T.R.
EGE UNIVERSITY
Graduate School of Applied and Natural Science

SERVER AND NETWORK ASSISTED DASH (SAND) ARCHITECTURE UTILIZING SDN TECHNOLOGY

PhD Thesis

Reza SHOKRI KALAN

International Computer

İzmir
2020

T.R.
EGE UNIVERSITY
Graduate School of Applied and Natural Science

SERVER AND NETWORK ASSISTED DASH (SAND) ARCHITECTURE UTILIZING SDN TECHNOLOGY

Reza SHOKRI KALAN

Supervisor : Assoc. Prof. Dr. Müge SAYIT

International Computer
Information Technology Third Cycle Programme

İzmir
2020

Reza SHOKRI KALAN tarafından Doktora tezi olarak sunulan "Server and Network Assisted DASH (SAND) Architecture Utilizing SDN Technology" başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 10.01.2020 tarihinde yapılan tez savunma sınavında aday oybirliği/~~oyçokluğu~~ ile başarılı bulunmuştur.

Jüri Üyeleri :		**İmza**
Jüri Başkanı	: Doç. Dr. Müge SAYIT	.....
Raportör Üye	: Dr.Öğr.Üyesi İlker KOCABAŞ	.....
Üye	: Doç. Dr. Ali. C. BEĞEN	.....
Üye	: Prof. Dr. Mehmet Emin DALKILIÇ	.....
Üye	: Dr.Öğr.Üyesi Deniz ÖZSOYELLER	.....

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Doktora Tezi olarak sunduğum "Server and Network Assisted DASH (SAND) Architecture Utilizing SDN Technology" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

10.01.2020

Reza SHOKRI KALAN

ÖZET

YAZILIM TANIMLI AĞ (YTA) TEKNOLOJİSİNDEN YARARLANAN SAND MİMARİSİ

SHOKRI KALAN, Reza

Doktora Tezi, Uluslararası Bilgisayar Anabilim Dalı
Tez Danışmanı: Doç. Dr. Müge SAYIT
Ocak 2020, 143 sayfa

Uyarlanabilir video akışlandırma teknolojisi, ağ durumunu göz önünde bulundurarak elde edilen kalitede akış sağlamayı amaçlayan bir teknolojidir. DASH istemcileri, ağ koşulları hakkında daha az bilgiye sahip oldukları için düşük kalite ve adaletsiz deneyim kalitesiyle karşılaşabilirler. Ağın ve kullanılabilir kaynakların durumuyla ayrıntılı bilgi sağlamak, daha iyi adaptasyon ve sonuçta daha iyi bir deneyim kalitesinin elde edilmesiyle sonuçlanabilir. Ancak, birleştirilmiş altyapıya dayanan geleneksel ağ mimarisi, ağ sağlayıcılarının ve ağ uygulaması geliştiricilerinin kendi yaklaşımlarını uygulamalarına kolayca izin vermemektedir. Yazılım Tanımlı Ağlar (SDN), veri ve kontrol düzlemlerini birbirinden ayrıştırarak ağa daha fazla esneklik ve programlanabilirlik sağlar. Bu tez, Sunucu ve Ağ Yardımlı DASH (SAND) ve SDN teknolojisini kullanarak daha iyi adaptasyon yapılmasını sağlamak amacıyla istemcilere ekstra bilgi sağlamak için yeni mimari ve algoritmalar sunmaktadır. Tezin katkıları şu şekilde listelenebilir, i) Deneyim Kalitesi (QoE) parametrelerini iyileştirmek için SDN teknolojisi yardımıyla video kodekleri ve ağ yollarının özelliklerini dikkate alarak katmanlı video akışlarını yöneten bir mimari sunulmuştur. ii) SDN tabanlı bir mimari ve SAND uyumlu DASH istemcisi geliştirilmiştir. Önerilen mimari, DASH istemcilerinin performansını arttırmaya ve ağ kaynaklarının istemciler arasında adil paylaştırılmasına yardımcı olmaktadır. iii) Ağ İşlevi Sanallaştırma (NFV) ve SDN yardımıyla SAND mimarisinde sanal önbellek kavramı sunulmuştur ve önbelleği en uygun lokasyona sanal olarak ilklendirmek veya taşımak için algoritmalar geliştirilmiştir. Ayrıca, önbelleği taşıma sırasında karşılaşılan "TCP hand-off" gecikme sorunu için çözüm sağlanmıştır. Mevcut çalışmalar ile karşılaştırıldığında bu çalışmada ağ kaynaklarının adil ve verimli kullanılması sağlanarak QoE metriklerinde daha iyi sonuçlar elde edildiği görülmüştür.

Anahtar sözcükler: deneyim kalitesi, yazılım tanımlı ağ, ağ işlevi sanallaştırma, uyarlanabilir video akışlandırma, DASH, DANE, SAND

ABSTRACT

SERVER AND NETWORK ASSISTED DASH (SAND) ARCHITECTURE UTILIZING SDN TECHNOLOGY

SHOKRI KALAN, Reza

PhD in International Computer
Supervisor: Assoc. Prof. Dr. Müge SAYIT
January 2020, 143 pages

Adaptive streaming is a technology aims to provide quality achieved streaming considering network conditions. DASH clients may experience low quality and unfairness because the client-driven approach has less intelligence about network conditions.Providing a global view of the network and available resources could result in better adaptation and consequently can improve the client's QoE. However, legacy network architecture based on the tightly coupled infrastructure does not allow network providers and network application developers to implement their own approaches. Software Defined Networks (SDN) brakes vertical integration network architecture into a logical central programmable model and gives more flexibility in network design and simplicity in network management and configuration.

This thesis is a collection of new frameworks and algorithms to provide extra information to the clients for better adaptation by leveraging Server and Network Assisted DASH (SAND) utilizing SDN technology. The contributions can be listed as, i) we present an architecture for managing flows of layered video to improve the QoE parameters with considering the characteristics of codecs and network paths by utilizing SDN. ii) We develop an SDN-based architecture and SAND-aware DASH client. Proposed architecture helps to improve the performance of DASH clients as well as providing fair sharing of the network resources among the clients. iii) We introduce virtual cache concept with NFV-SDN assisted SAND architecture and provide algorithms to virtually deploy or migrate cache. Also, we addressed the TCP hand-off issues during cache migration. Compared with stat-of-the-art approaches, simulation results show that proposed solutions achieve better results in terms of experienced quality, fairness, and utilizing network resources.

Keywords: quality of experience, software defined networks, network function virtualization, adaptive video streaming, DASH, DANE, SAND

ÖNSÖZ

Akıllı cihazların artan popülaritesi ile her gün daha fazla kişi internete bağlanmaktadır. Beklentiler, küresel nüfusun % 60'ının 2022'ye kadar İnternet kullanıcısı olacağını ve video trafiğinin tüm IP trafiğinin % 82'sini oluşturacağını göstermektedir. Ne yazık ki, internet hızı yavaş olduğu zaman, video istemcileri kesikli video izleme yada daha düşük kalitede video izleme deneyimi yaşar ancak, bu durum genelde olumlu karşılanmaz. Bu sorunu çözmek için, video servis sağlayıcıları aynı videonun çoklu kopyasını oluşturur. Her kopyanın farklı bit hızı vardır. Böylece istemciler otomatik olarak farklı internet bağlantı hızında uygun kaliteyi seçip sorunsuz bir video akışı sağlamaya çalışırlar. Bu tekniğe uyarlamalı akış adı verilir.

Literatürde önerilen istemci tabanlı ve sunucu tabanlı uyarlamalı akışlandırma çözümleri daha verimli hale getirilebilir. Ağ durumu hakkında uygulama katmanına güncel bilgi sağlamak, istemcilerin kaliteyi daha iyi adapte edebilmesine ve daha yüksek bit hızına sahip video görüntülemesine yardımcı olabilir. Bu amaçla, bu tezde NFV-SDN (Ağ İşlevi sanallaştırma Sanallaştırma- Yazılım Tanımlı Ağ) teknolojilerinin avantajları SAND mimarisi ile birleştirilmiştir. Böyle bir kombinasyon, ağ elemanları arasında mesaj iletimine imkan tanır ve uygun video sunucusu ve uygun video bit hizi seçmek için istemciye daha fazla bilgi sağlar. Simülasyon sonuçları, tez kapsamında önerilen mimari ve geliştirilen algoritmaların, temel ve benzer çalışmalarla karşılaştırıldığında daha iyi video izleme performansı sağladığını göstermektedir.

İZMİR

10.01.2020 Reza SHOKRI KALAN

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF TABLES

LIST OF TABLES (continued)

LIST OF ALGORITHM

LIST OF ABBREVIATIONS

Abbreviation	Explanation
3GPP	Third Generation Partnership Project
API	Application Programming Interface
BBA	Buffer Based Adaption
CDN	Content Delivery Network
DANE	DASH Aware Network Elements
DASH	Dynamic Adaptive Streaming Over HTTP
DDoS	Distributed Denial Of Service
DPI	Deep Packet Inspection
DRM	Digital Right Management
ETSI	European Telecommunication Standard Institute
FI	Future Internet
HAS	HTTP Adaptive Streaming
HLS	HTTP Live Streaming
ICN	Information Centric Networking
IETF	Internet Engineering Task Force
IP	Internet Protocol
MPD	Media Presentation Description
MSS	Microsoft Smooth Streaming
NBI	Northbound Interface
NFV	Network Function Virtualization
NOS	Network Operating System

LIST OF ABBREVIATIONS (continued)

Abbreviation	Explanation
OTT	Over-The-Top
PED	Parameters Enhancing Delivery
PER	Parameters Enhancing Reception
QoE	Quality Of Experience
QoS	Quality Of Service
RNE	Regular Network Elements
RTMP	Real Time Messaging Protocol
SAND	Server And Network-Assisted DASH
SBI	Southbound Interface
SDN	Software Defined Networks
SVC	Scalable Video Coding
TBA	Throughput Base Adaption
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VNF	Virtual Network Function
XML	Extensible Markup Language

1. INTRODUCTION

Content delivery over the Internet in the form of streaming is a technology used to allows players to display video or listen to audio almost immediately before the entire file is completely downloaded. A relatively fast Internet connection is required in order to provide high enough quality streaming services without re-buffering to the recipients. Adaptive streaming is a technology aims to provide quality achieved streaming with considering network condition. At the high-level abstraction, in the adaptive streaming different copy of the same video replicated with different qualities for distribution to the players, which can adaptively turn to a suitable quality. Such a behavior changes stream with the target of the optimizing display experience. Regarding the big effect of the adaptive technology in streaming, heterogeneous natures of network technologies and implementation are the other concerns. In the case of network performance, a traditional implementation based on the tightly coupled infrastructure does not make sense anymore. *Software Defined Networks* (SDN) as a new architecture developed to address this problem.

Streaming technologies need fast network connections, slow or interrupted network connections can cause problems. The most common form of the streaming problem is related to available network bandwidth. In lack of enough bandwidth media players experience re-buffering. Commercial adaptive streaming technologies include *HTTP Live Streaming* (HLS), *Microsoft Smooth Streaming* (MSS), and *Dynamic Adaptive Streaming over HTTP* (DASH) tries to fill players buffer's with appropriate content-in the from of bitrate- they need next. Bitrate that indicates the amount of data encoded per second is affected by network and player conditions which are changed in a timely fashion. Although all streaming technologies follow the same principle, play the stream with high enough quality without unacceptable re-buffering time, *MPEG-DASH* is the only international standardized solution.

Same as all streaming technologies, the basic idea behind *MPEG-DASH* is generating multiple copies of the same content with different bitrates or representations. Each representation divided into small same-size partitions (e.g., two, four or six seconds) which known as segment or chunk. All the segments are stored on a media web server and can easily be downloaded via a GET request. Besides, manifest file contains media structure and describe all related information about representations as well as *Uniform Resource Locations* (URL). Displaying starts with requesting and downloading the manifest file. As a consequence, the player requests the next segment based on manifest information, network available bandwidth, and local resources such as CPU utilization and buffer fullness. It is worth emphasiz-

ing that for each segment, adaptation to specific representation is done on the player side. Also, small segment size results in fast quality switching. The advantages provided by adaptive streaming technologies give more flexibility to players in order to achieve better *Quality of Experience* (QoE). However, adaptive streaming suffes performance issues while laying over on the traditional network platform.

Despite the impact of improving video streaming technologies, the performance of streaming applications also depends on underlying network conditions. The popularity of the *Transmission Control Protocol* (TCP), which is working in the transport layer has big impacts on overall network performance. Transmission rate heavily depends on TCP functions which influenced by the network and sender/receiver contentions. The potential of the large scale network such as Internet declines due to network overload. Transmitting high rate data to the receiver results in packet drops in (i) intermediate network devices due to the network congestion or (ii) receiver side because of the buffer fullness. In the case of network overload or buffer overflow, a congestion mechanism is used to reduce the transmission rate. Therefore, the majority of the TCP functions are adapting to the network and receiver condition. Furthermore, the architecture and design of the TCP protocol mainly influenced by end-to-end arguments or middleboxes.

Network configuration is the other concern that should be addressed. The heterogeneous nature of the technologies, infrastructures, and applications have made the management of such a system a complex task. While network condition is more ad hoc, it is hard to configure the network based on predicted policy. In addition, any changes in policy, or fault needs network devices reconfiguration. Current state-of-the-art network configuration is more time costly. Vertical integration in the current IP network enforcing automatic configuration and preventing innovation acceleration (e.g., moving to IPv6) in network infrastructure (Kreutz et al., 2014). As an example, due to the stationary of the current IP network, it takes more than five years to be fully designed, evaluated and deployed new routing algorithms. (Kreutz et al., 2014). Finally, a clean-state approach to completely redesign the current network is more costly and impossible in practice.

Network Function Virtualization (NFV) and SDN come to address the limitation of the current network infrastructure by separation of network functions. The NFV/SDN is an abstraction of the network virtualization which solves more networking challenges without touching the underlying infrastructures. SDN brakes vertical integration network architecture into a logical central programmable model by separating control logic (control plane) from the underlying network infrastruc-

ture (forwarding plane). The key consequence of this separation is more flexibility in network design and simplicity in network configuration and management while it is easy and possible to create, introduce and trace new abstraction. It is important to understand that logically centralized programmable architecture does not certainly need a physically centralized system (Koponen et al., 2010). However, even with the leveraging *SDN* technology, traditional networks based on TCP protocols have less intelligence about network requirements.

Doubtless, $DASH$ client-pull technology is more flexible, firewall-friendly and CDN-scalable than server-push technologies. But, the client-driven nature of $DASH$ also has some drawbacks. While service providers may not necessarily have control over the client's behavior, they may not be able to guarantee the premium quality of service with DASH (Cetinkaya et al., 2015). In order to enhance the delivery of DASH content, *Server and Network-Assisted DASH* (SAND) creates communications between different network elements (DASH-IF, 2017) (Jmal et al., 2017). *SAND* is an asynchronous network-to-client and network-to-network communication. With the aim of improving the efficiency of streaming, *SAND* exchange quality-related assisting information about real-time operational characteristics of networks. Such a solution tries to avoid delay or interfere with the delivery of the streaming media content. *MPEG-SAND* introduces a basic element to facilitate the envisaged network assistance functionality (DASH-IF, 2017). So, DASH clients can better adapt to network conditions and avoid interruptions during playback.

1.1 General Contribution of this Study

The main contribution of this study aims to improve the quality of streaming in the *SDN* networks by taking advantage of *SAND* technology into account. As a consequence, leveraging *SDN* technology with assisted *SAND* architecture allows us to observe the current network traffic and develop a flexible routing algorithm specific to the requirements of network applications. This thesis includes four contributions.

- The first contribution is to propose an optimization algorithm for a layer-based adaptive video streaming. We present fair network bandwidth allocation as an effective tool to improve QoE parameters with considering codec, path, and quality selection by utilizing *SDN*. The goal is to minimize interrupt during displays, switch-over between quality layers and maximize average received bitrate.
- The second contribution relies on leveraging *SAND* technology in adaptive

streaming. We proposed an SDN-based architecture and a SAND-aware DASH client that improves the viewer experience. The simulation results show that this architecture can help improve the performance of DASH clients as well as provide fair sharing of the network resources among the clients. The results also show that the proposed architecture can adapt to changing network and server conditions with the help of *SDN*.

- The third contribution is to propose a dynamic virtual cache (vDANE or vCache) placement algorithm. It uses a $PressureCache$ algorithm in order to allocate the first $vCache$ or *vDANE* in a suitable location in SDN networks. We show that the dynamic nature of the $InitialPressureCache$ results to better $vCache$ placement in the network where traffic pattern changes rapidly. Simulation results show that the proposed algorithm minimizes network cost in terms of bandwidth usage.
- The last contribution is to extend $InitialPressureCache$ algorithm for installing additional $vCache$. In lack of enough bandwidth, client experience re-buffering or low-quality video. The proposed $PressureCache$ algorithm overcomes this problem by installing an additional $vCache$ in a location that minimizes network resource consumption and maximizes client's QoE.

1.2 Outline of the Thesis

In *Section 2*, we provide fundamental information about adaptive streaming technologies as well as new network architectures rely on virtualization technology include NFV and *SDN*. We give a formal definition of the streaming and related challenges as well as making an overall survey on them.

In *Section 3*, we present an novel algorithm for optimal codec and path selection. We introduce the problems and dealt with: video coding, path selection, and routing algorithms. We conclude the chapter by presenting the general model for our algorithms.

In *Section 4*, we focus on the virtual cache placement and migration in the case of needs. By providing an algorithm with considering resource distance and available network bandwidth. The proposed algorithm has a dynamic nature and leverages *DASH-SAND* advantage in order to another cache witch newly installed in new location.

In *Section 5*, we propose an algorithm to achieve better QoE. The proposed algorithm, controller assists players to adapt suitable bitrate and enforce them to avoid greedy behaviors. Although, in the proposed system players have the freedom to request with the highest bitrate, in lack of enough bandwidth controller enforce them to not exceed specific bitrate.

In *Section 6*, we conclude our thesis and address published papers and future works.

2. BACKGROUND AND RELATED WORKS

In this section, we have a quick but accurate overview of streaming technologies, *SAND*, *SDN*, and *NFV* concept and standards. We introduce the problem, definition of video streaming and related technologies, and limitations in legacy networks. Finally, we provide a literature review in general form. Also, we discussed related works with more details in each chapter.

2.1 Problem

Streaming content either live or on-demand utilize the main part of IP network traffic. Unlike on-demand streaming, in live streaming content delivered to players in real-time as it happens. Providing low latency streaming contents to the end-user through heterogeneous networks which traffic pattern changes rapidly is a big challenge. Network resources and the mainly available bandwidth is the most critical point in streaming delivery. As shown in Figure. 2.1, lack of available bandwidth can affects playback. As a result, players experience poor quality or freezing streaming. A different class of solutions has been discussed to address this problem.

Despite the importance of adaptive streaming contributions, we believe that the overall performance, especially in video services, cannot be only based on user perception. The network architecture, request distribution and variation amount of the cross-traffic also have a big impact on client experience. While network condition and traffic pattern continually change utilizing the advantage of *SDN* technology give more intelligence to *DASH* clients to better adaption which results in better performance in them of *Quality of Service* (QoS) and QoE. While QoS is more pertinent with network engineering and traffic management, QoE is the exact video quality experienced by the client during display time.

2.2 Scalable Video Coding

Nowadays, the Internet and mobile networks are becoming a primary content distribution mechanisms. *Video coding* is used in different applications ranging from social media, video conferencing, wireless and Internet video streaming, to standard and high-definition TV broadcasting. While video content is delivered to a variety of decoding devices with different computational powers and heterogeneous display, flexible adaptation to desirable content is the key factor for achieving better quality under restricted network resources and conditions. Conventional single layer video streaming in a unicast manner does not make sense anymore. *Scalable Video Coding*

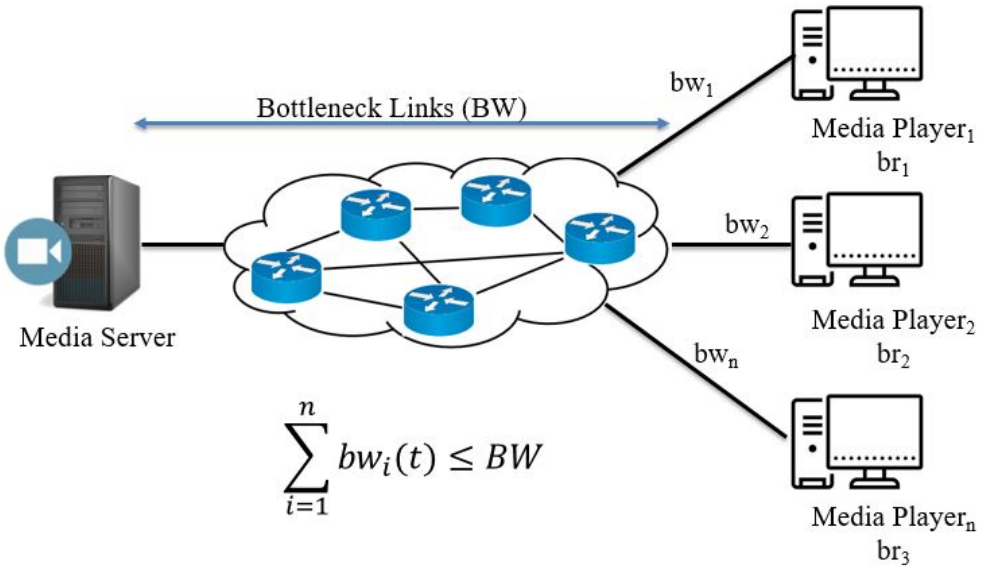


Figure 2.1. Video streaming and bottleneck link

(SVC) which has benefit from *H.264/MPEG4-AVC* standard, is a technology that encodes content in dependency layered format. In the context of quality, *scalability* indicates that a video decoder can retrievers video sequences without receiving all quality layers (Ji et al., 2011). In the SVC technology, several sub-streams with different qualities are transferring in one stream. As a rule, these streams are basic and enhancement ones.

While the first layer provides the base quality, enhanced layers increase the quality of display by adding extra data concerning improve bitrate. In other words, SVC introduces a way to transfer subsequent of the stream inside a single stream. Layer dependency forces players to receive layers (sub streams) in the orderly fashion. While weaker players which have the lowest bandwidth request and display base layer, a player with high enough bandwidth can experience better quality bay requesting base layer and a few enhanced layers. For example, in order to display video with layer two quality player should request base and first layers. This unique encoding scheme enables the quality of video segments to change incrementally and result in more flexibility and better adaptability in high dynamic networks (Elgabli et al., 2018). Overview and technical features of the SVC streaming can be found in (Schwarz et al., 2006).

2.3 Adaptive Streaming Technology

Toward effective streaming, shifting from traditional streaming technology (e.g., RTSP, RTMP) and backing to HTTP-based adaptive streaming was inevitable. Traditional UDP-based media delivery solutions have difficulty passing through firewalls and routers, but HTTP-based streaming has no such problems. Furthermore, leveraging HTTP-based web servers can address network scalability and enable providing better streaming service to edge clients with the benefit of caching content at the edge points. Adaptive streaming is a way to provides video streaming in a quality fashion. Adaptive means that players on the client's side automatically decide for adapting to the next segment of the video based on the observed throughput values and/or buffer occupancy. Three main streaming technology components are (i) Server which provides live or on-demand streaming and have a responsibility to offers feature like *Digital Rights Management* (DRM), (ii) player on the client's side to playback the media, and (iii) specific file formats. The key point is, all the streaming solutions leverage the advantage of compression technology known as encoding in order to maximize bandwidth utilization and retrieval in real-time.

Figure. 2.2, shows the concept of the adaptive streaming technology where video content should be transferred from source to multiple destinations with a different type of qualities and formats. Minimum latency between glass-to-glass is the key concern. After capturing and transferring source files to media servers, media contents should be encoded in different alternative bitrate or resolution in order to better adaptation. Transcoding (*compressing and encoding*) creates multiple files from a single source file. Each encoded bitrate is known as $representation$ and has its own URL address over $HTTP$ server that accessible by the client via $HTTP$ get request. Encoding more representations of the same content resulting in better adaptation, but it needs more processing time which increases to on the fly packaging cost. Encoded video is segmented into small size chunks, almost 2-10 seconds in length. Small size chunk help to fast adaption. Prepared video content is distributed into world geographic via *Content Delivery networks* (CDN).

Adaptive streaming and on top of them *DASH* are a performance management technique that offers lots of potential for *Over-the-Top* (OTT) media service toward streaming multimedia over the Internet. Bitrate flexibility of the adaptive streaming minimizes distribution costs. Thanks to adaptive streaming can deliver suitable quality streams to end users even with lacking in the connection speed and processing power. This advantage helps CDN and OTT companies, optimize delivery costs.

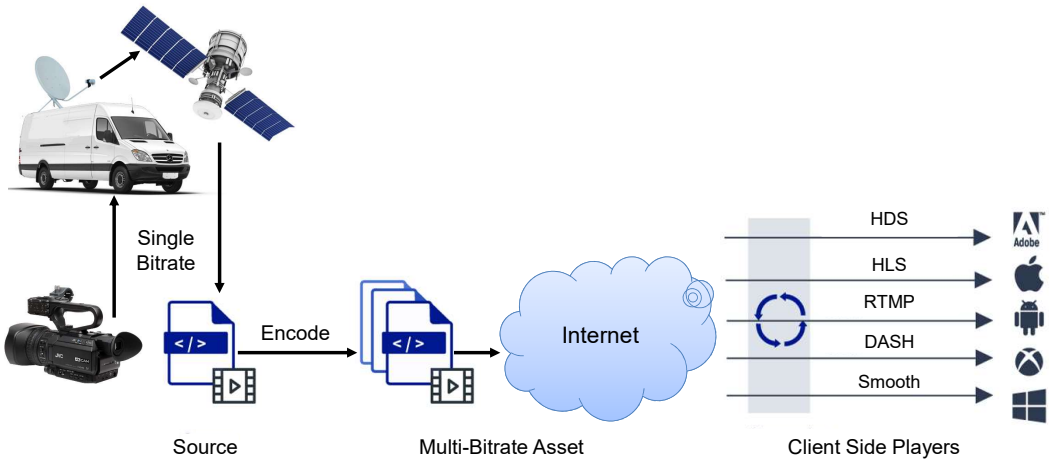


Figure 2.2. Video coding and adaptive video streaming

Adaptive streaming relies on the Internet which is an incredibly complex combination of heterogeneous networks and dynamic conditions. This complexity makes harder delivering video on the Internet. Unlike *Real Time Messaging Protocol* (RTMP) packet which has difficulty passing through the firewall, $HTTP$ base adaptive streaming protocols have firewall-friendly nature and can easily leverage available networks $HTTP$ servers and caching. Hence, $HTTP$ streaming has become a more popular solution in commercial deployments. It is worth emphasizing that, in terms of quality, the adaption feature does not deliver consistent quality.

2.3.1 HTTP Live Streaming

HLS introduced in 2009 in order to provide reliable and cost-effective video delivery over the Internet. A multimedia presentation is determined by a *Uniform Resource Identifier* (URI) to a playlist. A media playlist contains a list of video segments, while played sequentially, will display the video. An example of a media playlist is shown in Figure. 2.3. The first line refers to the format tag $\#EXTM3U$. While there is only one media format for audio, in the $variant$ stream, we see three video segments format each encoded with different bitrates (e.g., 350, 700, and 1000 kbps) and particular resolution, but with the same audio quality (96 kbps). A player on the client-side download this list and then requests and play video segment by segment while adapting to network conditions. Also, the $variant$ stream may contain a list of $renditions$. In the context of video streaming, rendition refers to an alternative copy of the same content with different properties. For example, content such as video recorded with different camera angles or different audio languages.

```
#EXTM3U
#EXT-X-VERSION:4
## Created with Unified Streaming Platform(version=1.9.5)

# AUDIO groups
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="audio-aac1-96",NAME="audio-TUR",LANGUAGE="TUR",AUTOSELECT=YES,DEFAULT=YES,CHANNELS="2"

# variants
#EXT-X-STREAM-INF:BANDWIDTH=521000,AVERAGE-BANDWIDTH=473000,CODECS="mp4a.40.2,avc1.4D401E",RESOLUTION=640x360,FRAME-RATE=25,AUDIO="audio-aac1-96",CLOSED-CAPTIONS=NONE
ch360-audio-TUR=96000-video=350000.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=929000,AVERAGE-BANDWIDTH=844000,CODECS="mp4a.40.2,avc1.4D401E",RESOLUTION=768x432,FRAME-RATE=25,AUDIO="audio-aac1-96",CLOSED-CAPTIONS=NONE
ch360-audio-TUR=96000-video=700000.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=1278000,AVERAGE-BANDWIDTH=1162000,CODECS="mp4a.40.2,avc1.64001F",RESOLUTION=1280x720,FRAME-RATE=25,AUDIO="audio-aac1-96",CLOSED-CAPTIONS=NONE
ch360-audio-TUR=96000-video=1000000.m3u8

# keyframes
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=52000,CODECS="avc1.4D401E",RESOLUTION=640x360,URI="keyframes/ch360-video=350000.m3u8"
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=103000,CODECS="avc1.4D401E",RESOLUTION=768x432,URI="keyframes/ch360-video=700000.m3u8"
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=146000,CODECS="avc1.64001F",RESOLUTION=1280x720,URI="keyframes/ch360-video=1000000.m3u8"
```

Figure 2.3. An example of the HLS playlist indicate media specification includes audio and video formats and bitrates

2.3.2 Dynamic Adaptive Streaming over HTTP

MPEG-DASH is a standardized low latency HTTP-based streaming method. *DASH* is codec agnostic and can supports H.264, HEVC/H.265, VP9, video codecs. *DASH* streaming content which is stored on HTTP server consists of media content in the form of small fixed-length segments, and *Media Presentation Description* (MPD) file explains a manifest of the available content, various representations, fixed or variable duration of segments, and *URL* addresses. Players start streaming by requesting *MPD* file and measure download time. By parsing received *MPD* file, players learn about characteristics of the media such as *URL* location of content, *DRM* and available representations as well as network available bandwidth. Based on achieved information client request a suitable alternative of the first segment with triggering $HTTP$ GET request. In order to fast startup, almost clients request a few segments with the lowest representation that have minimum bitrate until appropriate buffering. After that, clients continue fetching the next segments by running an adaptation algorithm with considering network bandwidth fluctuation and buffer occupancy. With respect to better understand the concept of *DASH*-as adaptive streaming-, we have some terminology.

- *Video Model:* In the *DASH* streaming, the video file is segmented into N equal size partitions known as segments(or chunks) and indexed as $1, 2, ..., N$ such that each segment represent P seconds of video. Each segment encoded in different bitrate called representation. Representations which encoded with higher bitrate provide better display quality, but consume more bandwidth while they have bigger size in bits. A media file that includes segments with different encoded bitrate and resolution, is stored in $HTTP$ media server.

- *Video Player:* The player downloads successive segments of the video from the media server and playback them after downloading is complete. In each adaptation, the player only requests one segment from the media server. Also,

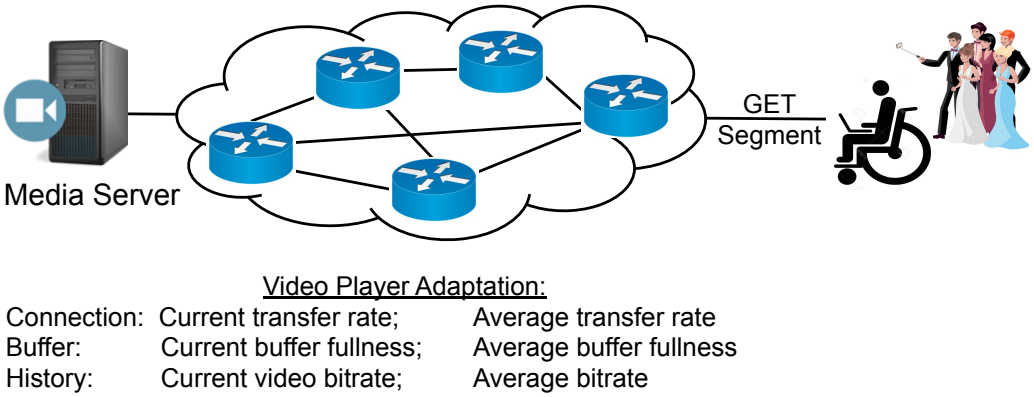


Figure 2.4. Conceptual model for DASH adaption algorithms.

the player has a limited buffer size (e.g., 24 seconds) to store downloaded video segments before playback.

- *Adaptation Algorithm:* Adaptation algorithms can be classified into three main classes; client-based adaptation, server-based adaptation, and network-assisted adaptation. Each adaptation category also can be divided into multiple adaptation algorithms. For example, client-based adaptation algorithms can be separated into two methods: *Throughput Base Adaption* (TBA), and *Buffer Based Adaption* (BBA). Besides the TBA and BBA algorithms, hybrid models also have been proposed. While the TBA algorithm estimates network bandwidth in order to adapt to successive suitable representation, the BBA considers only buffer occupancy. Figure. 2.4 illustrate conceptual model of DASH adaption algorithms.

- *Switch-over:* In the context of adaptive streaming, the term $switch - over$ refers to moving among different representations (quality layers). The *QoE* determined by both video quality which measured by bitrate value, and playback characteristics such as re-buffering and switch-over. Therefore, *DASH* streaming needs a tread-off between enhancing display quality and reducing video re-buffering as well as quantitative of switch-over.

2.4 Server And Network-Assisted DASH

It has proven that *DASH* client-pull technology is more flexible, firewall-friendly and CDN-scalable than server-push technologies. However, the distributed and client-driven nature of *DASH* also causes some challenges. For instance, service providers may not be able to guarantee the premium quality of service with

DASH (Thomas et al., 2016). Because, decentralized natures of the distributed clients, eliminate control over the client's behavior. To overcome this challenges *MPEG* introduced *DASH Assisting Network Elements* (DANE) in new *SAND* architecture. As a key feature of SAND architecture, enhancing *DASH* content delivery and improving the efficiency of streaming is achievable by enabling communication between different elements and exchange quality-related assisting information about real-time operational characteristics of networks. *SAND* offers asynchronous network-to-client and network-to-network communication (Thomas et al., 2016). Such a solution try to avoid delay or interfere with the delivery of the streaming media content. In the concept of *SAND* architecture four class of network elements are defined:

- *DASH client*: DASH client with SAND features, i.e. which can interpret SAND comments and act accordingly.
- DASH metric server: responsible for gathering *DASH* metrics from clients
- *DANE*: DANEs are network elements with minimum intelligence about *DASH*. For instance, the *DASH* server.
- *Regular Network Elements*: Although regular network elements (RNE) are presented in *DASH* infrastructure, they have not information about *DASH* format and characteristics.

The relation between SAND elements and message passing show in Figure. 2.5. Typically, *SAND* messages are delivered in *Extensible Markup Language* (XML) format using HTTP protocol. PER *(Parameters for Enhancing Reception)* and PED *(Parameters for Enhancing Delivery)* messages are sent by the DANEs to the clients and other DANEs, respectively, and they are used for providing information about the network to the clients or making content or infrastructure arrangements in DANEs for enhancing delivery. Status and metrics messages are sent by the clients to DANEs, which carries information about the current status of clients' internal parameters consisting of buffer level, requested quality and expected bandwidth for delivery (DASH-IF, 2017). On the other hand, DANEs trigger *DASH* clients to better adaption by sending information about cached segments, alternative segment availability, timing for delivery, network throughput, and QoS via PER messages (Kalan et al., 2019) (DASH-IF, 2017).

Message Types	Description
Parameters Enhancing Delivery (PED)	Exchanged between DANEs
Parameters Enhancing Reception (PER)	Sent from DANEs to DASH clients
Statues messages	Sent from DASH clients to metrics server

Table 2.1. SAND architecture and message types.

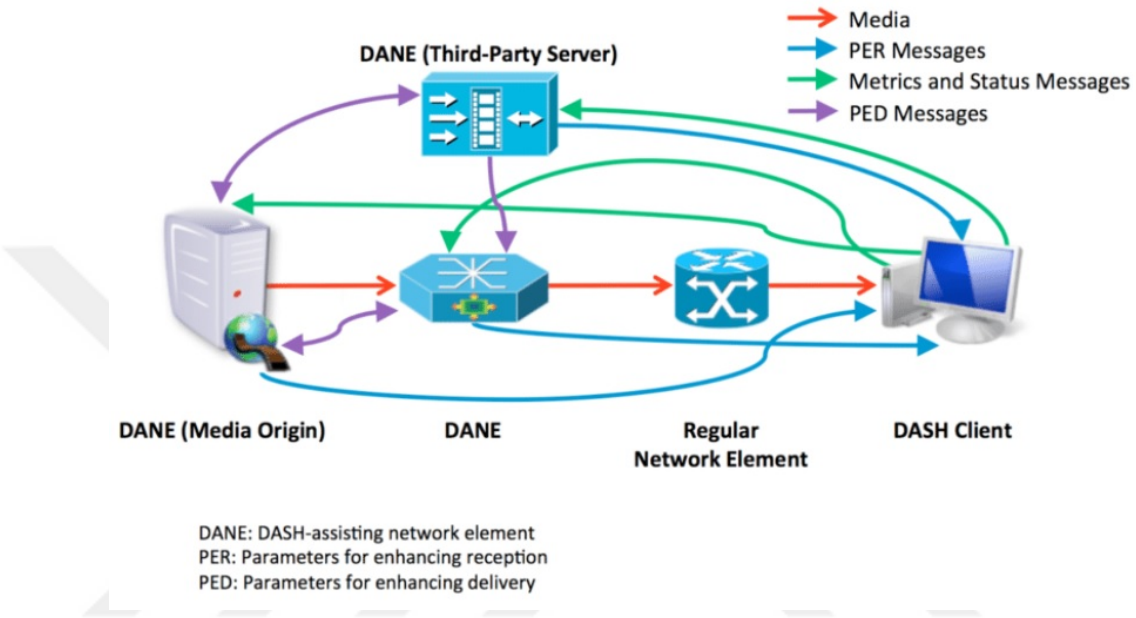


Figure 2.5. SAND architecture and message passing (Thomas et al., 2016)

2.5 Software Defined Networking

Besides video streaming technologies, the performance of these applications is also strictly coherent on underlying network conditions. Providing better network capacity helps to increase the quality of video sent to the clients. Therefore, *SDN* (Nunes et al., 2014), (Sezer et al., 2013), (Kim and Feamster, 2013) technology can be a good alternative for network operators offering services to video streaming providers. *SDN* is a new form of networking technology that transforms hardware and device-centric network architecture into a flexible, virtual, and programmable network in order to improve agility and provides fast innovation.

Conventional network architecture which is relied on the vertical design where data and control planes are bundled together makes network modification hard and complex task. Creating and introducing a new abstraction of networking with respect to simplifying management is conceivable by breaking the network control problems into traceable pieces (Kreutz et al., 2014). *SDN* architecture with sep-

arating control and data plans overcome this complexity and gives more agility to network functions. *SDN* is a preferred solution for high demand resources and unpredictable network traffic pattern was created in response to future networks demand. In a lack of *SDN* centralized control and management nature, current IP network operators need to configure each device individually.

Leveraging *SDN* allows centrally observing the current network traffic and developing a flexible routing algorithm specific to the requirements of applications running on top of the network. Network operators develop and implement an application-specific routing strategy by putting the network intelligence to the control plane. The control layer, monitors and manages the behavior of the network infrastructure and application through the *Southbound Interface* (SBI) and *Northbound Interface* (NBI). The control layer periodically communicates with network infrastructures to monitors and collects network status. In the context of segregation data and control planes well defined *Application Programming Interface* (API) can be leveraged between the *SDN* controller and underlay switches.

Technically, controller as *network brain* or *Network Operating System* (NOS) translates and forward application's provided functionality into infrastructure forwarding devices via $OpenFlow$-based well defined instruction set. Such a paradigm gives more agility to packet forwarding devices. By reducing control overhead from forwarding devices, data plane elements become highly efficient and programmable, but dumb packet forwarding devices. In addition to the central management capability and flexibility in routing of flows, *SDN Controller* also obtains flow statistics in its domain, such as throughput and packet loss.

The $OpenFlow$ protocol (McKeown et al., 2008), is the most popular communication protocol for transferring messages between the switches and the controller through the SBI. Also, NBI refers to the software interfaces between the controller and the application services that are run by collaborating with network operators. It is worth underscoring that logically centralized programmable model needs to guarantee an adequate level of performance, reliability and scalability (Koponen et al., 2010). While a single controller is enough for medium-size networks, it can be easily scalable via the west and the east API in enterprise networks. Furthermore, a two-tire *SDN Controller* hierarchy can be considered (Ali-Ahmad et al., 2013). while local controllers have fast responsibility, in top-level global controllers deals with a global decision. In the case of concerns with *SDN* scalability, resiliency to failures and convergence time after a failure a deep overview and technical details can be found in (Yeganeh et al., 2013).

In the context of security, *SDN* offers two classes of solutions: First class of solution involves using SDN in order to improve network security. $OpenFlow$ enabled devices at the entry point to the network make it possible to collect incoming information to the network which is useful in detection *Distributed Denial of Service* (DDoS) flooding attaches (Braga et al., 2010). Further, active security (Hand et al., 2013) provides a centralized programmable interface in order to: (i) gathering data from multiple resources, (ii) convergence collected data in security applicable format, (iii) reaction to block the attaches or decrees influence of attaches. The second class or solution relies on identifying critical security threats of *SDN* itself such as classification and prioritized application with considering that high priority rules generated by security application will not be affected by low priority applications (Porras et al., 2012).

2.5.1 Details of the SDN Architecture

In this section, we present, the most significant architecture view of the *SDN* technology. Technically, three main parts of the computer networks include: (i) At the top of architecture, management plane which includes software application and management tools or protocols, handle network policy. (ii) A central part is a control plane that interacted with underlay forwarding devices. (iii) Forwarding device executes routing information and travels data packet through network links. The control plane enforces network policy defined in the management plane into forwarding devices and fills forwarding tables with update related information. At first glance, tightly coupling control and data plans result in network resiliency, unfortunately, such a relatively static architecture reduces network flexibility and eliminate easier innovation.

In the context of *SDN* architecture, decoupling the control and data plans alleviate network complexity and make centrally control underlay devices (as a single big switch), feasible. The main concept of the *SDN* architecture relay on modifying heterogeneous forwarding devices to support data plane programmability via open and standard interfaces (e.g, OpenFlow) which are effortful in the legacy network because of the distributed nature of the control plane. As shown in the Figure. 2.6, top-down deceleration (or statement) of the *SDN* architecture includes five layers:

1. *Management plane:* This layer contains a list of the application using for traffic management (e,g. routing algorithm and load balancing), security (e.g, intrusions detection system) and network control protocol (e.g, SNMP). All

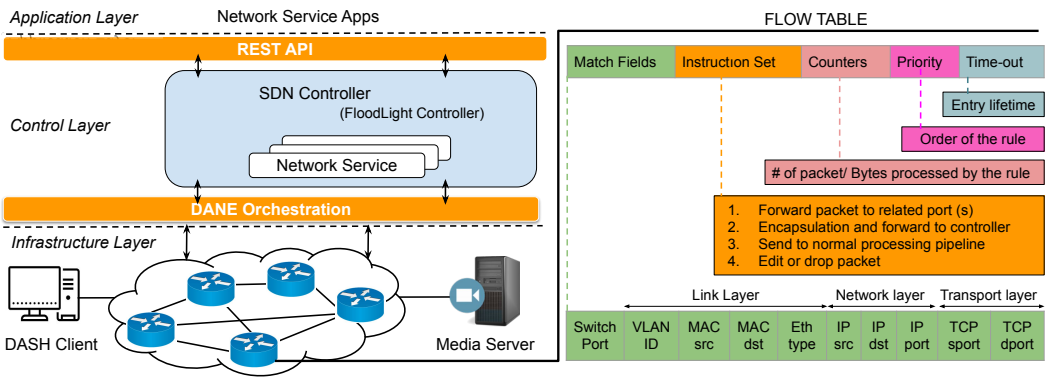


Figure 2.6. Illustration of the SDN architecture and flow table fields.

applications can leverage almost the same network information in order to policy management and effective decision making. Application's policies ultimately running by forwarding devices.

2. *Northbound Interface:* The northbound interface allows communication among the higher-level components such as application and NOS, known as the *SDN Controller*, via *RESTs APIs*. Typical, the purpose is to abstract the innerworkings of the network and convenience application developers.

3. *Network Operation System:* The NOS or controller as a core of the *SDN*, loads forwarding device's tables with application policies. Network basic functions (e.g, link manager, path manager, flow manager) facilitate controller in order to collect network metrics and statistics. Such abstraction of the network information alleviates management applications programming.

4. *Southbound Interface:* Southbound $APIs$ hide details of the individual network devices and facilitate efficient control over the network. This characteristic enables the controller as well as application to dynamically make changes according to real-time demands without struggle with low-level configuration.

5. *Data Plane:* Also known as forwarding plane, embrace programmable physical equipment, such as $OpenFlow$ switches, which have less intelligent about the network. Forwarding devices contain one or more forwarding tables and well-defined instruction set (e.g, flow rules) (Kreutz et al., 2015) used to take action on the incoming packet (e.g, forward, drop, and modify).

The main concept of the *SDN* architecture relies essentially on modifying forwarding devices to support flow tables. Compared with legacy networks in which infrastructure devices have autonomous decision making, equipment in the *SDN* just acts as simple forwarding devices. In other word, the absence of control in network equipment is the main difference between *SDN* and legacy networking. This new architecture is built on virtualization and standard interface such as $OpenFlow$.

Control plane fills $OpenFlow$ enabled data plane devices table. Data plane devices are responsible for packets forwarding based on matching rule, and periodically sending link's statistic to the controller. Flow tables -Inside $OpenFlow$ devices- define how the packet should be handled. If there is no defined rule for arriving packet, depend on configuration, the packet will either discarded or sent to the controller.

In summary, lack of the tightly coupled and integrated control in *SDN* equipment is the main difference between the *SDN* and traditional networks. In the traditional network, forwarding devices have an autonomous decision, however, in the SDN network act just as a simple forwarding element.

2.5.2 Why SDN?

Demand for new network architecture is undeniable, while traditional network based on the TCP/IP protocol is not efficient anymore. However, designing a new network protocol and architecture is more costly and in practice it is impossible. To address this problem we need network transforming. *NFV* (Mijumbi et al., 2016a) (Mijumbi et al., 2016b) is expected to allow the network provider to benefit of virtualization on the management of network functions by decoupling network functions from the physical devices. NFV has been proposed to shift hardware base middle-box processing to the application running on commoditized hardware (Kreutz et al., 2015). In this concept, *SDN* (Nunes et al., 2014) (Kim and Feamster, 2013) (Sezer et al., 2013) as a complementary technology, has the potential to dramatically impact the architecture of legacy networks and make NFV easier. *SDN* technology gives more flexibility and performance to network functions by decoupling the control plane and the data plane. This paradigm introduces programmable infrastructure in order to configure a network efficiently, to enhance network monitoring related skills and to provide improvement in network performance.

2.6 Network Function Virtualization

Network functions (e.g., cache, load balancing, and IDSs) are used to modify workload (e.g, packets and flows) in sophisticated ways (Gember-Jacobson et al., 2014). However, network functions as a temporary solution may bring new management challenges. Virtualization is based on the fact that the virtual machines installed on-demand and share the same hardware resources while the network should assign identical attributes to the computing layer (Chowdhury and Boutaba, 2010). Further, the on-demand nature of the construction, emigration, and annihilating are momentous of virtual machines. The flexibility introduced by the virtualization persuade telecommunication industry like *European Telecommunication Standard Institute* (ETSI), *Internet Engineering Task Force* (IETF), and *Third Generation Partnership Project* (3GPP) involves in *NFV*.

The individual concept of NFV has a successful experience in *MPLS* (virtualized path) (Xiao et al., 2000), *NAT* (Virtualized IP address), and *VLANs* (Virtualized L2) (Jain and Paul, 2013) domains. However, in the absence of global configuration, the distributed nature of virtualized functions rises configuration cost in time and complexity manner. More importantly, vendor-specific *NFV* solutions contract with *NFV* original concept (Mijumbi et al., 2016a). With the *SDN* assistance, distributed control and configuration can assemble in a single central point. As a consequence, the *SDN* platform should be open for application and hardware from multiple vendors. In the class of scalability and reliability, control redundancy should be considered as well.

The NFV orchestration which is in the charge of provisioning for virtualized network functions, controlled by the SDN controller through southbound interfaces (Li and Chen, 2015). The *SDN* controller computes the optimal function assignments (e.g., assigning a virtual cache function to a certain location or node) and translates the logic policy characteristic into optimized routing paths. The function assignments are enforced by the NFV orchestration and the controller steer the traffic traveling by installing forwarding rules into forwarding devices. Network operation transforming is triggered by using *NFV* and *SDN* technologies. *NFV* leverage virtualization in order to separate software instance from the hardware platform and gives more agility to network services by decoupling functionality from location (Han et al., 2015). However, by taking advantage of NFV-SDN, it is important to sure that the network performance remains at least as good as previous and does not bring new challenges such as security risk. In addition, *Virtual Network Function* (VNF) can be moved between different platforms. However, movement should not

have a negative impact on QoS parameters such as packet loss and delay.

Albeit NFV introduces a way to separating network services (e.g., caching, load balancing) from the underlying hardware by transferring network functions from dedicated hardware to general-purpose software such as hypervisor (Li and Chen, 2015) with respect to reduces capital expenditures (CAPEX) and operating expenses (OPEX), *SDN* rely on *NFV* and virtual centralized controller actualize hypervisor roles that control virtualized network functions. While *NFV* separate physical and virtual devices, *SDN* offers central control by decoupling the control plane from the data plane in network forwarding devices. With *NFV* assistance, *SDN* steer traffic dynamically based on user requirements. Thus, resource utilization can be achieved by a combination of *NFV-SDN*.

2.7 NFV and SDN Correlation

Although the concept of *NFV* and *SDN* are independent, they are in practice highly correlated and are mutually beneficial. The core similarity between them is that they both use network abstraction and depend heavily on virtualization. *NFV* and *SDN* are performing different roles, but leveraging *NFV-SDN* specifications enable networking architectures more flexible and result in network infrastructure agility. While *SDN* optimize network infrastructure by separating network control and forwarding function, *NFV* optimize the development of the network functions. As shown in Figure. 2.7, *NFV* works on the upper layer of the TCP/IP model and provides basic networking functions, while *SDN* orchestrates and configures network functions for specific uses. *SDN* benefit the advantage of virtualization in order to provide programmable configuration and optimize network resources quickly. A variety of programmable interfaces permits network programming. With *SDN* technology, flow-based routing, and dynamic traffic adjustment it is possible to meet fluctuating demands.

While *NFV* reduces the network operational expenditure, *SDN* leverage *NFV* to enhance its performance by simplifying the compatibility with the legacy network and facilitating its operation with making networks programmable. In addition, the middle-box -as a new and temporary network features in the case of network flexibility- brings new challenges in network management. Fortunately, leveraging *SDN* technology gives more flexibility to network management while provides a global view of the network. *SDN* and *NFV* architecture are now being commercially deployed to deliver network slicing feature. Network slicing is a form of virtual network architecture leverage *NFV-SDN* principle in order to partition (based

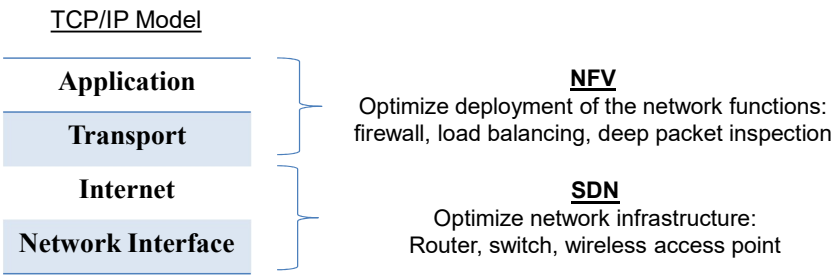


Figure 2.7. Illustration of the NFV-SDN correlation .

on topology, bandwidth, and traffic) fixed network into virtual elements that can be linked. In the sliced network, multiple controllers can coexist on top of common shared physical network infrastructure. Each controller (or virtual network) is customized to meet the specific characteristic of applications, services, and devices.

2.8 Content Delivery Networks

To display video, the request needs to dispatch from the client to the origin media server. Respond to every request only from the origin server is more costly and risky. Besides, distance latency leads to poor QoE. Caching a copy of the content in geographically distributed PoPs (Points of Presences) provides quick access to the contents and results in better network bandwidth utilization and improves QoE. Content Delivery Networks (CDNs) is a geographically distributed system have risen in massive popularity to address this problem. Leveraging edge' cache, where content stored near to the client, eliminates the number of requests from the origin server and reduce latency. Besides the reliable delivery of content, CDN also enhances origin server security. By leveraging CDN and based on real-time logs analytics and statistics, the OTT video service providers can determine where extra capacity is needed. Hence, in case of suffering from overload in cretin geography, it can provide extra bandwidth to make sure that everything is running smoothly.

virtual nodes can be easily moved from one location to another by leveraging a combination of the CDN and *NFV-SDN*. In such a domain, virtual cache (vCache) store and stream media content. However, content migration to the optimal position is the main challenge. Thus, the intelligent algorithm to find the optimal location for deploying cache has a significant impact on media delivery. In the context of the multimedia system, advantages of streaming with a CDN include:

- $Scalability$: CDN is the fastest and most reliable way to provides content near to numerous end users anywhere anytime.

- $Speed$: CDNs deliver content at low latency while they have high-speed connections. Multi-tier architecture and peering between edges eliminate streaming from the origin server. While overwhelming origin server increases response time, the light load origin server can provide sound quality service.
- $Quality$: CDNs deliver content at low latency while they have high-speed connections. Multi-tier architecture and peering between edges eliminate streaming from the origin server. While overwhelming origin server increases response time, the light load origin server can provide sound quality service.
- $Security$: In the context of security, streaming through a CDN can help prevents Distributed Denial of Service (DDoS) attacks. It comes from the fact, redundant CDN with multiple access points overcome a single point of failure.

2.9 Related Works in General

Align with introducing new streaming technologies, sophisticate enhancing in players equipment rise to support new streaming content and led to high-performance multimedia services expectation. The popularity of handset devices and wide-spread adaption of social media encourage providing streaming content over the real-time data platform such as OTT. However, providing high quality and affordable multimedia services over the Internet is the main challenge. Conventional network architecture can hardly handle the quality matter multimedia traffics while in it's content/service agnostic infrastructure quality issue was considered only in the network and transport layers.

Depend on the decision-making point for adaptation and requesting the next segment, bitrate adaptation schemes can be classified into multiple adaptation methods: client-based, server-based, and network-based adaptations. In the client-based adaptation, the client runs the rate adaptation algorithm which determines the quality of the segments to be requested. Despite the advantage characteristic of *DASH*, it has been shown that it suffers from three performance issues: unfairness, instability, and the under-utilization of the available bandwidth (Altamimi and Shirmohammadi, 2019), (Hemmati et al., 2015).

The client-driven nature of *DASH* has limited information and view about network conditions and other clients' behavior. Also, service providers may not necessarily have control over the client's behavior, therefore they may not be able to guarantee the premium quality of service with client base adaptation. Server-based adaption algorithms - such as MS-Stream (Bruneau-Queyreix et al., 2017), QAC

(De Cicco et al., 2011), and QCSS (Zhang et al., 2018) try to assign fair quality where multiple clients competing for the available bandwidth. However, while server store clients' information and maintain each client individually, the cost of high overhead and complexity are a drawback of the server-based algorithm.

Since the performance of the multimedia systems in terms of QoE relies on the underlying network conditions and server characteristics such as availability and distance, there is a tendency to get assistance from a network in today's multimedia architectures. Legacy network-based adaptation methods - like MP-DASH (Han et al., 2016) ,QDASH (Mok et al., 2012), BUFFEST (Krishnamoorthi et al., 2017), are monitoring system and run adaptation based on network data and statistic, without considering client's data. Network middle-box such as a proxy server can also assist the client during bitrate adaptation with offering network statistics and information like available bandwidth or suggestion suitable bitrate. However, monitoring network conditions by the middle-box devices have significant overhead and my cause to network congestion by itself specifically in traffic peak time. The main objective of our study is improving the end-user quality experience and maximize fairness by leveraging a centralized network controller without requiring any modification in client-side or overwhelming media server.

The best-effort nature of the current Internet architecture can not support the adaptation and customization of the available network resources in order to provide the content base (e.g., multimedia) requirements. As a consequence of providing insufficient QoS, the end-user suffers from a lack of QoE. Eventually, patching new features to the current Internet architecture towards meeting content or service aware quality also increases network complexity. The NFV-SDN as a refection of the *Future Internet* (FI) provides a new path for end-to-end adaption over the heterogeneous network infrastructures and tries to achieves better QoS/QoE.

In the context of the *SDN*, to the best of our knowledge, the most comprehensive related study on *DASH* can be divided into two main groups. Rely on the fact that mapping requests based on a global view of the network could provide better adaption (Mukerjee et al., 2015) (Liu et al., 2012) , the first group of study explaining the motivation of *DASH* over *SDN* technology without taking advantage of *SAND* architecture into account. The second group involves the context of *DASH* and *SAND* architecture. Leveraging *SAND* architecture gives more intelligence to network controller which has a global view of the network.

3. LAYERED ADAPTIVE VIDEO STREAMING

3.1 Design of A Layer-based Video Streaming System over Software-Defined Networks

In this section, we present a layered based adaptive video streaming architecture where SDN technology is utilized for routing of video layers and for assisting clients in the selection of the most appropriate video codec and video quality under the constraint of current network conditions. In the proposed architecture, the controller estimates the packet loss probability by taking video codec characteristics, bitrates of the layers and network capacity into account. Based on these estimations, the controller assists the client to select appropriate video codec and quality. Simulation results show that utilizing routing flexibility provided by the SDN and considering characteristics of the video streaming system can improve the performance of the video streaming.

3.1.1 Scalable Video Coding / Multiple Description Coding

As being one of the most popular application types used on the Internet, video streaming applications offer a wide range of usage scenarios, from live video streaming services to distribution of personal videos of users. Cisco forecasts for global IP video traffic growth states that IP video traffic is estimated to reach 82% of all consumer Internet traffic by 2021, up from 73 percent in 2018 (Cisco, 2018). The selection of transport layer protocol, quality adaptation techniques, and video codec type affect the performance of the video streaming applications. Hence, when designing a video streaming system architecture, these selections should be carefully considered. For almost a decade, *HTTP Adaptive Streaming* (HAS) has become a dominant technology for streaming video on the Internet.

In the HAS systems, in order to better adaptation different copy of the same content with different bitrates are stored in the origin server.HAS has a client-driven architecture. The client adjusts the suitable video bitrate based on observed parameters such as estimated network traffic or internal parameters such as buffer fullness level. For providing interoperability between HAS systems developed by different vendors, the MPEG working group proposed DASH standard (Hossfeld et al., 2015). DASH standardizes the MPD file format on the server-side. MPD giving the information related to the address and bitrate of the quality alternatives. It also standardizes parser functions implemented for extracting information from the MPD file on the client-side.

In SVC standard, the video file is encoded into a base layer and one or more enhancement layers. The base layer provides the minimum quality and can be decoded independently where each additional enhancement layer improves the quality of the video. The base layer is the most important and should never be lost, because, to decode an enhancement layer, the base layer and the previous enhancement layers are required. In other words, each enhancement layer needs all lower enhancement layers as well as a base layer for decoding due to layer dependency.

Like the SVC standard, MDC also encodes the video into layers called descriptions (Jiang et al., 2012). Similar to SVC, each description received by the client improves the video quality. However, in the MDC, the description layers are self-decodable and each layer carries basic information to decode the video independently. This characteristic of MDC causes extra overhead compared to SVC since each description contains redundant information to be decoded without requiring other descriptions. Thus, the MDC is more tolerant of loss compared to the SVC, but SVC provides better compression.

Besides the design characteristics of the video streaming applications, the performance of these applications is also strictly coherent on underlying network conditions. Providing better network capacity helps to increase the quality of the video sent to the clients. Therefore, SDN technology can be a good alternative for network operators offering services to video service providers. In a SDN domain network operators develop and implement an application-specific routing strategy by putting the network intelligence to the control plane (Nunes et al., 2014), (Sezer et al., 2013), (Kim and Feamster, 2013). The control layer monitors and manages the behavior of the network infrastructure and application through well-defined interfaces. Also, the control layer periodically communicates with network infrastructures to monitors and collects network status. Leveraging SDN allows centrally observing the current network traffic and developing a flexible routing algorithm specific to the requirements of applications running on top of the network.

In this study, we propose a video streaming system architecture based on the SDN controller orchestration, which uses centralized behaviors to forward the packets fairly and improve the quality of video observing by the end-user. In the proposed system, the clients and media servers are authoritative for streaming both the SVC and the MDC encoded video. The SDN controller utilizes underlying network information to select the video codec type and the optimal number of layers the client should request.

3.1.2 Related works

In addition to the central management capability and flexibility in routing of flows, the SDN controller also obtains flow statistics in its domain, such as throughput and packet loss. Leveraging SDN technology allows to observe the current network traffic and develop a flexible routing algorithm specific to the requirements of network applications. Although SDN is a relatively new technology, due to its advantages, there are too many studies proposing video streaming services over SDN in the literature.

In a group of studies, SDN is utilized for selecting suitable streaming paths for transferring video packets from the server to the clients, thanks to the flexibility of SDN on determining flow routes. The performances obtained by various path selections strategy discussed in (Jarschel et al., 2013), where available bandwidth and the load of the paths are considered for path deciding. Assigning the shortest path under the constraints of the UDP streaming parameters such as $jitter$, packet loss and delay constraints is discussed in (Egilmez et al., 2012). Streaming path selecting based on the optimization model with considering traffic volume to be sent is presented in (Cetinkaya et al., 2014). These studies give the performance obtained by selecting streaming paths based on strategies considering various observed parameters.

Authors in (Bentaleb et al., 2016) launched a new SDN based dynamic resource allocation and management architecture for HAS systems, which aims to alleviate these scalability issues and improve the per-client QoE. The introduced architecture manages and allocates the network resources dynamically for each client based on its expected QoE. Authors in (Ramakrishnan et al., 2015) adopt an SDN-based architecture for dynamic bandwidth allocation across multiple competing HAS streams. They formulate the multi-client bandwidth allocation problem within the convex optimization framework. In (Petrangeli et al., 2017), the SDN controller monitors the network conditions, estimates possible freezes in the video experienced by the clients by using a machine learning algorithm and sends prioritization commands to the $OpenFlow$ switches based on this algorithm.

In another group of studies that utilizing SDN technology in order to increase the performance of video streaming applications, there are several approaches proposed where the controller explicitly signals the clients to assist the quality adaptation process. In (Nam et al., 2014), the controller monitors the client's performance in term of QoE and change video server or streaming path in the case of a perfor-

mance deterioration is detected. Open Flow-enabled system proposed in (Akhshabi et al., 2011), where an orchestrating module explicitly informs the DASH players which representations should be selected. In (Georgopoulos et al., 2013), the controller selects the suitable video resolution following the device type of clients and sends commands to the clients to download the selected resolution. The controller allocates bandwidth for DASH flows and signals the clients by considering reserved bandwidth values for them to select appropriate bitrate based on current network conditions in (Bagci et al., 2017).

Thanks for *MPEG* groups which standardized DASH and introduced SAND technology. In SAND architecture, DANE elements, which are aware of active DASH players can be used to optimize bottleneck links for DASH video traffic to improve the viewer's QoE (Hossfeld et al., 2015). SDN technology can be utilized in SAND architecture. In (Kleinrouweler et al., 2016), DASH clients send representation candidates that can be selected by them to the controller and selects the representation by considering the recommendation sent by the controller. According to the current model, the clients may request the representation recommended by the controller or they may discard the request (Kleinrouweler et al., 2016). In the SDN assisted DASH architecture proposed in (Bhat et al., 2017), the SDN controller sends bandwidth of the caches; and clients can select the cache and the representation based on this information.

Layered video streaming over SDN has been investigated in several studies in the literature. Improving the performance of SVC DASH in the SDN network discussed in (Cetinkaya et al., 2015). $OpenFlow$ based video streaming system for streaming SVC video between multi-server and multi-client is specified in (Xue et al., 2015). Routing base layer over the lossless path and enhancement layers over alternate paths are discussed in (Civanlar et al., 2010). Transferring base layer and enhancement layers over different streaming paths are proposed in (Egilmez et al., 2013), (Laga et al., 2014) and (Gangwal et al., 2016). The streaming paths are determined based on the output of the optimization models that consider packet loss and delay variation for each layer. Moreover, references (Yue et al., 2015), and (Uzakgider et al., 2015) focus on transferring SVC layers over selected streaming paths and determining the suitable number of layers based on these selected paths. The authors suggest solving the joint path and layer selection problem based on learning models taking different parameters into account. As well as, SDN technology can also be utilized for transferring MDC coded video over different paths. In the research (Noghani and Sunay, 2014), a video streaming architecture addressing MDC coded video distribution over SDN. The layers of MDC are forwarded

over different multicast trees constructed by SDN switches. The authors propose to serve different QoS class users by adjusting the number of MDC layers sent to each service class.

However, most of the prior work that implements a video streaming architecture over SDN focuses on selecting the streaming paths or selecting a suitable server. None of them focuses or considers the video codec type when determining the streaming paths. To the best of our knowledge, there has not been any previous study on the benefit of federate selecting optimal video codec type and the optimal number of video layers for video streaming over SDN while network conditions are dynamically changed. While in our architecture, the SDN controller determines the video codec type and the number of layers and sends this information to the client at the beginning of a streaming session. The clients are capable of decoding and playing of both MDC and SVC coded video while interpreting the messages received from the controller. The overview and design of the proposed layer-based video streaming system is given in the next section.

3.1.3 Architecture Overview

The network architecture is conceptually divided into three layers as shown in Figure. 3.1. The bottom layer represents forwarding devices that are responsible for packet functions such as editing, dropping, and forwarding packets to a specific port or controller. Besides forwarding action, forwarding devices send links statistic to the controller periodically. The middle layer represents *Network Operation System* (NOS) or controller. The controller gathers network statistics from client and forwarding devices for network management and decision making. The forwarding devices programmed by the controller with the assistance of the well-defined southbound API such as OpenFlow. The controller communicates with the application layer through northbound as well. Finally, the upper layer represents network service applications. Applications in this layer define the forwarding policies which ultimately translated by OpenFlow southbound interface and program the behavior of the forwarding devices. In the proposed architecture, the controller consists of two main function modules. While the basic functions are used for essential network services such as link and path management, the advanced functions have involved with video codec and optimal layer selection.

Note that, based on the controller performance evaluations presented in (Heller et al., 2012), it is possible to conduct that one controller is often enough to support the whole underlying network. Another point of concern is the fact that SDN networks

can be easily scalable, which is beyond the scope of this study. As shown in system architecture view proposed schema consist of five basic functions:

- *Host Manager:* Provides information about online clients such as location and the time of getting online.
- *Link Manager:* Obtains traffic information from switches. Each switch forwards port statistics to the controller periodically.
- *Path Manager:* Contains statistics of each path include the number of hops in path or number of the available path between source and destination.
- *Flow Manager:* Responsible for traffic management by updating the forwarding tables with new instruction set as a rule.
- *Forwarding manager:* Decides the actions (e.g., forward to a specific port, modify or drop packets) for the packets received from the switches. For example, the new client's request is forwarded to the controller, because at the beginning there is not an entry in the forwarding tables of the switches.

In addition to basic network functions, there are two advanced functions for video codec and optimal layer selection which triggered by the SDN controller on demand.

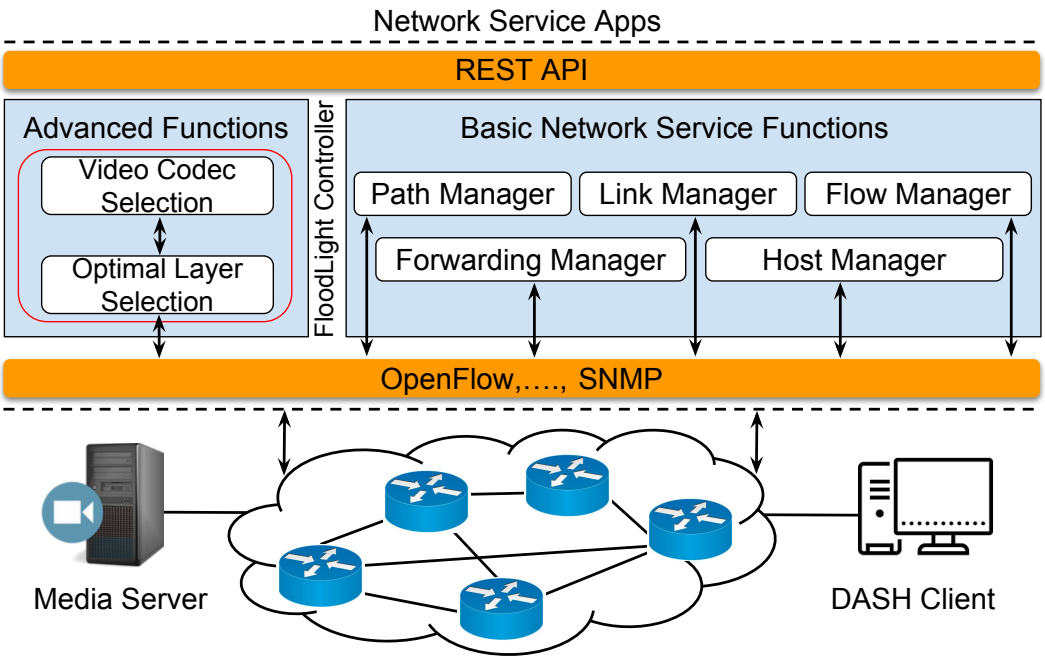


Figure 3.1. SDN controller and system architecture view

The proposed framework consists of a DASH application-aware centralized SDN controller to optimize the behavior of all the clients, DASH server which encodes and serves the video representations encoded with both codec, SVC and MDC; and DASH clients that can interpret the command received from the controller and have the capability of decoding SVC and MDC encoded videos. The controller tracks the clients via its Host Manager module. As shown in Figure. 3.2, When a new client joins to network and trigger establish the TCP connection to communicate with the origin server, it's request handled by the first-hop switch. While there is no entry for a new client in the switch's routing table, then switch forward the client request toward the controller in the form of *PACKET-IN* message to learn the streaming path for that client. In the controller, *Host Manager* module detects newly joined client after it receives a *PACKET-IN* message. The controller runs an algorithm that determines the video codec type, the optimal number of layers as well as the streaming paths of each layer. The controller uses the current network conditions as input to the algorithm. Based on this information, the controller calculates the likelihood of the packet losses at the link layer by considering the bitrates of the video layers and available bandwidth information of the paths that the layers will be transferred over. It then determines the video codec related parameters such as video codec type and the optimal number of video layers by running an optimization model that aims to minimize packet losses at the link layer and maximizes video bitrate. Note that, we focus on the estimation of packet losses at the link-layer since TCP is used and packets are not lost at the application layer. The controller signals the client via REST API for giving the output of the algorithm. At the same time sends the flow route information to the switches via $OpenFlow$ protocol on the southbound API. As shown in Figure. 3.3, the controller can determine different streaming paths for each video layer due to the server sends each video layer through different ports in our architecture. After establishing a TCP connection, the client starts to download the video layers. The controller informs the client with a new optimal number of layers if the network condition has changed. However, the controller does not run optimization, so the video codec type does not change anymore.

3.1.4 The Estimation of Packet Loss Rate at the Link Layer

In order to increase the quality, the clients should increase the number of layer request. However, as explained in the previous section, if the underlying network capacity is not adequate for sending as many as the number of requesting layers, two outcomes may be experienced: i) the download time of the video packets elongates and, hence, the client may suffer from the outage experience if base layer video packets are delayed; ii) the playout time of the delayed enhancement layer packets

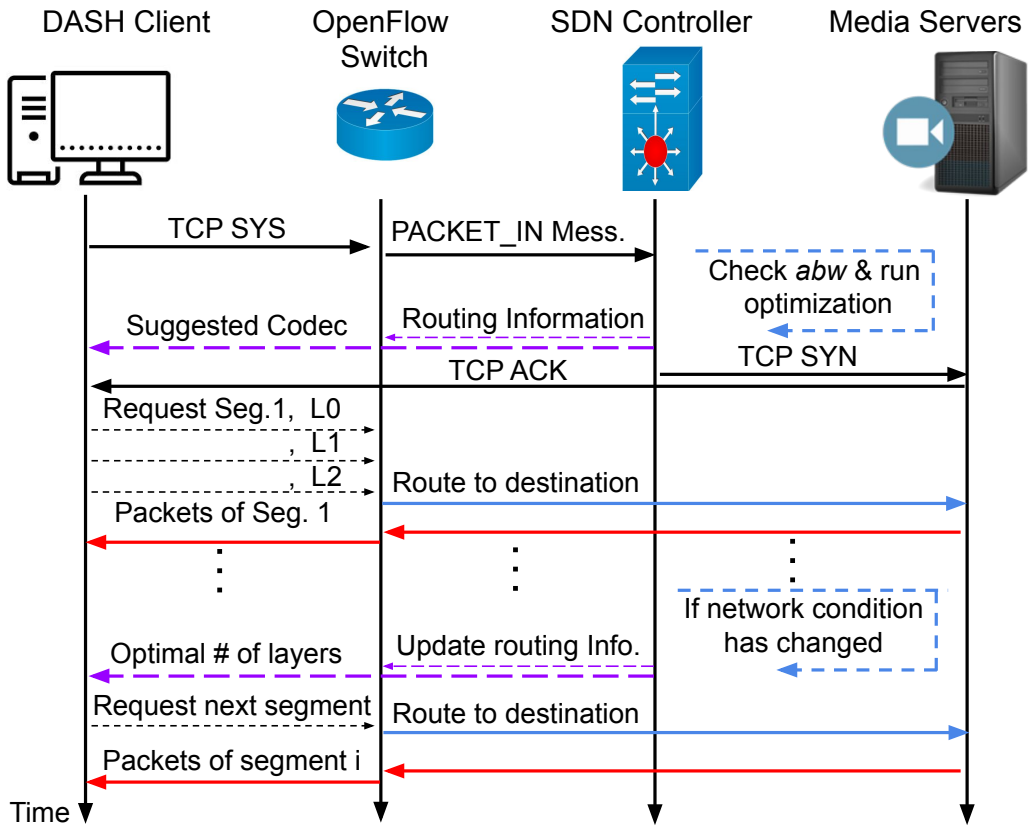


Figure 3.2. Connection setup and controller behaviors, updating forwarding devices flow table

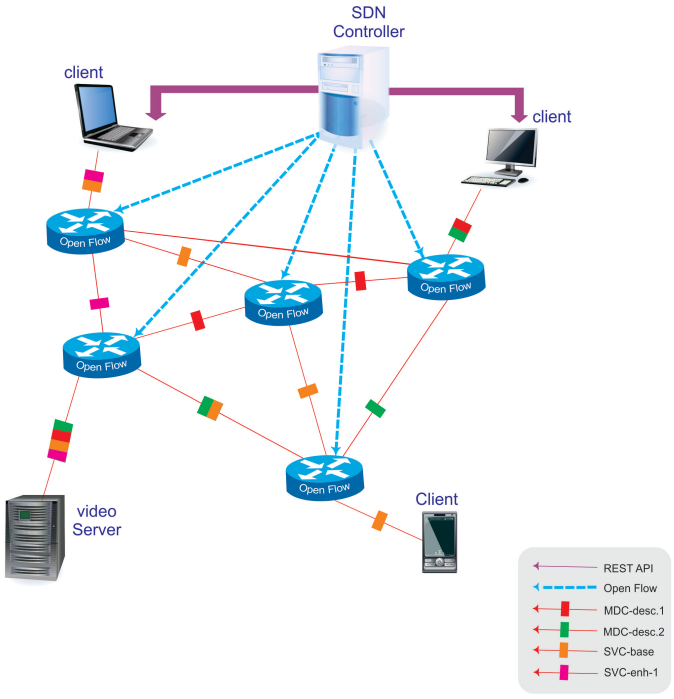


Figure 3.3. Overview of proposed video streaming system .

may pass and this causes packet losses at the application layer, i.e. the video cannot be played at higher quality even if the enhancement layer packets eventually arrives in the client. Clearly, the first outcome decreases QoE achieved by the client while the second outcome causes waste network resources.

In the proposed system, in order to determine the optimal number of layers to be requested, the controller estimates the likelihood of the packet delay at the client-side. Since TCP is used as the transport layer protocol for transferring video packets, the delay mainly occurs from the packet losses at the link layer and re-transmission of the lost packets. Hence, we calculate the expected packet loss ratio at the link layer in order to estimate quality deterioration and the wasting of network resources caused by delay.

The packet loss ratio is related to the available bandwidth of the streaming path and the bitrate of the video layers that will be transferred over that path. The controller periodically obtains information about the underlying network and traffic amount routed via switches. By using this information the controller calculates the total traffic transferred by the flows other than video streaming traffic for the end-to-end paths between the client and the server. For a given path p, available bandwidth that can be used by video streaming clients can be calculated by subtracting traffic value from the capacity of the path p. Suppose abw represents the available bandwidth value calculated by this way for a given path p. If L video layers are transferred over the path p, then the bandwidth portion that can be used for each layer roughly equals to abw/L due to TCP fairness. Hence, the expected packet loss ratio for a given path p can be calculated by using the formula given in formula (3.1), where plr_L and $bitrate_L$ represents the packet loss rate and bitrate of video layer L, respectively.

$$plr_L = 1 - \frac{abw}{L \times bitrate_L} \tag{3.1}$$

The controller estimates the packet loss ratio probability by taking the available bandwidth, video codec type, and the bitrate values of the video layers into account. If a MDC codec is used, the estimated total packet loss rate can be calculated by the given formula in (3.2). If the video is encoded with SVC, the packet loss ratio is calculated by taking a layer dependency into account as given in (3.3). In the formulas, plr_k represents the packet loss rate calculated as given in (3.1) and L represents the number of video layers.

$$plr_{estimated} = \sum_{k=0}^{L-1} plr_k \tag{3.2}$$

$$plr_{estimated} = \sum_{k=0}^{L-1} \left[plr_0 + \left(\prod_{j=1}^{k} (1 - plr_j) * plr_k \right) \right] \tag{3.3}$$

The estimated packet loss ratio given in (3.1) causes delay due to TCP retransmissions. The effect of packet losses at the application layer depends on the video codec type. If a SVC video is sent, layer dependency also affects the packet losses belonging to the enhancement layer. With SVC coded video, if a packet belonging to layer n is lost, other packets belonging to a higher layer than n cannot be decodable even if they arrived at an appropriate time. On the other hand, there is no inter-dependency between the layers of the MDC coded video.

We give an example to show the effects of the packet loss at the application layer and quality received with the multi-layer codec video. Suppose a video is encoded with one base and three enhancement layers using SVC. The same video is alternatively encoded with four layers using MDC. For instance, to achieve the quality provided by the three layers, the base and the first two enhancement layers (i.e. L_1 , L_2) should be received by the client using SVC coded video. On the other hand, MDC codec client just needs to receive any three layers to play the video with the same quality. Table 3.1 demonstrates the received video quality for the given scenarios for SVC and MDC clients. L_0 means the base layer for SVC coded video in the table. For both scenarios, the client plays the video with a higher quality with MDC codec than the quality received with SVC codec video.

Although the previous example shows the advantage of using the MDC codec, there is a bitrate overhead introduced by the MDC. A video layer encoded with the MDC has a higher bitrate than the bitrate of the equivalent video layer encoded with the SVC. This increases the loss probability of the packets at the link layer if the network capacity is not adequate. Therefore, there is a trade-off when selecting the optimal video codec for a given end-to-end streaming path capacity. In the next section, we explain the details of the optimization model that selects the optimal codec type according to the network conditions.

	Received Layers				# of Layers Played	
Scenario	L0	L1	L2	L3	SVC	MDC
1	+	-	+	-	1	2
2	+	+	0	+	2	3

Table 3.1. Example of SVC and MDC codec behaviors

3.1.5 Optimization Model for Selection of Video Codec Type and Optimal Number of Layers

The optimization model aims to maximize video quality under the constraint of network capacity. For this purpose, the controller runs an optimization model that maximizes the received bitrate while minimizing the expected packet loss ratio. In order to be able to determine the optimal number of layers to be served to the client, the controller needs to calculate the packet loss ratio by considering codec types. When a new client joins the system, the controller runs the algorithm given in **Algorithm 1**.

The algorithm determines the total packet loss ratio for all possible layers that can be served to the client for both codec types. The available bandwidth of the paths selecting for transferring the packets of L^{th} layer is calculated between the 2^{rd} and 6^{th} lines of the algorithm. The path having maximum available bandwidth is selected (2^{th} line) for each time determining the streaming path for the next layer (3^{th} line). Note that the path selection here is logical and the controller does not send the related forwarding rules to the switches for the selected paths yet.

When a path is selected for transferring the packets of i^{th} layer, the bitrate of the layer is reduced from the available bandwidth value of the path in line 4. Based on the selected paths for each layer, $plr_{estimated}$ values are calculated according to the codec type between 5^{th} and 9^{th} lines of the algorithm. The $plr^{total}_{i,c_{type}}$ variable calculated in line 10 represents the total packet loss ratio for the client being server i layers encoded with c_{type} of codec type.

After path assignment and calculating packet loss ratio values, the controller runs the optimization model. The objectives of the optimization model are given in (3.4) and (3.5) The model minimizes the estimated packet loss ratio calculated while maximizing the number of video layers, where the video is encoded with K layers.

Algorithm 1: *Packet Loss Ratio Estimation* Algorithm.

Input: define SVC & MDC
K : Total number of layer L
abw_i^p: The available bandwidth of the path *p* selecting for transferring packets of i^{th} layer.
br_i: the bitrate of i^{th} layer

Output: $plr_{i,c_{type}}^{total}$: packet loss ratio

```
foreach L = 0 to K do
    abw_pm ← abw of path having max abw
    abw_i^p ← abw_pm
    abw_pm ← abw_pm - br_i
    if c_type = MDC then
        calculate plr_estimated as given in (3.2)
    else
        calculate plr_estimated as given in (3.3)
    end
    plr_{i,c_type}^{total} = plr_estimated
end
```

$$\min plr_{L,c_{type}}^{total} \tag{3.4}$$

$$\max L \tag{3.5}$$

s.t.

$$L \leq K \tag{3.6}$$

We solve the proposed multi-objective optimization model by exhaustive searching. For this purpose, for a given video codec type and the number of video layers, the packet loss ratio is calculated. This produces $2 \times K$ solutions. The optimal solution among them is founded by searching the solution that minimizes the distance to the utopia point. The utopia point is the point where $plr_{L,c_{type}}^{total} = 0$ and $L = K$.

Typically, the number of layers used for multi-layered video is between 2 and 5. Therefore, the complexity of the model is low due to the limited number of video layers. The complexity of optimization model for n clients conclusion from $2 \times K \times n$, which lead to $O(n)$. Also, the complexity of looking for maximum available bandwidth and port assignment is $O(nlogn)$. Therefore, the time complexity of the proposed solution is $O(n) + O(nlogn) = O(nlogn)$.

The output of the optimization which gives the video codec type and the optimal number of video layers forwarded by the controller to the newly joined client. After receiving this information, the client starts to download video segments. The controller also sends the related flow's routing information to the switches. Although the controller signals the number of layers that the client should request at the beginning of the streaming session, the clients may change the number of layers based on the download speed and buffer occupancy in order to achieve a better quality of experience. When download speed is decreased, the client experience outage then reduces the number of layers suggested by the controller to avoid re-buffering.

3.1.6 Client Implementation

In the proposed system, the clients may receive SVC or MDC encoded video based on the information sent by the controller. Since the layer dependency and decoding procedure of SVC and MDC encoded video differs from each other, the client software runs differently according to the video codec type. When SVC encoded video is downloaded, the client must wait for the arrival of base layers in order to play the video if the buffer is empty. On the other hand, when the MDC encoded video is downloaded, the client must wait for the packets from at least one layer if the buffer is empty. For both encoding types, when the client buffer is empty, the playout freezes until the buffer level exceeds a pre-determined threshold.

Since the controller can assign different paths to each video layer, clients do not have to wait for video packets of one layer to arrive in order to send a request for downloading next layer packets. Thus, the client requests decided video layers concurrently. We follow the formula given in (Cetinkaya et al., 2015) for measuring available bandwidth. Suppose the client request n video layers from the server in the next request period. Let t_d represents the total time to download all requested layers. The t_d and available bandwidth are calculated based on a formula given in (3.7) and (3.8), respectively. In the formulas, L_i represents the i^{th} video layer. t_k is the receiving time of the last packet of k^{th} layer, $t_{request}$ is the time of the request, and br_k is the bitrate of the k^{th} layer.

$$t_d = \max(t_{L_0}, t_{L_1}, ..., t_{L_n}) - t_{request} \tag{3.7}$$

$$Estimated_{abw} = \frac{\sum_{i=0}^{n} br_{L_i}}{t_d} \tag{3.8}$$

3.1.7 Testbed and Experimental Setup

The sample topology used in the performance evaluation of the proposed system is illustrated in Figure. 3.3. The topology, emulated using Mininet (Lantz et al., 2010), consists of the $OpenFlow$ enabled switches, video clients and video server. The controller logic described in $Section$ 3.1.3 is implemented on Floodlight[1]. In the network topology, there are 10 clients have the capability of playing both MDC and SVC video. The arrival intervals of the clients are determined by the Poisson distribution having an average value of 5 seconds. The video server has both the SVC and the MDC encoded version of the same video. While the SVC video has one base and three enhancement layers, the MDC video has four description layers.

3.1.8 Experimental Results

To evaluate the performance of the proposed system, we measure the QoE parameters such as received video quality, the average received bitrate and the outages during the video playback. We also perform the best effort routing approach and maximum available bandwidth approach under the same set of parameters in order to compare the results obtained with the proposed system. The simulations are repeated 5 times and the average values of the results are given in Figures and Tables.

Figure. 3.4, illustrates the video playback quality in terms of the percentage per received video layers. In the simulations where best effort and maximum available bandwidth approach are tested with a combination of SVC and MDC clients, half of the clients play SVC coded video while the other half of the clients play MDC coded video. Figure. 3.4 shows that the clients in the proposed system play video at a higher quality than the clients in the other approaches.

In order to give the performance improvement provided by the proposed optimization model, we run simulations where both SVC and MDC clients receive video and show the comparative results in Figure. 3.5 and Table 3.2 for different approaches. Figure. 3.5 shows the average bitrate received by the clients over time. While the clients in the maximum available bandwidth and best effort approach have similar bitrate values, the clients in the proposed system received better bitrates.

[1] http://projectfloodlight.org

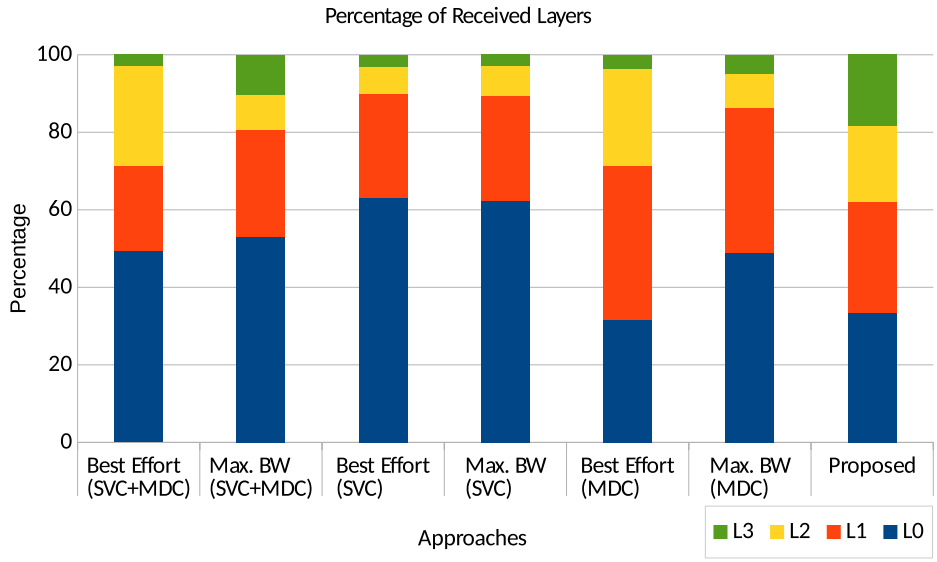


Figure 3.4. Video playback quality .

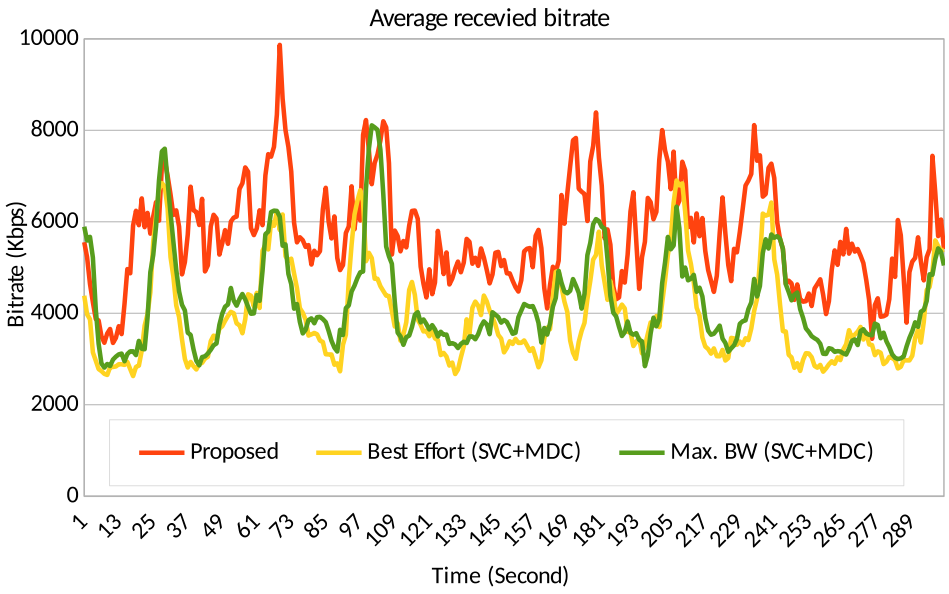


Figure 3.5. Average received bitrate .

When the links are congested and bandwidth is not enough to send the video data, decreasing download speed causes to drain client buffers, therefore video playback pauses. In Table 3.2, total outage duration in seconds and the number of outages observed in the three approaches are given. It is obvious that the proposed system in both total outage time and also in the number of the outage has a lower number. The reason for this is that the optimization algorithms in the proposed system, dynamically change the paths in order to increase bandwidth utilization.

	Best effort	Max. BW	Proposed
Duration (sec)	27	26	16
# of outage	12	12	8

Table 3.2. Outage duration's.

3.1.9 Summary

In this section, we proposed an SDN assisted video streaming architecture that utilizing network knowledge and path assignment exibility of SDN technology. When a client joins the system, the controller determines the video codec type and the optimal number of layers that should be sent to the client. The controller signals the selected video codec type and layer information to the client. The client implementation developed for the proposed architecture can receive the initial command from the controller and able to decode SVC and MDC coded video.

The determination of video codec type and number of video layers to be sent is based on the proposed optimization model in this study. The model takes the bitrate of video layers for both video codec type and available bandwidth information of the end-to-end path between the server and the client as input parameters. The aim of the optimization model is maximizing the video quality while minimizing the packet loss at the link layers to prevent retransmission and delay. When the network condition changes, switching between the video codec type may be necessary to provide good performance. We consider implementing this in future work. The performance of the proposed system is given by presenting received video quality, average received bitrate and outage duration. The proposed system is compared with 6 different approaches determining streaming paths and sending video codec types based on various strategies.

The simulation results show that the proposed system provides 46% and 36% increase in received bitrate when it is compared with the traditional best effort approach and maximum available approach, respectively. The proposed system also provides a 62% decrease in duration of outages and a 50% decrease in the number of outages when it is compared to both approaches.

Note: The results of this study published in (Kalan et al., 2017) *2017 8th international conference on the Network of the Future (NOF)*, and (Kalan et al., 2018a) *26 th IEEE Signal Processing and Communication Applications (IEEE SIU)*.

3.2 A Layered Adaptive Video Streaming Architecture: Codec, Path and Quality Selection by Utilizing SDN

In the system architecture in the previous work, when a client starts the video streaming application, the controller selects the video codec type, the optimal number of video layers and streaming paths of the video layers. However, the controller does not send recommendations related to the codec types of the video after streaming starts, even though the network condition is changed. In this current paper, we extend this architecture by proposing a heuristic algorithm that determines the optimal number of layers for each client and assigns paths for the determined layers. The algorithm is run by the controller periodically and streaming paths are re-routed by considering video codec type, packet loss probability and available bandwidth of the paths. In addition to this heuristic algorithm, we propose a DASH client architecture in which the clients communicate with the controller and utilizes the recommendation messages received from it. The clients decide the number of video layers that will be requested by taking buffer fullness and throughput estimation on the client side into account as well as the optimal number of layers recommended by the controller. The overview and design of the proposed layer-based video streaming system is given in the next section.

3.2.1 Codec and Quality Selection for Layered Video Streaming

Depends on client or network available resources and adaptation algorithm a client can adapt to different video quality during streaming time. When the clients request additional layers for which a path was not selected before, the controller should determine the paths for these newly requested layers. On the other hand, changing network conditions change the optimal number of video layers that can be transferred to the clients under the constraint of actual current available bandwidth values. In this work, we enhanced our previous work by adding a new module to the controller that provides to determine the optimal number of video layers due to changing network conditions. This module is called *Optimal Layer and Path Selection (OLS)* and the controller runs a heuristic algorithm within this module. The controller periodically measures the traffic amount on each streaming paths and if the traffic pattern is changed, it runs the heuristic algorithm in order to determine the number of layers to be requested by the clients with respect to adapt the current network conditions. The main idea behind the OLS is that the client could not request video with a higher bitrate than proposed by the controller and this approach provide to eliminate the client's greedy behavior. However, clients are free to request lower bitrate in the case of buffer drain and estimating re-buffering.

3.2.2 Optimal Layer and Path Selection (OLS) Algorithm

When a client starts the video streaming application, it requests the segments of the video encoded with the video codec type sent by the controller. The controller also determines the optimal number of layers at the beginning of the video streaming. However, since network conditions are dynamic due to cross traffic, fluctuations in traffic can cause quality switching by the clients. Hence, the number of layers that are determined at the beginning of the stream may not be optimal any longer. To align with network dynamism and determine the up-to-date optimal number of video layers, the controller periodically checks the changes in traffic amount on the paths and runs the OLS algorithm when the network condition considerably changed. The controller measures each path traffic and runs the OLS algorithm if the measurement result is above a cretin level of the pre-defined threshold value. This threshold can be set by the network operator. The bitrate of the minimum representation could be a good alternative for this threshold since the changes in traffic amount higher than this bitrate values may cause considerable quality changes in the received video.

The purpose of the algorithm is to determine the number of video layers for all clients in the system and the streaming paths for each layer that is transferred to the clients. The OLS algorithm determines the streaming paths and the number of video layers by taking into account the available bandwidths of the end-to-end paths, bitrate of video layers, and the total number of the layers transferred through each path. The algorithm works in two phases. In the first phase, the algorithm allocates paths for the packets of the first layer for each client concerning ensure that each client receives the video at least with minimum quality. For SVC, the first layer is the base layer due to the layer dependency rules, while an arbitrary layer can be selected as the first layer for the MDC coded video. In each iteration, the algorithm assigns a path with a maximum available bandwidth for each client and updates path information.

In the second phase, streaming paths for additional layers are assigned. The path allocation for the extra layer is done by considering the likelihood of packet losses. When a packet is lost at the link layer, due to TCP retransmission delay, the retransmitted packets may arrive in the client after their playout time is over, which can not be played. Unnecessary retransmissions increase the probability of packet losses as well as wasting network resources. Hence in the second phase, a path for an additional layer is assigned only if assigning a path for a new layer does not cause network congestion and unacceptable value for packet loss possibility.

The packet loss ratio is relevant to the available bandwidth of the streaming path, the number of video layers, and the bitrate of each layer that will be transferred through that path. Suppose abw represents the available bandwidth of a path p and n denotes the number of video layers is transferred through the path p. The available bandwidth of the path is the remaining bandwidth when the traffic of UDP and non-DASH TCP flows is subtracting from the original capacity. TCP fairness ensures that the bandwidth portion that can be used for transferring each layer roughly equals to abw/n on the path p. The controller calculates the expected packet loss ratio for the given video layer and path by using the formula given in (3.9), where plr_L and $bitrate_L$ represents the packet loss rate and bitrate of video layer L, respectively.

$$plr_L = 1 - \begin{cases} 1 & if \frac{abw}{n} \geq bitrate_L \\ \frac{abw}{n * bitrate_L} & otherwise \end{cases} \tag{3.9}$$

The packet loss rate is given in (3.9) refers to the loss probability of an arbitrary packet belonging to a TCP flow. When we consider the video packet losses at the application layer due to retransmission delay, we should also consider the layer dependencies, especially for SVC coded video. If a SVC layer packet is lost, then the upper enhancement layers of this lost packet can not be decoded on the client-side, hence they are also treated as lost packets. Therefore, in the calculation of packet loss probability, codec type should also be considered. The packet loss probability also depends on the number of flows sharing the same streaming paths and the bitrate of the video layers since packets losses are highly related to the capacity of the network and the traffic amount transferred over it. The formula for the calculation of packet loss based on codec type is given in (3.10). This formula is the modified version of the formula given in the previous study (Kalan et al., 2017). The modification should be necessary since the formula given in (Kalan et al., 2017) only considers the video layers transferred over the same paths, however, each flow affects other flows transferring over the same path as well as it affects the loss probability of other related layers transferred over different streaming paths. Consider a scenario that is illustrated in Figure. 3.6, which shows the reason why we should take into account the layers that are transferred over other paths. $L_i.$, $L_{i+1}.$ and $L_j.$ represent $i.$, $i+1.$ layers of video sent to the client 1 and $j.$ layer of the video sent to the client 2, respectively. Forwarding client' 2 flow over path 2 may lead to packet loss in the flows of client 1, $i.$ layer and packet loss in layer i has also effect on layer $i+1$ due to layer dependency. Hence, when packet loss in $h1\_SVCL_i$ has exceeded the threshold, it also affects the same client's upper layers and it should be considered in the calculation of packet loss probability.

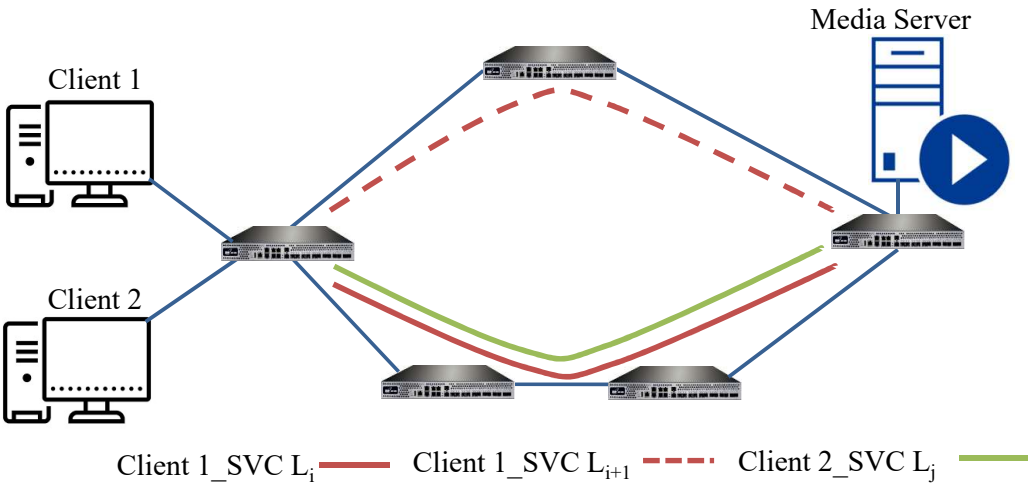


Figure 3.6. SDN controller and system architecture .

$$plr^{L}_{estimated} = \begin{cases} plr_L & if codectype = MDC \\ \sum_{k=0}^{L} \left\{ plr_0 + \left[\prod_{j=0}^{k-1} (1 - plr_j) * plr_k) \right] \right\} & if codectype = SVC \end{cases} \quad (3.10)$$

The OLS algorithm is given in **Algorithm 2**. In the first phase of the algorithm, where the paths for the first layers of each client are assigned, it is given between $2nd$ and $5th$ lines of the algorithm. In the second phase of the algorithm, the controller assigns paths for each additional layer for each client. The paths for additional layers are determined in sequence, in other words, a path is assigned for an additional layer after the paths for the same number of layers are assigned for all clients in the system. The path assignments are done unless the estimated packet loss ratios are under a certain threshold. Let plr^{p}_{L} represents the value obtained by using the formula (3.9) where p is the path that the packets of layer L are transferred through and thr_L represents the threshold determined for layer L.

When the path p is assigned for a new layer L, the controller estimates plr^{p}_{L} for the transferred packets through p. In addition, it estimates the effect of the newly assigned layer on all layers routing via the same path. When the number of layers in a path has been changed, the plr^{p}_{L} value should be controlled for all layers previously assigned to that p. A path assignment is done for new enhancement layer L if the new plr^{p}_{L} value is less than the defined threshold value (thr_L), otherwise it will be discarded. This scenario is repeated for all layers of all clients within the for loop given in the 11th and 31th lines of the algorithm.

Algorithm 2: *Optimal Layer Selection* Algorithm

Input: $P_{\text{Max-abw}}$:path with maximum available bandwidth
thr_{L}: threshold level determined for layer L
thr_{b}: threshold level determined for buffer
$base_{\text{lp}}$: selected path for base layer

Output: Optimal number of selected video layers

let plr^{p}_{L} denote the estimated packet loss ratio ($plr^{L}_{estimated}$) for given layer L and path p

foreach *Client* **do**
  $Base_{\text{lp}} \leftarrow P_{\text{Max-abw}}$
  $P_{\text{Max-abw}} \leftarrow P_{\text{Max-abw}}$ - bitrate of base layer
end
foreach *Layer* **do**
  foreach *client* **do**
    $P_{\text{Max-abw}} \leftarrow P_{\text{Max-abw}}$ - bitrate of layer L
    Calculate(plr^{p}_{L})
    if *(*$plr^{p}_{\text{L}} <= thr_{\text{L}}$*)* **then**
      foreach *layer* i *transferred via path* p **do**
        if *(*$plr^{p}_{\text{L}} <= thr_{\text{i}}$*)* **then**
          foreach *SVC Client in different path with* $L^{'} > L$ **do**
            Calculate($plr^{p^{'}}_{\text{L+1}}$)
            if *(*$plr^{p^{'}}_{\text{L}'} <= thr_{\text{L}'}$*)* **then**
              Continue;
            else
              remove assigned path
              Break
            end
          end
        else
          remove assigned path
          Break
        end
      end
    else
      remove assigned path
      Continue
    end
  end
end

The controller runs the OLS algorithm periodically, and when the network condition is changed, i.e. a new flow arrived or flow is terminated. The controller signals clients and switches with updated information according to the output of the algorithm. Typically, when a video is encoded with a layered coding, the number of layers is between 2 and 5 and this number can be considered as a constant. Therefore, the same as the $Algorithm$ 1, the time complexity of the proposed algorithm- for n clients - is $C * O(nlogn)$.

3.2.3 SDN-Assisted Video Client Implementation

The objective of adaptive video streaming is to maximize video quality by downloading video segments with higher bitrate while minimizing the outage duration. In our system, to minimize startup delay, the newly joined client downloads the first several segments from the lowest bitrate at the beginning of the streaming, which is a typical approach. When the clients are not assisted by the controller about the selection of the successive requested segment, due to the client-driven nature of HTTP adaptive video streaming, they decide for adapting to the next segment of the video based on the observed throughput values and/or buffer occupancy. If the client rate adaptation is run based on throughput, the client simply selects the highest representation where its bitrate is smaller than the measured bitrate. The buffer based rate adaptation algorithms take into consideration of the buffer fullness.

As explained in the previous section, in our SDN-assisted system, the controller sends a recommendation message for the number of layers that can be requested by the clients as an output of the OLS algorithm. Let L_{rec} represent the number of layers that are recommended by the controller. When L_{rec} value is received from the controller, one option for clients could be to start requesting the L_{rec} layers until the controller recommends another layer, i.e. quality level. However, the clients should also consider the current buffer level, which is one of the crucial internal parameters in order to prevent outage duration. When buffer fullness is considered as a part of the rate adaptation algorithm, clients adapt quality by considering minimum bitrate of the recommended layer from the controller on one hand, and their buffer level on the other hand. Let L_{n} represents the number of layers that are determined by the client, which equals to the possibly highest quality that can be received under the constraint of buffer occupancy. The recommended layer number by the controller may be higher or lesser than L_{n}. When a new recommendation is received from the controller, the client requests L_{rec} unless the buffer level is under a certain threshold and L_{rec} is higher than L_{n}. If the buffer fullness is under that threshold, the client sends a request for receiving L_{n} layers. Hence, the number of

layers requested by a client is always between the values given in (3.11).

$$min(L_n, L_{rec}) <= number\,of\,layers\,to\,be\,requested\,next <= L_{rec} \quad (3.11)$$

3.2.4 Testbed and Topology Setup

We use a Mininet emulator to set up the SDN environment for testing the performance of the proposed approach. Mininet provides an efficient platform for constructing SDN topologies, implementing and testing the SDN applications (Lantz et al., 2010). The controller modules are built on top of Floodlight software and communicate with underlying devices through $OpenFlow$. We run our experiments using the real world topology, known as "*Compuserve*", from the *Internet Topology Zoo* (Topology-Zoo, 2017). Figure. 3.7 illustrates the network topology which is used during the simulations. *Elephants Dream-II* (ED-II, 2017) is used as a streaming video. The video file consists of 327 segments with a duration of 2 seconds. On the client's side, total buffer length and threshold level are set to 24 and 8 seconds of video, respectively. There are 10 video clients are capable of streaming both SVC and MDC video. The video server also provides the same video with both SVC and MDC format. The SVC video has one base and two enhancement layers, while MDC video has three layers. Table 3.3 gives the bitrate of the layers for both codec type. The bitrates of the SVC enhancement layers are given regarding only the related layer. The bitrate of a SVC enhancement layers can also be represented cumulatively by considering layer dependencies. In that case, for example, the cumulative bitrate of the$L1$ equals to the sum of bitrate values of L0 and $L1$ given in Table 3.3.

During the simulation we used four different network scenarios and applied *Poisson distribution* to generate the bandwidth of the links with mean values of λ=8 Mbps, λ=10 Mbps, λ=12 Mbps and λ=15 Mbps. Total available bandwidth values between source and destination based on defined mean value (λ) are illustrated in Table 3.4. As shown, in order to analyze the behavior of algorithms, clients were placed behind bottlenecked links. We considered two different packet loss thresholds for each layer, which are given in Table 3.5. Tuning these values gives a bit of flexibility to the proposed algorithm to adapt appropriate bitrate. The estimated packet loss ratio should not exceed the threshold value in the same path layers when new client's traffic assigned to the path. Despite increasing threshold values increase received bitrate, it may result in increasing packet latency and video stalls.

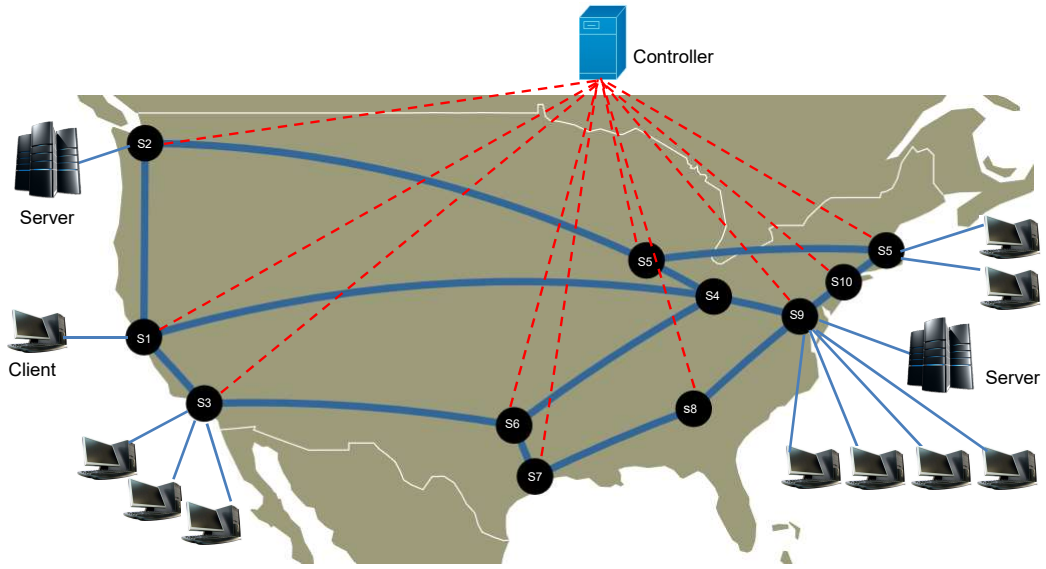


Figure 3.7. Compuserve network topology

Name	**L0**	**L1**	**L2**	**Total Size**
SVC Bitrate	2430	987	1750	5167
MDC Bitrate	2430	2430	2430	7290

Table 3.3. Elephants Dream(ED-II) representations.

Mean Value	**λ=8 Mbps**	**λ=10 Mbps**	**λ=12 Mbps**	**λ=15 Mbps**
Total Bandwidth	28.4 Mbps	32.3 Mbps	41.7 Mbps	58.5 Mbps

Table 3.4. Total network bandwidth.

Threshold	**$\mathbf{Thr}_1$**	**$\mathbf{Thr}_2$**
SVC Layer 0	0.05	0.10
SVC Layer 1	0.10	0.20
SVC Layer 2	0.20	0.30
MDC Layers	0.10	0.20

Table 3.5. Packet loss threshold levels for different layers and Codecs.

While SVC has different threshold values for each layer, MDC only has one value for each threshold level. Different thresholds for each SVC layer was defined due to the layer dependency. During the test, the simulations were repeated 10 times for each threshold level. All test results represented in Figures and tables represent averaged values.

3.2.5 Evaluation Results

In order to evaluate the performance of the proposed solution, we measured the following QoE metrics: i) average received bitrate, ii) re-buffering duration, iii) a number of the video segments which received from each layer, and iv) the numbers of quality switches. Throughput Base Adaptation (TBA) and Buffer Base Adaptation (BBA) (Huang et al., 2014) with the same set of configurations are applied as a benchmark for comparing the results of the proposed algorithm. In the TBA algorithm, clients measure network throughput while downloading video segments and adapt to the next suitable segment regarding the current network bitrate. In the BBA algorithm, clients control buffer level and then request a segment from suitable quality based on buffer fullness proportionally so that the clients request the highest possible quality when the buffer fullness is high and request the lowest quality when its buffer is almost empty. When TBA or BBA approach is used, the controller forwards the client's requested layers thought the links with maximum available bandwidth without running the OLS algorithm.

The video bitrate received by the clients as a function of the network capacity is given in Figure. 3.8. The clients using TBA and BBA algorithms have higher bitrate than the clients in the proposed approach in a congested network. However, as it is going to be shown later, the clients in TBA and BBA approaches have a higher re-buffering duration which is unacceptable. Noteworthy, displaying video with less duration of video stalls is so much preferable than the small quality degradation. Bitrate adaptation requires a tread-off between enhancing video quality and reducing the probability of freezes video during display time (**?**). While requesting video from the lowest bitrate results in poor video quality, requesting video from the upper layer increase the probability of re-buffering. The client-driven nature of the TBA and BBA algorithms only relies on greedy behavior. The main advantage of the OLS algorithm is making a trade-off between bitrate and delay. Therefore, in the long term, it outperforms than the other algorithms. When comparing with the TBA, the proposed approach based on the OLS algorithm performs better for both threshold levels. More precisely, the performance improvement provided by the proposed approach becomes more clear while network bandwidth is increased. Greedy behavior of the TBA algorithm encourages clients to maximize bandwidth utilization by requesting the highest quality possible, however, this may lead to congestion and as a result, sudden decrease in quality level. Thus, clients rapidly switch between different representations which decreases QoE. In the last scenario ($\lambda = 15$ Mbps), the clients in both TBA and OLS approaches obtained the same bitrate, but again the OLS has a less re-buffering duration, which shows the overall performance

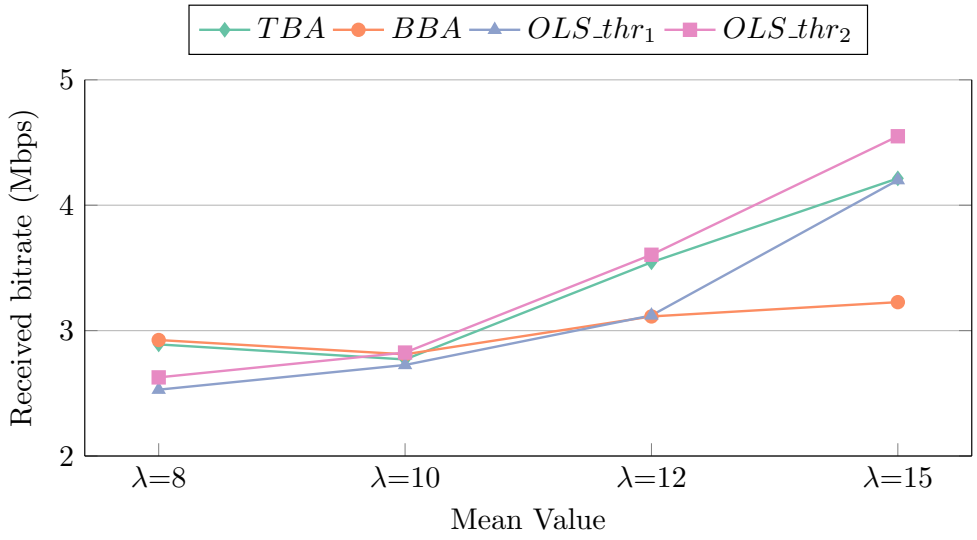


Figure 3.8. Average received bitrate.

of the proposed algorithm is better than that of the other approaches.

The average received bitrates as a function of time is shown in Figure. 3.9. It is clear that, in all assigned bandwidth, the OLS algorithm has better performance in terms of received bits per second. Impact of the proposed algorithm more evident in Figure. 3.9(a) where network links have less bandwidth ($\lambda = 8$ Mbps). By increasing network bandwidth clients in all solutions achieves higher bitrate. It is important to highlight that, the proposed OLS_thr_2 algorithm provides better performance than others when network capacity is increased, but it is also experienced more re-buffering as shown in 3.6, Table 3.7. By looking at threshold values in the proposed algorithm, it is possible to conduct that increasing threshold value can improve the bitrates of the received segments, but it also causes more delay and results in extra re-buffering. While other approaches have a significant reduction in re-buffering duration, the OLS_thr_2 suffers from the same re-buffering time.

Receiving more video segments from higher layers provides to play the video with better quality. Figure. 3.10 shows the received number of segments from each video layer during the simulations. In the first scenario, the BBA algorithms received more segments from enhancement layer 1 while the client with TBA and OLS_{thr1} approach received more video segments from the base layer. At the first glance, it may derive that the BBA algorithm outperforms while the network has less bandwidth, but it is worth underlying that they have an undesirable delay while trying to download video from higher bitrate which causes to unacceptable re-buffering.

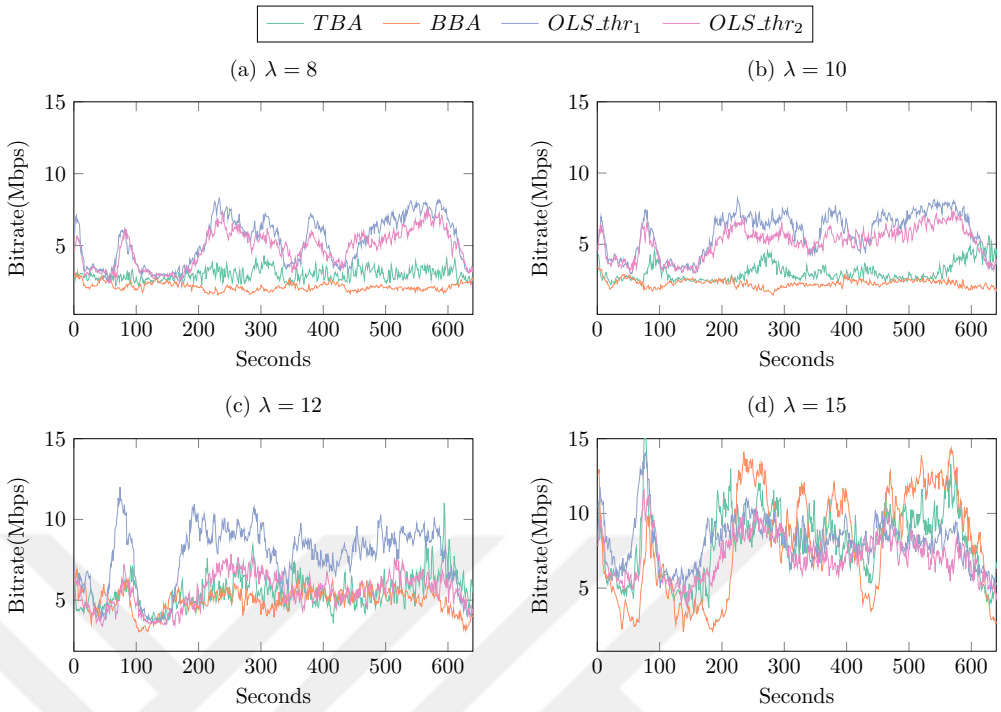


Figure 3.9. Average received bitrate as a function of time.

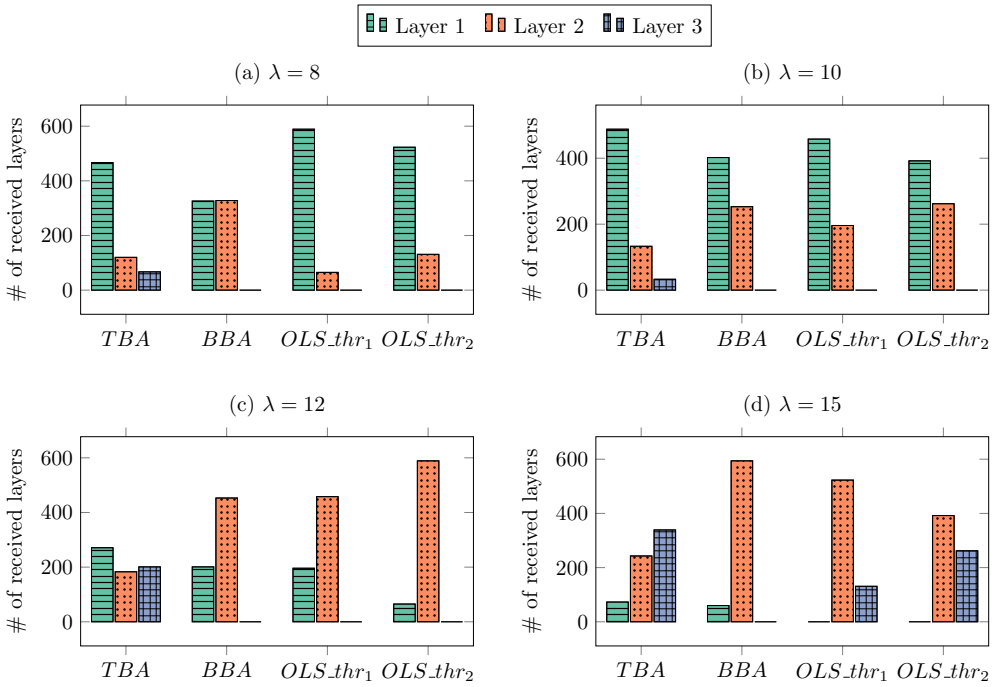


Figure 3.10. Average number of received layers per quality levels.

In the other scenarios where the link's bandwidths are increased, we observed that the OLS algorithm provides a client to adapt quality wiser and receive more segments form higher layers. This result shows that the clients with the OLS approach receive a minimum number of segments from the base layer among all approaches only when the network has enough capacity to transfer packets from higher layers.

When the network links are congested, the client adapts to lower bitrate in order to avoid outages. As shown in Table 3.6, Table 3.7,when ($\lambda = 8$ Mbps), the proposed algorithm has less re-buffering. But, the TBA and BBA algorithms have an unacceptable value in terms of re-buffering in both forms of duration and frequently repeated. It is because the client by itself has less information about the underlying network. As a result, the greedy behavior of the client in a short period may not ensure appropriate bitrate adaption. On the contrary, since the controller runs the OLS algorithm by using its knowledge about the underlying network, it helps clients to request the highest number of video layers under the constraint of network capacity.

The disadvantage of giving more packet loss threshold for different layers can be absolved in OLS_thr_2. By increasing network capacity, the re-buffering duration reduces significantly in all approaches except OLS_thr_2. Increasing packet loss threshold value gives more flexibility to the client to request video segments from the layer with the higher bitrate. Hence, it can impact dramatically network traffic and leads to more delay and re-buffering.

Table 3.8 shows the number of quality switches during streaming. Less number of quality switches means that the client experiences stable video quality. It is observed that the OLS algorithm provides stable video quality in all scenarios. The main reason is that, it prevents clients to switch higher layers when clients estimate that the network has more bandwidth in a short period of time. In other words, the proposed approaches limit clients' greedy behaviors.

Mean Value	λ**=8 Mbps**	λ**=10 Mbps**	λ**=12 Mbps**	λ**=15 Mbps**
BBA	198	120	66	12
TBA	185	104	49	19
$OLS - thr_1$	51	35	24	9
$OLS - thr_2$	52	37	36	35

Table 3.6. Total re-buffering duration per client (seconds).

Mean Value	λ**=8 Mbps**	λ**=10 Mbps**	λ**=12 Mbps**	λ**=15 Mbps**
BBA	50	39	26	8
TBA	50	29	28	9
$OLS - thr_1$	16	14	11	6
$OLS - thr_2$	20	15	14	15

Table 3.7. Total number of re-buffering per client

Scenario 1(λ=8 Mbps)	TBA	BBA	$OLS - thr_1$	$OLS - thr_2$
+1	21.4	5.5	0	0.3
+2	32.7	9.6	0	0
-1	52.0	0	0	0
-2	1.3	9.0	0	0
Scenario 1(λ=10 Mbps)				
+1	30.3	8.9	0.3	0.4
+2	17.0	0	0	0
-1	44.5	8.3	0	0
-2	2.3	0	0	0
Scenario 3(λ=12 Mbps)				
+1	25.5	5.9	0.7	0.9
+2	38	0	0	0
-1	60.5	5.22	0	0
-2	2.3	0	0	0
Scenario 4(λ=15 Mbps)				
+1	24.3	2.6	0.8	0.6
+2	50	0	0.4	0.8
-1	71	1.7	0	0
-2	2.1	0	0	0

Table 3.8. Number of switches between different representations

In the table, $+(-)x$ shows the increase (decrease) in the video quality and x number of video layers are added (extracted) between the successive requests of the clients. For example, $+2$ represents that the client receives the next segment from two quality level higher than the last downloaded one. In all scenarios, it is clear that the OLS algorithm has the lowest value in terms of the number of increments and decrements. It is because the controller's messages in the OLS

algorithm assists clients to receive video segments from specific layers and restricts client greedy behaviors when the network has plenty of bandwidth in a short period of time. As well, the BBA algorithm avoids the high number of quality switches. On the contrary, in the TBA algorithm, clients have greedy behavior to receive video from upper layers. Hence, when network is congested clients receive a base layer for a short period of time, and then may request video segments from upper layers when network capacity changes. Hence, TBA has a higher number of quality switches when compared to other approaches.

3.3 Conclusion

SDN technology can provide increasing the performance of video streaming applications. It is also possible to achieve higher performance if the characteristics of video stream applications are considered when utilizing SDN technology.

Layered video streaming systems allow clients to obtain various video qualities from one encoded video file. In order to encode the video sequences with a layered codec, SVC or MDC can be used, which have different characteristics. In this paper, we proposed a video streaming system architecture where the SDN controller is aware of video codec types and video layer's bitrates. By taking into account layer dependency constraints of both codecs, estimated packet loss ratios and current network condition into account, the controller dynamically assigns streaming paths for each layer of the videos transferred to all clients. In addition to that, the controller recommends the client the optimal number of layers under the constraint of current network conditions.

We also proposed an SDN assisted $DASH$ client architecture, where clients and SDN controller communicate periodically and the client can be able to interpret the recommendations sent by the controller. The clients utilize these recommendations within the rate adaptation module and decide the requested video segments by considering its own rate adaptation logic, buffer fullness level and recommended the number of layers by the controller. Simulation results show that the proposed architecture provides an increase in received video quality up to 76% and up to 10% decrease in outage duration when it is compared to another approach where the paths with maximum available bandwidths are assigned to the clients.

4. VIRTUAL CACHE PLACEMENT AND MIGRATION

In this section, we present our algorithm for installing virtual cache in a suitable location as close as possible to clients and reduce the total cost of bandwidth usage. Web-caches play an important role in the architecture of a $DASH$ system, which can bring video files near to the clients. In such systems, the placement of the caches affects the performance of the video streaming applications.

In this study, we propose a SAND video streaming architecture utilizing both SDN and NFV concepts, whereby virtual cache instances are created on-demand by considering the live DASH attributes and characteristics. In this article, our contribution focus on three main topics. Firstly we introduce an algorithm for installing an additional cache when the number of clients in the network rapidly increases and the existing cache is not adequate. In the second phase, we revise our algorithm such that there is no cache in the network and we should find the first cache location. Finally, when network conditions and traffic patterns change we extend our study by proposing a new algorithm for cache migration.

4.1 Virtualized Cache Placement in SDN/NFV Assisted SAND Architecture

The performance of video streaming applications influenced when the network conditions are changed because of cross traffic or due to newly joined or recently leaving clients. To cope with this dynamism of network conditions, adaptive HTTP streaming applications have been developed, whereby the clients can change the video quality according to the changing throughput or buffer fullness. Adaptive streaming technologies such as Apple HLS, Microsoft MSS, and Adobe HDS are the ways to deliver multiple bitrates over the HTTP. Adaptive means that players on the client's side automatically detect Internet throughput and adjust to suitable quality in response. Adaptive streaming technologies are a performance management technique that offers lots of potential for OTT toward streaming multimedia over the Internet.

Caches in the form of *HTTP/web-caches* play an important role in the architecture of a $DASH$ system, and they can bring video files nearer to the clients than the original source. In such systems, the placement of the caches affects the performance of the video streaming applications. These virtual caches are defined as DANEs, and using such an approach gives the right level of flexibility and control. *vDANE*s are the virtualized caches with the knowledge of DASH characteristics,

which uses NFV hosted by the servers connected to the switches. We select close to optimal network nodes for virtual cache placement by taking into account characteristics such as the bandwidth of the paths, the places, and the number of the online clients, for a given network graph.

In this section, we select the optimal network nodes for cache placement by taking into account characteristics such as the bandwidth of the paths, the places, and the number of the online clients, for a given network graph. For this purpose, we use a placement algorithm which is an enhanced version of the Pressure algorithm proposed by Clegg (Clegg et al., 2013) which we call $PressureCache$, an implementation of virtualized $vDANEs$. The results show that the proposed approach gives better performance in terms of received quality than random cache placement.

SDN and NFV technologies can be utilized together by using them as components of a video streaming system architecture. Today, the biggest video streaming companies deploy CDN servers to bring video content closer to their customers. The virtualization of CDN servers can be performed by defining CDN functions as $VNFs$ that run on top of the physical elements within $ISPs$ (N. Bouten et al., 2015). Based on the co-operation between $ISPs$ and OTT video service providers, $vCDN$ instances can be created in several points in the network in order to create a streaming service infrastructure that can deliver video packets to the clients efficiently (Ibn-Khedher et al., 2017).

4.1.1 Related Works

In conventional web and CDN caching mechanism, content is proactively pushed to edge servers rely on prior knowledge of content demand and network structure which means that cache location is almost static. Caching content in edge devices in *Information Centric Network* (ICN) is not so efficient while data plane and control plan are implemented inside the forwarding device which affects forwarding device performance. Further, by pushing content to all edges, unnecessarily same content replicated in all edge routers/switches which affect forwarding device performance and consequently reduces overall network performance. On the other hand, avoiding content similarity in edge devices cause more convergence between forwarding devices and has a side effect on network traffic. Synchronizing between forwarding devices and controller keep update forwarding table in edge devices and reduce forwarding table size. Routing decisions in central control can accelerate service delivery and provides more agility to forwarding devices.

Certainly, the cache content popularity such as $PopCache$ (Suksomboon et al., 2013) and $BEACON$ (Xiaoqiang et al., 2016) have a big impact on cache performance, but none of these studies considered the impact of the user access behavior. Consequently, the selection of important nodes towards reducing the number of content acquisition hops (He et al., 2017) (Gomes et al., 2017) is a key factor for improving network performance. In (Tang et al., 2019), clients request content from a connected edge, and edge as a nearest content resources respond if the requested content is available. Otherwise, the request is forwarded until the corresponding resources are returned. Based on the prediction algorithm, the edge node decides whether and where to cache the requested content. Further, the server as a central controller pushes data content to the edge node. Again, the proposed algorithm more focus on prediction user requests and pre-fetching rather than place the minimal number of cache in optimal locations.

The main limitation of the centralized cache approach is that in a large network it not perform well. As a simple solution, a large network can be sliced and cache placed in each domain. Network slicing helps to reduce cache servers load and end-to-end delay, but dividing network into multiple domains as well as cache placement algorithm needs to be considered. To address those challenges, authors in (Badshah et al., 2019) proposed multiple cache servers based on closeness centrality, betweenness centrality, and path-stretch values. However, the proposed model has static nature which means that the cache location is fixed even the network's traffic pattern has changed. Also, clients suffer performance when cache servers are placed behind bottleneck links.

Rely on the concept of SDN technology, this study aim caching content in edge equipment's named *vDANE*. Compared with traditional web caching and CDN caching, a virtual cache is ubiquitous and transparent to applications, cache locations are dynamic and also separation data and control plane easier network traffic management. The main advantage of physical forwarding devices and the logically central controller has easier network monitoring and management. As a consequence, provides a mechanism to place caches in different suitable locations which leads to better performance. The approaches proposed for localization of the virtual functions are developed for conventional Internet applications rather than focusing on video streaming applications' requirements (Clayman et al., 2014), (Moens and Turck, 2014). Optimal cache localization policy which jointly considers network capacity, the load and migration of the virtual $CDNs$ is proposed in (Ibn-Khedher et al., 2017). Although these approaches provide remarkable solutions to virtual cache placement problems, they differ from our study since our implementation is

developed by taking $DASH$ and $SAND$ into account. The authors did not focus on adaptive video streaming, cache selection, and considering the number of clients as well.

In terms of network resource utilization, optimal content placement more depends on the geographical distribution of requests, request intensity, and content popularity. The content placement which discussed in (Claeys et al., 2014) improves bandwidth usage by proactively caching content inside the ISPs network. However, proactive caching does not satisfy the geographical distribution and dynamic nature of the video-on-demand requests. Furthermore, frequently migrating content to reconfigure the proactive placement result in more bandwidth usage. Authors in (Claeys et al., 2016) proposed *Integer Linear Program* (ILP) model to address frequently migration multi-tenant content and server selection. Despite the similarity in terms of problem objectives, both proposed models cannot apply to our scenario as they rely on VoD and long term prediction (e.g., a week) which is not suitable in live streaming. However, they did not launch advanced of the SDN technologies into account.

The advantages and flexibility offered by NFV and SDN technologies have encouraged researchers to utilize them in caching infrastructure for video streaming architectures. Authors in (N. Bouten et al., 2015) provided a hierarchical network architecture consists of three layer structure (core, aggregation, and access layers) for multimedia delivery on NFV. The efficient placement of virtualized network functions is discussed in (Moens and De Turck, 2014), where authors introduced hybrid solution inside providers network with a combination of the legacy network and NFV named $NFV-P$. Regards a good abstraction of the deployment of NFV in an operator's network it seems like that author in both works did not consider QoE parameters. Intelligent migration algorithm for deciding the optimal placement for virtual content delivery functions is presented in OMAC (Ibn-Khedher et al., 2016a), However, this study focused on minimizing the migration cost and not the content retrieval delay or the response time.

The co-operation of CDN companies and ISP network providers can be beneficial for DASH applications (Cetinkaya and Sayit, 2016). While $SAND$ architecture introduces a message exchange between DASH clients and network-aware elements, content providers can leverage SAND in order to conduct the $DASH$ client to specific CDN servers. The impact of CDN and ICN for DASH streaming has been investigated by Zhe et al. in (Li et al., 2012). The authors present a network-friendly DASH architecture under the joint management of both $CDNs$ and $ISPs$

and propose a communication protocol between virtual and original CDN. However, proposed architecture does not aim at determining the location of a cache server or proxy, but determining a copy of requested content.

The approach presented in (Heikkinen et al., 2016), combines the $SAND$ standard with CDN management features for $DASH$ video streaming by implementing content aware-caching and by deploying content and service-aware networks elements. In the proposed model, a monitoring and management server receives metric signals from clients and communicates with CDN servers via PER messages in order to inquire about the status of the servers and network conditions. Nevertheless, CDN server placement and *NFV-SDN* related solutions were not studied in this work. Although there are several $SAND$ architecture utilizing SDN advantages proposed in the literature (Kleinrouweler et al., 2016), (Kalan et al., 2017), our study differs from these studies since virtualization concepts and cache placement problems are not previously considered. We, therefore, focus our study on the behavior of the SAND architecture to place the cache in an optimal location in the *NFV-SDN* networks for streaming video with better QoE.

Virtualized networking such as *NFV-SDN* is good for addressing the high demand for resources, unpredictable traffic patterns, and agility in network configuration. In cache as a service (Georgopoulos et al., 2014), the authors introduce $OpenCache$ architecture which leverage SDN and$OpenFlow$ to provide a control plane that orchestrates caching and steers the content to the clients as close as possible. The objective of $OpenCache$ is more related to what content should be cached, but the proposed method does not consider cache placement, network bandwidth, or traffic patterns.

Leveraging NFV increases network flexibility and reduces cost by having the displacement of network functions over virtual instances deployed on generic servers. Efficient and dynamic placement of *virtual network functions* is discussed in (Clayman et al., 2014). In the proposed solution of (Moens and Turck, 2014), when the network load rises the base load is handled by physical hardware while virtual instances deal with the overflow. However, this study does not elaborate on the specific data and traffic in practice. NFV based multimedia delivery is reported in (N. Bouten et al., 2015). This study more focuses on the optimal allocation of the data center and intention to find the trade-off between distributed and centralized topology and less attention to real multimedia traffic.

More recently, with the facility of *Network Slicing* which utilizes $NFVs$ and virtualized apps, it becomes easier to adapt the performance of the slice to service needs. Aspects such as migrating caches to optimize performance become more flexible. In this section, we propose a SAND video streaming architecture utilizing NFV-SDN concepts, whereby virtual cache instances are created on-demand by considering the live DASH attributes and characteristics. These virtual caches are defined as *DANEs*, and using such an approach gives the right level of flexibility and control. $vDANEs$ are the virtualized caches with the knowledge of $DASH$ characteristics, which uses NFV hosted by the servers connected to the switches. We select close to optimal network nodes for virtual cache placement by taking into account characteristics such as the bandwidth of the paths, the places, and the number of the online clients, for a given network graph. The architecture of the proposed system discussed in the next section.

4.1.2 System Architecture and Design

The system design for $vDANEs$ is presented here, showing the overall architecture and the cache selection and cache placement algorithm. The controller and the caches are setups as *DANEs* in this architecture. We use NFV to create *vDANE* instances which are added based on the output of the modules running on the controller. The architecture is illustrated in Figure 4.1, and shows the layering devised.

The top layer consists of the *Controller*, where *DANE Orchestration* and control plane functions related to the proposed architecture are managed. This layer provides all of the modules and functions for *NFV-SDN* – the *Basic Network Service Functions*, and for managing DANEs – the *DANE Modules*. This top layer also has a REST API that can be used by higher-level Apps and Services. There are three DANE modules within the controller, and these modules determine (i) if a new *vDANE* instance should be added to the network, (ii) where it should be placed, and (iii) to which one of clients it should connect.

The middle layer is represented by an abstraction of *vDANE*s, made up of all the of the NFVs executing DANEs across the network. The bottom layer is the physical infrastructure, which consists of OpenFlow enabled switches and links.

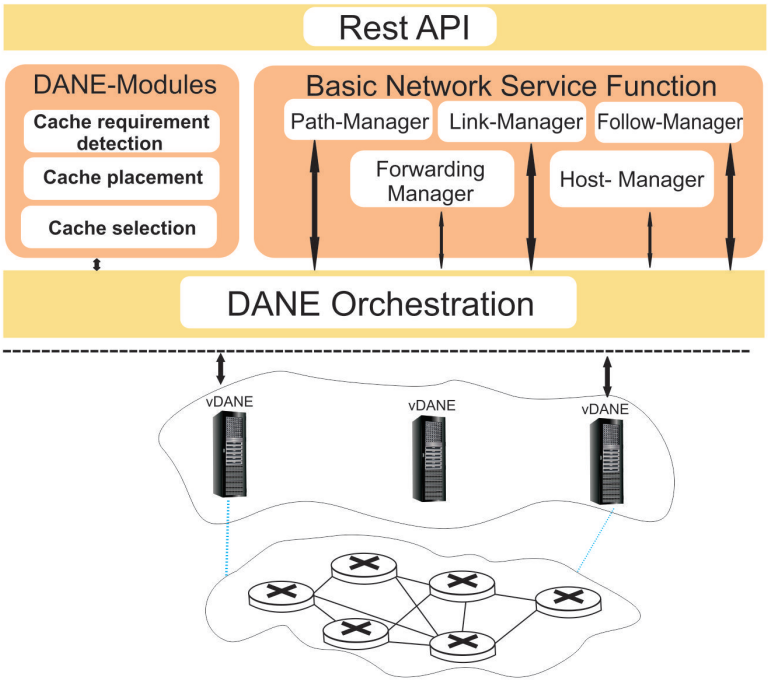


Figure 4.1. vDANE modoules and system architecture view

4.1.3 Cache Selection and Cache Placement

$vDANEs$ are designed to be aware of the $DASH$ system and its application-specific parameters, such as the number and the bitrates of the media representations. $vDANEs$ periodically send the average quality of the received video representation requested by the clients back to the *Controller*. This information is retrieved by the *Cache requirement detection* module running in the controller. The Cache requirement detection module checks if the average quality, received by the clients, is under a specific threshold, called *t*he quality threshold. This threshold can be specified by either the network operator or the video streaming system company. For example, it can be selected as the bitrate of the representation having moderate quality. If the average requested quality is under the quality threshold, the controller then decides to create a new *vDANE* instance. The placement of the new *vDANE* is determined by the *Cache placement* module of the controller. When the Cache requirement detection module triggers, it signals the Cache placement module, and the placement process starts.

For the selection of the location of *vDANE*, we use a placement algorithm which is a modified version of the $Pressure$ algorithm proposed by Clegg et al. (Clegg et al., 2013) and extended in (Tuncer et al., 2015), which we call the *PressureCache* algorithm. This algorithm has never been used for determining cache

placement in the literature. In the $PressureCache$ algorithm a score, called the $PressureCache$ score, is calculated for all potential switches that can be selected as hosting a *vDANE* instance. The $PressureCache$ score is calculated for switch i by using formula (4.1) and (4.2), which are modified versions of the score calculation formula given in (Clegg et al., 2013). $P(i)$ refers to $PressureCache$ score of switch i.

$$avgb_j = \frac{\sum_{c=1}^{k} b_j^c}{k} \tag{4.1}$$

$$P(i) = \frac{\sum_{j'=1, i \neq j}^{n} [max(1, avgb_j - b_{ij'})]}{n} \tag{4.2}$$

The score calculation was modified in this study since the formula in (Clegg et al., 2013) only considers the number of hops between the switches and does not consider the bandwidth values of the paths. However, the available bandwidth of the paths between the clients and the server is one of the main criteria that directly affects the performance of the video streaming application. In addition, the number of clients connected to a switch, which is also a criterion that effects available bandwidth of the paths between clients and the caches, was not considered in the calculation given in (Clegg et al., 2013). Also, the shortest path may not lead to better performance because it rapidly changes to bottleneck points while requests increase. In this paper, we consider the available bandwidth of the paths as well as the number of clients connected to a switch when calculating the score for the switches.

In formula (4.1), b_j^c refers to bandwidth of the path between switch j and the *vDANE* which the clients transferred packets from, where c and k represent the clients and number of the clients connected to switch j, respectively. The formula gives the average bandwidth value for the paths between the clients and their associated $vDANEs$. Note that, if there is no client connected to a switch, the bandwidth values related to that switch's links are not used in the calculation of the formula. In formula (4.2), $b_{ij'}$ is the bandwidth of the path between the switch i and switch j', and n is the number of clients connected to the switch i. Switch j refers to the switch that host the first *vDANE*.

The $PressureCache$ algorithm is given in **Algorithm 3**. The algorithm selects an almost optimal point for *vDANE* placement, by considering the bandwidth of the paths between the $vDANEs$ and the clients and also between potential places

Algorithm 3: $PressureCache$ Algorithm

Input: Set of switches
let *selected* nodes hosting vDANE function;
foreach *switch i not in selected* **do**
    Compute *Pressure Score(i)*;
end
Output: Select the switch with the lowest score

for a new *vDANE* instance and the clients. While the $Pressure$ algorithm proposed in Tuncer (Tuncer et al., 2015) selects more than one switch to connect local manager nodes to by considering the number of hops between switches, the $PressureCache$ algorithm only selects the most optimal switch for *vDANE* placement and makes this selection by using the $PressureCache$ score. After the placement of the new *vDANE* is determined, the controller creates a new *vDANE* instance connected to the specified switch. Creating a new instance of a *vDANE* brings about the further problem of selecting the *vDANE* for each client to connect to. *vDANE* selection for each client is done by the Cache selection module of the Controller. This module compares the maximum bandwidth of the paths between each client and the *vDANE* instances and selects the *vDANE* with maximum available bandwidth.

4.1.4 Testbed and Simulation Setup

In order to evaluate the performance of the $PressureCache$ algorithm with respect to improving the QoE, we applied our experiments over three network topologies and used *Elephants Dream* (ED-II) media dataset (ED-II, 2017) for streaming video and benefit from the network-friendly nature of the SVC which is an extension of the $H.264$ standard. The SVC codec provides one base layer and one or more enhancement layers. The base layer has the lowest quality and can be decoded independently, but enhancement layers depend on the base layer and previous layers to improve video quality. Thus, clients need to receive the base layer and all the enhancement layers to achieve topmost quality. As shown in the representations Table 4.1, there is one base layer (L0) and two enhancement layers (L1 and L2). Each of the layers includes 327 video segments with an equal length of two seconds of video, giving a total video length of 654 seconds. The base layer, $L0$, has an encoding bitrate of 2400 $kbps$, L1 has a bitrate of 3417 $kbps$, and $L2$ has a bitrate of 5167 $kbps$.

For performance evaluation, we use the following QoE metrics: (i) average received bitrate, (ii) outage duration, and (iii) number of the video segments received from each quality layer. The three topologies used are two real-world topologies

Representation	L0	L1	L2
Bitrate	2400	3417	5167

Table 4.1. Elephants Dream (ED-II) data set representations.

Topology	#Nodes	#Links
Custom	8	11
Compuserve	11	14
BellCanada	42	58

Table 4.2. Network topologies specification.

Topology	PressureCache	Random
Custom	4051	3546
Compuserve	4289	3784
BellCanada	4718	3313

Table 4.3. Average received bitrate ($kbps$).

Topology	PressureCache	Random
Custom	223	498
Compuserve	420	432
BellCanada	81	750

Table 4.4. Outage duration ($milliseconds$).

from the *Internet Topology Zoo* (Topology-Zoo, 2017) and a custom one. The number of nodes and links are presented in Table 4.2.

This study aims to find the optimal location for the next incoming *vDANE* while the number of clients increases and leads to a decrease received bitrate on the client-side. As discussed in Section 3, the controller aligns with SAND to estimate a better location for *vDANE* placement. We also choose random feasible places-with maximal and minimum distance from the current vDANE- which have an adequate bandwidth for placement in order to compare with optimal placement and we refer this approach as a random approach. Mininet emulator, which consists of OpenFlow enabled switches is adapted for simulation. DASH clients join to the network based on Poisson distributions. The average interval time and the number of clients are set to 5 and 10, respectively.

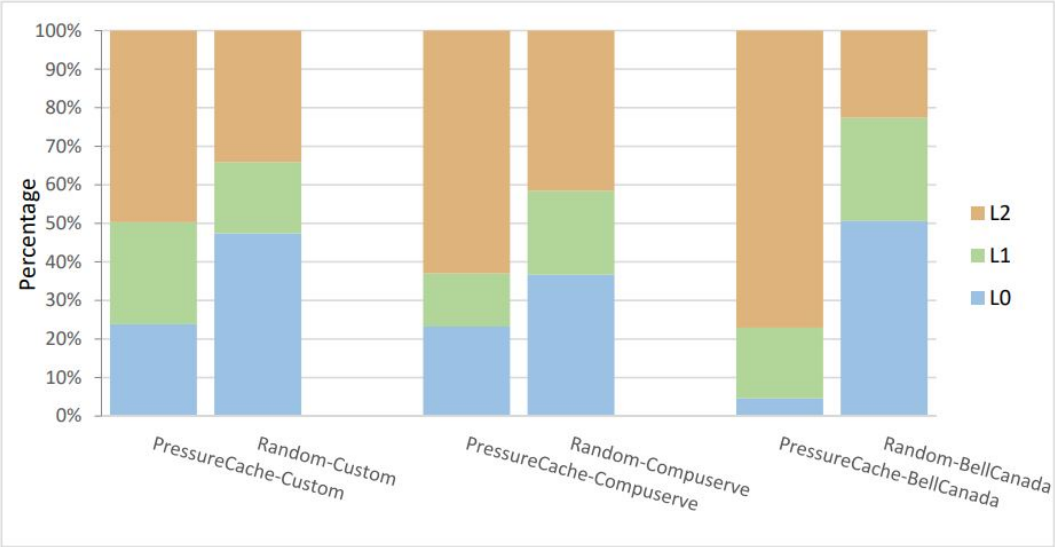


Figure 4.2. Received video layer and playback quality

4.1.5 Performance Evaluation and Experimental Results

In order to investigate the performance of the previously described setup, we conduct a simulation which focuses on the QoE parameters during the streaming. Our experiment includes close to an optimal virtual server that is selected based on the $PressureCache$ algorithm. We compare those results with an experiment using randomly placed servers which are randomly distributed in overlay networks and have the capability to serve $DASH$ clients. In order to derive tolerable and acceptable values, all of the topologies are simulated 5 times and the average value of the results are shown in the tables and figures. Also, instead of reporting individual value for each random server, we just give average value due to the space limitations.

We see in Figure 4.2 that the $PressureCache$ algorithm outperforms the random selection. In this Figure, $L0$ refers to lower video quality, $L1$ the enhancement 1, and $L2$ has higher bitrate or representation quality. A client who receives more video from the $L0$ layer will be displaying video with lower quality. In contrast, receiving more video segments from layer $L2$ is proof that the client is experiencing better quality video. It is clear that clients in the $PressureCache$ setup have received far more video segments of a higher bitrate, while in the random approach, clients have received more segments from layer $L0$, the lower bitrate. We observe that the BellCanada network shows particularly good video quality using the $PressureCache$. Table 4.3 show the average received bitrate in both $PressureCache$ and random selection. The numbers show that in all the three topologies, the $PressureCache$ approach derives a better performance since it has maximum received bitrate.

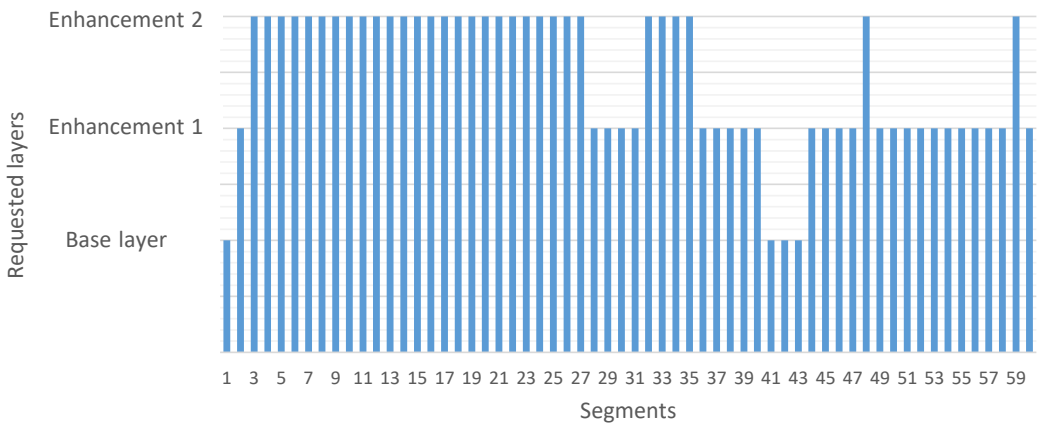


Figure 4.3. Requested video layer per segment

Increasing network traffic can result in reducing download speed and can lead to draining a client's buffer and consequently pausing video playback. In Table 4.4 the total outage duration (while the buffer is empty and the client has to wait) per client is presented for both $PressureCache$ and random selection. It is obvious from the data that the $PressureCache$ has fewer buffer outages in all topologies and hence clients achieve an uninterrupted display. This is because the $PressureCache$ algorithms utilize the network links and use bandwidth in a desirable manner. Forwarding a client's request to the qualified improves network bandwidth utilization and reduce delay.

Figure 4.3 shows the first 60 segments of a single sample of the requested layers during streaming, shown as a sample timeline. We see that the requested layer starts at the Base layer (L0), goes to Enhancement 1 (L1) and then to Enhancement 2 (L2). The requested layer is mostly at L2 but drops to L1, and occasionally L0, depending on network conditions.

During the tests, the simulation is repeated 5 times for each topology. In order to show the average values of the requested layers across each of the runs, we have mapped the values observed, which are from a discrete domain of L0, L1, L2, into values in the continuous domain. We set L0 $\rightarrow$ 0.0, L1 $\rightarrow$ 1.0, and L2 $\rightarrow$ 2.0. Using this approach we can calculate the average values of the mapped requested layers. The averaged time-line trend of mapped values is depicted in Figure 4.4. As shown, at the beginning of the streaming, the client gets a lower bitrate layer in order to fill the buffer and get a fast startup. Then it adaptively downloads video segments from the best layer as possible. In the $PressureCache$ approach, clients request and received most of their video segments from enhancement layers which have a higher bitrate, while in the random case, the clients get a lower bitrate.

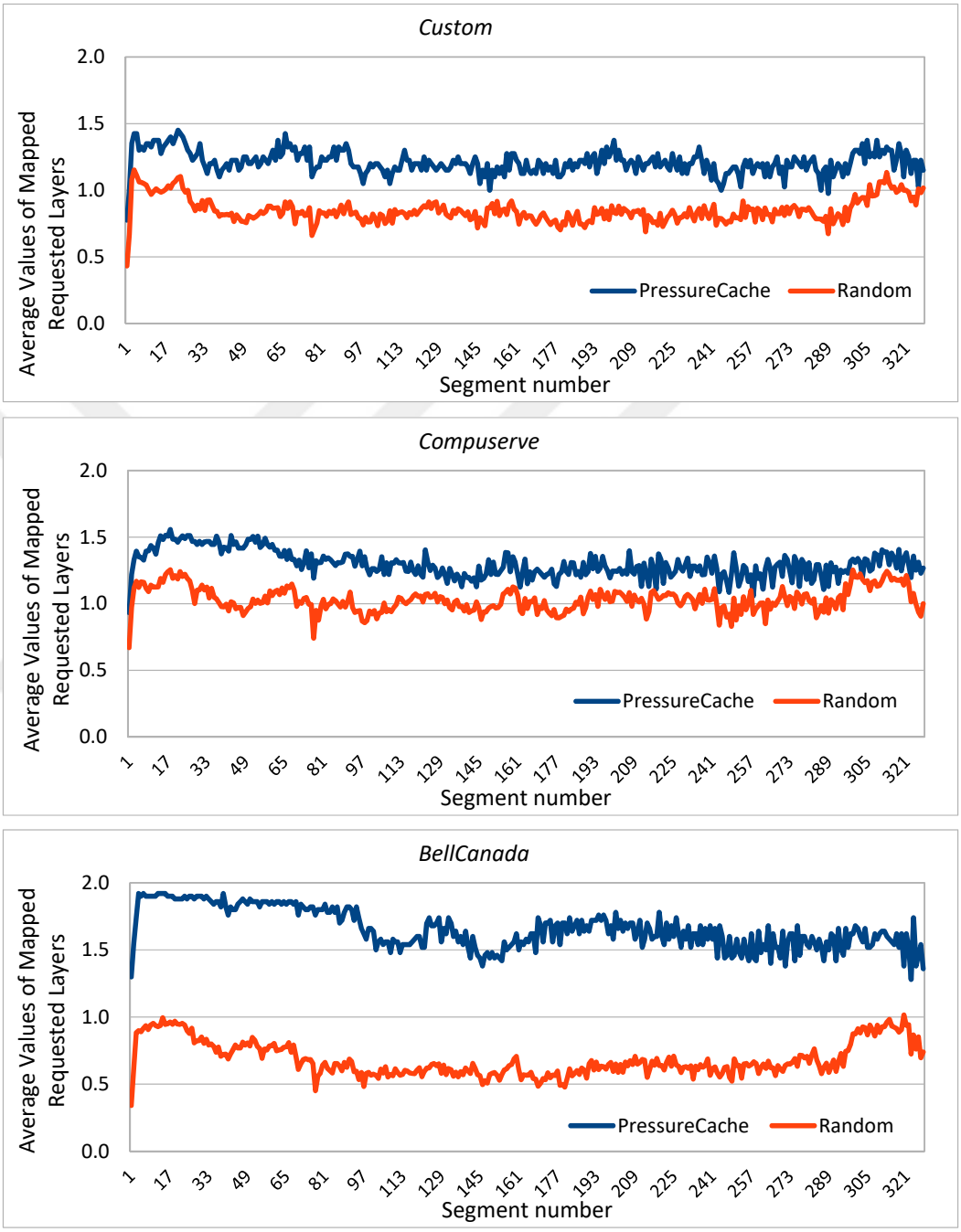


Figure 4.4. Averaged time-line trend of mapped values-requested per segment

4.1.6 Conclusion

In this section, we presented an architecture utilizing *NFV-SDNV* concepts as a proposal for $SAND$, which is a standardization process managed by the $MPEG$ Group. For this purpose, we introduced a concept called *vDANE*, which are virtual web-caches that are aware of $DASH$ characteristics, and we proposed a placement algorithm for those $vDANEs$. In order to show the performance of the proposed architecture, we comparatively tested the proposed algorithm over different custom and real-world topologies. The results show that the proposed algorithm provides up to 42% increase in received video bitrate and up to 90% decrease in outage duration. This means that clients in the $PressureCache$ approach receive higher bitrate and experience better quality, while the average of random *vDANE* placement drives to lower bitrates.

As future works, we plan to enhance our study by improving the algorithm by considering dynamic network topologies. We also plan to develop an approach for the first *vDANE* placement and *vDANE* migration as well. In the beginning, we will address *vDANE* placement and dynamically update *vDANE* location by taking client and network statistics into account.

$Note$: The results of this study proposed in (Clayman et al., 2018) *IEEE International Black Sea Conference on Communications and Networking*, 4-7 June 2018 – Batumi, Georgia

4.2 Optimal Cache Placement And Migration For Improving The Performance Of Virtualized SAND

Nowadays, video streaming over $HTTP$ is one of the most dominant Internet applications, using adaptive video techniques. Rely on this popularity, network-assisted approaches have been proposed and are being standardized in order to provide high QoE for the end-users of such applications. As discussed in the previous section, while caches are one of the main components of the *SAND* architecture, the location and the connectivity of these caches play an important role in the user's QoE. The nature of *SAND* and *DANE* provides a good foundation for software-controlled virtualized $DASH$ environments, and in this paper, we propose a cache location algorithm and a cache migration algorithm for virtualized *DANE* deployments. The optimal locations for the virtualized *DANEs* are determined by the SDN controller and migrate it based on gathered statistics. The performance of the resulting system shows that, when *NFV-SDN* technologies are leveraged in such systems, software-controlled virtualized approaches can provide an increase in QoE.

In this current work, we consider a scenario where a video streaming company, in co-operation with the ISP, has allocated a network slice to allow the deployment of $vDANEs$, and the ISP is using SDN to connect the nodes. We focus on cache placement and migration problems and propose an approach for selecting the optimal location of instances of $vDANEs$. Since caches help reduce latency by putting the video content nearby the clients, the location of the caches (and vDANEs specifically) plays an important role in the performance of video streaming applications. The placement of $vDANEs$ and the migration functions are executed by an SDN controller, whereby the controller selects an optimal location for installing the initial virtual *vDANE* using NFV. When the network conditions are changed because of cross traffic or due to newly joined or recently leaving clients, the *SDN controller* runs the algorithm for cache migration. For determining the optimal point for a instance, both the resource availability and the performance requirements in terms of network bandwidth are considered. When a *vDANE* instance is migrated from one location to another, SAND messages are sent to the clients so that they can connect to the new *vDANE* in its new location.

The contributions of this section are to address: (i) an approach for selecting a node among feasible nodes for creating an instance of *vDANE* function; (ii) moving the instance to another node when network conditions are changed; (iii) an implementation of the *SAND* architecture which uses $vDANEs$ in the SDN network.

Note: $vDANEs$ behave as virtual caches which are managed by the $ISPs$, and due to the nature of *vDANE*, we will use the terms $'vDANE'$ and $'vCache'$ interchangeably.

4.2.1 Proposed SAND architecture with vDANEs

Our previous work (Clayman et al., 2018) proposed a *DANE* implementation utilizing the NFV concept, which we called the *vDANE*. Also, we introduced an algorithm for selecting the placement of an additional when a new cache is required. In this work, we study the problem of selecting the location of the initial and the migration of it considering $SAND$ characteristics. Moreover, we consider the replacement of the virtualized cache function and connectivity, as well as bandwidth and hop counts of the links, where only available bandwidth is considered in (Clayman et al., 2018).

$DASH$ clients adapt quality based on throughput and request video segments from a cache. Requesting segments from a remote cache can influence the video throughput and buffer fullness, hence the perceived quality, due to a possible increase in latency and a decrease in bandwidth. By bringing the cache nearer to clients, it reduces latency and may improve network conditions in terms of bandwidth. In order to provide a higher QoE, cache placement should be done by considering link bandwidth and delay. The goal of our algorithm is to find an optimal node for virtual cache placement to improve the QoE at the clients.

In our scenario, the SDN controller has information about IP addresses of $DASH$ clients as well as the bitrate of the video file's representations, by means of the cooperation between the network operator and video streaming company. The system architecture and controller modules, illustrated in Figure 4.5, provides *DANE* function modules for: (i) *Cache Placement* and (ii) *Cache Migration*. The controller modules include basic network service functions that are responsible for collecting traffic statistics, monitoring hosts and flows, and uploading flow rules to the switches. The controller also sends and receives *DANE* Messages as part of the $SAND$ architecture.

The *Cache Placement* module is triggered when the first $DASH$ client comes online and sends a request to start video streaming, and it selects an optimal point for the *vDANE* instance. The *C*ache Migration module is triggered when the controller detects the location of *vDANE* should be changed. Since network capacity is dynamic due to cross traffic and a different number of $DASH$ clients being con-

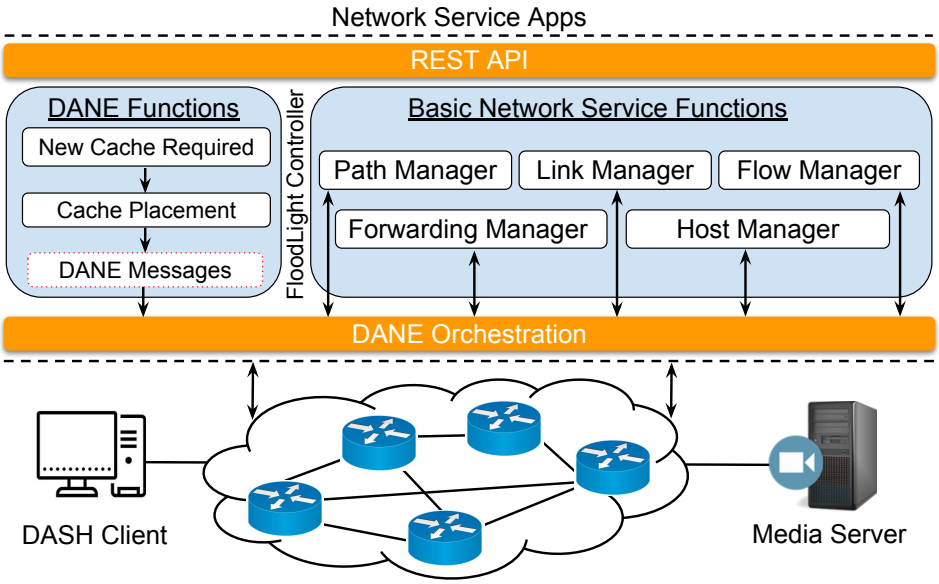


Figure 4.5. SAND Architecture with vDANE orchestration

nected/ disconnected over time from various points of the network, this module changes the location of the *vDANE* functions to provide seamless service, under this dynamic environment. The Cache Migration module then sends DANE Messages to ensure that clients can renew their TCP connection and informing them about the IP address of the new server.

4.2.2 Selecting the Initial Cache Location

Our study aims to find the optimal location for the placement of the initial *vDANE* instance and then to migrate it, in such a way that it mainly reduces the traffic and still provides high QoE for DASH clients. In order to minimize the total network traffic and to efficiently utilize network bandwidth, the hop count should be minimized. Reducing the distance between clients and content sources, it can eliminate forwarding flows over inessential links. Taking advantage of minimum hop count can improve latency and the total amount of flows that travel across network paths. Minimum hop count is considered in the $HotSpot$ algorithms (Mamatas et al., 2010), but this approach may result in lower performance if virtual caches are placed behind $bottlenecked$ links. To overcome this obstacle, besides using the distance factor, in terms of hop count, we take the total amount of bandwidth and the number of connectivity into account as well.

In the proposed cache placement algorithm, $vDANEs$ are instantiated at the network node which provides minimizes distance to the clients and maximizes the bandwidth. To achieve this, we develop *PressureInitialCache* (PIC) algorithm that

estimates the pressure score of the candidate nodes. This algorithm was developed by following and modifying the $PressureCache$ algorithm proposed in our previous work which is based on a $Pressure$ formula discussed in (Clegg et al., 2013) and enhanced in (Tuncer et al., 2015). The output of the *PressureInitialCache* algorithm is a score which is calculated for all potential switches that can be selected as hosting a *vDANE* instance. The *PressureInitialCache* calculated this score for switch j by considering distance in terms of hop counts and pressure in terms of client density.

The *PressureInitialCache* algorithm is presented here. Assume in any network, $H(t)$ represents a set of online clients at time t, d_{ij} is the distance between nodes i and j in terms of hop count, and P is the set of the path between these nodes. In the proposed framework, $node$ refers to the network elements that can run virtualized network functions. Consider a virtual cache instance is running on the node j. Suppose that the client connected to the node i requests video content from the cache. By considering that, the cache sends packets to the clients which connected to the node i, by a rate of r_i per each time unit. Accordingly, r_i unit of traffic will be passed from node j to node i over d_{ij} hops in each time unit. Hence, the total traffic amount exchanged per time unit will roughly equal the value given by formula in (4.3).

$$T(t) = \sum_{i \epsilon H(t)} r_i d_{ij} \tag{4.3}$$

In order to reduce total network traffic and response time as well, the candidate cache node should have minimum distance with clients (d_{ij}) and provide maximum available bandwidth and connectivity. The cache location which gives the maximum value of *bandwidth / distance* is optimal. Suppose L represents number of the links in the network, $B = \sum_{l=1}^{L} b_l$ is the total network bandwidth and $B^{'} = \sum_{l=1}^{L_j} b_l$ is the total bandwidth of the links that directly connected to node j. Here, b_l refers to the bandwidth of the link l and L_j refers to the number of links which j connected to. While B represents the total capacity of the network, $B^{'}$ represents the maximum traffic amount that can be transferred by the node j. $D(j) = \sum_{i=h+1}^{H} d_{ij}$ refers to distance between the node j and clients > 1 hop away, where h represents the number of clients connected to j^{th} node through one-hop distance, H is the number of all clients and $D^{'}(j) = \sum_{i=1}^{h} d_{ij}$.

The context of locality has a remarkable impact on performance. On the contrary, a long-distance connection has a negative effect due to the RTT delay and consumed links bandwidth while traveling through more of them. By considering these facts, we evaluate the candidate locations to run the initial *vDANE* network function by using formula (4.4). This formula sets a pressure score, which is indicated by $P(j)$, for all possible candidate node locations denoted by j. Note that, this formula gives slightly more importance to distance than the bandwidth. The reason for that is, distance-related metrics generally affect $jitter$ and initial waiting time parameters of video streaming applications and when available bandwidth is adequate, these parameters can be manageable with a rational buffering mechanism on the client's side. Since client with direct connection to the cache node has only one hop distance, the d_{ij} in the denominator equals to 1, which makes $\sum_{i=1}^{h} d_{ij} = h$.

$$P(j) = \frac{\frac{B}{B'}}{\frac{D'}{D}} = \frac{\sum_{l=1}^{L} b_l * \sum_{i=h+1}^{H} d_{ij}}{\sum_{l=1}^{L_j} b_l * \sum_{i=1}^{h} d_{ij}} = \frac{\sum_{l=1}^{L} b_l * \sum_{i=h+1}^{H} d_{ij}}{h * \sum_{l=1}^{L_j} b_l} \tag{4.4}$$

Finally, in order to increase reliability, the number of links connected to a candidate node also has been taken into account. Note that, more connection links provide more reliability. Lets L_j indicated the total number of connected links to the candidate node. With considering all parameters, *PressureInitialCache* algorithm is running to calculated pressure score for node j by using formula (4.5). The node with the minimum score is selected as a location for the initial cache in the network.

$$P(j) = \frac{\sum_{l=1}^{L} b_l * \sum_{i=h+1}^{H} d_{ij}}{\sum_{l=1}^{L_j} b_l * h * L_j} \tag{4.5}$$

4.2.3 Cache Migration

When a $DASH$ client requests to start the video streaming application, if there is no running *vDANE* instance on the network, the controller triggers the *Cache Placement* module. As shown in ***Algorithm 4***, the *Pressure score*, is calculated for all potential switches that can be selected as hosting a *vDANE* instance. The switch which has a minimum score considered as the optimal place for the initial *vDANE* instance.

Algorithm 4: *PressureInitialCache* Algorithm

```
Input: Set of switches
while true do
    monitor network condition
    if network condition has changed then
        foreach switch j do
            Compute PressureScore (j)
        end
    end
end
Output: Select the switch with the lowest pressure score
```

Topologies	#Nodes	#Links	#Clients	Average bandwidth (Mbps)
Custom	8	11	10	λ=7, λ=10, λ=15
Compuserve	11	14	12	λ=7, λ=10, λ=15
BellCanada	43	58	20	λ=7, λ=10, λ=15

Table 4.5. Network topologies properties and configuration

Representations	#R1	#R2	#R3	#R4	#R5	#R6
Bitrate	2133	2484	3078	3526	3840	4219

Table 4.6. Big Buck Bunny representations (kb)

After the initial *vDANE* instance starts, the controller continuously monitors network traffic, client behavior, and collect statistics from forwarding devices. When the number of clients in the system changes or the controller determines a significant change in the traffic pattern, it runs the $PressureInitialCache$ algorithm and calculates scores for each node again. Changes in traffic amount are determined based on a predefined certain *quality threshold*, which can be assigned by either video streaming company or network operator. If the location of *vDANE* is changed according to the new score, the controller sends PER enforcement messages to the clients to change their request to *vDANE* in the new location. Further, the shortest paths are selected for each client for streaming packets.

If the average quality in terms of bitrate or representation received by the clients is under the *quality threshold*, the controller then decides to deploy additional *vDANE*. An algorithm that decides when a new cache instance should be installed is proposed in our previous work (Clayman et al., 2018).

4.2.4 Test bed and Simulation Setup

Same as previous works, we have implemented and run simulations using the $Mininet$ emulator in order to evaluate the performance of the proposed algorithm. The *FloodLight* SDN controller, with the assistance of $OpenFlow$ as a southbound interface, was used during the simulations. Three different network topologies were used to evaluate the performance of the $PressureInitialCache$ algorithm, which include a $Custom$ topology and two from the Internet Topology Zoo (Topology-Zoo, 2017): the *Compuserve* and *BellCanada* topologies, as shown in Table 4.5. The number of the DASH clients are set to *10*, *12*, and *20*, for of the network topologies, respectively. A *Poisson distribution* with different mean values: λ=7, λ=10, λ=15 Mbps, were used for defining the network link bandwidths. Network links do not have the same bandwidth, but there is no big difference in links' capacity. For example, in the first setting, bandwidth is set to (λ=7 Mb) which means that links capacity could even be 5Mb or 8Mb.

During simulation, the *Big Buck Bunny* video (ITEC, 2018) with six different representations is used for streaming. Each representation contains 299 video segments with an equal length of 2 seconds, so the total length of each representation is 598 seconds of video. As shown in Table 4.6, the first representation $(R1)$ has the lowest bitrate, and the last representation $(R6)$ provides the highest bitrate or video quality. To achieve better QoE, clients should request and receive as many segments as possible from representations with a higher bitrate. The clients run a throughput based rate adaptation algorithm, where they decide the video quality based on the observed throughput values. Clients are randomly distributed over all of the nodes, and the DASH clients join and leave the network based on a *Poisson distribution*, which has a mean equals to 3 seconds.

4.2.5 Performance Evaluations and Experiment Results

In order to provide comparable performance results, we also implemented two other algorithms: one of them is *Best effort*, and the other one is the $HotSpot$ approach described in (Mamatas et al., 2010). In the *Best effort* approach, a cache is placed near to the center of topology. Therefore, the average distance between all the clients and the cache is minimum with this approach. The alternate $HotSpot$ algorithm benefits from the concept of the locality by introducing $F(i) = a^3 + b^2 + c$ as a pressure function. In this function, a, b and c, refer to the number of clients with one, two, and three hops away from node i, respectively. Unlike *Best effort*, $HotSpot$ places caches in a location that has the highest client density. In the simulations,

all the parameter settings were equal for all of the approaches. Each simulation was repeated 10 times and the average values are given in the tables.

In Tables 4.7, 4.8 and 4.9, the metrics for the average received video quality, the average startup delay, and the average outage duration values obtained from the simulations are given for the $Custom$, $Compuserve$, and $BellCanada$ topologies, respectively. In the tables, the *Average received video quality* values shows the representation bitrate received on the client's side; the *Average startup delay* is the parameter that represents the latency of starting a video after user requests to play a video, and the *Average outage duration* is the duration of video stalls.

If we examined the received video quality values, we see that the minimum quality obtained is with *Best effort* approach. The main reason for this is that this approach only considers the locality, but it does not consider the number of clients connecting to the network or the distribution of the connection points. Mainly, in the *Best effort* approach, the location of the caches may cause some bottleneck points in the network, hence clients start sending a request for segments with lower quality, which has a smaller size, and in turn, outage duration values may reduce. We see that effect when we examine the values given in the tables, where the clients experience lower outage duration than clients in the $HotSpot$ approach. For all approaches, we see that the observed parameters get better when the network capacity gets higher, in terms of available bitrates, as expected. We observe that the proposed $PressureInitialCache$ approach outperforms the other two approaches in different sizes of network topologies in all simulations.

The percentage of received video segments belonging to each representation are given in Figures 4.6, 4.7, and 4.8. Here, $R1$ represents the lowest quality, $R2$ through to $R5$ represents the medium quality representations, and $R6$ represents the highest quality. The numbers show that the clients in the $PressureInitialCache$ approach receive the biggest share of the highest quality $R6$ segments in the $Custom$ and $Compuserve$ topologies, which is a good result. Since $BellCanada$ is a relatively bigger topology, the distance between the cache and the clients affects the quality and the smallest percentage belonging to the highest quality among the results related to all topologies is observed in $BellCanada$ topology. Nevertheless, the clients with the proposed $PressureInitialCache$ approach received the highest number of segments from the top representation – again a good result.

Parameters	Average received video quality (Kbps)			Average startup delay (sec)			Average outage duration (sec)		
Bandwidth (Mbps)	**λ=7**	**λ=10**	**λ=15**	**λ=7**	**λ=10**	**λ=15**	**λ=7**	**λ=10**	**λ=15**
Best effort	2309	2865	3532	13	7.9	5.8	450	53	5
HotSpot	2546	3532	3440	13.4	5	6.1	757	4	17
PressureInitialCache	3013	3670	3680	11.2	4.9	5.1	128	2	2

Table 4.7. Performance results: Custom topology.

Parameters	Average received video quality (Kbps)			Average startup delay (sec)			Average outage duration (sec)		
Bandwidth (Mbps)	**λ=7**	**λ=10**	**λ=15**	**λ=7**	**λ=10**	**λ=15**	**λ=7**	**λ=10**	**λ=15**
Best effort	2799	3050	3442	10.3	7.9	7.6	351	133	34
HotSpot	2587	3050	2945	14.5	7.9	6.2	615	133	42
PressureInitialCache	3146	3066	3517	10.1	7.9	6.1	180	102	26

Table 4.8. Performance results: Compuserve topology.

Parameters	Average received video quality (Kbps)			Average startup delay (sec)			Average outage duration (sec)		
Bandwidth (Mbps)	**λ=7**	**λ=10**	**λ=15**	**λ=7**	**λ=10**	**λ=15**	**λ=7**	**λ=10**	**λ=15**
Best effort	2304	2451	2883	26	15	9	386	250	147
HotSpot	2650	2542	2917	19	11	9	662	672	417
PressureInitialCache	2600	2607	3107	13	11	7	340	233	87

Table 4.9. Performance results: BellCanada topology.

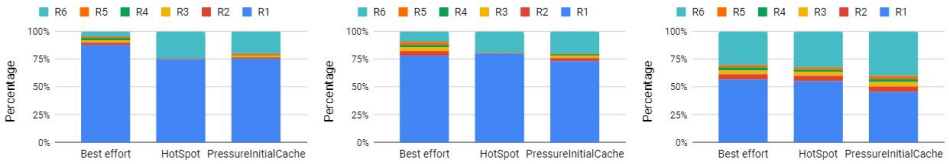
In order to show that the improvement in the received quality provided by $PressureInitialCache$ is maintained throughout the whole run, we averaged the received quality as a function of time. These values are comparatively given for all approaches for the $BellCanada$ topology in Figure. 4.9. Average received quality values are obtained by averaging the bitrate of the representations downloaded by all clients in all simulations. When we examine the data, the clients using the $PressureInitialCache$ approach always play the video with higher quality, and it is consistent for each segment of the video. This shows that, for dynamic adaptive video streaming applications, optimizing localization by considering the density of the clients has a positive impact on the received quality. However, if the network has limited capacity, where $\lambda = 7$ Mbps, we see a small quality degradation of the proposed approach. The reason for that is that after a *vDANE* is migrated, and during the TCP re-establishment process, the clients stop receiving traffic and measure throughput as zero and so request the lowest quality since clients use throughput based adaptation. A buffer based rate adaptation would perform better in that case. This shows that the performance of the virtualized environment is highly connected with the rate adaptation algorithm used by DASH applications.

(a) λ=7 Mbps (b) λ=10 Mbps (c) λ=15 Mbps

Figure 4.6. Distribution of the received quality level – R1 to R6: Custom topology

(a) λ=7 Mbps (b) λ=10 Mbps (c) λ=15 Mbps

Figure 4.7. Distribution of the received quality level – R1 to R6: Compuserve topology



(a) λ=7 Mbps (b) λ=10 Mbps (c) λ=15 Mbps

Figure 4.8. Distribution of the received quality level – R1 to R6: BellCanada topology

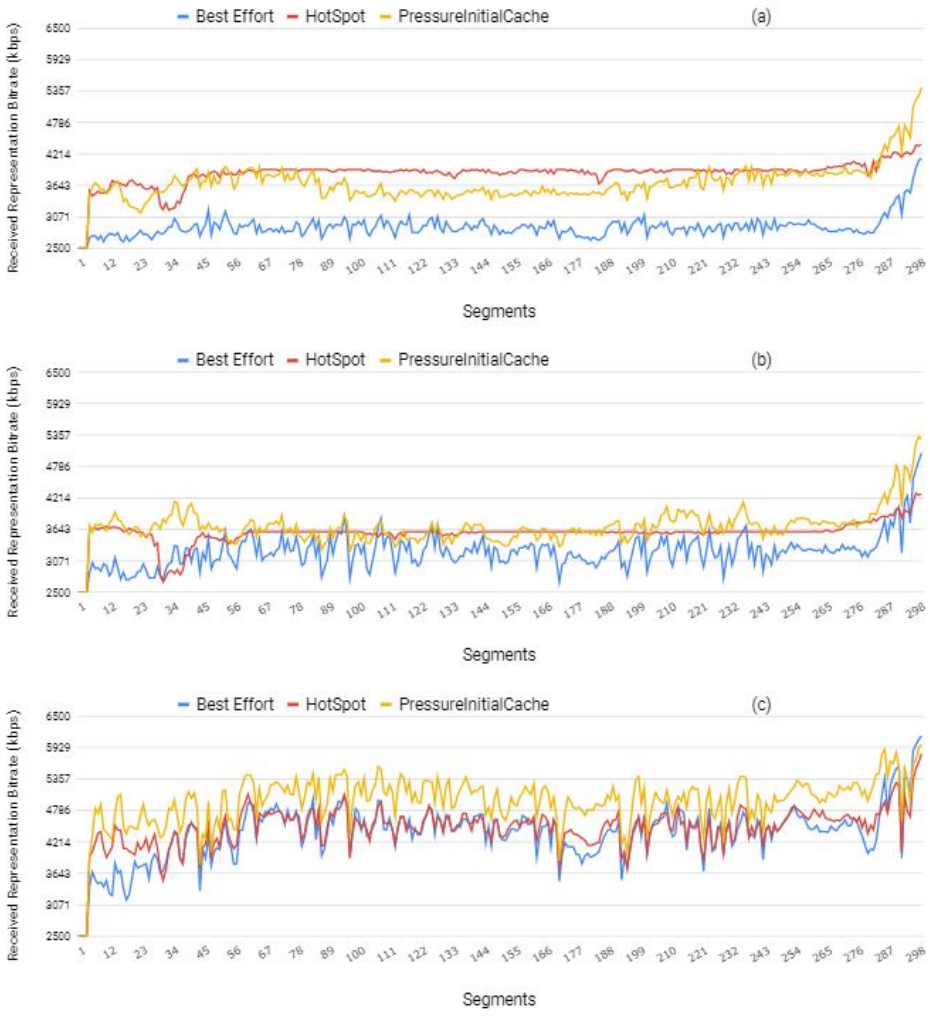


Figure 4.9. Bitrate of the received video representation per segment: (a) λ=7 Mbps, (b) λ=10 Mbps, and (c) λ=15 Mbps)

To demonstrate the impact of the proposed approach in larger networks where more than one cache instance is required, we also ran additional simulations by using the *BellCanada* topology with 100 clients. In the first set of simulations, the first cache is placed in random locations, whereas the proposed algorithm is used in the second set of simulations. For both sets of simulations, when the number of online clients increases, some second caches are installed by using our algorithm given in (Clayman et al., 2018). Simulation results indicated that with the random approach, clients received 14% of the video segments from the highest quality, on average, while receiving 17% when $PressureInitialCache$e was utilized. The average video bitrate received by clients in the random approach was equal to 2466 Kbps, and was equal to 2571 Kbps when using $PressureInitialCache$. This means that end-users experienced better video display with our proposed algorithm. Significant improvement is observed in the outage duration values, where the total outage

duration is as low as 168 seconds for the proposed algorithm, but with the random approach results in 637 seconds of outage duration – which is totally unacceptable. For these tests, we configured paths with a lower bandwidth of λ=7 to show the behavior of the clients.

4.2.6 Summary

In this section, we proposed a $SAND$ architecture that deploys $vDANEs$ and an approach for initiating and migrating $vDANEs$, the virtual *DANE* cache instances. Such an approach is made possible by softwarized networks using the deployment of *NFV-SDN* technology in $ISPs$. One of the objectives of our work is the cache placement algorithm used to calculate an optimal point for *vDANE* placement. This is done by considering both the number of hops and the available bandwidth between the potential location and the clients. The nodes having high connectivity are also considered, as it is an important criterion when reliability issues are taken into account. Locations with more connection links, more available bandwidth and fewer hops from the clients have more chances to be selected.

In order to show the performance improvement that can be provided by the proposed software-controlled approach, the comparative performance results, conducted on Mininet, were obtained and presented by implementing two cache placement algorithms proposed in the literature. We tested the performance over different types and sizes of the network, and the results show that the proposed approach outperforms the other two approaches in all simulations. The observed throughput improvement provided by the proposed approach, when compared to *Best effort* and $HotSpot$ approaches is up to 30.5% and 19% respectively. Our algorithm also provided a decrease in the re-buffering duration of up to 49% and 70%, and a decrease in startup delay of up to 50% and 32% when compared to these other approaches. These results show us that, as well as implementing a video streaming specific cache localization algorithm, deploying localization by using software-based approaches in virtualized environments, makes it easier to adapt to dynamic conditions by changing the location of the virtualized components.

In our future work, we plan to implement a content distribution scheme and jointly consider *vDANE* location and content availability when selecting the location of $vDANEs$. We will evaluate against real network slices when they are deployed by $ISPs$, as well as utilizing a larger number of clients.

$Note$: The results of this study published in (Kalan et al., 2019), *IEEE Conference on Network Softwarization (NetSoft 2019) 24-28 June 2019- Paris, France*

4.3 SAND Assisted Virtualized Cache Placement in Highly Dynamic SDN Network

In this section we address the problem of *Cache Placement* in the dynamic network where network topology and traffic patterns are changing rapidly. The goal of our study is to develop an efficient virtual cache placement in order to minimize the load of the links and improve QoE on the client's side. We develop algorithms based on $PressureCache$ to achieve optimal cache placement and maximizing reliability. For this purpose, as a contribution of this study, we consider nodes to place the *vDANE* instances by considering the number of links connected to the nodes, the available bandwidth of these links and the number of connected clients.

In the general case, adding K virtual cache to the underlying infrastructure can improve network latency by factor k, placing network function in suitable location and programming flows have a big effect on network performance as well. Virtual cache $(vCache)$ in the form of $vDANEs$ migrates resource near to the end-users. Content distributors leverage virtualization features in the NFV platforms to deploy caches as $VNFs$ ($vCaches$) instead of physical appliances. Such a solution leads to dynamic configuration and reduces cost, but providing sustain the quality is recognized as a challenge since dynamic configuration may cause interruption of the provided services. The $vCache$ migration is essential to the use of network resources efficiently and improves the performance of the overall *NFV-SDN* based CDN function in terms of network operator cost reduction and high streaming quality (Ibn-Khedher et al., 2016b). Due to the voluminous traffic to be delivered to video streaming clients, providing efficient algorithms requires knowledge about network conditions and client distribution in order to migrate $vCache$ in an appropriate location.

As discussed, $DASH$ standard developed by the $MPEG$ enables high quality video streaming over the Internet delivered from conventional $HTTP$ servers. Another ongoing standardization process related to DASH architecture is $SAND$ which has been being developed by the $MPEG$ group. DASH clients adapt quality by taking observed network conditions into account and request video from an HTTP server. However, requesting content from a remote server has a negative impact because of the constraints on latency and bandwidth. Latency may arise due to the distance between the server and the client. Furthermore, forwarding flows through end-to-end paths while traversing a set of middle-boxes being used from traffic shaping and performance (e.g, proxy or caching) to security such as firewalls, *Intrusion Detection System* (IDS) and *Deep Packet Inspection* (DPI), come

with high infrastructure and operational costs (Sherry et al., 2012). Mounting cache near to clients, reduce latency as well as the total amount network traffic flows over the network.

The objectives of the proposed approach in this study are i) selecting a candidate node among feasible nodes in order to deploy vCache. The selected node should have adequate bandwidth and appropriate location regarding the attachment points of the online clients. ii) move the current vCache to another location or deploy additional vCaches when network conditions change. In our previous work (Clayman et al., 2018), we discussed finding the second cache, for the given network topology and the distribution of the client's attachment points to the network. Determining the location of the initial vCache and its migration was studied in our other previous work (Kalan et al., 2019). In this current study, we focus on the problem of vCache placement and migration in a more general form by enhancing our previous studies. We propose an architecture that selects an almost optimal location for installing the initial virtual cache (*vDANE*) as a network function, migration vCache, and installing additional vCache on demand. For this purpose, an adaptive approach with considering resource availability and the performance requirements in terms of end user's QoE is developed.

4.3.1 Problem Statement and Background

Using cache reduces origin server traffic and response time when requested resources are close to the end-users. The optimally of the cache locations is dependent on the network traffic patterns, client distribution, and mobility during the time. Hence, the candidate node for hosting cache should be close to high-density locations where there are more clients connected to the network. It is important to highlight that providing caching at all network elements (e.g, forwarding devices) is technically feasible (Ghodsi et al., 2011).

Pervasive and centralized caching approaches are common forms of caching mechanism. In the pervasive caching, all forwarding devices have the capabilities of content caching that provides low latency. But, pervasive caching is costly and inefficient because of redundancy. Furthermore, it can affect network performance because of extra overhead. It also reduces the cache hit rate. In the centralized caching only dedicated forwarding devices have caching capacity. Hence, it reduces content redundancy, increases the cache hit rate. The flexibility introduced by the NFV-SDN reduces forwarding device processing while the routing algorithm is running by the central controller.

Authors in (Rossi and Rossini, 2012), consider deploying more cache space in the "core" routers of the network. On the contrary, authors in (Psaras et al., 2012) conduct putting cache at the edge of the network results in better performance. Authors in (Wang et al., 2013) consider network size, content popularity, and objective replacement strategy for investigating the allocation problem. However, those proposed solutions more focused on cache allocation policy and performance of caching rather than cache placement which is not the intention of this thesis. The priority of is cache placement in the optimal location which reduces total network traffic and provides better quality for end-users.

Early effort in this direction have already disused in (Badshah et al., 2020). Authors in this study compute the k number of the most important switches that carry more network traffic and then place cache servers by attaching a cache server to one of the selected switches at a time. In this study SDN controller with the assistance of the OpenFlow is used for network monitoring and traffic engineering. To consider important switches, the authors assume that the network is running long time (days or weeks) and has enough traffic logs to analyses and make a decision. The majority of our proposed approach is high dynamic nature and agility in decision making according to current network conditions.

We aim to provide a cost-effective approach to address this problem while ensuring an acceptable level of QoE such as re-buffering, oscillation bitrate and display quality in video streaming applications. The placement phase consists of determining how many virtual caches are necessary and where to place them to meet demand. We provide a heuristic solution with considering both available network bandwidth and distance by taking advanced of the *NFV-SDN* and *SAND* assistance into account. The core contribution of this study aims to i) find near to optimal location for placing the first and additional caches in the highly dynamic network. ii) Move caches to the appropriate locations based on request distribution with considering available resources. iii) Decide forward the client's request to a suitable cache.

The placement of the virtual function has a big effect on the provided service. Therefore, the virtual functions should be placed where they will be used in a more efficient and cost-effective way. While the network is changing rapidly, it is not sufficient to make good initial placement only based on static conditions. Our previous works (Kalan et al., 2019) address this challenge by introducing optimal cache placement and migration algorithms in the high dynamic network where network traffic and client distribution are changes rapidly. Another work by the

authors in this area (Clayman et al., 2018), describe additional cache placement to meet demand by considering the live DASH attributes and characteristics. Proposed SAND video streaming architecture utilizing both SDN and NFV concepts. In both studies, virtual caches are defined as DANEs, gives the right level of flexibility and control. The $vDANEs$ are the *caches with the knowledge of DASH characteristics, which uses NFV hosted by the servers connected to the switches*. However, proposed approaches neither discuses the dynamic nature of the network topology and client movement nor the efficient cache migration and hand-off problem. Furthermore, the first cache location was considered as fixed in the proposed algorithm in (Clayman et al., 2018).

4.3.2 System and Controller Architecture

As shown in Figure. 4.10, in our system architecture, controller gathers network information from network nodes (e.g., switches, servers, clients) by taking advantage of the southbound interface and basic network service functions. In addition to basic network functions, there are three DANE modules inside the controller:

- *Cache Placement:* By analyzing the performance of the network and DASH clients, it is possible to determine the current resources (e.g., remote server and limited bandwidth) are not enough to keep the quality at a reasonable rate. When the network size or the number of clients increases, the additional cache should be installed to meet demand.

- *Cache Migration:* If it is decided that the cache instance is going to be initiated, it should be placed in a suitable location so that this decision causes an increase in QoE and network utilization due to the decrease in network traffic. In lack of enough resources or even deploying cache in an inappropriate location, clients switch the quality to the lower alternatives and more interruptions during display could be experienced. In such a case the cache should be moved to a suitable location.

- *DANE Messages:* Connecting a client to the appropriate cache improves both network resource utilization and client QoE. When there is more than one cache in the network, the controller enforces a client connects to the appropriate cache with the assistance of the DANE PER messages. As a consequence, a client experiences better video quality.

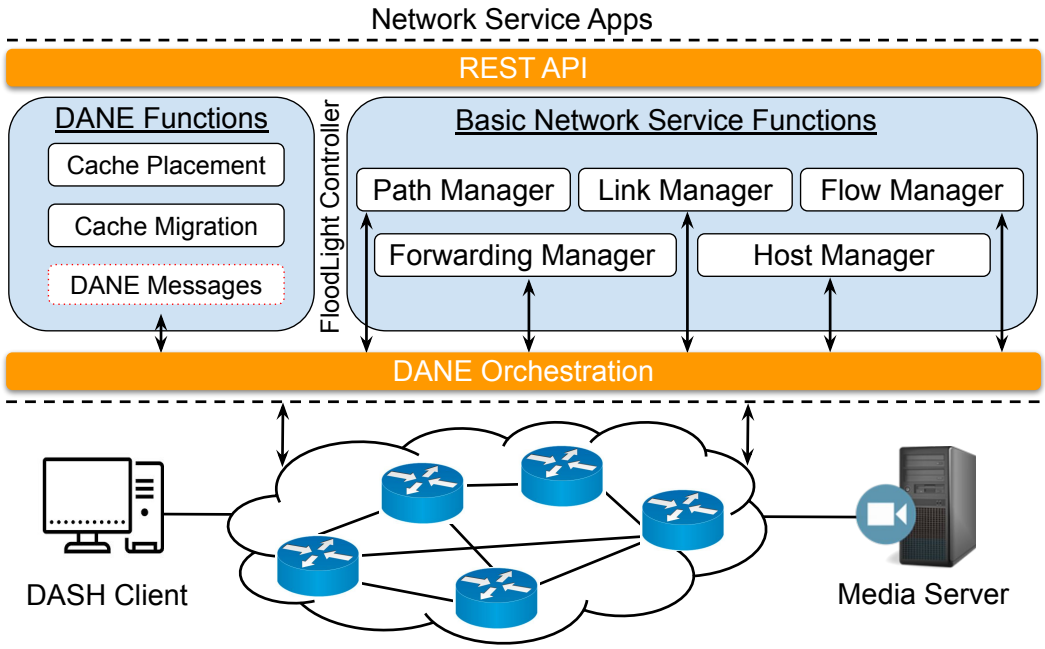


Figure 4.10. Illustration of the SDN architecture and orchestration DANEs functions.

4.3.3 Initial Virtual Cache Placement based on Pressure Score Estimations

Recall from the formula (4.5), it can be considered $B = \sum_{l=1}^{L} b_l$ as constant since total network bandwidth is the same for all clients and it affects the pressure score for each candidate at the same impact. Therefore, we can ignore it from the formula. After updating (removing total network bandwidth as a constant), formula (4.5) can be rewritten as in formula (4.6).

$$p(j) = \frac{\sum_{i=1}^{H-h} d_{ij}}{\sum_{l=1}^{L_j} b_l * h * L_j} \tag{4.6}$$

Beside end-to-end bandwidth limitation that varies from time to time, each physical hardware that hosted vDANE has the limited processing power, as well. By increasing the number of incoming requests to the specific *vDANE*, its performance may degrade since it becomes a bottleneck processing and forwarding point. Hence, the consumption amount of processing unit, which is denoted as pu, is another important parameter that should be considered. Candidate node with less pu utilization is preferred. With considering all parameters, *PressureInitialCache* algorithm is running to calculated pressure score for node j by using formula (4.7). The node with the minimum score is selected as a location for the initial cache in the network.

$$P(j) = \frac{\sum_{i=1}^{H-h} d_{ij} * pu}{\sum_{l=1}^{L_j} bl * h * L_j} \tag{4.7}$$

4.3.4 Deploying Initial vDANE Instances

Media server -as well as $vDANEs$- periodically send network information including the average quality of the received video representations requested by the clients to the controller. The *New Cache Required* as a DANE-module running inside the controller, retrieves this information and determines a new cache is necessary. Determination is based on predefined certain *quality threshold*, which is set by either video streaming company or network operator. When the average quality in the form of bitrate or representation, received by the clients falls under the *quality threshold*, the controller decides to deploy initial or additional *vDANE* instance. The controller triggers the *Cache Placement* module in demand for initial *vDANE*. The *Cache Placement* module runs the $PressureInitialCache$ algorithm given in the ***Algorithm. 4***. The additional cache will be installed if the initial cache does not provide expected improvement in network traffic.

As mentioned before, the output of the $PressureInitialCache$ algorithm is a node having the minimum score, which is calculated for all potential switches that can be selected as hosting a *vDANE* instance. $P(j)$ with minimum score considered as the almost optimal location for first *vDANE* deploying. The main objective of the algorithm is finding an optimal location for *vDANE* placement, by considering the number of hops and available bandwidth between potential location and the clients. In the case of reliability, the number of connected links in a potential location considered as well. The location with more connection links and available bandwidth and fewer hop counts from the clients has more chances to be selected.

To the best of our knowledge, this algorithm has never been used for determining cache placement in the literature. The closest study in (Tuncer et al., 2015) which relied on distributed management layer and control layer known as local manager and local controller. There is more than one switch connect to each local manager nodes observing the number of hops. But the *PressureInitialCache* algorithm only selects one switch for the initial *vDANE* placement. After determining an appropriate node (e.g, switch) as a location for deploying *vDANE* instance, the controller creates a new *vDANE* instance connected to the specific switch. Then, the controller, update routing devices' forwarding tables and forward clients request to the recently deployed *vDANE* via the shortest path.

Authors in (Claeys et al., 2016) proposed a hybrid model by dividing the total network's available caching space such that *"one part is reserved to proactively push content in the different caching locations, while the rest is used to implement reactive caching"*. Although this study provides reactivity migration to the proactive placement when unexpected changes in the request pattern happen, in contrast to our approach it is still relying on the pre-knowledge of the content popularity and the geographical distribution of requests. The complexity of the content popularity prediction is another concern, the ISP decides which content when will be stored and where. In our proposed approach, there is no prediction of storing content into caches and where caches should be placed. Furthermore, the new additional cache will insert into the network only in the case of necessary. Without a doubt, leasing more cache space into the network improves network bandwidth usage, but it is a costly solution.

4.3.5 Cache Migration

Network controller continuously monitors network condition and gather statistic parameters from underlay infrastructure as well as the number of DASH client and cross-traffic pattern. Unlike legacy networks, it was shown that network monitoring can be achieved more efficiently by using $OpenFlow$ statistics (Megyesi et al., 2016) (Singh et al., 2018) (Adrichem et al., 2014). For example, instead of probing the end-to-end path, the SDN controller can keep track of available bandwidth on each link and hence estimate end-to-end available bandwidth (Megyesi et al., 2016). Any changes in the environment that affect the current configuration should be reported to the controller and consequently, the controller triggers appropriate functions. In the case of any changes in network topology, traffic pattern or even $quality\ threshold$, the controller again runs the $PressureInitialCache$ algorithm with respect to an estimated new location based on update data. If the output of the algorithm refers to new location for *vDANE*, the controller virtually migrates *vDANE* to the new place and assemble it. Further, the controller leverage SAND technology and sent a PER message to enforce the client connects to the new *vDANE*. It is worth to recall that the shortest path is used for streaming video.

4.3.6 TCP Hand-off

TCP hand-off is a set of actions that happen when the connection removed from one interface and connects to another. During this short period clients stop sending and receiving data. Traffic redirection, load balancing, and roaming are the main reasons. The hand-off also appears in the CDN networks, where CDN architectures leverage a redirect mechanism to initiate a connection between a client and the most appropriate server (Binder et al., 2015). As shown in Figure .4.11, in the case of *vDANE* migration, the current connection should be terminated and a new connection re-initiate with *vDANE* instance that recommended by the controller via PER enforcement messages. Unfortunately, additional TCP handshake results in latency and clients may experience quality degradation or re-buffering specifically in the lack of enough network bandwidth.

While SDN architecture decouples control logic from the forwarding hardware, forwarding devices act as dump elements and just repeat role dedicated by the SDN controller. These roles that stored inside the routing (forwarding) table define the behavior of the forwarding device against the new incoming request. Forwarding device asks from the controller when there is no entry in the forwarding table for a new incoming request. Figure. 4.11 visualized connection set up when a new client joins the network. While at the beginning there is no entry for this client in the switch, it forwards the client's request toward controller in the form of PACKET_IN message. The controller returns appropriate action in the form of PACKET_OUT message. After receiving PACKET_OUT message from the SDN controller and establishing the handshaking process, switch asks about related routing information from the SDN controller. The controller updates the switch's routing table with the related routing information and consequently client connects to the server (or *vDANE*) and starts streaming by requesting and downloading a sequence of chunks. In the case of *vDANE* migration or installing additional *vDANE*, the SDN controller updates the switch's routing table. At the same time, the SDN controller trigger clients to connect new *vDANE* via PER message. Obviously, the connection and establishing process takes more time, therefore, during hand-off time clients may experience re-buffering.

4.3.7 Testbed and Simulation Setup

For the sake of providing a comparison, we implemented the simulation using $Mininet$ emulator to evaluate the performance of the proposed *PressureInitialCache* algorithm. The agility of the $Mininet$ (Lantz et al., 2010) provides an easy

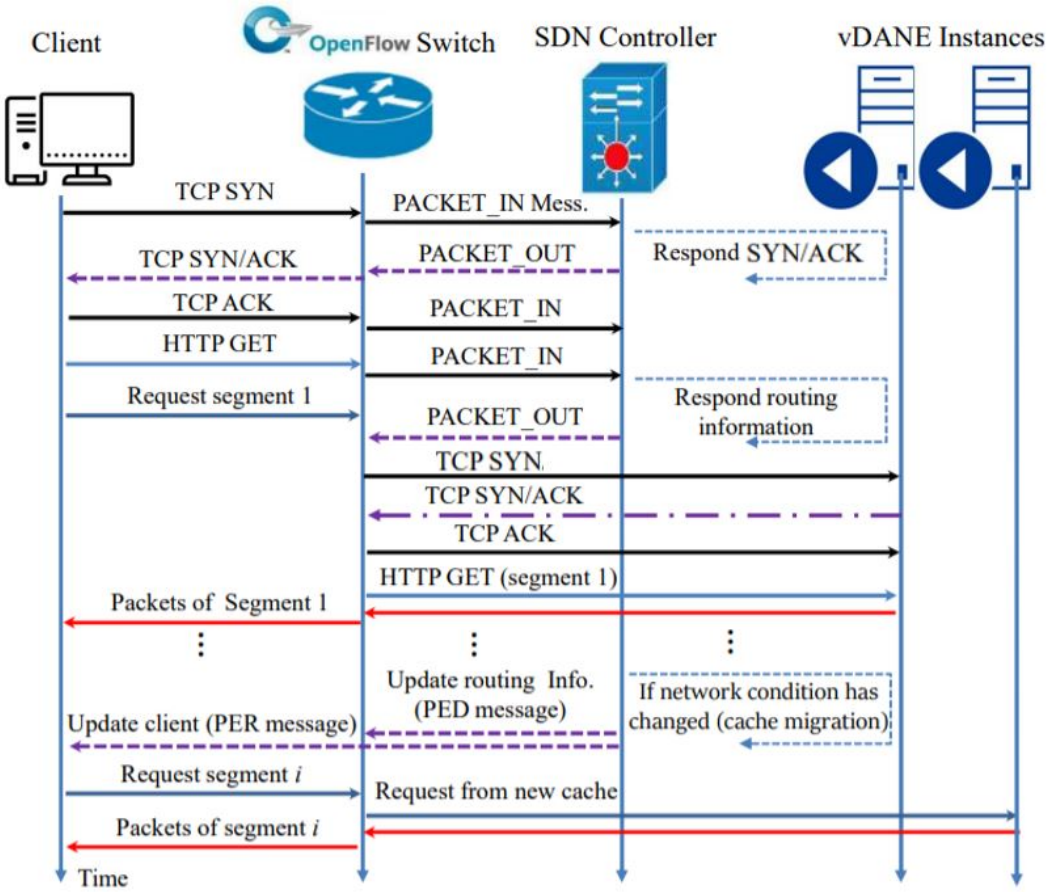


Figure 4.11. Illustration of the TCP handshaking and connection initialization architecture.

way to prototype and evaluate SDN protocols and applications. In the context of the SDN controller, $FloodLight$ (FloodLight, 2017) controller with the assistance of the $OpenFlow$ as a southbound interface is used during the simulation. Combination of the $Mininet$ and $OpenFlow$ switch in a virtualized container, provide same semantics of software-based $OpenFlow$ switches (Kreutz et al., 2015). We compared the proposed algorithm with two different algorithms, where one of them is *Best effort* and the other one is the $HotSpot$ approach (Mamatas et al., 2010) which is a kind of $Pressure$ algorithm. In this study, we consider the same value for all node's processing unit.

As shown in Table 4.10, three different network topologies was applied to evaluate the performance of the *PressureInitialCache* algorithm, which include one $Custom$ topology which is shown in Figure. 4.12, and two from the Internet Topology Zoo (Topology-Zoo, 2017) include $Compuserve$ and $BellCanada$ topologies. $Poisson$ distribution with different mean (λ=20, 25, 30 Mb) values are used for generating the network link bandwidths. The number of nodes and links is the same as the previous section configuration, but the bandwidth of each link increased due

to the number of online clients. Network links do not have the same bandwidth, but there is no big difference in links' capacity. For example, in the first setting, bandwidth is set to (λ=20 Mb) which means that links capacity could even be 15Mb or 25Mb. Also, the number of the DASH clients are set to 30, 40, and 50 for $Custom$, $Compuserve$, and *BellCanada* topologies respectively. In order to show the effect of the increasing number of online clients which causes to install additional cache, we also repeated simulation with a new set of online clients shown in the last column.

Big Buck Bunny (ITEC, 2018) which was illustrated in Table 4.6, is used for streaming video during simulation. There are six representations include one base layer and five enhancement. While the first representation (R1) has the lowest quality, the last representation (R6) has the highest quality. Each representation contains 299 video segments with equal length of 2 seconds video, in total 598 seconds of video length. To achieve better QoE, clients should request and receive segments from an enhancement layer with a higher bitrate as more as possible. The client randomly distributed over all nodes and the same distribution is used in all algorithms.

Network topologies	**Average available bandwidth**	**# Clients PressureInitialCache**	**# Clients PressureCache**
Custom	(λ=20, 25, 30 Mb)	30	35
Compuserve	(λ=20, 25, 30 Mb)	40	50
BellCanada	(λ=20, 25, 30 Mb)	50	100

Table 4.10. Network topologies and configuration

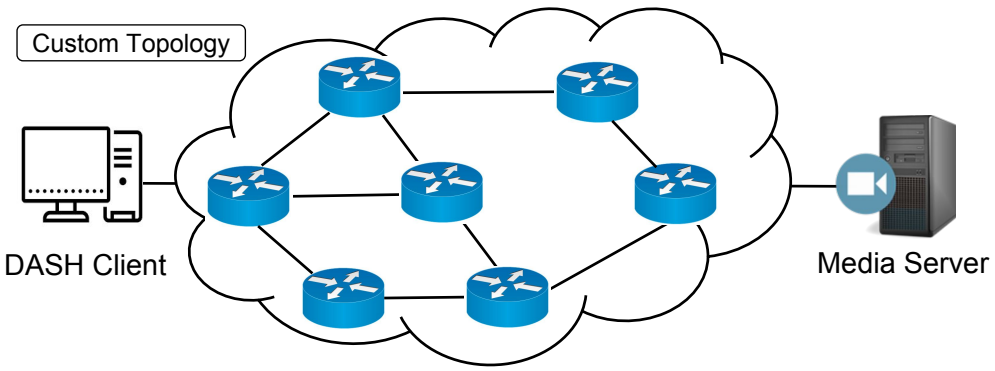


Figure 4.12. Illustration of the $Custom$ topology architecture.

DASH clients join the network based on *Poisson* distribution and start requesting a suitable representation after downloading the MPD file. Adaptive streaming allows clients to adapt the highest bitrate to achieve better display quality. This adaptation can be based on buffer occupancy, throughput or a hybrid model. In this study, we considered throughput base adaptation. On average, clients' interval time is set to 1 second. The client distribution network is totally random. During simulation time client may leave the network and new clients join the network. Hence, the simulation domain has a dynamic nature. Each simulation is repeated 10 times and the average values are given in tables and Figures. There are three different network topologies, each of them has three different configurations and bandwidth. Since all algorithms repeated 10 times in each topology, in total 270 simulations repeated.

4.3.8 Performance Evaluation and Experimental Results

In the simulation scenarios, we evaluated the proposed approach using three different network topologies each with three different configurations. Moreover, the shortest path algorithm is used for routing to the $vDANEs$. To enable evaluate results of the proposed algorithm, we implemented two different algorithms. The first one is *Best effort*, which *vDANE* takes place in the location where forwarding devices carry more network traffic considering (Badshah et al., 2019). $HotSpot$ as a second algorithm introduce $F(N_i) = a^3 + b^2 + c$ function intending to derive to the optimum location by taking advantage of the locality and place cache in the highest tensity location. In this function, a, b and c refer to the number of clients with one, two, and three hop count base distance from node N_i, respectively.

Tables 4.11, 4.12, and 4.13 list quality parameters in the form of the average value obtained from the simulation of the $Custom$, $Compuserve$, and $BellCanada$ topologies respectively. For description: Received video quality implies experienced bitrate in clients' side, startup delay indicates latency before starting display when client trigger request and outage duration defines video freezing time during display, which has a negative impact on end-user QoE. It is obtained from the tables that *Best effort* has least received bitrate while $HotSpot$ and $PressureInitialCache$ achieves better results. It is because *Best effort* algorithm lie caches in the center of the network by the assumption of the minimum distance between clients and caches. Deploring caches in the center of a network without considering client distribution may lead to inefficient performance, especially when caches in places behind bottleneck links. On the contrary, the $HotSpot$ algorithm benefits from the locality and hence it has higher received bitrate comparing with *Best effort*, but it is still lower than $PressureInitialCache$ algorithm.

Parameters	Average received video quality (Kbps)			Average startup delay (sec)			Average outage duration (sec)		
Bandwidth (Mbps)	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**
Best Effort	2513	2774	3078	10	10	12	121	84	12
HotSpot	2761	3057	3220	11	9	8	205	43	14
PressureInitialCache	2906	3160	3160	12	13	13	109	41	14

Table 4.11. Performance results: Custom topology.

Parameters	Average received video quality (Kbps)			Average startup delay (sec)			Average outage duration (sec)		
Bandwidth (Mbps)	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**
Best Effort	2446	2852	3196	17	16	12	140	55	26
HotSpot	2563	2877	3083	16	14	13	256	78	68
PressureInitialCache	2471	2888	3232	24	17	16	168	112	23

Table 4.12. Performance results: Compuserve topology.

Parameters	Average received video quality (Kbps)			Average startup delay (sec)			Average outage duration (sec)		
Bandwidth (Mbps)	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**
Best Effort	2321	2373	2397	24	20	18	733	683	585
HotSpot	2568	2701	2955	17	14	13	356	235	100
PressureInitialCache	2859	3056	3185	24	22	20	197	144	97

Table 4.13. Performance results: BellCanada topology.

The dynamic nature of the $PressureInitialCache$ algorithm displacement cache into the appropriate location. Since the process of displacement related to any changes in traffic patterns or client distribution, it's expected that startup time and in sum case outage duration (e.g, only first two value of the Compuserve topology) is taking a little more time during handover. Similar results were obtained with the other thresholds values. It is worth emphasizing that assigned bandwidths are very tight, thus we expect startup delay in all scenarios and it is rising with increasing network size. In all cases, increasing network bandwidth consequence to better performance. As we expected, our proposed algorithm outperforms other algorithms in all network topologies and bandwidth.

In the context of QoE, received representation is another important parameter. Receiving more video segments form higher video quality or representation means that clients experience display with higher quality. Figures 4.13, 4.14, and 4.15 shows percentage of the received video segments from each quality layer. While quality layer $L1$ represents the lowest quality, $L6$ represents the highest video quality. The client achieves better display quality by receiving more segments from the upper layers. Observed percentages indicate that clients in the *PressureInitialCache* received more segments from the highest layer, which means better QoE. The other important observation is that, by increasing network size and the number of online clients the *PressureInitialCache* algorithm shows more efficiency. The performance of the proposed algorithm is more evident in the $BellCanada$ topology which is a relatively bigger one. In small size network topologies, client and content are close to each other, hence the effect of the proposed algorithm is not more obvious.

The long term received representation is another valuable factor to illustrate the effectiveness of our proposed algorithm. Figures 4.16, 4.17, and 4.18 illustrate whole video streaming include 299 video segments. As see in the Figures, efficiency of the *Best effort*, *HotSpot* and *PressureInitialCache* are shown with blue, red and yellow color respectively. These results show reflect received video quality as a fraction of time. bigger value means that clients are displayed video with better quality. Without a doubt, receiving a high-quality video with more outage and interrupt during display time never provides a better QoE. Therefore, receiving more segments form high-quality representations without considering outage duration does not sense anymore. By considering this fact we see that the proposed algorithm achieves better performance. This means that clients display video with better quality when leverage *PressureInitialCache* algorithm. It is clear that by increasing $Lambda$ value clients have more freedom to adapt high bitrate video segments, and therefore when $Lambda = 30$ all algorithms have good performance, but again

PressureInitialCache has better adaptation with high-quality video representations. In Custom topology (Figure 4.16), when $Lambda$ value is set to 20Mb, the *HotSpot* algorithm outperform. This means that the concept of the locality, with considering clients' density and deploying cache as close as possible to clients, has a positive effect on the received video quality.

As mentioned before, by increasing network size and number of clients, re-buffering duration rises up in all algorithms due to the long distance between the source and destination nodes. The main advantage of our algorithm is installing caches in appropriate locations in such a way that it provides less distance and high bandwidth between clients and content sources. As a result, *PressureInitialCache* achievers to higher video quality with less outage time. Simulation results prove the efficiency of our algorithm in the first cache placement. In the next section, we will discourse about installing extra cache on demand.

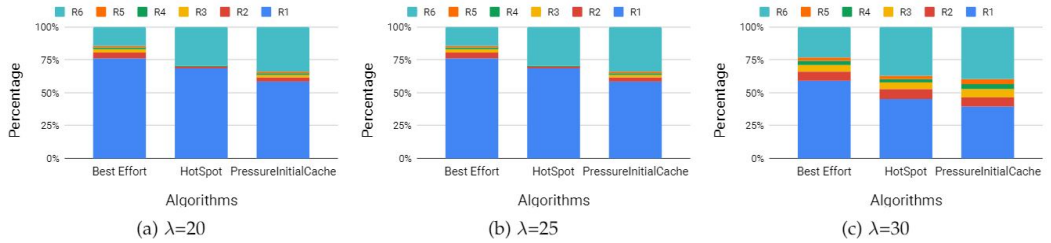


Figure 4.13. First Cache. Distribution of the received quality level – R1 to R6: Custom topology

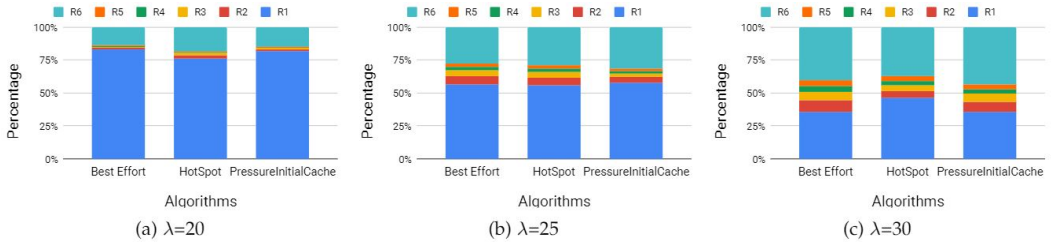


Figure 4.14. First Cache. Distribution of the received quality level – R1 to R6: Compuserve topology

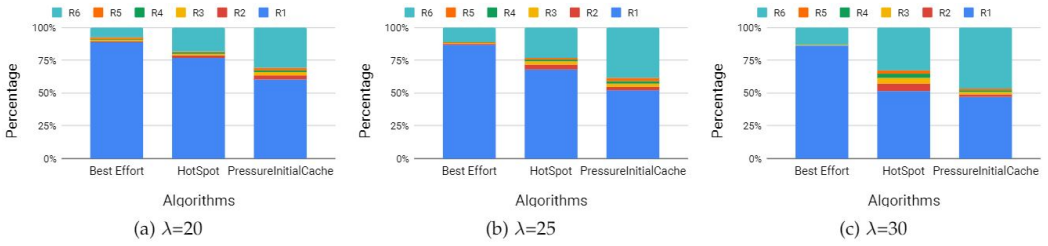
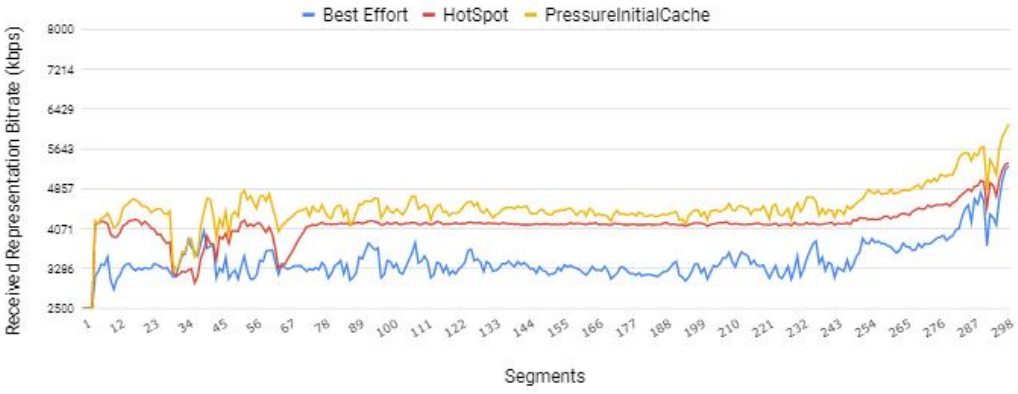
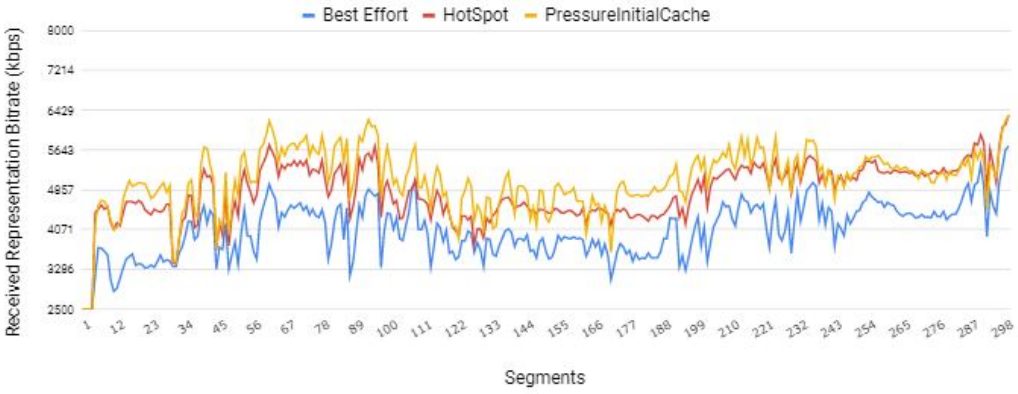


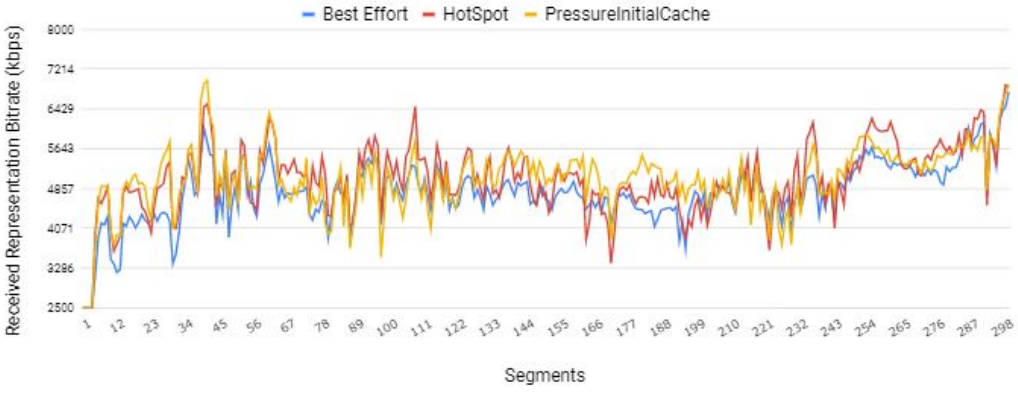
Figure 4.15. First Cache. Distribution of the received quality level – R1 to R6: BellCanada topology



(a) λ=20

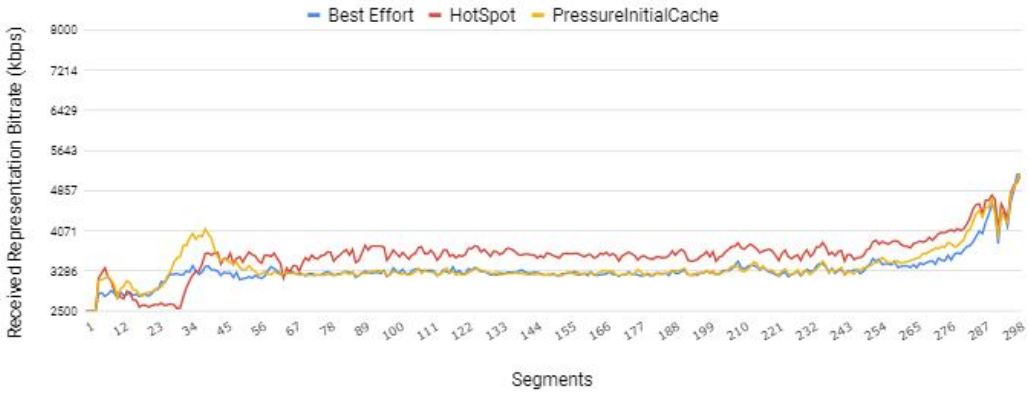


(b) λ=25

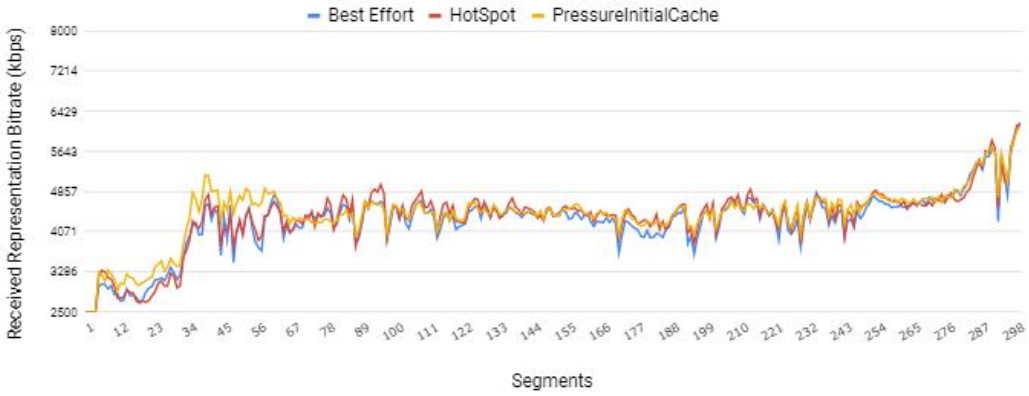


(c) λ=30

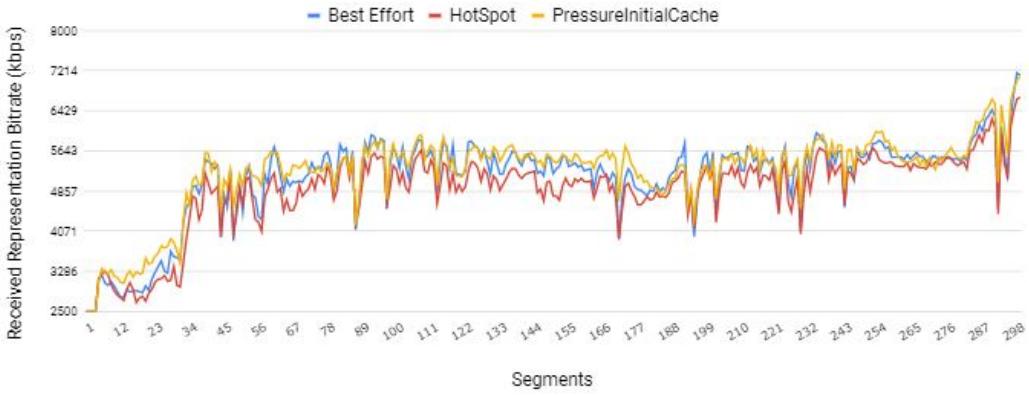
Figure 4.16. First Cache. Received bitrate per segment – Custom topology: (a) λ=20 Mbps, (b) λ=25 Mbps, and (c) λ=30 Mbps)



(a) λ=20

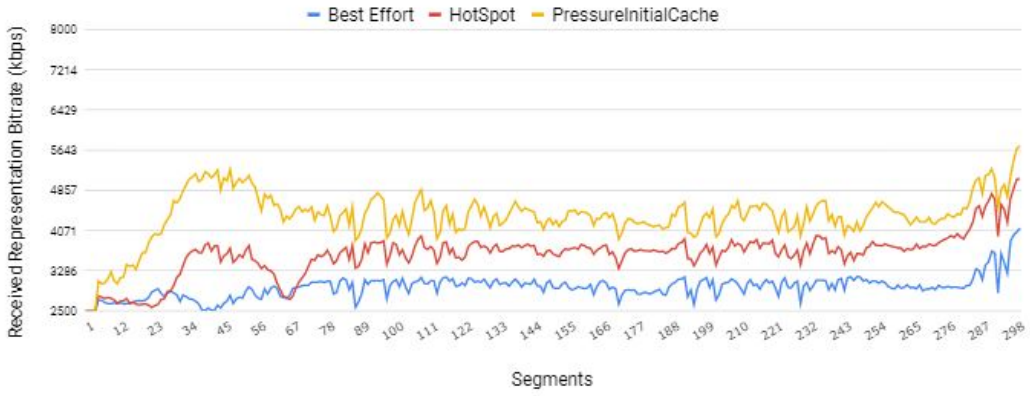


(b) λ=25

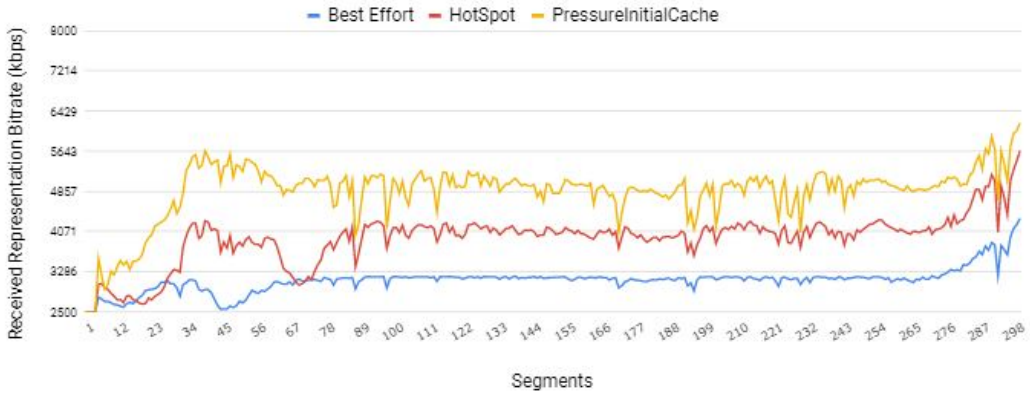


(c) λ=30

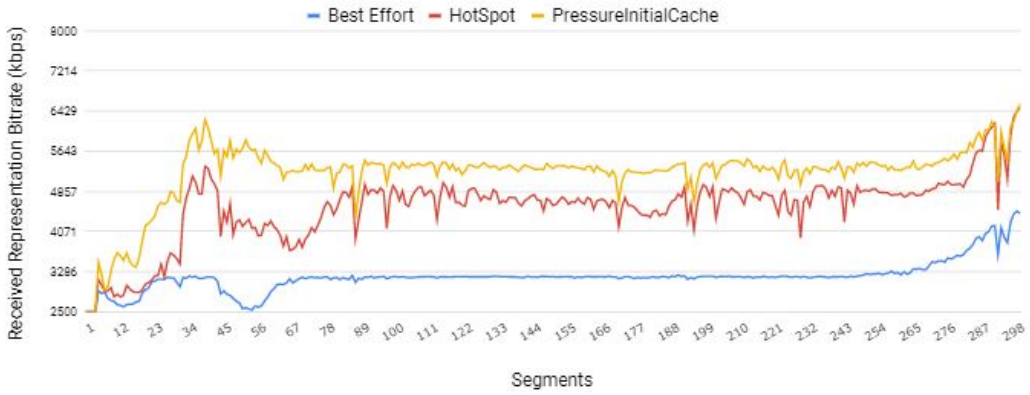
Figure 4.17. First Cache. Received bitrate per segment – Compuserve topology: (a) λ=20 Mbps, (b) λ=25 Mbps, and (c) λ=30 Mbps)



(a) λ=20



(b) λ=25



(c) λ=30

Figure 4.18. First Cache. Received bitrate per segment – BellCanada topology: (a) λ=20 Mbps, (b) λ=25 Mbps, and (c) λ=30 Mbps)

4.3.9 Additional Cache Installation

By analyzing the performance of installing virtual cache in different network topologies, it is possible to conduct that one cache often is not enough to keep the quality at a reasonable level. By increasing network size or number of clients, the additional cache should be installed to meet demand (Clayman et al., 2018). In lack of enough resources, clients cause lower video quality and more interruptions during display. We addressed this challenge in our previous work (Clayman et al., 2018) where the first cache randomly takes place. However, in this study, we provide a new solution based on a predefined cache location.

When new clients join the network, network traffic patterns changes and the SDN controller trigger *PressureInitialCache* algorithm in order to calculate the first cache location. After determining and installing the first cache location, the controller monitors clients received the video and decides to install a new cache if clients received video quality is under the predefined threshold. After installing a new cache in the new location, part of the network traffic removed from the first cache and forward to the new one. Typically, clients connect to the closest cache except in bottleneck conditions.

In order to the trade-off between distance and bandwidth, we introduced *formula* (4.8) which measures distance between each client with the first vCache and a new candidate one. It also considers available bandwidth provided by the candidate node. Candidate vCache should provide more bandwidth and less distance to connected clients. In this formula, $P_{j'}$ refers to the score of the second vCache position. $P_{j'}$ with the maximum score is preferred. Distance between client i with first and additional vCaches are illustrated with d_{ij} and $d_{ij'}$ respectively. Lets $L_{j'}$ indicates the total number of connected links to the candidate node j' and b_l is the bandwidth of each connected link. With respect to estimates suitable location for installing second vCache, we introduced $PressureCache$ algorithm which is developed based on the *formula* (4.8)

$$P(j^{'}) = \sum_{i\epsilon(H-h)} (d_{ij} - d_{ij'})) * \sum_{l=1}^{L_{j'}} b_l \tag{4.8}$$

With SAND technology, SDN has changed to new kinds of network management, decision making, and forwarding process. In general, clients are connected to the nearest cache and request video segments. However, when a client suffering

from a bottleneck link, the SDN controller with the assistance of the SAND technology informs the client to connect to the appropriate vCache. This process starts by sending a PER message to the client and enforces it connect to another cache in respect of enhancing client's received quality as well as managing network traffic. During this period, which called *hand-off* time, the client does not receive any video segment from the cache and in lack of enough segment in the buffer, it may experience re-buffering and interrupt video displaying due to delay.

Another important issue is related to how fast the cache placement or cache migration process in done. The fact is, in large scale networks, it is hard to assume the network snapshot at any point while network patterns frequently change. However, in the SDN domain, network monitoring can be achieved more efficiently by using OpenFlow statistics. Besides, clients periodically send average received bitrate to the controller. Suppose P denote video segment's size which is equal to 2 seconds in the *Big Buck Bunny* video asset and the client's buffer size is equal to 24 seconds of video. Video is going to display when a client has at least 8 seconds of buffered video. Typically, in order to fast startup, clients request the first 4 segments from lower representations which have less bitrate. Thus, during simulation client sends average received bitrate every 10 seconds -just bigger than $4 * P$ seconds. The network controller checks the whole online client's average bitrate and runs the $PressureCache$ algorithm if the total average is under the threshold level. This threshold value might be defined by the content provider. For example, in our study, we considered that each client should be able to display video with minimum quality without unacceptable delay. If the network pattern or client received quality does not change anymore, the controller waits for the next 10 seconds again without running the algorithm.

In the second scenario, the number of online clients has been increased to 35, 50, and 100 for $Custom$, $Compuserve$, and $BellCanada$ topologies respectively. Again, each topology has been tested with three different bandwidths (λ=20, 25, 30 Mb). While the number of online clients increased in each topology, an additional cache needs to be installed in a new location. The requirement for the additional cache is more evidence in the $BellCanada$ topology where the number of clients has increased 100%. As expected, simulation results show that the $PressureCache$ algorithm achieves better performance when compared with the $Best\ effort$ and $HotSpot$ algorithm - again good results.

Parameters	Average received video quality (Kbps)			Average startup delay (sec)			Average outage duration (sec)		
Bandwidth (Mbps)	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**
Best Effort	2791	2908	3333	10.4	12	8.6	64	45	10
HotSpot	3241	3393	3512	10.3	9	8.1	80	45	4
PressureCache	3385	3409	3547	13	10.3	11.9	27	23	9

Table 4.14. Performance results: Custom topology.

Parameters	Average received video quality (Kbps)			Average startup delay (sec)			Average outage duration (sec)		
Bandwidth (Mbps)	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**
Best Effort	2603	2935	3067	16.7	14.6	13.4	194	109	52
HotSpot	3093	3157	3291	14.1	14	13	110	37	55
PressureCache	2967	3326	3639	17.4	17	15	97	55	17

Table 4.15. Performance results: Compuserve topology.

Parameters	Average received video quality (Kbps)			Average startup delay (sec)			Average outage duration (sec)		
Bandwidth (Mbps)	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**	**λ=20**	**λ=25**	**λ=30**
Best Effort	2694	2837	2810	14.7	18.5	14.7	498	404	312
HotSpot	3084	3416	3487	14.6	15	14.5	55	30	27
PressureCache	14.5	15	14.5	82	27	23			

Table 4.16. Performance results: BellCanada topology.

In practice, it is expected that the proposed algorithm significantly outperforms two other algorithms in terms of QoE parameters. Results about received video quality, startup delay, and outage duration are shown in Tables 4.14, 4.15, and 4.16. In all cases, received video quality, startup delay, and outage duration tend to improve as the value of λ increases. In other words, when the network bandwidths are increasing DASH clients have a better adaptation to appropriate representation and experience better video quality.

In the context of startup delay, it seems that the proposed algorithm has a little limitation in the small size networks (e.g.,*Custom topology*), and it is expected because of less distance between clients and caches. However, in the $BellCanada$ topology, which is the biggest one in terms of size and number of online clients, the proposed algorithm outperforming than other algorithms.

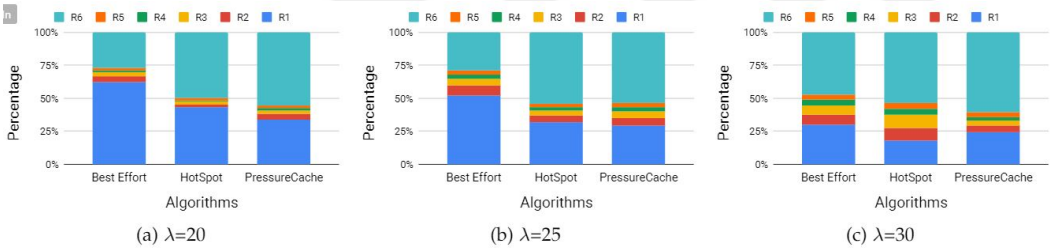
Similar to the first test results, the proposed approach derives to better performance in the second test as well, where we installed additional caches. Figures. 4.19, 4.20, and 4.21 represent the number of received segments from each representation. We found that installing vCache in any random location or even in the center of network topology never leads to better results. As seen, *Best effort* in all topologies has low performance. On the contrary, relying on locality issues is the main reason that $HotSpot$ and $PressureCache$ drives to better results. Also, the dynamic nature of the $PressureCache$ gives superiority to clients to download more segments from higher representation.

The emergence of deploying cache in the edges of the network is more evident when the size of the network increase. Because in large-scale networks, either packet propagation or intermediate forwarding devices' delay has a negative impact on packet delivery. Network delay cause to buffer drain in the client-side as well as impact the network throughput. Therefore, in both buffer base or throughput base adaption techniques client adapt to lower representation in order to avoid re-buffering. Hence, client receives video segments from lower representations.

Figures. 4.22, 4.23, and 4.24 shows long term downloaded video segment by the client in $Custom$, $Compuserve$, and $BellCanada$ topologies respectively. In all algorithms, at the beginning of the streaming, the client gets video from the lowest representation to fill the buffer and fast startup. Then it adaptively downloads video from the highest representation as possible. As a result, the $PressureCache$ algorithm earns better performance in a long time. In the middle of simulation new

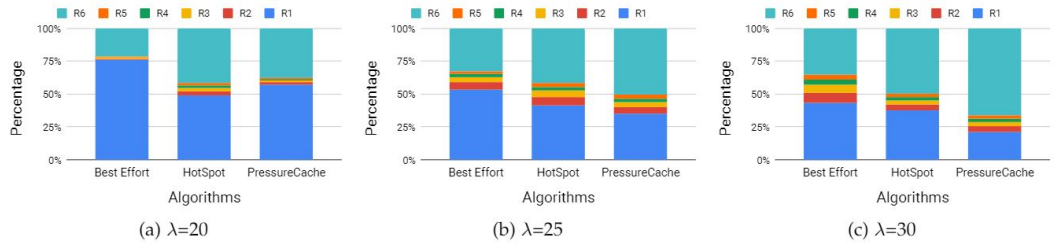
clients join the network and change the traffic patterns. An increasing number of online clients cause to install new vCache in a suitable location. As seen, in all topologies and configuration with different bandwidth values, the quality of the received representation increased because more vCaches provide more resources in the form of bandwidth and reduce the distance in the form of hop counts. Thus, generally in all algorithms and specifically in the proposed algorithm clients adapt to higher representations. The reason for better adaption in the $PressureCache$ algorithm mainly depends on (i) vCache placement and migration, and (ii) forwarding client to more proper vCaches.

When there is more than one vCache are available in the network, clients connected to the closest vCache. But, sometimes, pressure on one vCache is increases or cross-traffic cause to chocks paths. In such a condition, clients in both *Best effort* and *HotSpot* algorithms are still continuing to request video from the nearest vCaches. However, in the $PressureCache$ algorithm, the controller leverages SAND technology in order to improve clients as well as network performance. SAND archi-



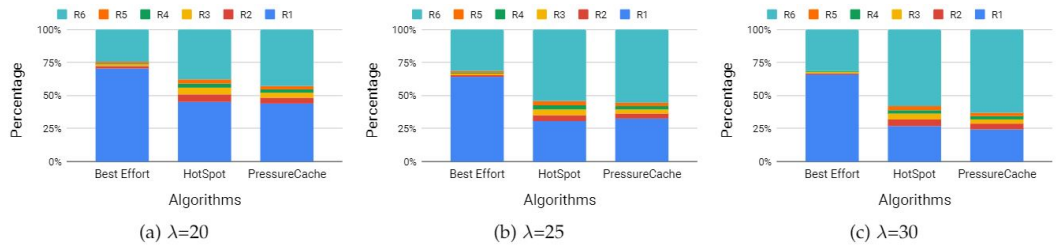
(a) λ=20 (b) λ=25 (c) λ=30

Figure 4.19. Additional Cache. Distribution of the received Quality level – Custom Topology



(a) λ=20 (b) λ=25 (c) λ=30

Figure 4.20. Additional Cache. Distribution of the received Quality level – Compuserve Topology



(a) λ=20 (b) λ=25 (c) λ=30

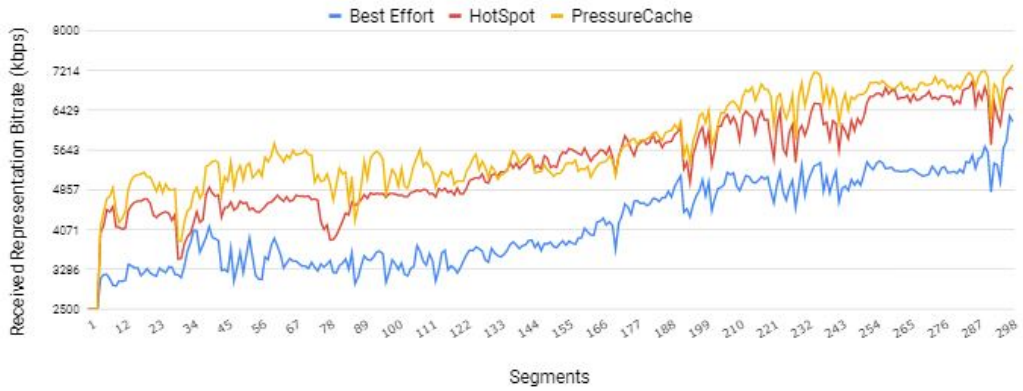
Figure 4.21. Additional Cache. Distribution of the received Quality level – BellCanada Topology

tecture sends a PER message to a dedicated set of the clients and enhances/enforce them to request a video from specific vCache even it was located in the far distance. When clients receive $Enforcement\ PER$ messages start Hand-off process and connect to the new vCache.

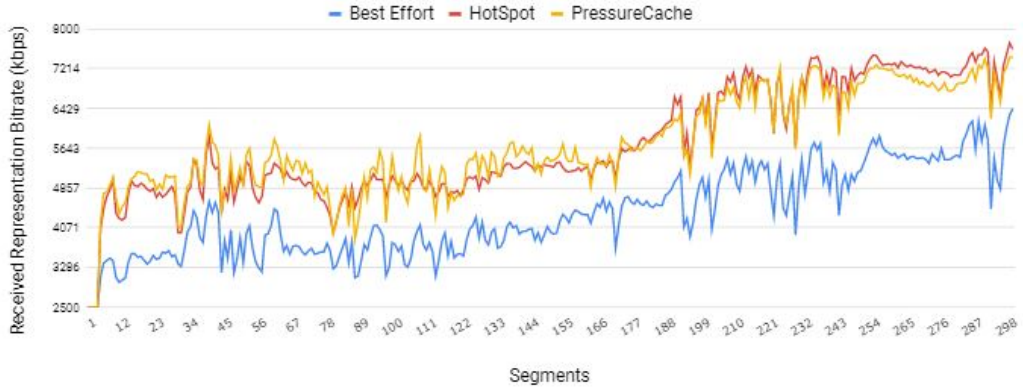
Figure .4.25, illustrates a single sample of the requested video segments during the first 240 seconds (120 segments) streaming of the $BellCanada$ topology in the proposed algorithm. Recall from Table .4.6, first representation (R1) $\rightarrow$ 1.0, R2 $\rightarrow$ 2.0 , R3$\rightarrow$ 3.0 , R4 $\rightarrow$ 4.0 , R5 $\rightarrow$ 5.0 , R6 $\rightarrow$ 6.0. At the beginning of the streaming, clients request the video from the first representation in order to fill the buffer and fast startup. After that, the client adapts to a suitable representation based on throughput estimation. In the second 60 th, an increasing number of online clients or cross-traffic affects client adaption. Hence received video bitrate on clients' side decreases. Moving vCache into a new location or installing additional vCache in the appropriate location improves clients' resource availability in the form of bandwidth and distance and clients starting to request and receives video from high bitrate representations. In the 120 th second, we see that client request video from the highest possible representation. Joining more and more clients into the network force controller runs an additional vCache installation algorithm and deploy new vCache in the appropriate location and forwards the client request to suitable vCache. Effect of installing additional vCache clearly visible after passing 160 seconds (or segments 80).

4.4 Cache Migration And Hand-Off Latency

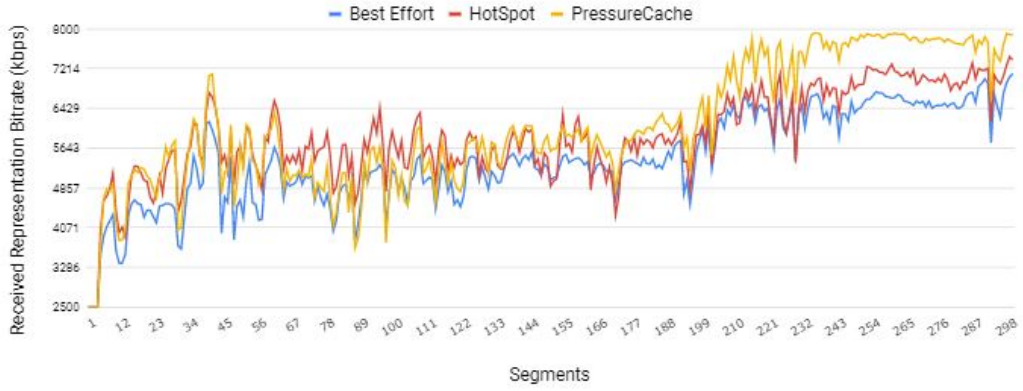
As mentioned before, after starting the initial *vDANE* instance, the controller continues collecting statistics of the network and monitoring DASH and cross-traffic patterns. For this purpose, we deploy a SAND metrics server as a northbound application running in the SDN controller. The clients periodically send the current buffer fullness level to the controller. Note that, the traffic introduced by status and metrics messages can be negligible when compared to traffic caused by the transmission of the video files. When the distribution or number of the clients in the system changes or it determines a change in traffic amount, the controller calculates pressure values for each node again. If the location of *vDANE* is changed according to the new outcome of pressure values, the controller sends PER enforcement messages to the clients to change their server. In addition, the shortest paths between the clients and the new *vDANE* location are selected for each client for streaming packets.



(a) λ=20

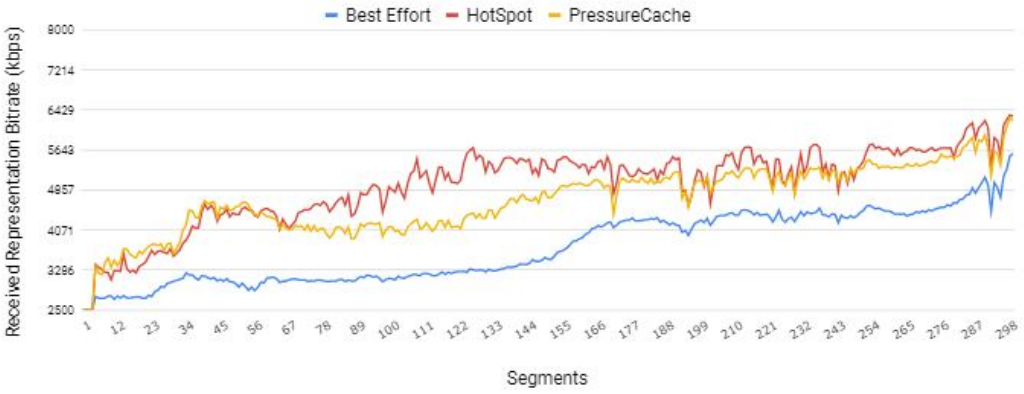


(b) λ=25

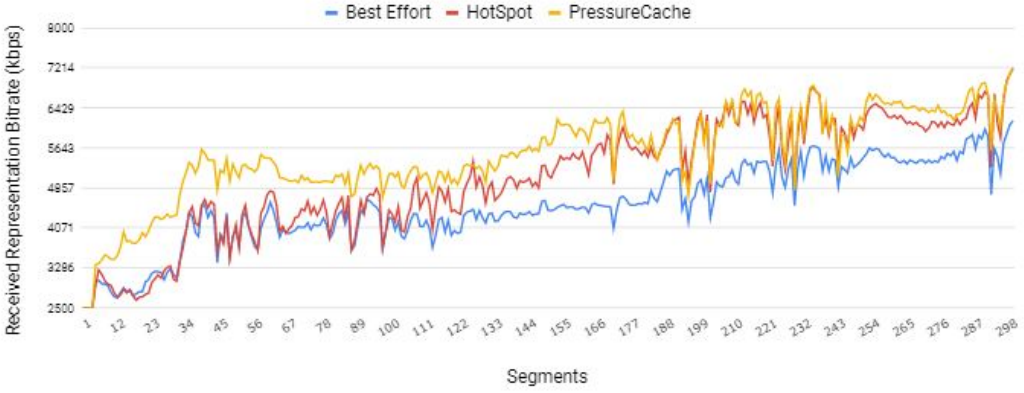


(c) λ=30

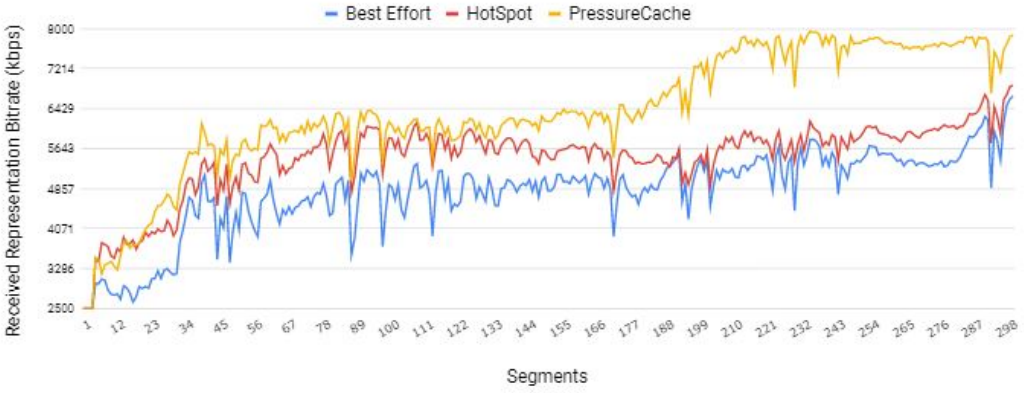
Figure 4.22. Additional Cache. Received bitrate per segment – Custom topology: (a) λ=20 Mbps, (b) λ=25 Mbps, and (c) λ=30 Mbps)



(a) λ=20

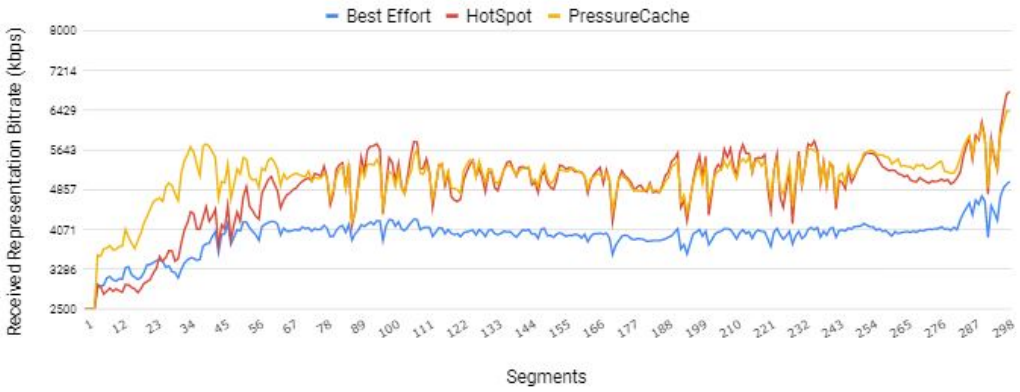


(b) λ=25

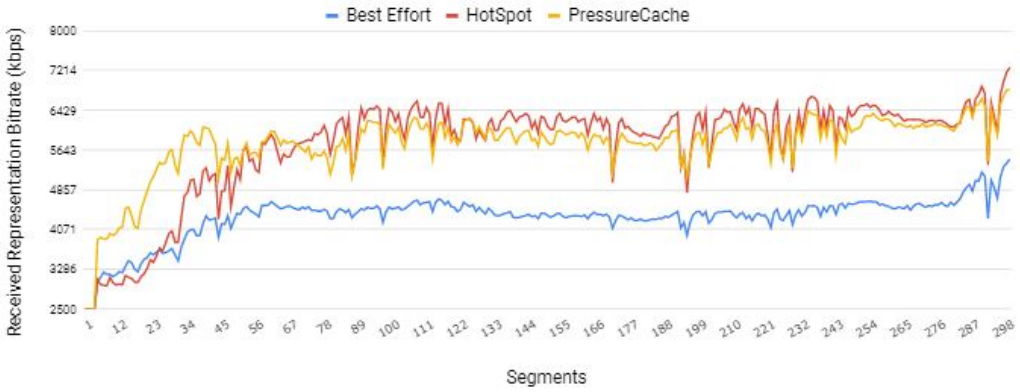


(c) λ=30

Figure 4.23. Additional Cache. Received bitrate per segment – Compuserve topology: (a) λ=20 Mbps, (b) λ=25 Mbps, and (c) λ=30 Mbps)



(a) λ=20



(b) λ=25

(c) λ=30

Figure 4.24. Additional Cache. Received bitrate per segment – BellCanada topology: (a) λ=20 Mbps, (b) λ=25 Mbps, and (c) λ=30 Mbps)

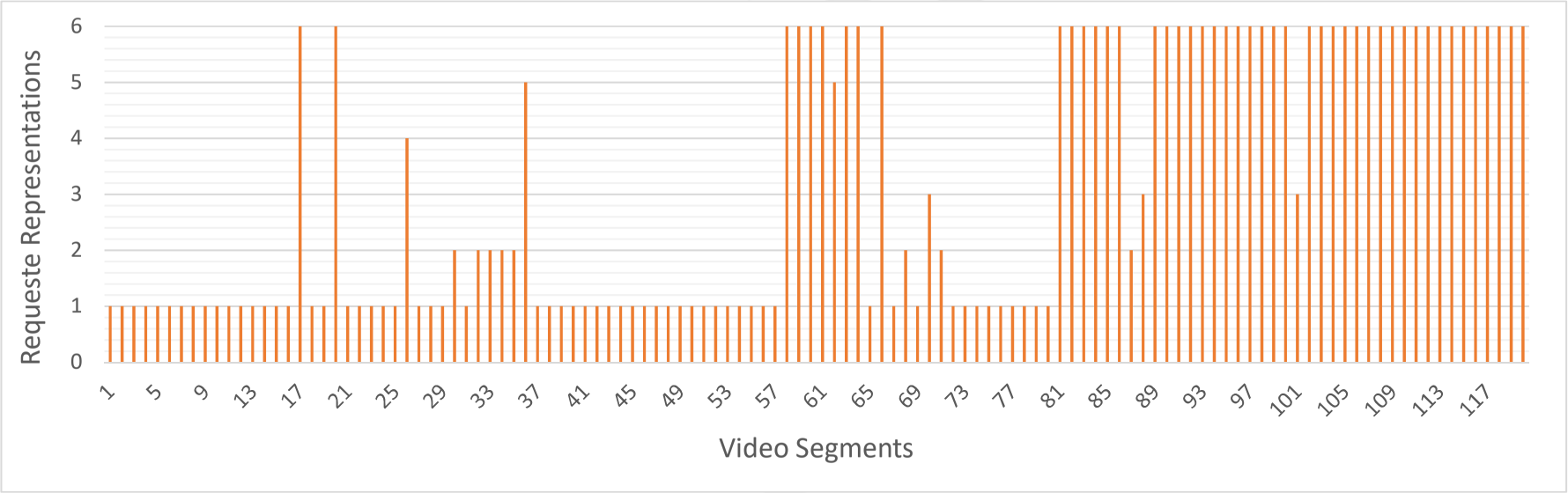


Figure 4.25. Clients adaption and requested representations per segment – BellCanada topology:

However, the experiments show that this approach has a drawback. When *vDANE* is migrated to its new location, a small change in network condition -for example connecting a few new clients close to the first *vDANE* location - may result to the minimum pressure value at the previous location and consequently, SDN controller triggers the migration process again to migrate *vDANE* to its previous location. This process continues unless remarkable changes reshape the traffic pattern completely. We call this phenomenon as *Ping-Pong* effect.

Algorithm 5 shows the cache migration algorithm run by the SDN controller. To avoid the $Ping - Pong$ effect and minimize the number of migrations by eliminating the unnecessary ones, we use a different metric to be utilized in the migration decision process executed by the SDN controller. The difference metric is calculated as $1 - (P'/P)$, where P and P' refer to PIC score of old and new locations of *vDANE*, respectively. And it should be above a certain threshold value, δ, for the migration process to start as given in the first line of the algorithm.

Each client has a buffer with a size of b seconds, which can store a maximum b/t segments, where t is the size of a segment in terms of seconds. Supposing fast startup and also avoid re-buffering, it is assumed that the client requests the first 4 segments ($4 * t$ seconds length) with a minimum bit rate representation. In 4^{th} line, algorithm checks buffer fullness and timer. If the client has enough segment in its buffer, then the SDN controller enforces the client to start the next request form new vCache. A client with a more buffered segment has less probability to enter the buffering stage during switching between current and new vCache.

Algorithm 5: *Cache Migration* Algorithm.

```
   Input:
   δ: threshold value for migration
   P: PIC value of the current vDANE
   P′: PIC value of the new vDANE
 1 if (1-(P'/P) >= δ) then
 2     run timer
 3     foreach client = 1 to H do
 4         if (buffer fullness>= thr) or timer>= thr then
 5             update routing devices forwarding tables
 6             enforce client request seg. from new vDANE
 7         else
 8             continue
 9         end
10     end
11 end
```

The proposed algorithm forces the clients whose buffer contains more data to connect to the new *vDANE* first. This approach brings two advantages. First, while the clients have more buffered data, it takes time to buffer to get empty and experience re-buffering except clients have less buffered data. Second, by moving the clients to new vCache in two phases, i.e. migrating one client group after another, the clients which are still connected to old vCache may utilize more prosperous bandwidth and download some segments faster before starting hand-off.

The evaluation of the proposed cache algorithm is based on two QoE parameters; throughput or received bitrate, and buffer fullness or re-buffering duration. We observed that our architecture with *vDANE migration* provided improvement in throughput when compared to *Best effort* and $HotSpot$ approaches, up to 14% and 7%, respectively. Furthermore, the proposed algorithm has better performance in terms of quality metrics when compared with our previous study (Kalan et al., 2019). Considering buffer fullness indicates that our algorithm also a decrease in re-buffering duration 50% and 82% when compared to *Best effort*, $HotSpot$, respectively, which shows the good impact on QoE of our approach.

4.5 Conclusion

The optimally of the cache locations is dependent on the network traffic patterns, client distribution, and mobility during the time. Hence, the candidate node for hosting cache should be close to high-density locations where there are more clients connected to the network. Providing caching at all network elements is technically feasible, however, caching content in all network elements can affect network performance because of extra overhead. Caching content only in dedicated forwarding devices reduces content redundancy and increases the cache hit rate.

Pushing content to edges reduces access time while the client request content from a cache as close as possible. By analyzing the performance of installing virtual cache in different network topologies, it is possible to conduct that one cache often is not enough to keep the quality at a reasonable level. By increasing network size or number of clients, additional vCache should be installed to meet demand. In lack of enough resources, clients cause lower video quality and consequently more interrupt during display. To address this problem additional vCaches need to be deployed in appropriate locations.

Rely on the fact that adapting requests based on a global view of the network could provide better adaption, the network-assisted approach has been proposed in this study. Introduced algorithm with leveraging SAND technology and *NFV-SDN* architecture gives more intelligence to a client to better adapt. Simulation results show that the proposed algorithm improves network resource utilization and QoE on clients' side by installing virtual cache near to optimal location and handling cache emigration based on client distribution and network traffic pattern.

Simulation results in all the three studies show that the proposed algorithms achieve better performance compared with *Best effort* and $HotSpot$ algorithms. leveraging SAND technology in *NFV-SDN* gives more intelligence to the network controller and clients to better adaptation. Such intelligent, augmented network elements utilize network resources in a more efficient way. As a consequence better QoS for service providers and higher QoE for end users.

$Note$: Part of this study has accepted in *23rd Conference on Innovation in Clouds, Internet and Networks (ICIN 2020)*-Paris.

The comprehensive results of this study is ready for submission in journal of *Multimedia Tools and Applications*

5. IMPLEMENTATION OF SAND ARCHITECTURE USING SDN

The SDN programmable infrastructure offers a wide range of flexibility for designing application-specific routing algorithms and the SAND architecture that provides clients to make more intelligent decisions about rate adaptation. In this study, we propose an SDN-based architecture and a SAND-aware DASH client that improves the viewer experience. The simulation results show that this architecture can help improve the performance of DASH clients as well as provide fair sharing of the network resources among the clients. The results also show that the proposed architecture can adapt to changing network and server conditions with the help of SDN.

5.1 Introduction

The $DASH$ standard developed by the $MPEG$ defines the format and functionality of the media presentation description that includes the media-related information and media partitions (Sodagar, 2011). The representations encoded at various qualities stored on a DASH server while segmented into smaller portions. This segmentation provides DASH clients to adapt quality by requesting segments of different representations over time. DASH clients request the video quality according to their rate adaptation algorithm. The rate adaptation algorithm may select the representation to be requested by considering bandwidth (Rainer et al., 2012) (Fouladi et al., 2018), buffer status (Spiteri et al., 2016) or both (Mao et al., 2017). An extensive survey on the subject is (Bentaleb et al., 2018).

The goal of the rate adaptation algorithms is to achieve the best quality QoE under the estimated network conditions based on the observations made at the client-side. Yet, the clients can potentially do a better job, if assistance from the server or network is provided. The *MPEG* has recently completed the standardization of the SAND. SAND proposes an architecture where the clients get information and/or guidance from $DANE$ and server(s), where these elements aim to help clients achieve better QoE. Besides, various approaches such as enhancing the delivery and altering encoding of the video files are also considered (Thomas et al., 2017).

In this work, we propose a SAND architecture using SDN elements. In proposed SAND architecture, the cache servers act as a virtual DANE introduced in our earlier works (Clayman et al., 2018) (Kalan et al., 2019). Here, we consider a video streaming provider that disseminates video files by using virtualized CDN servers. Also, SDN is used for carrying video data packets that are routed through the back-

bone network. The contributions of this paper are twofold. First, we implement the SAND protocol by using SAND messages and SAND network elements, and second, we develop a SAND-aware DASH client that adapts video quality jointly considering the SAND messages and its internal parameters such as buffer level and estimated throughput.

5.2 HTTP-Based Streaming Over SDN

Since SAND is a recently proposed architecture, a study giving the details and performance results of an SAND implementation is not more proposed in the literature yet. The first proposals related to DASH over SDN in the literature can be seen as the antecedent works of SDN supported SAND architectures. In (Seddiki et al., 2014), the authors proposed a QoS model by introducing a traffic shaping method that defined packet forwarding rules according to the type of the protocol. By assigning priority to the HTTP flows, DASH clients achieved a higher throughput when compared to the case that traffic shaping was not used. To provide fairness among DASH clients, authors in (Georgopoulos et al., 2013) proposed that the clients requested bitrates according to the commands received from the controller. No routing strategy for the flows was proposed in this work. In (Nam et al., 2014), HTTP streaming by using SDN capabilities was proposed. In the study, the controller periodically obtained information such as re-buffering events and playing information from the clients. According to the status of the clients and obtained network information from the SDN, the controller decided to change the server or the streaming path in the SDN domain. Video streaming for transferring layered video data over SDN networks was discussed in (Kalan et al., 2017), where the controller recommended a suitable video layer for DASH clients based on network conditions.

Using network devices to assist clients allows them to adapt the streaming rate better to the changing network conditions. Authors in (Kleinrouweler et al., 2015) implemented an adaptation assistant in the form of the HTTP proxy server. The proxy server processes the manifest to obtain the characteristics of the stream that are later used for fair sharing of the bandwidth.

A good view of using SAND architecture in SDN networks was given by Kleinrouweler (Kleinrouweler et al., 2016), focusing on controller assisting clients to improve QoE. Using SAND was also discussed in (Kleinrouweler et al., 2017), where the authors proposed assisted adaption rule algorithms based on DANE and $BOLA$ (Spiteri et al., 2016) functionality concerning decrease the number of quality switches and reduce playback outages. Bitrate adaptation algorithm for DASH

by considering wireless bottleneck utilizing the SAND architecture was discussed in (Li et al., 2016). Another contribution toward improving QoE in HTTP adaptive streaming by taking advantage of flexibility introduced by the SDN and SAND was discussed in $SDNDASH$ (Bentaleb et al., 2016). However, the proposed methods did not consider changing the path and media server while the network path was congested.

5.3 Server And Network-Assisted DASH

In the SAND architecture, there are network entities called DANEs, server(s), clients as well as regular network elements. DANEs are used for enhancing QoE by sending messages to the clients, servers and other DANEs. Cache servers or *third-party servers* (TPS) that are aware of DASH manifests and segments, and receive clients' metric and status messages can be considered as DANEs (Thomas et al., 2016).

PED, PER, and client status and metric messages are defined as SAND messages. While PED messages are exchanged between two DANEs, or a server and a DANE; PER messages are sent to the clients. For example, a cache server can send a PER message indicating the available segments on the cache to the clients, hence, the clients can request the segments that exist on the cache incurring a shorter delay. In this example, the cache server is a DANE for being aware of DASH segments and sending messages to the clients to improve quality. A status message can be sent by a client to a cache server, indicating the segments to be requested soon, and this way, the cache server can pro-actively fetch the related content (Thomas et al., 2016). The messages sent to the clients can be either an assistance message or an enforcement/error message. Assistance messages can be used by clients to improve their quality experience. Enforcement or error messages require clients to operate in a particular way (Thomas et al., 2016).

In the next section, we give the implementation details of the proposed architecture in which we define the controller and $OpenFlow$ enabled switches as DANEs, and the messages sent by the controller as PED messages. We also give the details of possible examples of PER messages and proposals for enforcement and assistance messages sent by DANEs to the clients. Our study differs from the literature since it proposes a complete SAND architecture to use SDN components as well as a novel approach for rate adaptation for the SAND-aware DASH clients.

5.4 SAND Architecture Implementation Over SDN Backbone Network

5.4.1 Network Assistance via SDN

In this section, we give the implementation details of the proposed SAND architecture using SDN. In this architecture, we use an SDN network as the backbone. The controller is responsible for managing this backbone SDN network. Other entities of the ISPs in SAND architecture such as origin server, CDN servers, and the clients are not necessarily part of the SDN network. We define the controller and OpenFlow enabled switches in SDN networks as DANEs. A TPS runs as a northbound application at the controller, which is responsible for collecting the metric and status messages from the clients, and sending assistance and enforcement messages to them. The clients periodically send the information about the selected representation and buffer fullness to the TPS application at the controller. TPS application can be considered as a bridge between the video streaming provider and backbone network management. Thus, TPS can have information regarding the upload bandwidth capacity of the CDN servers. It keeps track of the number of clients connected to each CDN server to measure how much bandwidth is left on each server.

In addition to the northbound applications, several modules are running on the controller as shown in Figure. 5.1. Following this architecture, both the clients and CDN servers communicate with the TPS, which collects high-level abstraction of the network resources including connection points, available paths, and traffic patterns. The TPS converts the service request into network service functions. The *Path Manager* module determines the paths on the backbone SDN network between the clients' ISPs and the CDN servers' ISPs on the Internet. The *Link Manager* module measures the active traffic and calculates the available bandwidth of these paths. Both the *Path Manager* and *Link Manager* modules transmit the gathered information to the TPS, which can decide to change the streaming path according to the heuristic given below and pass new path information to the *Flow Manager* module that sets the path between the server and clients by signaling the switches.

As mentioned earlier, the TPS collects streaming performance related parameters via metric and status messages sent by the clients. Let *abwS* represents the available bandwidth of CDN server *S* and *abwS-C* represents the available bandwidth of the backbone path between server *S* and client *C*. The TPS measures an effective available bandwidth by calculating *min(abwS, abwS-C)* values for each server-client pair. There may be more than one path between a server and a client.

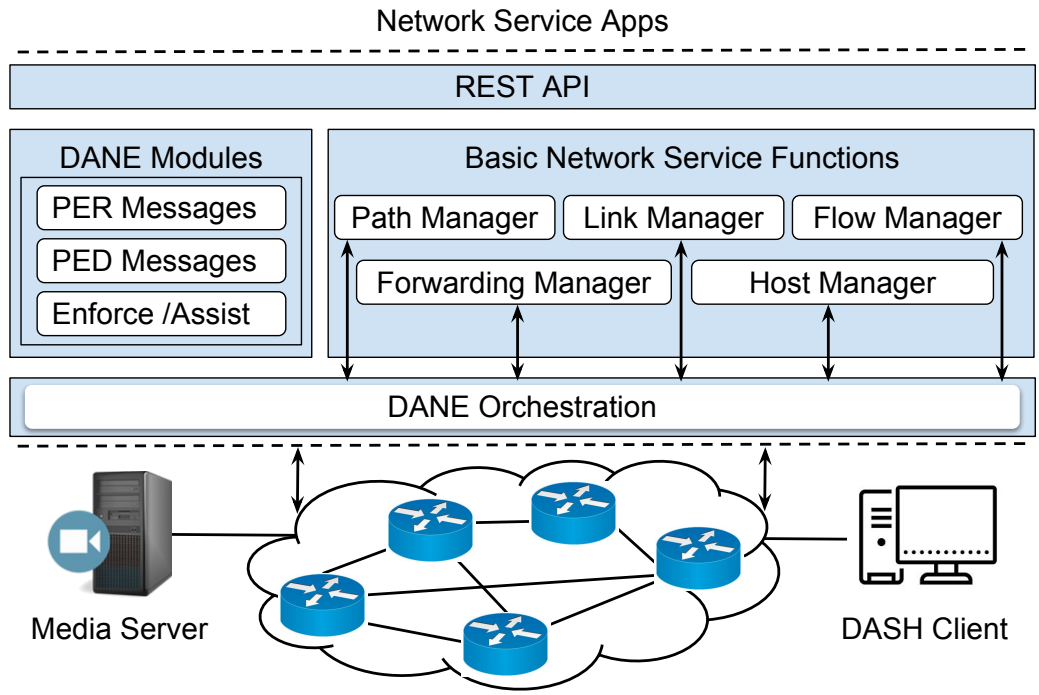


Figure 5.1. Illustration of the SDN functions in the proposed system architectur.

The best routing path pair providing the maximum effective available bandwidth for each client in the system is determined as the optimal server-path pair. Optimal server-path pair can be dynamically updated according to the network conditions.

When a client starts the video streaming application, the local DNS of the client sends a DNS request to the authoritative DNS of the CDN company. The first DNS request message sent by the client, is first retrieved by an $OpenFlow$ switch at the backbone. While the switch has no entry for the new client in *Flow Tables*, it forwards the request to the TPS application, which determines the optimal server-path pair for that client, and sends the following messages:

- A PED message carrying the IP address of the selected CDN server to the authoritative DNS of the CDN company to forward that address to the local DNS of the client.
- A PED message to the OpenFlow switches on the selected path so that the switches update their flow tables according to the route selected by the TPS.
- A PER message to the client carries the suggested representation for requesting the next segments. This representation can be determined by considering the bitrate of the representations or the effective bandwidth value.

The client starts sending periodic metric and status messages after starting to download segments from the server. When the TPS receives a status message indicating the requested representation and buffers fullness ratio from the client, the TPS first checks whether the client switched to lower quality or the buffer fullness is at a critical level. If quality degradation or critical state in buffer level is detected, and the backbone path is congested or the current server of the client is heavily loaded, the TPS determines the optimal server-path pair again for that client. At this point, there are two possible cases:

- *Case 1:* The effective available bandwidth of the optimal server-path pair is higher than the bitrate of the requested representation. In this case, the TPS sends the following messages:
  - A PED message to the $OpenFlow$ switches about the selected path as mentioned above.
  - A PER message to the client including an enforcement message signaling the IP address of the new server, if the server is changed, as well as an assistance message carrying a suggested minimum representation bitrate indicating that client should not request a representation with a lower bitrate.
- *Case 2:* The current server-path pair is the optimal server-path pair for that client, or the capacity of the optimal server-path pair is limited. In this case, the backbone is congested and the TPS sends the following message:
  - A PER message to the client including an enforcement message signaling that the client should not request a bitrate higher than the specified value.

The TPS has the responsibility to forward the client requests to a suitable server via the best routing path. When the network is congested or the server load is high, the TPS may assist or enforce the clients to request a video encoded at a specific bitrate. According to the algorithm given in *Algorithm 6*, the client does not request a representation with a bitrate higher than the suggested one, if it receives an enforcement message. A similar approach can be taken when an assistance message is received and the client agrees to implement the suggestion. For example, at the beginning of the streaming session, this decision can be made according to the user's preference. The client implements the decision each time sending a request to the server until it receives from the TPS a reset message. If the client accepts the

suggestion when it receives an assistance message, it sends a message indicating its decision to the TPS. The TPS may use this information for further analysis of the consequences of the selected actions.

5.4.2 SAND-Aware Client Design

In this section, we give a rate adaptation algorithm for SAND-aware DASH clients. In the proposed system, the DASH client receives PER messages from the TPS. According to the received type of PER message, the clients run the algorithm given in **Algorithm 6**. The client decides on the next segment based on the buffer level by using the BOLA algorithm and requests the video from the lowest available bitrate, if there is not enough data in the playback buffer, i.e., the buffer occupancy is below the threshold. Otherwise, the client can request the video from higher bitrates, which may be limited by the DANE. In the case of assistance, the client sets the next bitrate to be requested to the minimum of the outcomes of the BOLA algorithm and the DANE. This ensures that the client never requests a bitrate higher than the one signaled by the DANE. This way, the TPS provides a fair sharing of the network resources among the clients. In case an enforcement message is received, the client starts requesting the segments from the new server suggested by the TPS. Network traffic engineering and resource management are two main outcomes of this awareness that lead to better QoE achieved by the clients.

5.5 Testbed And Simulation Setup

Mininet emulator was used to evaluating the performance of the proposed system. In the Mininet topology, there are more than one CDN server, clients and a controller. DASH clients communicate with the controller and exchange status message as explained in the previous sections. DASH clients send and receive their packets through their access networks, which are connected to the core (backbone) network. As a controller, we used the $FloodLight$ controller, which was developed by open community developers. Also, for testing and evaluation of the proposed system, we implemented a Java-based DASH client. The illustration of the simulation topology is given in Figure. 5.2.

Big Buck Bunny (ITEC, 2018) was used for streaming video during the simulations. DASH clients join the network based on *Poisson* distribution and start requesting a suitable representation after downloading the manifest file. The average interval time and the number of DASH clients are set to 2 and 10, respectively. Also, we used $Compuserve$ network topology with two different parameter set-

tings. This topology is a real network topology consisting of 11 switches and 14 links. In the settings, *Poisson* distribution with different mean (λ) and standard deviation (σ) values were used for generating the network link bandwidths. In the first set up, the standard deviation was adjusted to ($0 \leq \sigma \leq 5$), which means all network links had roughly the same bandwidth. In the second set up, the standard deviation was higher ($5 \leq \sigma \leq 15$), thus, the network links had different bandwidth values.

Algorithm 6: *Assisted DASH Adaption* Algorithm.

```
   Input:
         qBola: rep.id determined by BOLA
         qDane: rep.id received from DANE
         selected rep.id : determined by the client
         requested rep.id : HTTP GET to the server
         fDane: true if previous rep.id equals qDane
 1 let rep.id point to representation ID;
 2 foreach HTTP GET request do
 3     requested rep.id ← min(qDane , qBola);
 4     switch last message type received from the TPS do
 5         case assistance message do
 6             if ((buffer.level >= threshold) OR fDane) then
 7                 if (qBola >= qDane) then
 8                     fDane← true;
 9                 else
10                     fDane← false;
11                 end
12             else
13                 fDane← false;
14             end
15             return requested rep.id;
16         end
17         case enforcement message do
18             if (message carries a new server IP ) then
19                 connect_to_new_server();
20             end
21             if (message carries a rep.id ) then
22                 return rep.id;
23             end
24         end
25         case reset message do
26             requested rep.id ← selected rep.id;
27             return requested rep.id;
28         end
29     end
30 end
   Output: Requested representation
```

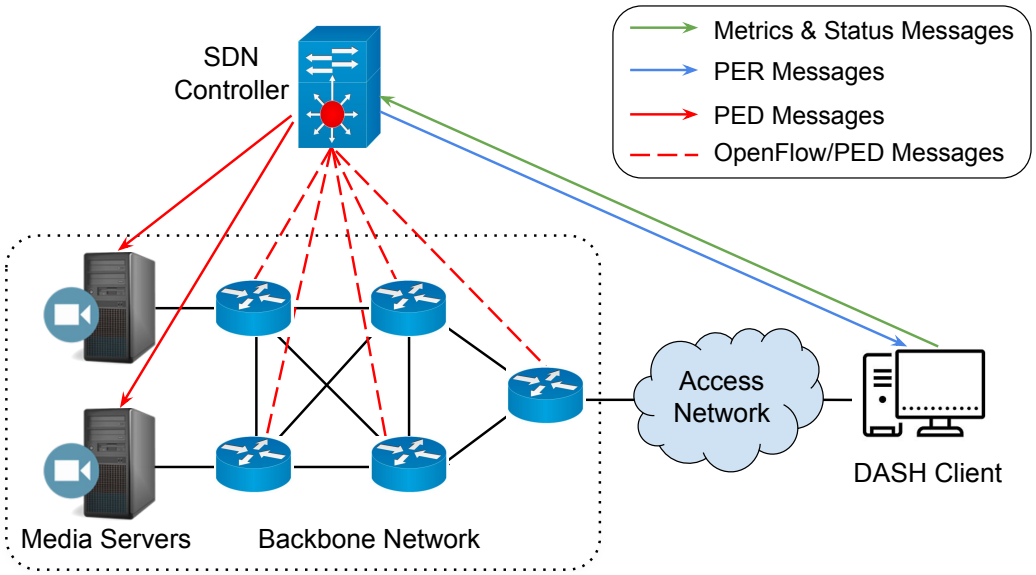


Figure 5.2. Illustration of the simulation setup in the proposed system.

5.5.1 Performance Evaluation and Experimental Results

In order to investigate the performance of the proposed approach, we conducted a set of simulations with a focus on the QoE metrics: (i) average received bitrate, (ii) outage duration (re-buffering), (iii) startup delay, and (iv) number of the requested segments from each representation. We compared the proposed system with two different approaches, where one of them is the conventional (in terms of rate adaptation algorithm) system with DASH client that is not supported by DANE, and the other one is the assisted adaptation rule system (Kleinrouweler et al., 2017) utilizing DANE and BOLA (Spiteri et al., 2016). We refer to this second system as DANE-assisted BOLA. While DANE-assisted BOLA selects an appropriate video representation based on the current buffer level and the messages received from the DANE, in the conventional system, the client requests a suitable video representation based on the calculated throughput without the network support. Each simulation was repeated five times and averaged values of the results are presented. It is also worth emphasizing that the buffer-based approach has been tested as well, however, its performance was similar to the throughput approach, and hence, its results are omitted here.

Parameters	(a) Average received video quality (Kbps)			(b) Average startup delay (sec)			(c) Average outage duration (sec)		
Bandwidth (Mbps)	λ=**20**	λ=**25**	λ=**30**	λ=**20**	λ=**25**	λ=**30**	λ=**20**	λ=**25**	λ=**30**
Conventional	2370	3431	4908	7.1	6.7	5.8	277	115	40
DANE assisted BOLA	2405	3214	5235	5.1	5.4	5.8	95	27	43
Proposed	2510	3443	5584	5.3	5.5	4.7	94	27	2

Table 5.1. Simulation results, with $0 \leq \sigma \leq 5$.

Parameters	(a) Average received video quality (Kbps)			(b) Average startup delay (sec)			(c) Average outage duration (sec)		
Bandwidth (Mbps)	λ=**20**	λ=**25**	λ=**30**	λ=**20**	λ=**25**	λ=**30**	λ=**20**	λ=**25**	λ=**30**
Conventional	2838	3679	8900	10.2	10.8	10.8	342	340	78
DANE assisted BOLA	2877	3669	9317	10.1	9.6	11.8	364	330	80
Proposed	3864	4380	12510	7.2	8.2	7.0	31	43	1

Table 5.2. Simulation results, with $5 \leq \sigma \leq 15$.

In Table 5.1 and 5.2, throughput, startup delay and outage duration values are given for all approaches. The results show that the proposed approach outperforms the other approaches in terms of the measured QoE metrics. Table 5.1(a) shows that DANE-assisted clients can use network bandwidth and increase received bitrate even requesting from higher bitrate representation has been restricted by DANE enforcement messages due to congested network or limited bandwidth. Besides, in the proposed approach, PER messages cause clients to request video representation from the proposed server, which has high available bandwidth and lower load. Therefore, the clients in the proposed approach achieve better video representation than other approaches. Furthermore, fair bandwidth utilization provided by DANE reduces startup delay as shown in Table 5.1(b).

In the proposed approach, SAND elements orchestrate video streaming components in case of traffic volume changes. When network traffic changes rapidly, PER messages direct the clients to request representation from a suitable server while the controller selects paths that have adequate bandwidth to transmit video packets. An increase in traffic volume leads to a reduction in download speeds and causes buffer draining on the client-side. As a result, video playback may pause. In Table 5.1(c), total outage durations experienced by the clients are presented for all approaches. It is observed that the clients in the proposed approach have less buffering outages compared to the other approaches. The main reason is that the proposed SAND architecture directs the clients to select appropriate representations by enforcing them not to exceed the quality suggested by the DANE when the network bandwidth is limited. The proposed approach performs better when $5 \leq \sigma \leq 15$ as observed from Table 5.2. In the proposed approach, the DANE may suggest or enforce the client to request specific representation from a specific server when the traffic volume changes. Hence, the clients may make a better decision to adapt quality based on the PER messages sent by the DANE.

Figure. 5.3(a) shows the percentages of the received segments per representations. As mentioned before, L1 and L6 refer to the lowest and highest representation bitrate, respectively. Receiving more video segments with a lower bitrate naturally decreases the quality. In general, the proposed approach outperforms the two other methods. Since the conventional and DANE-assisted BOLA approaches receive more segments from the low quality representations, the clients using these approaches play a video with lower quality than that of the proposed approach. The performance differences between the approaches become clearer when the graphs are given in Figure. 5.3(b) are examined, where the network links have more asynchronous bandwidth values.

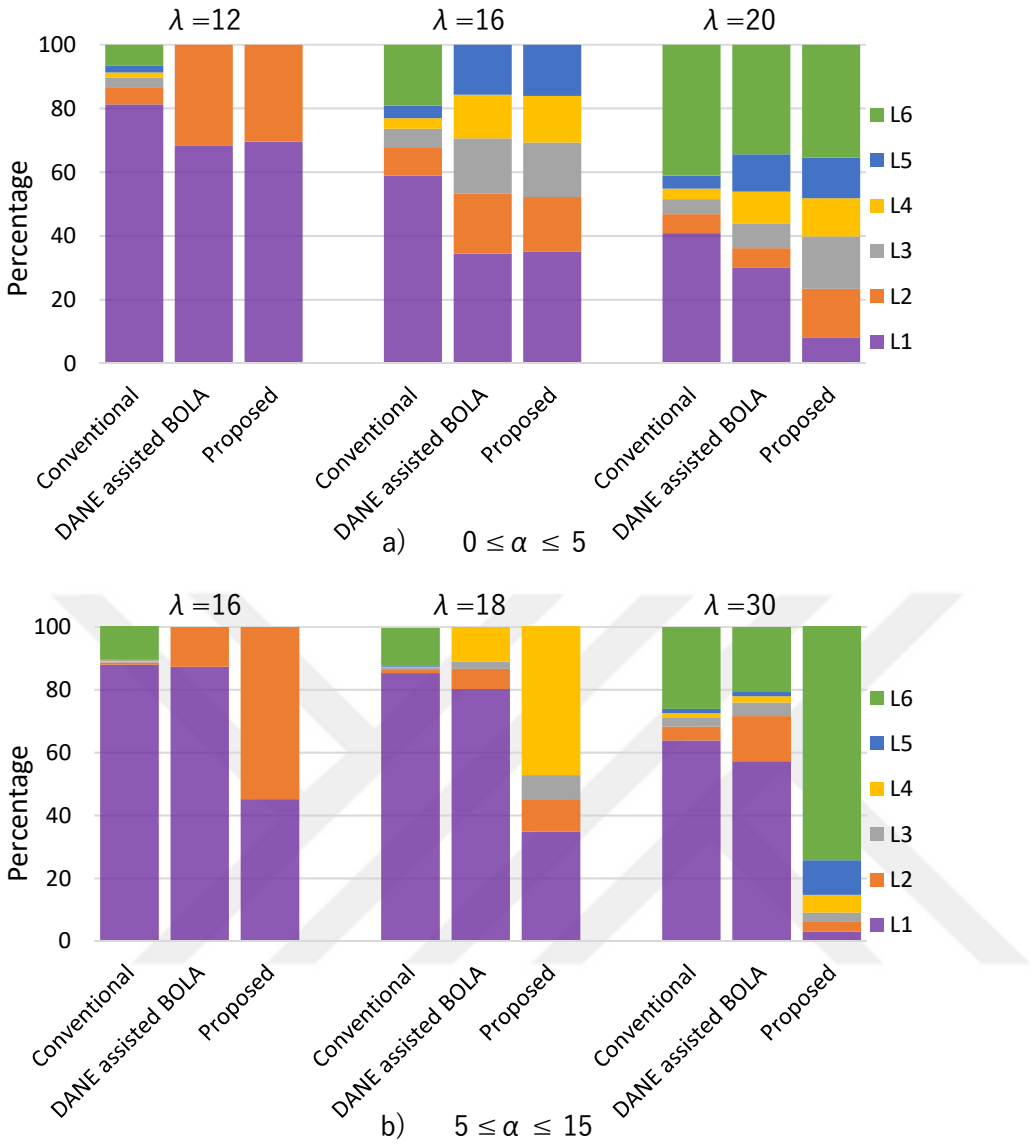


Figure 5.3. Distribution of the received quality level.

Since each client implements its rate adaptation algorithm in the conventional approach, the video quality experienced by different clients may differ even they share the same network environment. Undoubtedly, using DANE reduces the standard deviation by enforcing the clients to never exceed the specific quality levels even when the network bandwidth is free for a short period. However, the greedy behavior of the conventional approach causes the clients to request a video representation from the highest possible quality levels when estimating the network has more available bandwidth. Unfortunately, clients should decrease quality when available bandwidth decreases. Therefore, resource allocation in the conventional approaches is not fair enough and players experience different video quality because they have a limited view of the network activity in the lack of SAND architecture.

Leveraging SAND and DANE may help clients receive segments from the representations having similar qualities in case they are properly assisted by DANEs. This provides that network resources are shared fairly among clients, since each client uses a roughly equal amount of network bandwidth. In the proposed approach, the TPS estimates maximum quality level based on the network information and enforce clients not to exceed that threshold. Thus, all clients receive videos in more uniform quality. As shown in Figure. 5.4, in the proposed approach, the video quality experienced by the clients is similar while they receive higher bitrate with the steady video quality. On the contrary, the other approaches have a higher deviation, which means that clients experience video with more varying quality.

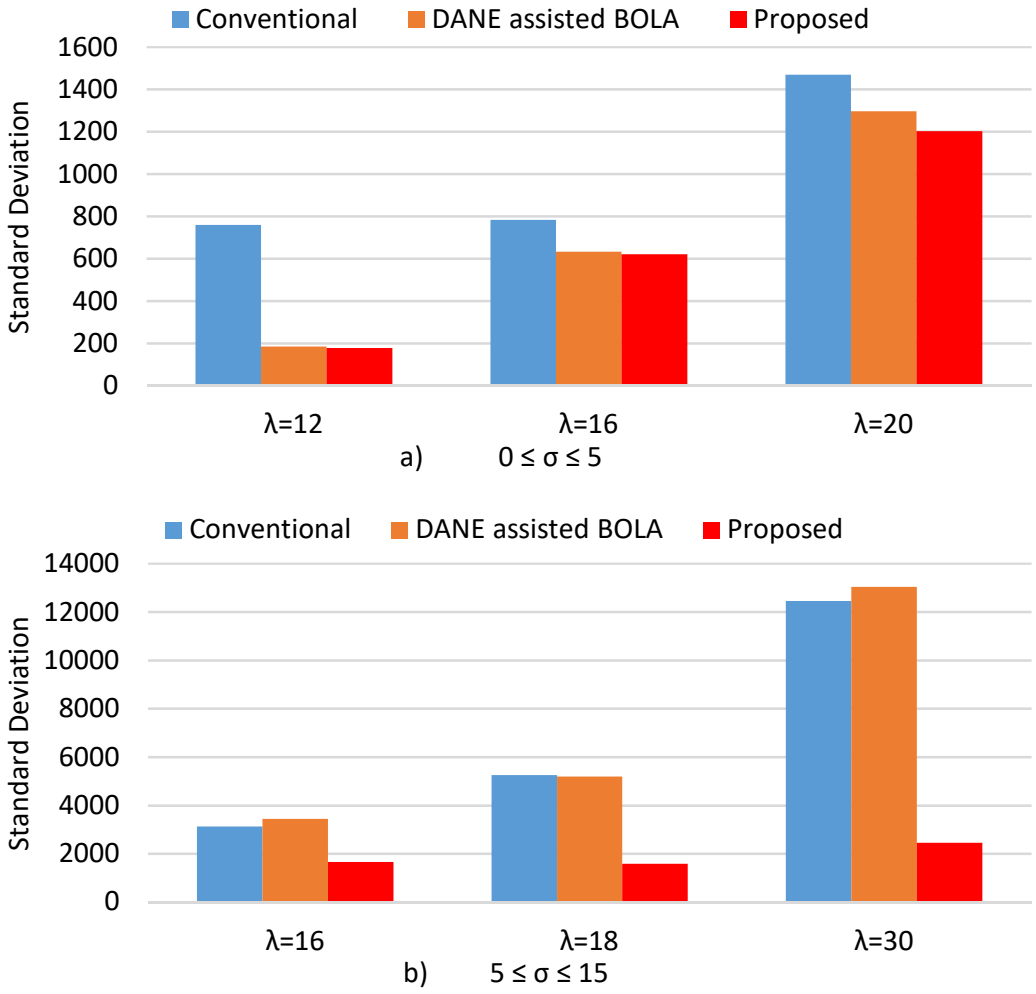


Figure 5.4. Standard deviation of the received video bitrate.

5.6 Conclusion

SDN is a versatile approach for providing network support for SAND. In this paper, we developed an SDN-based SAND architecture and designed a SAND-aware DASH client. Based on the simulation results, the proposed approach provided improvements up to 40 and 34% in received video bitrate while providing reductions up to 78 and 80% in outage duration, and 54 and 68% in startup delay when compared to the conventional and DANE assisted BOLA methods, respectively. This means that SAND architecture in the proposed approach enables the clients to achieve better adaptation, while the network conditions change rapidly. This awareness leads clients to achieve better QoE.

Planned future work and improvements include enhancing our study by adding a content management infrastructure. We plan to continue to use virtualized caches and develop an algorithm for determining the contents in these virtualized caches.

Note : The results of this study published in (Kalan et al., 2018b), *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 27-29 November 2018 – Verona, Italy

6. CONCLUSION

In this section we summarize the findings of the thesis and discuss some future perspectives.

6.1 General View

In this thesis, we have investigated adaptive streaming technology relied on SAND technology to experience better QoE. In this way, we leveraged NFV and SDN advantages concerning overcome legacy network limitations. NFV-SDN provides more flexibility for network management. SDN is an emerging paradigm where the data and control planes are separated, which gives more agility and flexibility to network management while a central controller has a big picture of the underlying network infrastructure. SDN technology allows network operators to develop and implement application-specific, innovative routing strategies as well as gather real-time network related information such as traffic volume and packet loss rate. Therefore, SDN could be a promising technology to provide network support for SAND applications. The SDN programmable infrastructure that offers a wide range of flexibility for designing application-specific routing algorithms and the SAND architecture that provides clients to make more intelligent decisions about rate adaptation.

In the SAND architecture, there are network entities called DANEs, server(s), clients as well as regular network elements. DANEs are used for enhancing QoE by sending messages to the clients, servers, and other DANEs. Parameters Enhancing Delivery (PED), Parameters Enhancing Reception (PER), and client status and metric messages are defined as SAND messages. While PED messages are exchanged between two DANEs, or a server and a DANE; PER messages are sent to the clients. The messages sent to the clients can be either an assistance message or an enforcement/error message. Assistance messages can be used by the clients to improve their performance. Enforcement or error messages require clients to operate in a particular way.

6.2 Contributions

The first part of this study introduces an algorithm for efficiently using network resources while supporting for fair bandwidth allocation to SVC clients. Withe the consideration of the packet loss ratio in the application layer, we introduced an optimization algorithm for the selection of video codec type and an optimal number

of layers. The aim of the proposed (***Algorithm 1:*** *Packet Loss Ratio Estimation*) algorithm is to maximize the quality of video at the client-side while considering the constant capacity of the network. Toward conduct to the optimal number of video layers, the SDN controller calculates the packet loss ratio by considering video codec types.

We extended our first proposed algorithm by focusing on layer dependency and packet loss ration in multiple paths, while each layer can be routed via different paths and affected by other streams on the same path. In our SDN-assisted system, we implemented the OLS (***Algorithm 2:*** *Optimal Layer and Path Selection*) algorithm. An output of the OLS algorithm, the controller assists clients by sending the optimal number of layers as well as video codec types which can be requested. We have conducted that the OLS algorithm achieves better results in therm of QoE metrics. In addition, it is discussed and proof that the OLS algorithm belongs to the class of $(C * O(nlogn))$ complexity, where C is a constant value refers to the number of video layers.

As a second contributing, we leveraged advantages of the network function virtualization with features of the MPEG-SAND architecture and introduced virtual DANE (vDANE or vCache). The vDANEs are the virtualized caches with the knowledge of DASH characteristics, which uses NFV hosted by the servers connected to the switches. Contribution of the SAND and NFV-SDN can assist the client in better adaption. We introduced our cache placement algorithm by taking advanced of the SAND technology into account. As a first attempt, we introduced the concept of vDANE and proposed an algorithm for selecting the placement of an additional vDANE when a new cache is required (Clayman et al., 2018). The proposed (***Algorithm 3:*** *PressureCache*) algorithm aims toward installing additional vDANEs in appropriate locations while the first one was previously installed in a fixed location.

In (Kalan et al., 2019), we proposed the problem of selecting the location of the initial $vDANE$ and the migration of it considering SAND characteristics. While in the former study only available bandwidth is considered, in the latter study, the replacement of the virtualized cache function and connectivity as well as bandwidth and hop counts of the links also is charged. The dynamic nature of the proposed (***Algorithm 4:*** *InitialPressureCache*) algorithm gives more flexibility to service providers to immigrate vDANE to a suitable location with respect to reduces network traffic and latency. As a consequence, the client experiences better QoE. Also, we addressed the *hand-off* problem by introducing (***Algorithm 5:*** *Cache Mi-*

gration).

SDN is a considerable alternative for providing network support for SAND architectures, we develop a SAND architecture by utilizing network observation and routing capabilities of SDN, and design a SAND aware DASH client for maximizing the advantages provided by the SAND architecture. SAND architecture in the proposed approach gives more intelligence to the clients in order to achieve better adaption while network condition changes rapidly. This intelligence leads clients to have better QoE. Furthermore, proposed (***Algorithm 6:*** *Assisted DASH Adaptive*) algorithm eliminate clients greedy behaviour. Thus, almost smooth traffic can be considered as another output of the proposed algorithm.

6.3 Future Prospects

As future work, we aim to address unified streaming where there is only one source to stream all content to all devices. Such an architecture allows the origin server to ingest one format and package it on-the-fly to all devices. Unify streaming improves cache capacity while it is stored only one copy for all formats. Besides, streaming a unique format reduces network traffic and CDN cost as well.

6.4 Publication

- Journals

    1. A Layered Adaptive Video Streaming Architecture: Codec, Path and Quality Selection by Utilizing SDN (is ready for submission)
    2. SAND Assisted Virtualized Cache Placement in Highly Dynamic SDN Network (is ready for submission)

- IEEE Conferences

    1. RS Kalan, M Sayit, S Clayman, *Multimedia Service Management with Virtualized Cache M.igration*, 23rd Conference on Innovation in Clouds, Internet and Networks (IEEE ICIN 2020), Paris- France, (Accepted)
    2. RS. Kalan, M. Sayit, S. Clayman, *Optimal Cache Placement and Migration for Improving the Performance of Virtualized SAND*, IEEE Conference on Network Softwarization (IEEE NetSoft 2019), Paris-France
    3. RS. Kalan, M. Sayit, AC. Begen, *Implementation of SAND Architecture Using SDN*, 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (IEEE NFV-SDN 2018), Verona- Italy
    4. S. Clayman, RS. Kalan, M. Sayit, *Virtualized Cache Placement in an SDN/NFV Assisted SAND Architecture*, IEEE International Black Sea Conference on Communications and Networking (IEEE BlackSeaCom 2018), Batumi- Georgia
    5. RS. Kalan, S. Bikas, B. Kheibari, M. Sayit, *SDN-CPS: Selecting Codec, Path and Server for Video Streaming over SDN*, 26th Signal Processing and Communications Applications Conference (IEEE SIU 2018), Çeşme- Turkey
    6. RS Kalan, C Cetinkaya, M Sayit, *Design of a Layer-Based Video Streaming System over Software-Defined Networks*, 8th International Conference on the Network of the Future (IEEE NOF 2017), London- England
    7. K. Herguner, RS. Kalan, C. Centinkaya, M. Sayit, *Towards QoS-Aware Routing for DASH Utilizing MPTCP over SDN*, IEEE Conference on Network Function Virtualization and Software Defined Networks (IEEE NFV-SDN 2017), Berlin-Germany

REFERENCES

Adrichem, N.L.M., Doerr, C. and Kuipers, F.A., 2014. OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks. In 2014 IEEE Network Operations and Management Symposium (NOMS). p. 1–8.

Akhshabi, S., Begen, A.C. and Dovrolis, C., 2011. An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over HTTP. In Proceedings of the Second Annual ACM Conference on Multimedia Systems. ACM, New York, NY, USA, MMSys '11, p. 157–168.

Ali-Ahmad, H., Cicconetti, C., de la Oliva, A., Dräxler, M., Gupta, R., Mancuso, V., Roullet, L. and Sciancalepore, V., 2013. CROWD: an SDN approach for DenseNets.

Altamimi, S. and Shirmohammadi, S., 2019. Client-server Cooperative and Fair DASH Video Streaming. In Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video. ACM, New York, NY, USA, NOSSDAV '19, p. 1–6.

Badshah, J., Kamran, M., Shah, N. and Abid, S.A., 2019. An Improved Method to Deploy Cache Servers in Software Defined Network-based Information Centric Networking for Big Data. *Journal of Grid Computing*, 17(2):255–277.

Badshah, J., Mohaia Alhaisoni, M., Shah, N. and Kamran, M., 2020. Cache Servers Placement Based on Important Switches for SDN-Based ICN. *Electronics*, 9(1):39.

Bagci, K.T., Sahin, K.E. and Tekalp, A.M., 2017. Compete or Collaborate: Architectures for Collaborative DASH Video Over Future Networks. *IEEE Transactions on Multimedia*, 19(10):2152–2165.

Bentaleb, A., Begen, A.C. and Zimmermann, R., 2016. SDNDASH: Improving QoE of HTTP Adaptive Streaming Using Software Defined Networking. In Proceedings of the 2016 ACM on Multimedia Conference. ACM, New York, NY, USA, MM '16, p. 1296–1305.

Bentaleb, A., Taani, B., Begen, A.C., Timmerer, C. and Zimmermann, R., 2018. A Survey on Bitrate Adaptation Schemes for Streaming Media over HTTP. *IEEE Communications Surveys Tutorials*.

Bhat, D., Rizk, A., Zink, M. and Steinmetz, R., 2017. Network Assisted Content Distribution for Adaptive Bitrate Video Streaming. In Proceedings of the 8th ACM on Multimedia Systems Conference. ACM, New York, NY, USA, MMSys'17, p. 62–75.

REFERENCES (continued)

Binder, A., Boros, T. and Kotuliak, I., 2015. A SDN Based Method of TCP Connection Handover. In Information and Communication Technology, I. Khalil, E. Neuhold, A.M. Tjoa, L.D. Xu and I. You, eds.. Springer International Publishing, Cham, p. 13–19.

Braga, R., de Souza Mota, E. and Passito, A., 2010. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In LCN. volume 10, p. 408–415.

Bruneau-Queyreix, J., Lacaud, M., Negru, D., Batalla, J.M. and Borcoci, E., 2017. MS-Stream: A multiple-source adaptive streaming solution enhancing consumer's perceived quality. In 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC). IEEE, p. 427–434.

Cetinkaya, C., Karayer, E., Sayit, M. and Hellge, C., 2014. SDN for segment based flow routing of DASH. In 2014 IEEE Fourth International Conference on Consumer Electronics Berlin (ICCE-Berlin). p. 74–77.

Cetinkaya, C., Ozveren, Y. and Sayit, M., 2015. An SDN-assisted system design for improving performance of SVC-DASH. In 2015 Federated Conference on Computer Science and Information Systems (FedCSIS). p. 819–826.

Cetinkaya, C. and Sayit, M., 2016. Video-on-demand system architecture with ALTO-SDN integration. In 2016 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom). p. 1–5.

Chowdhury, N.M.K. and Boutaba, R., 2010. A Survey of Network Virtualization. *Comput. Netw.*, 54(5):862–876.

Cisco, 2018. The Zettabyte Era: Trends and Analysis. URL `Available:https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html`.

Civanlar, S., Parlakisik, M., Tekalp, A.M., Gorkemli, B., Kaytaz, B. and Onem, E., 2010. A QoS-enabled OpenFlow environment for Scalable Video streaming. In 2010 IEEE Globecom Workshops. p. 351–356.

Claeys, M., Tuncer, D., Famaey, J., Charalambides, M., Latré, S., Pavlou, G. and De Turck, F., 2016. Hybrid multi-tenant cache management for virtualized ISP networks. *Journal of Network and Computer Applications*, 68:28–41.

Claeys, M., Tuncer, D., Famaey, J., Charalambides, M., Latré, S., Pavlou, G. and Turck, F.D., 2014. Proactive multi-tenant cache management for virtualized ISP networks. *10th International Conference on Network and Service Management (CNSM) and Workshop*, p. 82–90.

REFERENCES (continued)

Clayman, S., Kalan, R.S. and Sayit, M., 2018. Virtualized Cache Placement in an SDN/NFV Assisted SAND Architecture. In 2018 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom). p. 1–5.

Clayman, S., Maini, E., Galis, A., Manzalini, A. and Mazzocca, N., 2014. The Dynamic Placement of Virtual Network Functions. In IEEE/IFIP 2014.

Clegg, R.G., Clayman, S., Pavlou, G., Mamatas, L. and Galis, A., 2013. On the selection of management/monitoring nodes in highly dynamic networks. *Computers, IEEE Trans. on*, 62(6):1207–1220.

DASH-IF, 2017. DASH-IF Position Paper: Server and Network Assisted DASH (SAND), ISO/IEC 23009-5, published December 2016.

De Cicco, L., Mascolo, S. and Palmisano, V., 2011. Feedback control for adaptive live video streaming. In Proceedings of the second annual ACM conference on Multimedia systems. ACM, p. 145–156.

ED-II, 2017. Elephants Dream-II-1080p, [Online] http://concert.itec.aau.at/SVCDataset/, accessed 2018. URL `http://concert.itec.aau.at/SVCDataset/`.

Egilmez, H.E., Civanlar, S. and Tekalp, A.M., 2013. An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks. *IEEE Transactions on Multimedia*, 15(3):710–715.

Egilmez, H.E., Dane, S.T., Bagci, K.T. and Tekalp, A.M., 2012. OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks. In Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference. p. 1–8.

Elgabli, A., Aggarwal, V., Hao, S., Qian, F. and Sen, S., 2018. DBABP: Robust Rate Adaptation Algorithm for SVC Video Streaming. *CoRR*, abs/1805.00041.

FloodLight, 2017. Project FloodLight, FloodLight. URL `http://www.projectfloodlight.org/floodlight//`.

Fouladi, S., Emmons, J., Orbay, E., Wu, C., Wahby, R.S. and Winstein, K., 2018. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In USENIX NSDI.

Gangwal, A., Gupta, M., Gaur, M.S., Laxmi, V. and Conti, M., 2016. ELBA: Efficient Layer Based Routing Algorithm in SDN. In 2016 25th International Conference on Computer Communication and Networks (ICCCN). p. 1–7.

REFERENCES (continued)

Gember-Jacobson, A., Viswanathan, R., Prakash, C., Grandl, R., Khalid, J., Das, S. and Akella, A., 2014. OpenNF: Enabling Innovation in Network Function Control. In Proceedings of the 2014 ACM Conference on SIGCOMM. ACM, New York, NY, USA, SIGCOMM '14, p. 163–174.

Georgopoulos, P., Broadbent, M., Plattner, B. and Race, N., 2014. Cache as a service: Leveraging SDN to efficiently and transparently support video-on-demand on the last mile. In 2014 23rd International Conference on Computer Communication and Networks (ICCCN). p. 1–9.

Georgopoulos, P., Elkhatib, Y., Broadbent, M., Mu, M. and Race, N., 2013. Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming. In Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking. FhMN '13, p. 15–20.

Ghodsi, A., Shenker, S., Koponen, T., Singla, A., Raghavan, B. and Wilcox, J.R., 2011. Information-centric networking: seeing the forest for the trees. In HotNets-X.

Gomes, A.S., Sousa, B., Palma, D., Fonseca, V., Zhao, Z., Monteiro, E., Braun, T., Simoes, P. and Cordeiro, L., 2017. Edge caching with mobility prediction in virtualized LTE mobile networks. *Future Generation Computer Systems*, 70:148–162.

Han, B., Gopalakrishnan, V., Ji, L. and Lee, S., 2015. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97.

Han, B., Qian, F., Ji, L. and Gopalakrishnan, V., 2016. MP-DASH: Adaptive Video Streaming Over Preference-Aware Multipath. In Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies. ACM, New York, NY, USA, CoNEXT '16, p. 129–143.

Hand, R., Ton, M. and Keller, E., 2013. Active security. In Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks. ACM, pp. 17.

He, S., Tian, H. and Lyu, X., 2017. Edge Popularity Prediction Based on Social-Driven Propagation Dynamics. *IEEE Communications Letters*, 21(5):1027–1030.

Heikkinen, A., Ojanperä, T. and Vehkaperä, J., 2016. Dynamic cache optimization for DASH clients in Content Delivery Networks. In 2016 13th IEEE Annual (CCNC). p. 980–983.

REFERENCES (continued)

Heller, B., Sherwood, R. and McKeown, N., 2012. The Controller Placement Problem. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks. ACM, New York, NY, USA, HotSDN '12, p. 7–12.

Hemmati, M., Yassine, A. and Shirmohammadi, S., 2015. A Dec-POMDP model for congestion avoidance and fair allocation of network bandwidth in rate-adaptive video streaming. In 2015 IEEE Symposium Series on Computational Intelligence. IEEE, p. 1182–1189.

Hossfeld, T., Seufert, M., Sieber, C., Zinner, T. and Tran-Gia, P., 2015. Identifying QoE Optimal Adaptation of HTTP Adaptive Streaming Based on Subjective Studies. *Comput. Netw.*, 81(C):320–332.

Huang, T., Johari, R., McKeown, N., Trunnell, M. and Watson, M., 2014. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In Proceedings of the 2014 ACM Conference on SIGCOMM. ACM, New York, NY, USA, SIGCOMM '14, p. 187–198.

Ibn-Khedher, H., Abd-Elrahman, E. and Afifi, H., 2016a. OMAC: Optimal migration algorithm for virtual CDN. In 2016 23rd International Conference on Telecommunications (ICT). p. 1–6.

Ibn-Khedher, H., Abd-Elrahman, E., Kamal, A.E. and Afifi, H., 2017. OPAC: An optimal placement algorithm for virtual CDN. *computer Networks*, 120:12–27.

Ibn-Khedher, H., Hadji, M., Abd-Elrahman, E., Afifi, H. and Kamal, A.E., 2016b. Scalable and Cost Efficient Algorithms for Virtual CDN Migration. In 2016 IEEE 41st Conference on Local Computer Networks (LCN). p. 112–120.

ITEC, 2018. Big Buck Bunny. URL `http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/BigBuckBunny/2sec/`.

Jain, R. and Paul, S., 2013. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, 51(11):24–31.

Jarschel, M., Wamser, F., Hohn, T., Zinner, T. and Tran-Gia, P., 2013. SDN-Based Application-Aware Networking on the Example of YouTube Video Streaming. In 2013 Second European Workshop on Software Defined Networks. p. 87–92.

REFERENCES (continued)

Ji, X., Huang, J., Chiang, M., Lafruit, G. and Catthoor, F., 2011. Scheduling and Resource Allocation for SVC Streaming over OFDM Downlink Systems. In Recent Advances on Video Coding, J.D.S. Lorente, ed., IntechOpen, Rijeka, chapter 10.

Jiang, J., Sekar, V. and Zhang, H., 2012. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies. ACM, New York, NY, USA, CoNEXT '12, p. 97–108.

Jmal, R., Simon, G. and Chaari, L., 2017. Network-assisted strategy for dash over CCN. In 2017 IEEE International Conference on Multimedia and Expo (ICME). IEEE, p. 13–18.

Kalan, R.S., Bikas, S., Kheibari, B. and Sayit, M., 2018a. SDN-CPS: Selecting codec, path and server for video streaming over SDN. In 2018 26th Signal Processing and Communications Applications Conference (SIU). p. 1–4.

Kalan, R.S., Cetinkaya, C. and Sayit, M., 2017. Design of a layer-based video streaming system over software-defined networks. In 2017 8th International Conference on the Network of the Future (NOF).

Kalan, R.S., Sayit, M. and Begen, A.C., 2018b. Implementation of SAND Architecture Using SDN. In 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). p. 1–6.

Kalan, R.S., Sayit, M. and Clayman, S., 2019. Optimal Cache Placement and Migration for Improving the Performance of Virtualized SAND. In IEEE Conference on Network Softwarization 24-28 June 2019- Paris, France.

Kalan, R.S., Sayit, M. and Clayman, S., 2019. Optimal Cache Placement and Migration for Improving the Performance of Virtualized SAND. In NETSOFT2019.

Kim, H. and Feamster, N., 2013. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119.

Kleinrouweler, J.W., Cabrero, S. and Cesar, P., 2016. Delivering Stable High-quality Video: An SDN Architecture with DASH Assisting Network Elements. In Proceedings of the 7th International Conference on Multimedia Systems. ACM, New York, NY, USA, MMSys '16, p. 4:1–4:10.

REFERENCES (continued)

Kleinrouweler, J.W., Cabrero, S., van der Mei, R. and Cesar, P., 2015. Modeling Stability and Bitrate of Network-Assisted HTTP Adaptive Streaming Players. In International Teletraffic Congress, 2015.

Kleinrouweler, J.W., Meixner, B. and Cesar, P., 2017. Improving Video Quality in Crowded Networks Using a DANE. In ACM NOSSDAV.

Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T. et al., 2010. Onix: A distributed control platform for large-scale production networks. In OSDI. volume 10, p. 1–6.

Kreutz, D., Ramos, F., Verissimo, P., Rothenberg, C.E., Azodolmolky, S. and Uhlig, S., 2014. Software-defined networking: A comprehensive survey. *arXiv preprint arXiv:1406.0440*.

Kreutz, D., Ramos, F.M.V., Veríssimo, P.E., Rothenberg, C.E., Azodolmolky, S. and Uhlig, S., 2015. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76.

Krishnamoorthi, V., Carlsson, N., Halepovic, E. and Petajan, E., 2017. BUFFEST: Predicting buffer conditions and real-time requirements of HTTP (S) adaptive streaming clients. In Proceedings of the 8th ACM on Multimedia Systems Conference. ACM, p. 76–87.

Laga, S., Cleemput, T.V., Raemdonck, F.V., Vanhoutte, F., Bouten, N., Claeys, M. and Turck, F.D., 2014. Optimizing scalable video delivery through OpenFlow layer-based routing. In 2014 IEEE Network Operations and Management Symposium (NOMS). p. 1–4.

Lantz, B., Heller, B. and McKeown, N., 2010. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, New York, NY, USA, p. 19:1–19:6.

Li, Y. and Chen, M., 2015. Software-Defined Network Function Virtualization: A Survey. *IEEE Access*, 3:2542–2553.

Li, Z., Sbaï, M.K., Hadjadj-Aoul, Y., Gravey, A., Alliez, D., Garnier, J., Madec, G., Simon, G. and Singh, K., 2012. Network friendly video distribution. In 2012 Third International Conference on The Network of the Future (NOF). p. 1–8.

REFERENCES (continued)

Li, Z., Zhao, S., Medhi, D. and Bouazizi, I., 2016. Wireless video traffic bottleneck coordination with a DASH SAND framework. In VCIP. p. 1–4.

Liu, X., Dobrian, F., Milner, H., Jiang, J., Sekar, V., Stoica, I. and Zhang, H., 2012. A Case for a Coordinated Internet Video Control Plane. In Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. ACM, New York, NY, USA, SIGCOMM '12, p. 359–370.

Mamatas, L., Clayman, S., Charalambides, M., Galis, A. and Pavlou, G., 2010. Towards an Information Management Overlay for emerging networks. In 2010 IEEE Network Operations and Management Symposium - NOMS 2010. p. 527–534.

Mao, H., Netravali, R. and Alizadeh, M., 2017. Neural Adaptive Video Streaming with Pensieve. In ACM SIGCOMM.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J., 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Megyesi, P., Botta, A., Aceto, G., Pescapè, A. and Molnár, S., 2016. Available Bandwidth Measurement in Software Defined Networks. In Proceedings of the 31st Annual ACM Symposium on Applied Computing. ACM, New York, NY, USA, SAC '16, p. 651–657.

Mijumbi, R., Serrat, J., Gorricho, J., Bouten, N., Turck, F.D. and Boutaba, R., 2016a. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262.

Mijumbi, R., Serrat, J., Gorricho, J., Latre, S., Charalambides, M. and Lopez, D., 2016b. Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*, 54(1):98–105.

Moens, H. and De Turck, F., 2014. VNF-P: A model for efficient placement of virtualized network functions. In 10th International Conference on Network and Service Management (CNSM) and Workshop. IEEE, p. 418–423.

Moens, H. and Turck, F.D., 2014. VNF-P: A model for efficient placement of virtualized network functions. In CNSM 2014. p. 418–423.

Mok, R.K., Luo, X., Chan, E.W. and Chang, R.K., 2012. QDASH: a QoE-aware DASH system. In Proceedings of the 3rd Multimedia Systems Conference. ACM, p. 11–22.

REFERENCES (continued)

Mukerjee, M.K., Naylor, D., Jiang, J., Han, D., Seshan, S. and Zhang, H., 2015. Practical, Real-time Centralized Control for CDN-based Live Video Delivery. ACM, SIGCOMM '2015, p. 311–324.

N. Bouten, J. Famaey, R.M.B.N., J. Serrat, S.L. and Turck, F., 2015. Towards NFV-based multimedia delivery. In IFIP/IEEE 2015. p. 738–741.

Nam, H., Kim, K.H., Kim, J.Y. and Schulzrinne, H., 2014. Towards QoE-aware video streaming using SDN. In 2014 IEEE Global Communications Conference. p. 1317–1322.

Noghani, K.A. and Sunay, M.O., 2014. Streaming Multicast Video over Software-Defined Networks. In 2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems. p. 551–556.

Nunes, B.A.A., Mendonca, M., Nguyen, X., Obraczka, K. and Turletti, T., 2014. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys Tutorials*, 16(3):1617–1634.

Petrangeli, S., Wu, T., Wauters, T., Huysegems, R., Bostoen, T. and De Turck, F., 2017. A Machine Learning-based Framework for Preventing Video Freezes in HTTP Adaptive Streaming. *J. Netw. Comput. Appl.*, 94(C):78–92.

Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M. and Gu, G., 2012. A security enforcement kernel for OpenFlow networks. In Proceedings of the first workshop on Hot topics in software defined networks. ACM, p. 121–126.

Psaras, I., Chai, W.K. and Pavlou, G., 2012. Probabilistic in-network caching for information-centric networks. In Proceedings of the second edition of the ICN workshop on Information-centric networking. p. 55–60.

Rainer, B., Lederer, S., Müller, C. and Timmerer, C., 2012. A seamless Web integration of adaptive HTTP streaming. In EUSIPCO.

Ramakrishnan, S., Zhu, X., Chan, F. and Kambhatla, K., 2015. SDN Based QoE Optimization for HTTP-Based Adaptive Video Streaming. In 2015 IEEE International Symposium on Multimedia (ISM). p. 120–123.

Rossi, D. and Rossini, G., 2012. On sizing CCN content stores by exploiting topological information. In 2012 Proceedings IEEE INFOCOM Workshops. p. 280–285.

REFERENCES (continued)

Schwarz, H., Marpe, D. and Wiegand, T., 2006. Overview of the Scalable H.264/MPEG4-AVC Extension. In 2006 International Conference on Image Processing. p. 161–164.

Seddiki, M.S., Shahbaz, M., Donovan, S., Grover, S., Park, M., Feamster, N. and Song, Y.Q., 2014. FlowQoS: QoS for the Rest of Us. In ACM HotSDN.

Sezer, S., Scott-Hayward, S., Chouhan, P.K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M. and Rao, N., 2013. Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43.

Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S. and Sekar, V., 2012. Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service. In Proceedings of the ACM SIGCOMM. ACM, p. 13–24.

Singh, M., Varyani, N., Singh, J. and Haribabu, K., 2018. Estimation of End-to-End Available Bandwidth and Link Capacity in SDN. In Ubiquitous Communications and Network Computing, N. Kumar and A. Thakre, eds.. Springer International Publishing, Cham, p. 130–141.

Sodagar, I., 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia*.

Spiteri, K., Urgaonkar, R. and Sitaraman, R.K., 2016. BOLA: Near-optimal bitrate adaptation for online videos. *IEEE INFOCOM*.

Suksomboon, K., Tarnoi, S., Ji, Y., Koibuchi, M., Fukuda, K., Abe, S., Motonori, N., Aoki, M., Urushidani, S. and Yamada, S., 2013. PopCache: Cache more or less based on content popularity for information-centric networking. In 38th Annual IEEE Conference on Local Computer Networks. p. 236–243.

Tang, Y., Guo, K., Ma, J., Shen, Y. and Chi, T., 2019. A smart caching mechanism for mobile multimedia in information centric networking with edge computing. *Future Generation Comp. Syst.*, 91:590–600.

Thomas, E., van Deventer, M.O., Stockhammer, T., Begen, A.C. and Famaey, J., 2017. Enhancing MPEG DASH Performance via Server and Network Assistance. *SMPTE Motion Imaging Journal*, 126(1):22–27.

Thomas, E., van Deventer, O., Stockhammer, T., Begen, A., Champel, M.L. and Oyman, O., 2016. Applications and deployments of Server And Network assisted DASH (SAND).

REFERENCES (continued)

Topology-Zoo, 2017. The Internet Topology-Zoo, [Online] http://www.topology-zoo.org/ , accessed 2017. URL `http://www.topology-zoo.org/`.

Tuncer, D., Charalambides, M., Clayman, S. and Pavlou, G., 2015. Adaptive Resource Management and Control in Software Defined Networks. *IEEE TNSM2015*, 12(1):18–33.

Uzakgider, T., Cetinkaya, C. and Sayit, M., 2015. Learning-based Approach for Layered Adaptive Video Streaming over SDN. *Comput. Netw.*, 92(P2):357–368.

Wang, Y., Li, Z., Tyson, G., Uhlig, S. and Xie, G., 2013. Optimal cache allocation for content-centric networking. In 2013 21st IEEE International Conference on Network Protocols (ICNP). IEEE, p. 1–10.

Xiao, X., Hannan, A., Bailey, B. and Ni, L.M., 2000. Traffic engineering with MPLS in the Internet. *IEEE Network*, 14(2):28–33.

Xiaoqiang, Z., Min, Z. and Muqing, W., 2016. An in-network caching scheme based on betweenness and content popularity prediction in content-centric networking. In 2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC).

Xue, N., Chen, X., Gong, L., Li, S., Hu, D. and Zhu, Z., 2015. Demonstration of OpenFlow-Controlled Network Orchestration for Adaptive SVC Video Manycast. *IEEE Transactions on Multimedia*, 17(9):1617–1629.

Yeganeh, S.H., Tootoonchian, A. and Ganjali, Y., 2013. On scalability of software-defined networking. *IEEE Communications Magazine*, 51(2):136–141.

Yue, Y., Ran, Y., Chen, S., Yang, B., Sun, L. and Yang, J., 2015. Joint Routing and Layer Selecting for Scalable Video Transmission in SDN. In 2015 IEEE Globecom Workshops (GC Wkshps). p. 1–6.

Zhang, L., Wang, S. and Chang, R.N., 2018. QCSS: A QoE-Aware Control Plane for Adaptive Streaming Service over Mobile Edge Computing Infrastructures. In 2018 IEEE International Conference on Web Services (ICWS). p. 139–146.

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to my supervisor Dr. *Muge Sayit*, Associate Professor in the $MCRG$ lab-University of EGE, for her constant guidance, valuable comments, and prompt suggestions, as well as for her continuous leading towards the successful completion of this Ph.D. research. She provided me with an environment in which I smoothly and freely performed my scientific research.

I also would like to express my appreciation to my research advisor *Ali.C Begen*, associate Professor in the Department of Computer Engineering at Ozegin University, for his support and targeted observations and thorough comments, and *Ilker Kocabas*, assistant Professor in International Computer Institute - University of EGE for all valuable and constructive feedback's during the progress of this research.

I would like to thank the Scientific and Technological Research Council of Turkey ($TUBITAK$) Electric, Electronic and Informatics Research Group (EEEAG) under grant 115E449, and *Digiturk beIN MEDIA GROUP*.

I would like to thank all academic, managerial and technical staff of the International Computer Institute for providing me with a comfortable research environment. A special thanks go to my friends Moharram Challenger and Vaid Khalipour Akram who patiently give me hand to join International Computer Institute as well as during my Ph.D. education. Special thanks to former graduated Ph.D. students specifically Can Umit and Yasin Yiğit for their efforts in preparing and sharing with me the LaTeX template of the thesis.

Finally, I would like to express my sincere thanks to parents who love me without any expectations, my family members for their support in every aspect of my life.

CURRICULUM VITAE

Reza Shokri Kalan

Address: Ege University International Computer Institute- Izmir/Turkey
Mobile: +90 552 246 2086
E-mail: reza.shokri@hotmail.com, reza.shokri.kalan@gmail.com
Linkedin: www.linkedin.com.in/reza.shokri-13069a36

Education Background

MSc: 2011-2013, Easter Mediterranean University, Computer Engineering-Cyprus
BSc: 1988-2003, Islamic Azad University, Computer Engineering- Shabestar-Iran

Academic Experience

2013-2014: Algorithm design , Roshdiyeh University
2007-2008: Advanced programming, Applied Science and Technology University
2007-2008: Database management, Aptech international computer education center
2004-2008: Computer networks, Industrial Management Institute
2004-2008: Computer networks, Tehran Institute of Technology

Projects

2019-2020: research in *'Central multimedia system for improving quality of experience in Content-Centric distribution'* Funding Organization: Under $TUBITAK$ Funding (TEYDEB Project, 3180915)

2015-2017: Researcher in *'The Integration of ALTO (Application-Layer Traffic Optimization) Protocol and SDN (Software Defined Networks) Technology for Maximizing Quality of Experience in Video Streaming Applications'*, under $TUBITAK$ funding (115E449)

Work Experiment

2019—-: Network architecture and CDN specialist, Digiturk beIN MEDIA GROUP
2003-2011: Computer engineer and database manager, CHOLOR PARS company

Publications

R. Shokri Kalan, M. Sayit, S. Clayman, *"Multimedia Service Management with Virtualized Cache Migration"* 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN) 24-27Feb 2020, Paris, France (Accepted)

R. Shokri Kalan, M. Sayit, S. Clayman, *"Optimal Cache Placement and Migration for Improving the Performance of Virtualized SAND"* Software Defined Networking and Network Function Virtualization Performance (IEEE PVE-SDN 2019), 24-28 June 2019, Paris, France

R. Shokri Kalan, M. Sayit, A. Begen,*" Implementation of SAND Architecture Using SDN"*, IEEE Conference on Network Function Virtualization and Software Defined Networks (IEEE NFV-SDN 2018), 27-29 November 2018, Verona, Italy

S. Clayman, R. Shokri Kalan, M. Sayit,*" Virtualized Cache Placement in an SD-N/NFV Assisted SAND Architecture"*, IEEE International Black Sea Conference on Communication and Networking (IEEE BlackSeaCom 2018), 4-7 June 2018, Batumi, Georgia

R. Shokri Kalan, S. Bikas, B. Khibari, M. Sayit,*" SDN-CPS: Selecting Codec, Path and Server for Video Streaming over SDN"*, 26th. IEEE Signal Processing and Communications Application Conference (SIU2018), 2-5 May 2018,Cesme, Izmir, Turkey

R. Shokri Kalan, C. Cetinkaya, M. Sayit, "Design of A Layer-based Video Streaming System over Software Defined Networks" 8th International Conference on the Network of the Future (NoF 2017) 22-24, November 2017, London, UK

K. Herguner, R. Shokri Kalan, C. Cetinkaya, M. Sayit, *"Towards QoS-aware Routing for DASH Utilization MPTCP over SDN"*, IEEE Conference on Network Function Virtualization and software Defined Networks (IEEE NFV-SDN), 6-8 November 2017, Berlin, Germany

R. Shokri Kalan, *"Improving Video-on-Demand by Multi-channel/Multicast Approach in Cellular Networks"*, (IEEE BlackSeaCom 2017), 5-8 June 2017, Istanbul, Turkey

R. Shokri Kalan, H. Khodaei, E. Ashena, *"Toward Secure and Effective Mobile Multimedia: Big Picture"*, 2nd International Conference on knowledge-based Research in Computer Engineering IT, September 2017, Tehran, Iran

R. Shokri Kalan, M. Osman Unalir, *"Leveraging Big Data Technology for Small and Mediumsized Enterprises (SMEs)"*, 6th International Conference on Computer and Knowledge Engineering (ICCKE2016), October 2016, Iran

R. Shokri Kalan, E. Ashena,*" Big Data: Mobile Apps in Attendance or Betrayal"*, The 4th National Conference on Applied Research in Electrical, Mechanical, Computer and IT Engineering" 2018, Shiraz- Iran R. Shokri Kalan, I. Kocabas,*" Big Data Analytics in Healthcare: Social Biological Data"*, International Journal of Engineering Inventions(IJEI) Volume 5, Issue 11 (Dec 2016)

R. Shokri Kalan, I. Kocabas, *"Adaptive Tools And Technology In Big Data Analytics"*, Journal of Multidisciplinary Engineering Science and Technology (JMEST), Vol.3-Issue 1 (January- 2016)

R. Shokri Kalan, G. Oz, P. Shams,*" Energy consumption management in data center using by virtualization"*, International Conference on Networking and Future Internet (ICNFI), pp.48-54, 2012, Istanbul

R. Shokri Kalan, N. Lotfi, *"Quantum computing and cryptography"*, 5th National Conference of Computer and Electronic, 2002, Iran

Books:

2018: Wireless and cellular networks
2013: Data center architecture and design
2010: Wireless communication and networks
2009: Introduction to Cisco engineering

Foreign Languages

English, Turkish