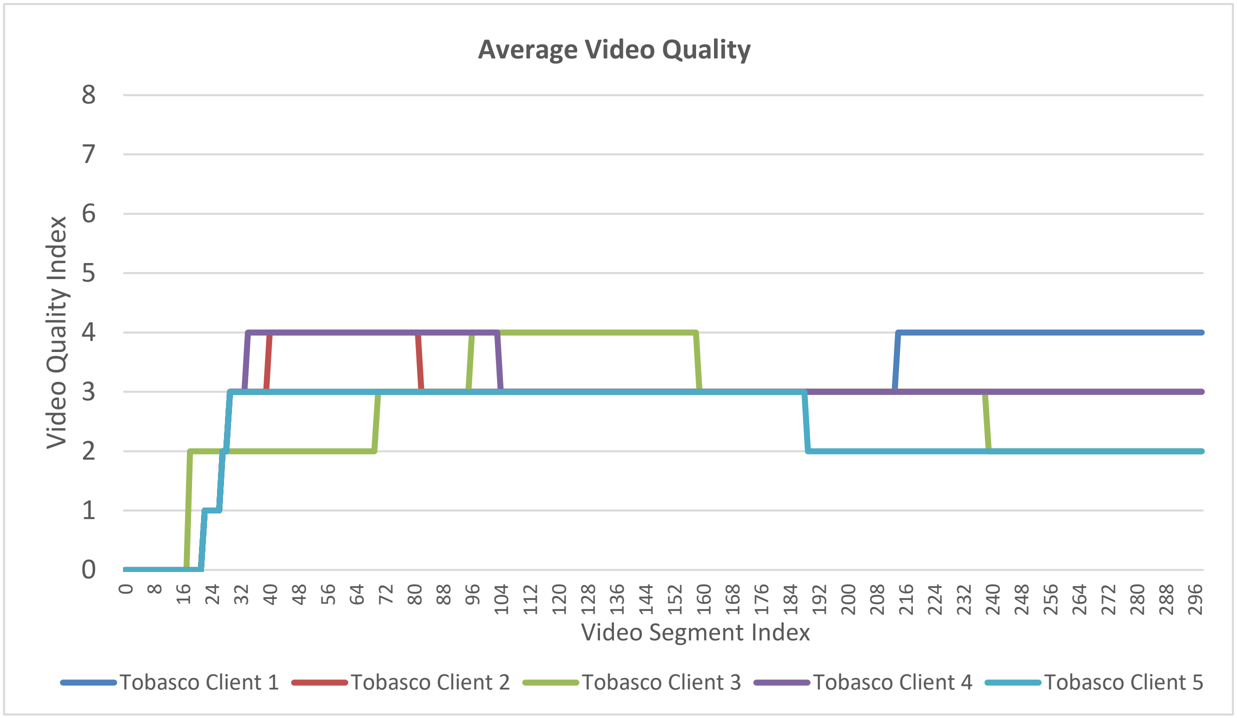


Fig. 4.5: Average video quality of TOBASCO for 5 clients scenario

Figure 4.6 shows the average playback video quality of SHANZ-I adaptation algorithm. It has achieved higher average playback quality level with better stability. It also achieved better fairness amongst its client. All the clients played the video without any playback interruptions.

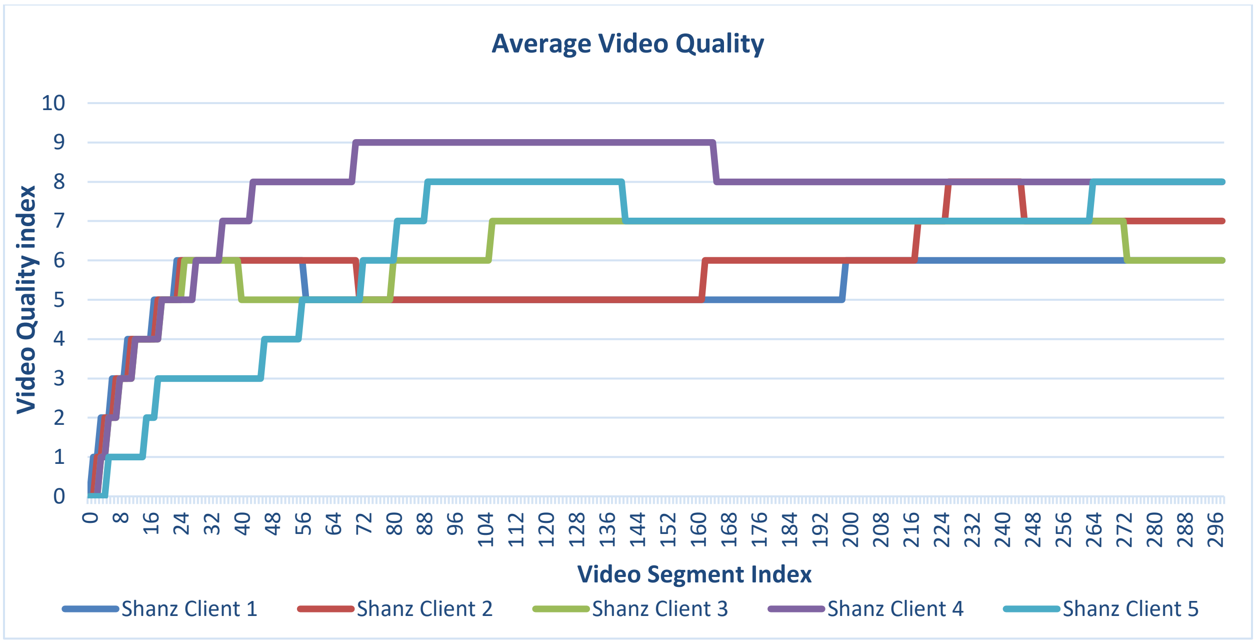


Fig. 4.6: Average video quality of SHANZ-I for 5 clients scenario

Figure 4.7 shows the average playback video quality of SHANZ-II adaptation algorithm. It has achieved higher average playback quality level with better stability. All the clients played the video without any playback interruptions. It also achieved better fairness amongst its client.

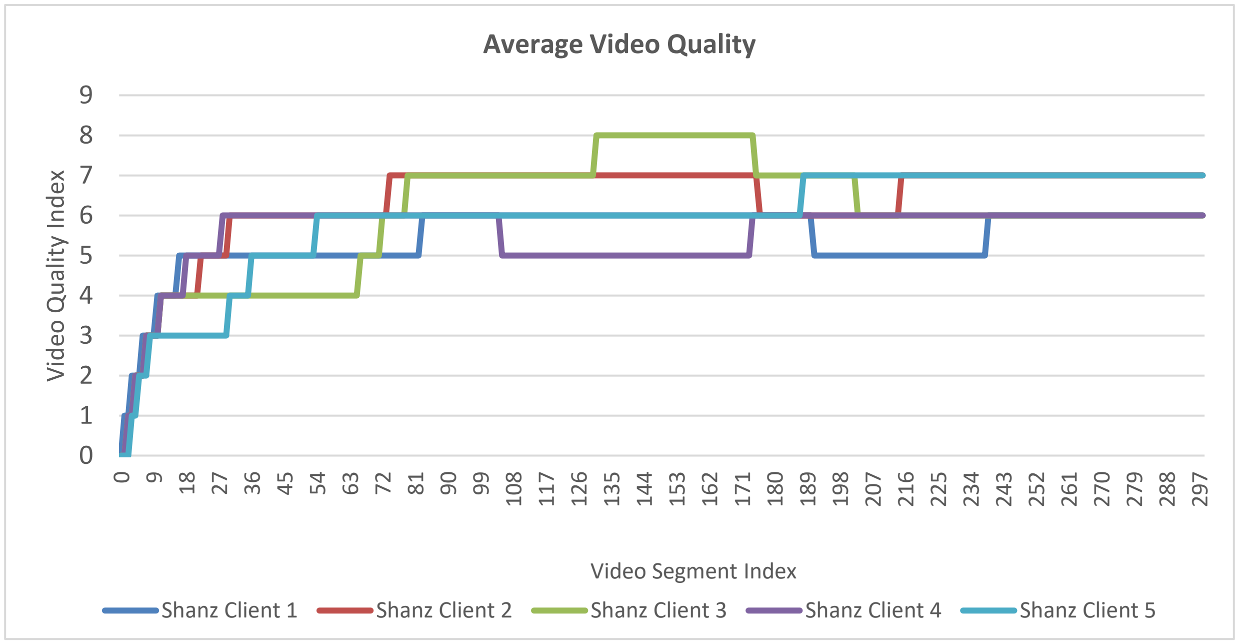


Fig. 4.7: Average video quality of SHANZ-II for 5 clients scenario

Figures from 4.8 to 4.12 below show the graph of video stability index of the algorithms. FESTIVE algorithm is the least stable algorithm, with highest fluctuations. PANDA algorithm also showed least stability with highest amplitude of video quality oscillations. PANDA showed sudden drop in its stability index due to sharp fluctuations in video quality. AAASH was the most stable algorithm due to underutilization of the link. Stability index of SHANZ-I algorithm, also kept on showing continuous minor fluctuations, but overall its stability is better as compared to other algorithms. SHANZ-II algorithm was the most stable algorithm when compared with other algorithms.

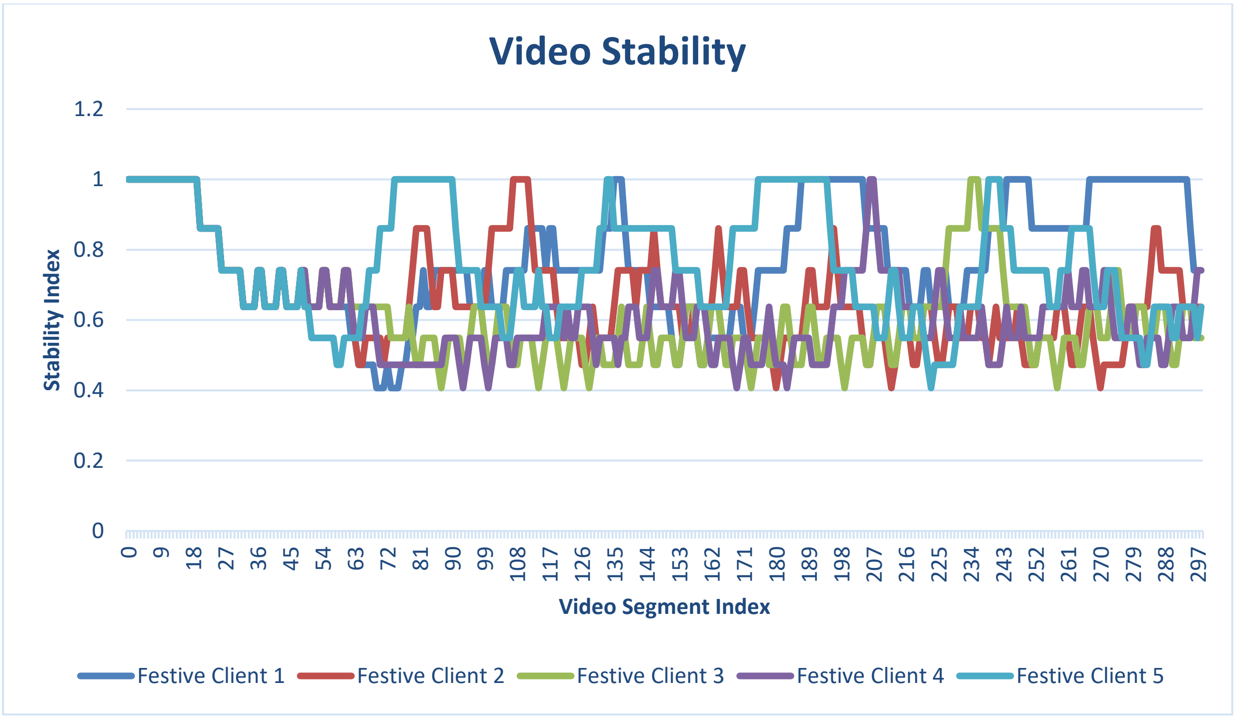


Fig. 4.8: Stability Index of FESTIVE for 5 clients scenario

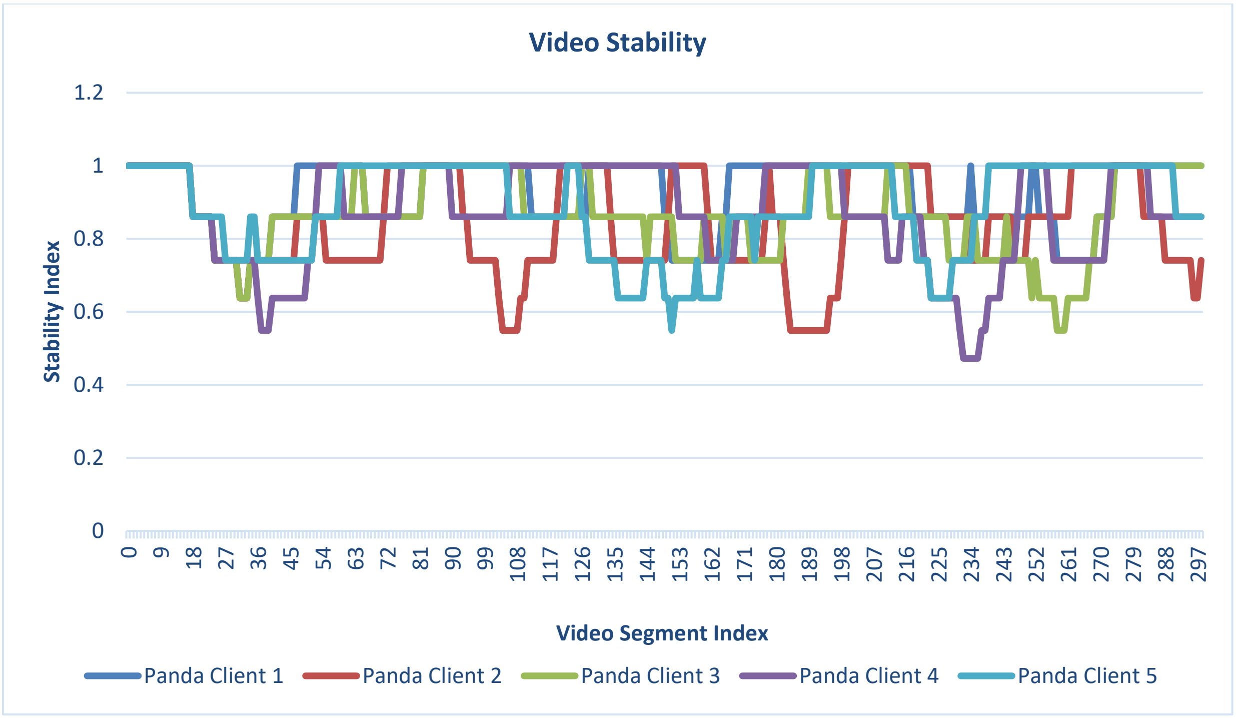


Fig. 4.9. Stability Index of PANDA for 5 clients scenario

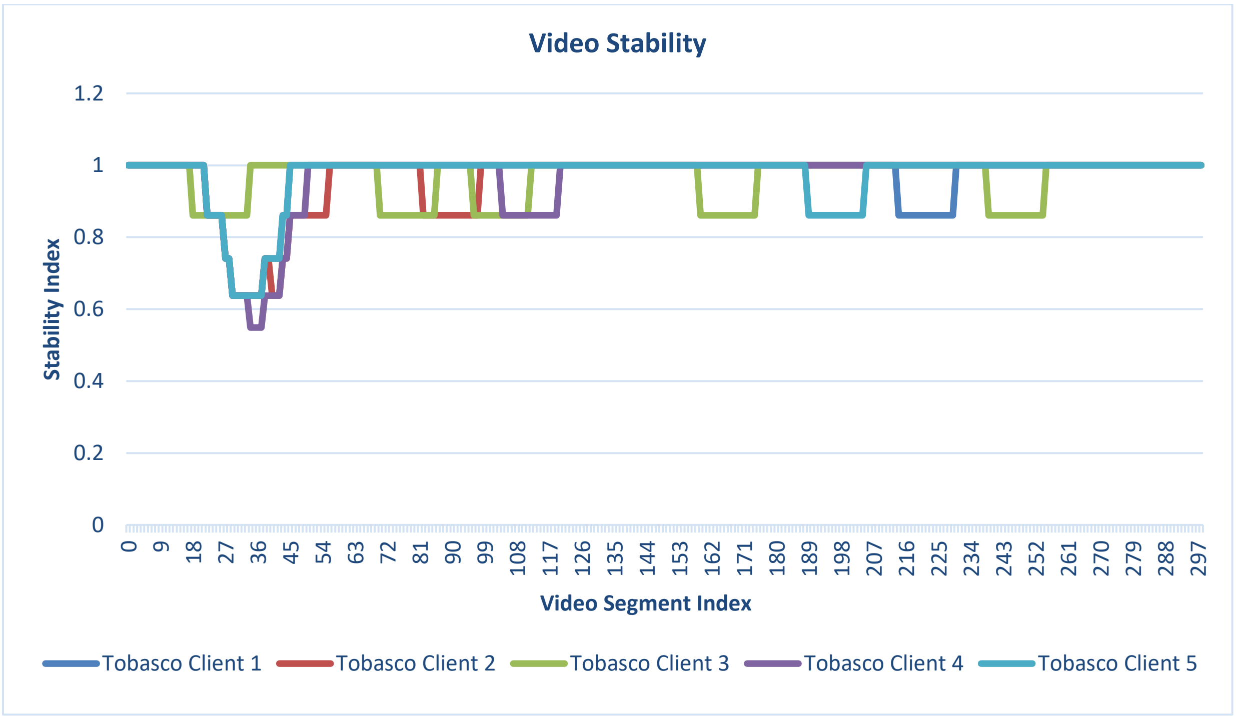


Fig. 4.10: Stability Index of TOBASCO for 5 clients scenario

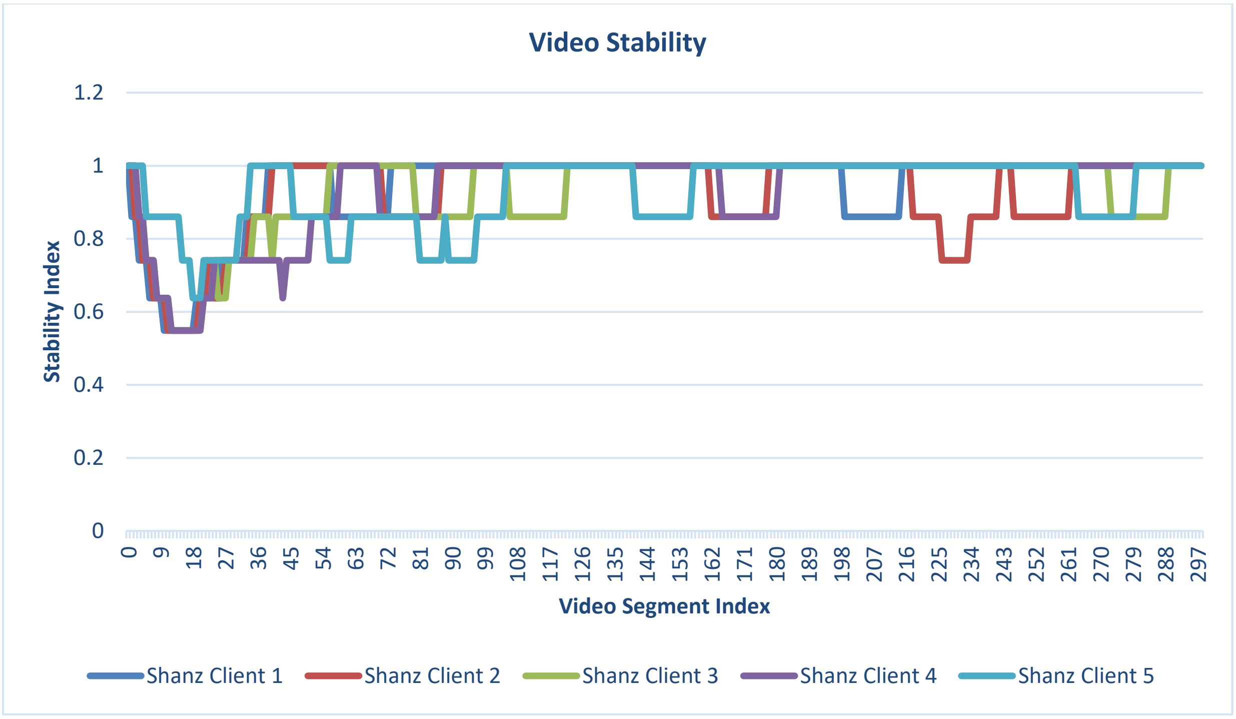


Fig. 4.11: Stability Index of SHANZ-I for 5 clients scenario

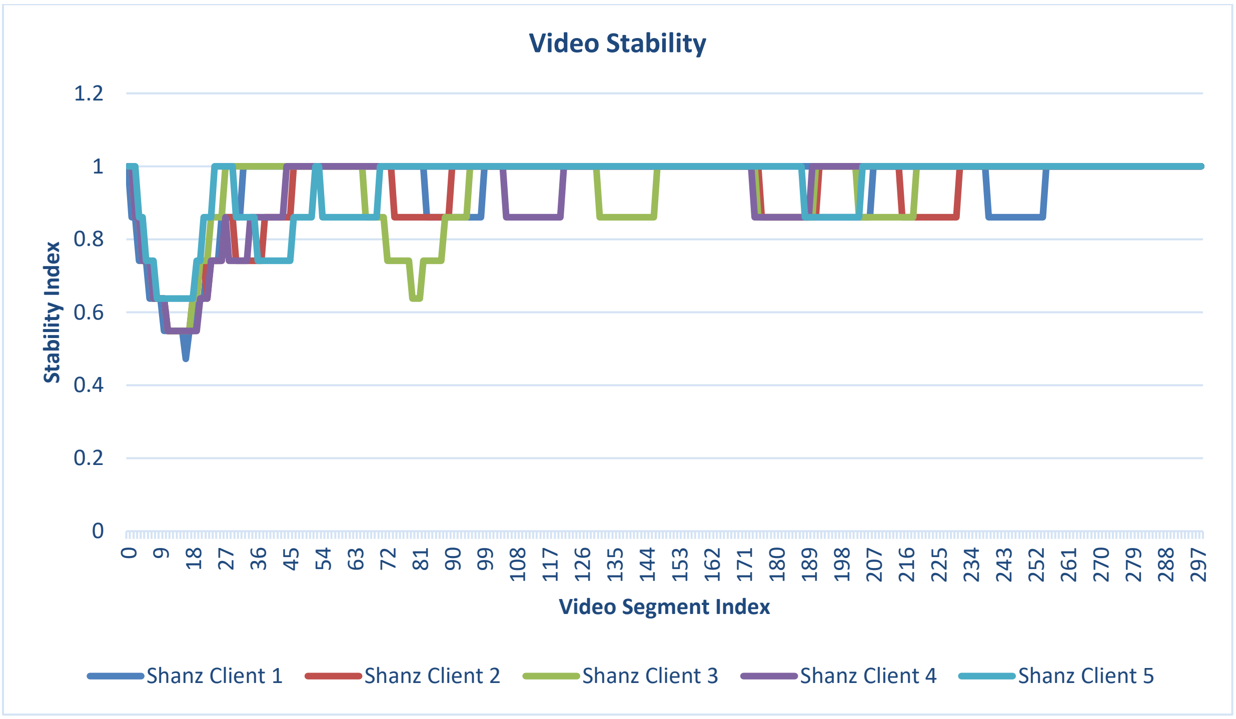


Fig. 4.12: Stability Index of SHANZ-II in 5 clients scenario

4.2.3 Congested Network Scenario

In the final stage of our experiment, we have tested the adaptation algorithms in network congestion under a bottleneck link. A constant background traffic was added to increase congestion in the network. In this scenario, number of clients increased to 7, and the results were compared.

Table 4.8 shows QoE metrics of FESTIVE adaptation algorithm for 7 clients. With increase of congestion in the network, video quality oscillations of FESTIVE further increased. Client 2 faces maximum number of playback stalling events. Unfairness among the clients also increase with increase of network congestion.

TABLE 4.8. QOE OF FESTIVE FOR CONGESTED NETWORK SCENARIO

FESTIVE QoE Metrics	**Client 1**	**Client 2**	**Client 3**	**Client 4**	**Client 5**	**Client 6**	**Client 7**
Playback Interruptions	1	12	3	1	2	3	1
Avg. Video Quality	5	1	1	5	2	6	1
Video Quality Transitions	42	38	37	51	35	61	40
Avg. Buffer Size	27	24	23	27	24	28	22

The results of PANDA algorithm for 7 clients scenario are shown in Table 4.9. Video quality shifts of PANDA increased, due to congestion in the network. However, its average video quality is still better than that of FESTIVE with lesser video quality shifts.

TABLE 4.9. QOE OF PANDA FOR CONGESTED NETWORK SCENARIO

PANDA QoE Metrics	**Client 1**	**Client 2**	**Client 3**	**Client 4**	**Client 5**	**Client 6**	**Client 7**
Playback Interruptions	3	1	2	4	3	1	2
Avg. Video Quality	2	6	3	4	7	2	2
Video Quality Transitions	19	27	17	21	21	17	23
Avg. Buffer Size	28	27	28	24	24	27	27

QoE metrics of AAASH is shown in Table 4.10. The performance of AAASH algorithm also decreased due to increased network congestion, as it was before. The stability of the algorithm has decreased due to more video quality oscillations. Some of the clients have also shown playback interruptions due to congestion in the network.

TABLE 4.10. QOE OF AAASH FOR CONGESTED NETWORK SCENARIO

AAASH QoE Metrics	**Client 1**	**Client 2**	**Client 3**	**Client 4**	**Client 5**	**Client 6**	**Client 7**
Playback Interruptions	3	2	1	2	0	2	1
Avg. Video Quality	3	2	2	4	3	1	0
Video Quality Transitions	3	5	4	5	5	7	6
Avg. Buffer Size	29	31	27	28	31	30	29

QoE of SHANZ-I for network congestion scenario is described in Table 4.11. Clients of SHANZ still have higher average video quality as compared to other adaptation algorithms. The algorithm showed better fairness and stability amongst the clients, as compared to other three algorithms, without any playback interruption.

TABLE 4.11. QOE OF SHANZ-I ALGORITHM FOR CONGESTED NETWORK SCENARIO

SHANZ QoE Metrics	**Client 1**	**Client 2**	**Client 3**	**Client 4**	**Client 5**	**Client 6**	**Client 7**
Playback Interruptions	0	0	0	0	0	0	0
Avg. Video Quality	6	5	4	4	3	4	3
Video Quality Transitions	11	8	8	11	7	9	10
Avg. Buffer Size	30	27	29	27	29	28	26

QoE of SHANZ-II algorithm for network congestion scenario is described in Table 4.12. Clients of SHANZ-II showed highest average video quality level as compared to other adaptation algorithms. Our proposed algorithm showed better fairness and stability amongst the clients, as compared to other three algorithms, without any playback interruption.

TABLE 4.12. QOE OF SHANZ-II ALGORITHM FOR CONGESTED NETWORK SCENARIO

SHANZ QoE Metrics	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7
Playback Interruptions	0	0	0	0	0	0	0
Avg. Video Quality	4	3	4	3	5	5	3
Video Quality Transitions	8	7	9	8	11	10	9
Avg. Buffer Size	30	27	29	27	29	28	26

Figures 4.13 to 4.17 show average video quality of the algorithms for congested network scenario. It shows that video quality oscillations are increased for all the algorithms. FESTIVE was most affected algorithm in this case, having highest number of video quality oscillations with least stability. PANDA algorithm still has the sharp fluctuations in the video quality. TOBASCO showed better stability but poor video quality. SHANZ-I and SHANZ-II algorithms still managed to maintain higher average video quality with lesser number of video quality oscillations, when compared with other algorithms.

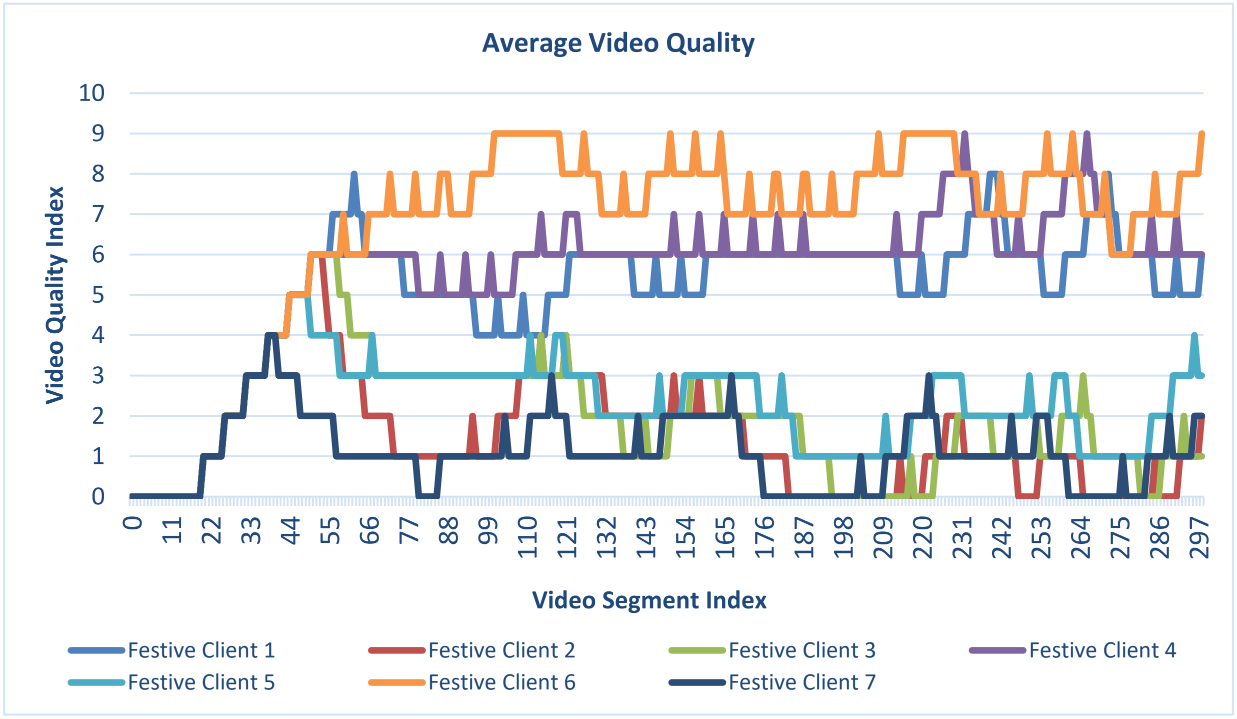


Fig. 4.13: Average Video Quality of FESTIVE for congested network scenario

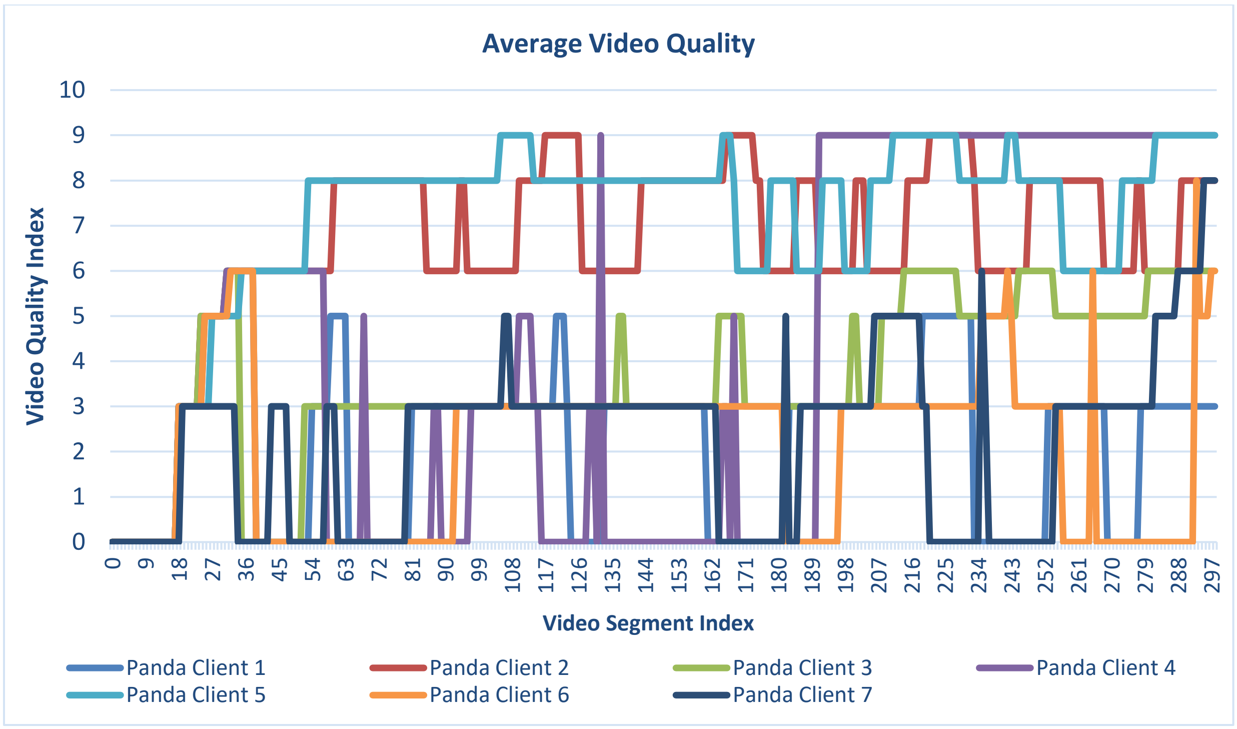


Fig. 4.14: Average Video Quality of PANDA for congested network scenario

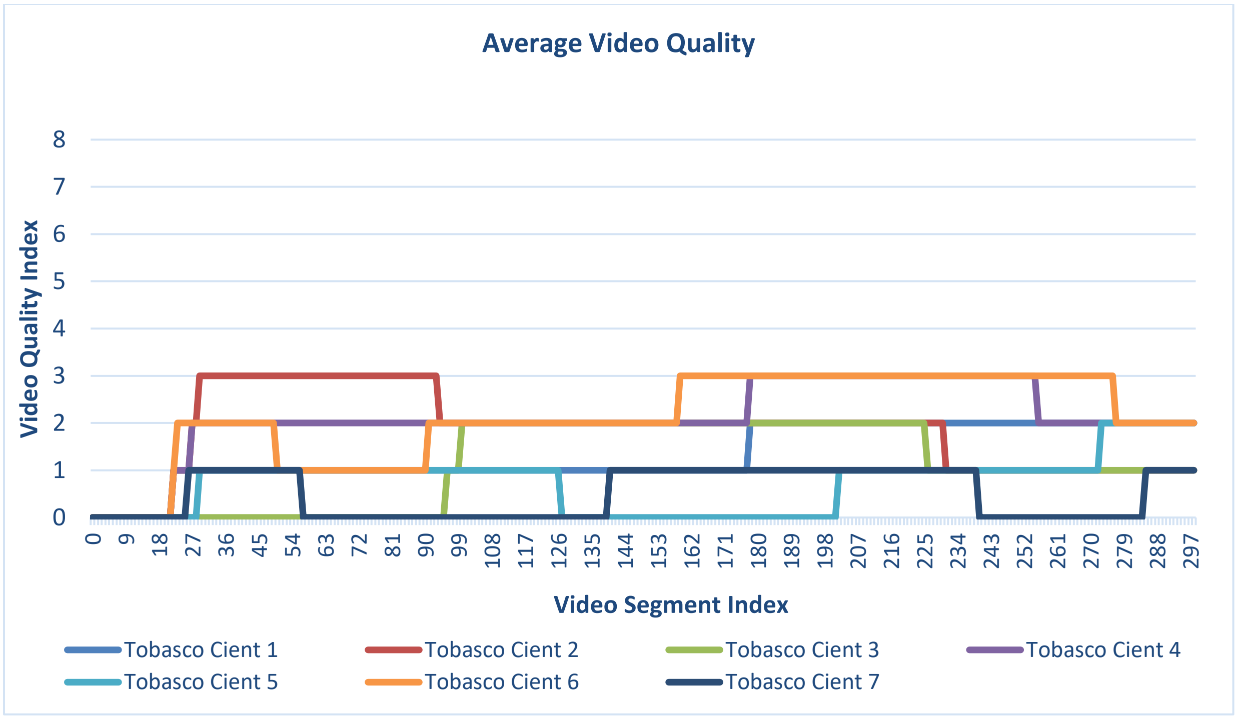


Fig. 4.15: Average Video Quality of TOBASCO for congested network scenario

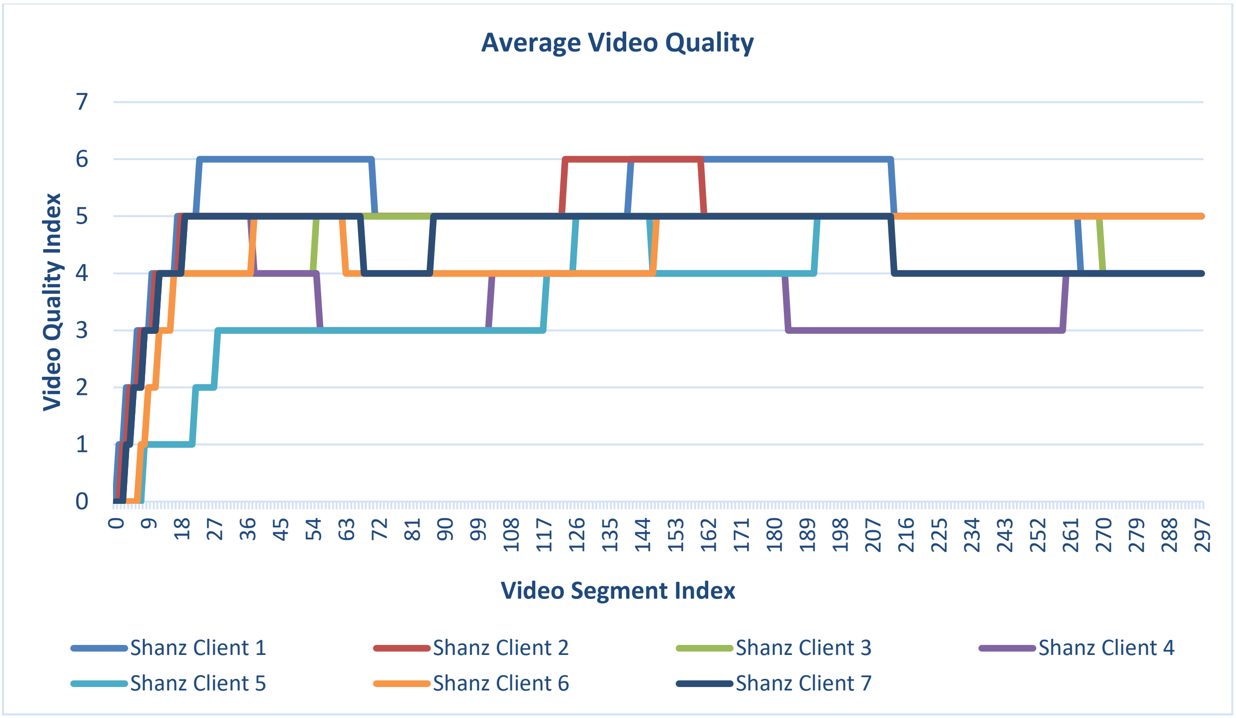


Fig. 4.16: Average Video Quality of SHANZ-I for congested network scenario

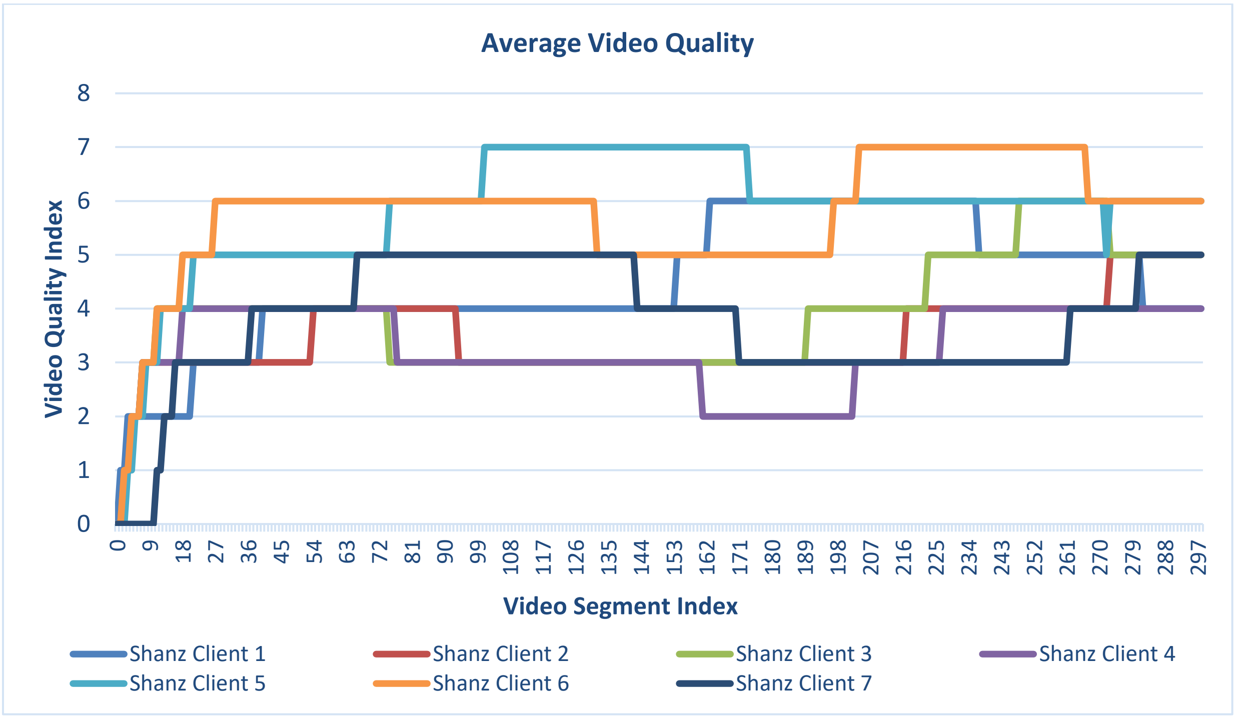


Fig. 4.17: Average Video Quality of SHANZ-II for congested network scenario

Figures 4.18 to 4.22 shows the stability index of the adaptation algorithms in the congested network scenario. It shows that FESTIVE has the highest rate of oscillations with least stability. The instability of PANDA algorithm also decreased, with highest amplitude of video quality oscillations. TOBASCO algorithm was most stable algorithm because of underutilization of the link. Stability of SHANZ-I and SHANZ-II algorithm was also decreased, but it remained higher than its minimum threshold value of 0.54. Our proposed algorithms showed better stability in comparison to other algorithms.

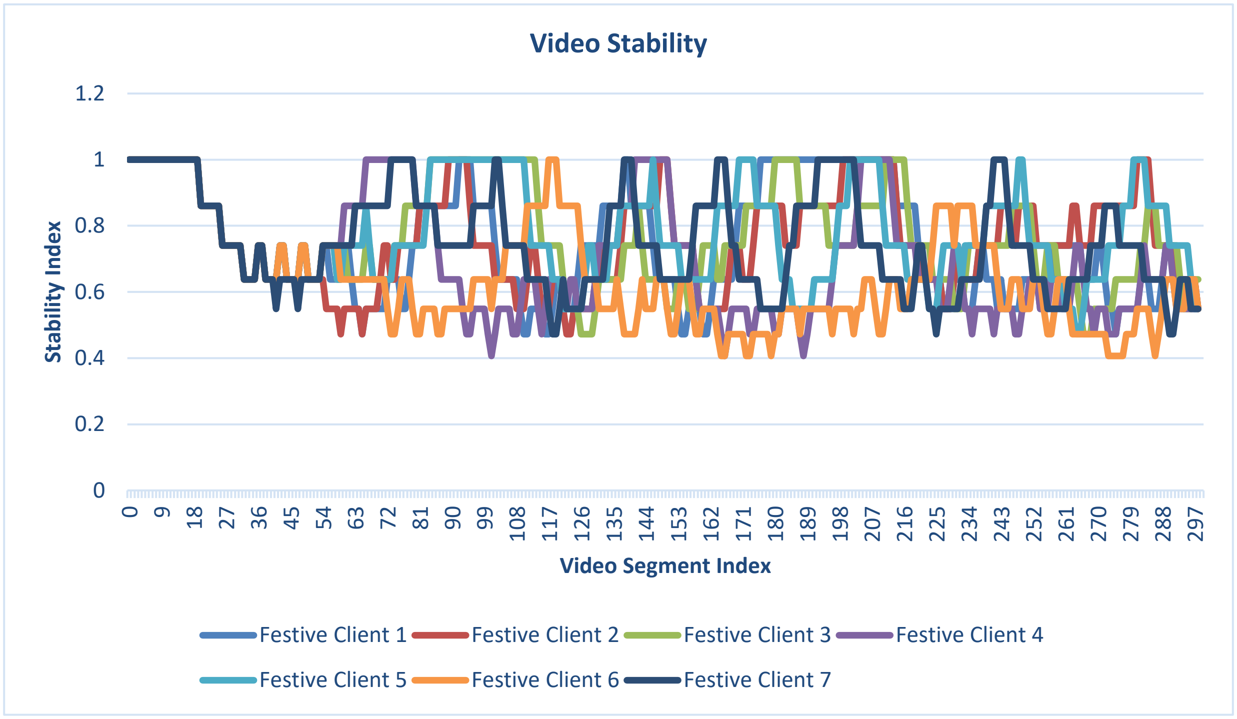


Fig. 4.18: Stability Index of FESTIVE for congested network scenario

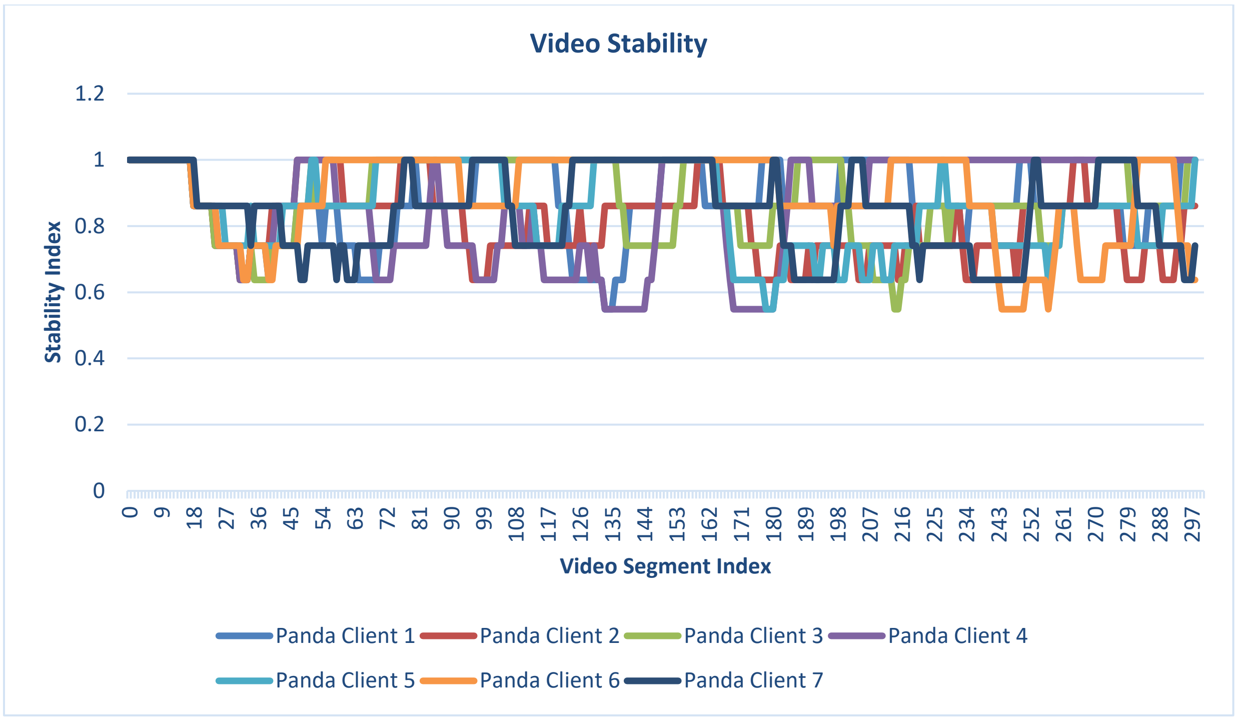


Fig. 4.19: Stability Index of PANDA for congested network scenario

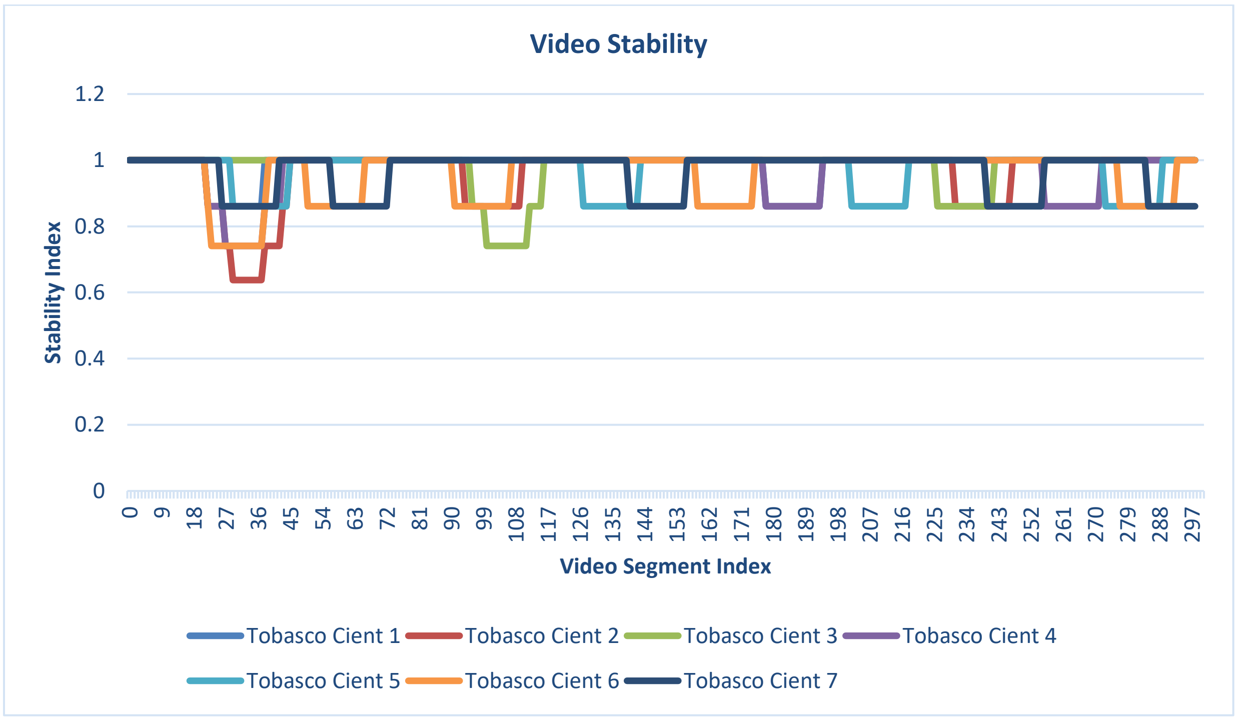


Fig. 4.20: Stability Index of TOBASCO for congested network scenario

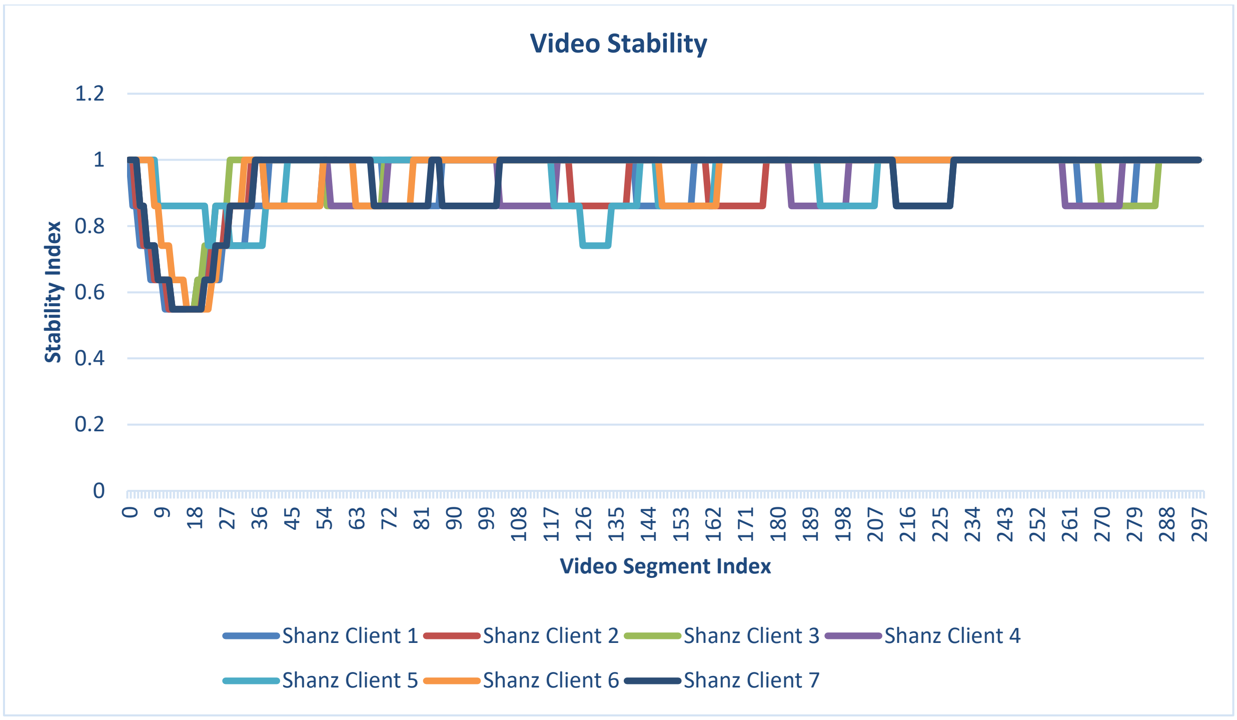


Fig. 4.21: Stability Index of SHANZ-I for congested network scenario

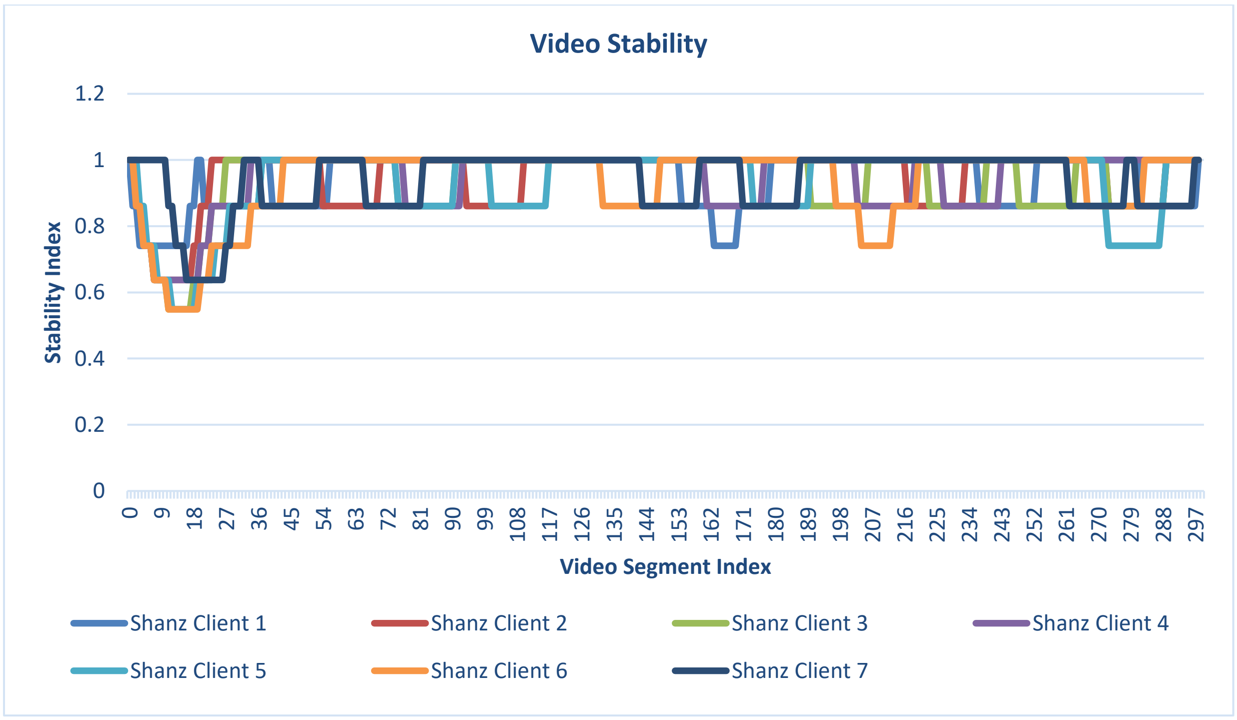


Fig. 4.22: Stability Index of SHANZ-II for congested network scenario

4.2.4 Analysis of Results

Experimental results verify that our proposed algorithm maintains a balance in its stability and efficiency even in drastic network conditions. SHANZ-I and SHANZ-II algorithms has achieved higher average quality level, with better stability in all the test cases with lesser number of video quality oscillations as compared to other algorithms. The algorithms has also maintained a level of fairness amongst its clients in all the test cases. The algorithm ensured seamless video streaming amongst all the clients in all the test cases, without any playback interruption.

SHANZ-I algorithm maintained the balance between stability and efficiency of the algorithm. It achieved higher average playback quality level and also maintained its stability. SHANZ-II algorithm maintained the stability of the algorithm but also reduced the start-up delay and latency of the video by using server push mechanism. Both of the proposed algorithms maintained the level of fairness amongst their clients.

After the comparison of our proposed algorithms with state of the art techniques in multiple test cases, our algorithms achieved better quality of experience metrics and outperformed other algorithms.

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 Conclusion

MPEG DASH has become the standard protocol for HTTP based adaptive video streaming. With increase in the popularity of HAS, there has been a sharp increase in the video traffic over the Internet. When multiple clients compete for bandwidth in the same network, it creates a bottleneck link. This has led to more issues like unfairness and congestion in the network. This work presents a dynamic rate adaptation algorithm, which can evolve with changing network conditions to maintain the balance between stability and efficiency of the algorithm by adaptive step-up function and using a feedback control mechanism to maximize the stability function. The algorithm also managed to maintain fairness amongst the clients by introducing randomized delay to eliminate bandwidth overestimation issues. For the evaluation of our proposed algorithm, we used public available dataset named as “Big Buck Bunny”. This video was an animated video of mpeg dash format and consists of total duration of 9 minutes and 56 seconds. The video segment size of the video was 2 seconds.

The algorithm was evaluated in three different test cases and the results were compared with three other popular algorithms, namely, FESTIVE, PANDA and AAASH. In the first case, the algorithm was tested on a single client with the bandwidth of 10Mbps. In the second case, the algorithm was tested on 5 clients and in the third test case the algorithm was tested on 7 clients in network congestion scenario. For the analysis of quality metrics, different parameters were compared like, average video quality, total number of video stalling events, average buffer size, total number of buffering events and the stability of the video. After evaluation of the algorithm in multiple test cases, the results demonstrated that the algorithm outperformed other algorithms. The algorithm showed better stability of the video in all the test cases and achieved the highest average video quality as compared to other algorithms. Finally, our algorithm also managed to maintain fairness among the clients, while other algorithm showed lesser fairness in presence of multiple clients.

5.2 Future Work

Increased video traffic over the internet has given rise to new research challenges. Video content over the internet consumes up to three quarters of the total bandwidth of the network. A large number of users in a same network watch videos on different heterogeneous platforms. It increases unfairness among the clients and results in congestion in the network. Therefore, an intelligent mechanism is required, which predict the future network state and react accordingly before the performance of the network degrades. Secondly, there should be a mechanism such that different network entities should interact with each other to give feedback about the current state of the network. A new framework of MPEG DASH named as SAND has given an excellent framework for intelligent and network aware video streaming. This framework can be used as a roadmap for building a new infrastructure for video streaming. Secondly, Media centric IOT is another research area, which requires the contribution for resolving challenges in efficient multimedia streaming in nodes, with using minimum computational power and bandwidth.

REFERENCES

[1] "Cisco Visual Networking Index: Forecast and Trends, 2017–2022," White Paper November 26, 2018.

[2] Sandvine, "Sandvine Global Internet Phenomena Report," 2015, Available: https://www.sandvine.com/downloads/general/global-internet-phenomena/2015/global-internet-phenomena-report-latin-america-and-north-america.pdf.

[3] H. Schulzrinne, "Real Time Streaming Protocol version 2.0," 2016.

[4] J. Goldberg, M. Westerlund, and T. Zeng, "A Network Address Translator (NAT) Traversal Mechanism for Media Controlled by the Real-Time Streaming Protocol (RTSP)," 2070-1721, 2016.

[5] Microsoft. (2015). *Microsoft Smooth Streaming*. Available: http://www.iis.net/downloads/microsoft/smooth-streaming

[6] Apple. (2015). *Apple HTTP Live Streaming*. Available: https://developer.apple.com/streaming/

[7] Adobe. (2015). *Adobe HTTP Dynamic Streaming*. Available: http://www.adobe.com/products/hds-dynamic-streaming.html

[8] I. I.-D. 3, "Information technology – Coding of audio-visual objects – Part 12: ISO base media file format – Amendment 3: DASH support and RTP reception hint track processing," January 2011.

[9] I. J. C. G. Sodagar, Apr, "White paper on MPEG-DASH Standard, MPEG-DASH: The Standard for Multimedia Streaming Over Internet," 2012.

[10] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, T. J. I. T. o. c. Wiegand, and s. f. v. technology, "Comparison of the coding efficiency of video coding standards—including high efficiency video coding (HEVC)," vol. 22, no. 12, pp. 1669-1684, 2012.

[11] S. Kim, K. J. K. T. o. I. Chung, and I. Systems, "A Bandwidth Estimation Scheme to Improve the QoE of HTTP Adaptive Streaming in the Multiple Client Environment," vol. 12, no. 1, 2018.

[12] V. K. Adhikari *et al.*, "Measurement study of Netflix, Hulu, and a tale of three CDNs," vol. 23, no. 6, pp. 1984-1997, 2015.

[13] C. J. D. M. R. Smith, "By the numbers: 20 amazing Netflix statistics and facts," 2014.

[14] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, R. J. I. C. S. Zimmermann, and Tutorials, "A Survey on Bitrate Adaptation Schemes for Streaming Media over HTTP," 2018.

[15] M. Seufert *et al.*, "A survey on quality of experience of HTTP adaptive streaming," vol. 17, no. 1, pp. 469-492, 2015.

[16] V. K. Adhikari *et al.*, "Measurement Study of Netflix, Hulu, and a Tale of Three CDNs," *IEEE/ACM Transactions on Networking,* vol. 23, no. 6, pp. 1984-1997, 2015.

[17] M. Zink, J. Schmitt, and R. J. I. T. o. M. Steinmetz, "Layer-encoded video in scalable adaptive streaming," vol. 7, no. 1, pp. 75-84, 2005.

[18] C. Müller, S. Lederer, and C. Timmerer, "An evaluation of dynamic adaptive streaming over HTTP in vehicular environments," in *Proceedings of the 4th Workshop on Mobile Video*, 2012, pp. 37-42: ACM.

[19] J. Yao, S. S. Kanhere, I. Hossain, and M. Hassan, "Empirical evaluation of HTTP adaptive streaming under vehicular mobility," in *International Conference on Research in Networking*, 2011, pp. 92-105: Springer.

[20] S. Alcock and R. J. A. S. C. C. R. Nelson, "Application flow control in YouTube video streams," vol. 41, no. 2, pp. 24-30, 2011.

[21] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis, "Trickle: Rate limiting youtube video streaming," in *Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX}{ATC} 12)*, 2012, pp. 191-196.

[22] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen, "Server-based traffic shaping for stabilizing oscillating adaptive streaming players," in *Proceeding of the 23rd ACM workshop on network and operating systems support for digital audio and video*, 2013, pp. 19-24: ACM.

[23] K. Satoda, H. Yoshida, H. Ito, and K. Ozawa, "Adaptive video pacing method based on the prediction of stochastic TCP throughput," in *2012 IEEE Global Communications Conference (GLOBECOM)*, 2012, pp. 1944-1950: IEEE.

[24] C. R. A. Ford, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," I. E. T. Force, Ed., ed, Jan. 2013.

[25] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon, "Cross-layer scheduler for video streaming over MPTCP," in *Proceedings of the 7th International Conference on Multimedia Systems*, 2016, p. 7: ACM.

[26] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of multipath tcp performance over wireless networks," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013, pp. 455-468: ACM.

[27] J. Wu, C. Yuen, B. Cheng, M. Wang, and J. J. I. T. o. M. C. Chen, "Streaming high-quality mobile video with multipath TCP in heterogeneous wireless networks," vol. 15, no. 9, pp. 2345-2361, 2016.

[28] G. Cofano, L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, and S. Mascolo, "Design and experimental evaluation of network-assisted strategies for HTTP adaptive streaming," in *Proceedings of the 7th International Conference on Multimedia Systems*, 2016, p. 3: ACM.

[29] R. Houdaille and S. Gouache, "Shaping HTTP adaptive streams for a better user experience," in *Proceedings of the 3rd Multimedia Systems Conference*, 2012, pp. 1-9: ACM.

[30] R. K. Mok, X. Luo, E. W. Chan, and R. K. Chang, "QDASH: a QoE-aware DASH system," in *Proceedings of the 3rd Multimedia Systems Conference*, 2012, pp. 11-22: ACM.

[31] J. Chen, R. Mahindra, M. A. Khojastepour, S. Rangarajan, and M. Chiang, "A scheduling framework for adaptive video delivery over cellular networks," in *Proceedings of the 19th annual international conference on Mobile computing & networking*, 2013, pp. 389-400: ACM.

[32] T. C. Thang, Q.-D. Ho, J. W. Kang, and A. T. J. I. T. o. C. E. Pham, "Adaptive streaming of audiovisual content using MPEG DASH," vol. 58, no. 1, pp. 78-85, 2012.

[33] Z. Li *et al.*, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," vol. 32, no. 4, pp. 719-733, 2014.

[34] J. Jiang, V. Sekar, and H. J. I. A. T. o. N. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," vol. 22, no. 1, pp. 326-340, 2014.

[35] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive HTTP streaming," in *Proceedings of the second annual ACM conference on Multimedia systems*, 2011, pp. 169-174: ACM.

[36] L. De Cicco and S. J. I. A. T. o. N. Mascolo, "An adaptive video streaming control system: Modeling, validation, and performance evaluation," vol. 22, no. 2, pp. 526-539, 2014.

[37] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. J. S. P. I. C. Gabbouj, "Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network," vol. 27, no. 4, pp. 288-311, 2012.

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

[39] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. J. A. S. C. C. R. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," vol. 44, no. 4, pp. 187-198, 2015.

[40] G. Tian and Y. Liu, "Towards agile and smooth video adaptation in dynamic HTTP streaming," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, 2012, pp. 109-120: ACM.

[41] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz, "Adaptation algorithm for adaptive streaming over HTTP," in *2012 19th international packet video workshop (PV)*, 2012, pp. 173-178: IEEE.

[42] C. Sieber, T. Hoßfeld, T. Zinner, P. Tran-Gia, and C. Timmerer, "Implementation and User-centric Comparison of a Novel Adaptation Logic for DASH with SVC," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, 2013, pp. 1318-1323: IEEE.

[43] A. Beben, P. Wiśniewski, J. M. Batalla, and P. Krawiec, "ABMA+: lightweight and efficient algorithm for HTTP adaptive streaming," in *Proceedings of the 7th International Conference on Multimedia Systems*, 2016, p. 2: ACM.

[44] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, 2016, pp. 1-9: IEEE.

[45] W. Huang, Y. Zhou, X. Xie, D. Wu, M. Chen, and E. J. I. T. o. B. Ngai, "Buffer state is enough: Simplifying the design of QoE-aware HTTP adaptive video streaming," vol. 64, no. 2, pp. 590-601, 2018.

[46] P. Juluri, V. Tamarapalli, and D. Medhi, "SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP," in *Communication Workshop (ICCW), 2015 IEEE International Conference on*, 2015, pp. 1765-1770: IEEE.

[47] P. Wisniewski, A. Beben, J. M. Batalla, and P. Krawiec, "On delimiting video rebuffering for stream-switching adaptive applications," in *2015 IEEE International Conference on Communications (ICC)*, 2015, pp. 6867-6873: IEEE.

[48] 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*, 2013, pp. 1-8: IEEE.

[49] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *ACM SIGCOMM Computer Communication Review*, 2015, vol. 45, no. 4, pp. 325-338: ACM.

[50] Y. Sun *et al.*, "CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 272-285: ACM.

[51] C. Wang, A. Rizk, and M. Zink, "SQUAD: A spectrum-based quality adaptation for dynamic adaptive streaming over HTTP," in *Proceedings of the 7th International Conference on Multimedia Systems*, 2016, p. 1: ACM.

[52] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using openflow-assisted adaptive video streaming," in *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*, 2013, pp. 15-20: ACM.

[53] (March, 2018). *NS-3 (ns-3.28 ed.)*. Available: https://www.nsnam.org

[54] Big Buck Bunny [Online]. Available: http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/BigBuckBunny/2sec/

[55] P. Gawłowicz, S. Zehl, A. Zubow, and A. J. a. p. a. Wolisz, "Nxwlan: Neighborhood extensible wlan," 2016.