SDN-based Controllable-P2P-assisted CDN for HTTP Adaptive Live Video Streaming over Edge Access Networks

by

Selin Nacaklı

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in

Electrical and Electronics Engineering

KOÇ
UNIVERSITY

March 6, 2020

SDN-based Controllable-P2P-assisted CDN
for HTTP Adaptive Live Video Streaming
over Edge Access Networks

Koç University
Graduate School of Sciences and Engineering
This is to certify that I have examined this copy of a doctoral dissertation by

Selin Nacaklı

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____
Prof. Dr. Ahmet Murat Tekalp

_____
Prof. Dr. Özgür Barış Akan

_____
Prof. Dr. Öznur Özkasap

_____
Assoc. Prof. Dr. Berk Canberk

_____
Asst. Prof. Dr. Ali Cengiz Beğen

Date: _____

To my dearest parents and beloved husband...

ABSTRACT

Live video services over the Internet, including over-the-top (OTT) and IPTV services, are becoming ever more popular. These services are based on the client-server model using hyper-text transmission protocol (HTTP) adaptive streaming (HAS). Since there's a maximum number of concurrent streams that a server can handle, popular solutions to provide demand-scalability include content-distribution networks (CDN) and peer-to-peer (P2P) streaming. Today OTT video services employ cloud CDN solutions to cache content in distributed edge points of presence. Cloud CDNs not only ease the load on the origin server, but also accelerate content delivery with lower latency. However, hosting or renting CDN infrastructure can be costly. Recent developments in the WebRTC protocol that offers easy to use P2P media and data channels have led to renewed interest in P2P video streaming. However, the uncontrolled nature of P2P video delivery over the Internet remains as a critical problem.

In the meantime, we observe two major industry trends: i) cloud service providers are moving towards network edges for lower latency and higher bandwidth access (also known as fog computing), ii) network service providers (NSP) are replacing closed and proprietary hardware-based access technologies with disaggregated and virtualized software running on edge clouds to manage their edge access networks.

Recognizing the shortcomings of current CDN and P2P video solutions and the potential of SDN-based unified cloud and edge-access network technologies, we propose a hybrid P2P-assisted CDN architecture hosted at edge-access datacenters operated by network service providers. In the proposed architecture, the NSP employs an SDN-enabled cloud platform to manage each edge network domain. An important feature of the proposed hybrid service architecture is that both the CDN access by clients and P2P video streaming between clients are controlled by the network service provider

within each SDN-based edge network domain to optimize video service key performance indicators (KPI). This controllable P2P-assisted edge-managed video service reduces the load on CDN servers while overcoming quality of experience (QoE) fluctuations per flow and unfairness between multiple heterogeneous video-resolution clients over a reserved network slice. Other advantages of this service include: i) better video quality, lower delay and no buffering for clients; ii) minimization of cross-ISP traffic; iii) avoiding illegal, unauthorized usage of P2P services. Experimental results show that the proposed managed P2P-assisted CDN service deployed at SDN-enabled network edge platforms performs better than the state of the art CDN or P2P services alone according to our KPI. Finally, there are no solutions to the best of our knowledge that address all of these problems in the literature using P2P-assisted services combining SDN, WebRTC and edge computing.

ÖZETÇE

Günümüzde OTT ve IPTV servisleri de dahil olmak üzere Internet üzerinden canlı video hizmetleri giderek daha popüler hale gelmektedir. Bu hizmetler, hiper metin aktarım protokolü (HTTP) uyarlamalı akış (HAS) kullanan kullanıcı-sunucu modeline dayanmaktadır. Bir sunucunun kullanabileceği eşzamanlı akış sayısında maksimum bir kısıt olduğundan, isteğe bağlı ölçeklenebilirlik sağlayan popüler çözümler arasında içerik dağıtım ağları (CDN) ve eşler arası (P2P) akış bulunmaktadır. Bugün OTT video hizmetleri, içeriği dağıtılmış uç noktalarda önbelleğe almak için bulut CDN çözümlerini kullanmaktadır. Bulut CDN'leri yalnızca kaynak sunucudaki yükü azaltmakla kalmayıp, aynı zamanda düşük gecikmeyle içerik dağıtımını hızlandırmaktadır. Ancak, CDN altyapısını barındırmak veya kiralamak pahalı olabilir. P2P ortamının ve veri kanallarının kullanımını kolaylaştıran WebRTC protokolündeki son gelişmeler, P2P video akışına olan ilginin artmasını sağlamıştır. Fakat, Internet üzerinden P2P video gönderiminin kontrolsüz doğası kritik bir problem olarak kalmaktadır.

Bir yandan da, iki büyük endüstri trendi gözlemlenmektedir: i) bulut hizmeti sağlayıcıları düşük gecikme süresi ve daha yüksek bant genişliği erişimi için ağ kenarlarına doğru ilerlemektedir (sis bilişim olarak da bilinir), ii) ağ hizmeti sağlayıcıları (NSP) kenar erişim ağlarını yönetmek için kapalı ve özel donanım tabanlı erişim teknolojilerini kenar bulutlarda çalışan ayrıştırılmış ve sanallaştırılmış yazılım ile değiştirmektedir.

Mevcut CDN ve P2P video çözümlerinin eksikliklerini ve SDN tabanlı bulut ve uçtan erişimli ağ teknolojilerinin potansiyelini kabul ederek, ağ servis sağlayıcıları tarafından işletilen kenar erişim veri merkezlerinde barındırılan hibrit bir P2P destekli CDN mimarisi öneriyoruz. Önerilen mimaride, NSP, her bir kenar ağ alanını yönetmek için SDN özellikli bir bulut platformu kullanmaktadır. Önerilen hibrit servis mi-

marisinin önemli bir özelliği, video servisi anahtar performans göstergelerini (KPI) optimize etmek için hem kullanıcılar tarafından CDN erişiminin hem de kullanıcılar arasında P2P video akışının, her bir SDN tabanlı uç ağ etki alanındaki ağ servis sağlayıcısı tarafından kontrol edilmesidir. Bu kontrol edilebilir P2P destekli kenar yönetimli video servisi, CDN sunucularındaki yükü azaltırken, akış başına deneyim kalitesi (QoE) dalgalanmalarının ve ayrılmış bir ağ dilimi üzerindeki birden fazla heterojen video çözünürlüğü kullanıcısı arasındaki adaletsizliğin üstesinden gelir. Bu hizmetin diğer avantajları şunlardır: i) daha iyi video kalitesi, düşük gecikme ve kullanıcılar için tamponlama olmaması; ii) çapraz ISS trafiğini en aza indirmesi; iii) P2P hizmetlerinin izinsiz kullanılmasından kaçınması. Deneysel sonuçlar, SDN destekli kenar ağ platformlarında çalıştırılan önerilen kontrol edilebilir P2P destekli CDN servisinin, KPI'mıza göre tek başına son teknoloji CDN veya P2P hizmetlerinden daha iyi performans gösterdiğini göstermektedir. Son olarak, SDN, WebRTC ve kenar bilişimini birleştiren P2P destekli servisleri kullanarak tüm bu sorunları gideren bir çözüm literatürde bulunmamaktadır.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude and respect to my advisor Prof. Dr. Ahmet Murat Tekalp. Without his guidance, encouragement, persistent help and everlasting patience this dissertation would not have been possible. I also would like to thank to the members of my thesis committee Prof. Dr. Özgür Barış Akan, Prof. Dr. Öznur Özkasap, Assoc. Prof. Dr. Berk Canberk, Asst. Prof. Dr. Ali Cengiz Beğen for their valuable comments.

First of all, I would like to thank my dearest parents Hatice and Necmettin Yılmaz for their strong love and support. They had full confidence in me even when I had little faith in myself. Next, I thank to my greatest chance in life, my beloved husband, İlker Nacaklı. He has been there for me whenever I needed even a little support. I feel very fortunate to have him in my life. Furthermore, I would like to thank my mother-in-law, Kerime Nacaklı, who supported me hugely in the last months of my thesis period. In addition I would like to thank Kadir Tolga Bağcı for all the assistance he gave me. Also, I am very grateful to Gonca Bakar, Rıza Arda Kırmızıoğlu, Ogün Kırmemiş, Akın Yılmaz and Serkan Sülün for all the fun and collaboration we had. Last but not the least, I would like to thank to all my family, friends and research lab colleagues for their full support.

I also acknowledge the support of TUBITAK (The Scientific and Technological Research Council of Turkey) Grants #113E254 and #115E299.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

ALTO	Application-Layer Traffic Optimization
API	Application Program Interface
BGP	Border Gateway Protocol
CDN	Content Distribution Network
CDN APP	Content Distribution Network Application
CDN PoP	CDN Point of Presence
CORD	Central office re-architected as a datacenter
DASH	Dynamic Adaptive Streaming over HTTP
DiffServ	Differentiated Services
E2E	End-to-End
HAS	HTTP Adaptive Streaming
HTTP	Hyper Text Transmission Protocol
IntServ	Integrated Services
IoT	Internet-of-Things
IP	Internet Protocol
IPTV	Internet Protocol Television
ISP	Internet Service Provider
KPI	Key Performance Indicator
M-CORD	Mobile CORD
MPLS	Multiprotocol Label Switching
NFV	Network Function Virtualization
NSP	Network Service Provider
OSPF	Open Shortest Path First

OTT	Over-the-top
P2P	Peer-to-Peer
P2P APP	Peer-to-Peer Application
QoE	Quality of Experience
QoS	Quality of Service
R-CORD	Residential CORD
REST API	Representational State Transfer API
SDN	Software Defined Networking
SEBA	SDN Enabled Broadband Access
SLA	Service Level Aggrement
TTFB	Time to First Bite
UI	User Interface
VSP	Video Service Provider
WebRTC	Web Real-Time Communications

Chapter 1

INTRODUCTION

1.1 Motivation

1.1.1 Content Delivery Models

Live video services, including over-the-top (OTT) and IPTV services, over the Internet are becoming ever more popular. These services are based on the client-server model using adaptive streaming over hyper-text transmission protocol (HTTP). A fundamental limitation of the client-server model is that the video server may get overloaded as demand increases, since there's a maximum number of concurrent streams that a server can handle. Popular solutions to provide demand-scalability include: i) content-distribution networks (CDN), ii) multi-cast streaming, and iii) peer-to-peer (P2P) streaming.

OTT video services commonly employ CDN that stores cached content in multiple geographical locations, known as points of presence. Static and dynamic CDNs not only ease the load on the origin server, but also reduce latency. However, hosting or renting CDN infrastructure can be costly. On the other hand, IPTV services employ Internet protocol (IP) multi-cast streaming over multicast-enabled private networks, where a single stream that originates from the media server is delivered to multiple clients. However, IP multi-cast protocol is not supported in the open Internet; thus, it is not available for OTT video services.

Apart from CDN and multicast, there is P2P video streaming which employs a tracker that maintains a list of clients and the video content that they receive such that every video client simultaneously consumes and delivers a video stream to

other clients enrolled in the service. Recent developments in the WebRTC protocol that offers easy to use P2P media and data channels has led to renewed interest in P2P-assisted video streaming. There have been several concerns about the use of uncontrolled nature of P2P video in the Internet, dominated by bittorrent traffic, which include: i) unauthorized content sharing (content piracy) that is an important concern for content owners, ii) increasing cross-ISP traffic that is an extra cost for network service providers.

The shortcomings of these traditional video delivery solutions necessitate new approaches such as: i) hybrid CDN-P2P video delivery, since CDN-only solution does not scale in a cost-effective manner, ii) controllable-P2P video delivery, which is a new business model for P2P-assisted video services that are controlled by the video service provider (VSP) and/or network service provider (NSP).

1.1.2 Industry Trends

In the meantime, we observe two major industry trends: i) cloud service providers are moving towards network edges for lower latency and higher bandwidth access (also known as fog computing), ii) network service providers (NSP) are replacing closed and proprietary hardware-based access technologies with disaggregated and virtualized software running on edge clouds to manage their edge access networks.

1.1.3 Proposed Solution

We propose a novel hybrid CDN and controllable-P2P streaming architecture for e2e managed video services over edge access networks, leveraging new industry trends in managing access networks as edge clouds, i.e., integrating SDN, edge computing, and WebRTC technologies. This SDN-based controllable-P2P assisted CDN architecture is a hybrid peer-to-peer (P2P) assisted content-distribution network (CDN) hosted at edge-access networks operated by network service providers using software defined networks (SDN), WebRTC data channel and edge computing technologies. In the proposed architecture, the edge access network consists of a group of peers con-

nected through the switches in the edge network and a multi-access edge computing unit, which includes the aggregation switch, SDN controller, CDN PoP (CDN point of presence), and five web servers namely, traffic engineering manager (TEM), ISP accountant, Peer-to-Peer APP, content accountant, and CDN APP which communicate with the SDN controller through REST API. This controllable video service maximizes QoE and minimizes cross-ISP traffic.

Furthermore, the proposed architecture addresses an important concern in OTT video services, which is the lack of service-level guarantees on user experience. Current Internet offers only best-effort service, which makes providing quality-of-experience (QoE) guarantees to clients not possible. There is a recent study that proposes managed client-server video streaming services over software-defined networks (SDN) to provide end-to-end (e2e) service-level guarantees over backbone and edge networks [Bagci et al., 2017]. This thesis extends the proposed managed streaming services over SDN framework to include P2P-assisted controllable video services to reduce the server load. Managed P2P-assisted streaming increases the streaming capabilities of VSP to achieve superior streaming quality without additional CDN infrastructure costs and/or bandwidth costs while offering e2e service-level guarantees to the clients.

Other advantages of managed P2P-assisted video services include: i) better video quality (higher bitrates), lower delay and no buffering for clients; ii) reduction of the load on the centralized or distributed CDN infrastructure of the VSP, and providing insurance against service outages during service-request peaks; iii) minimization of uncontrolled P2P traffic between different NSP; and iv) avoiding illegal, unauthorized usage of P2P services by means that will be explained further. To the best of our knowledge, there are no solutions that address all of these problems in the literature using P2P-assisted services combining SDN, WebRTC and edge computing.

1.2 Background and Related Work

This section of the thesis introduces the background and related works.

1.2.1 Software Defined Networking

Traditional computer networks are typically built from a large number of network devices such as routers, switches and numerous middle boxes, which manipulate traffic for purposes other than packet forwarding, with many complex protocols. Network operators are responsible for configuring policies to respond wide range of network events or applications. They have to transform these high level policies into low-level configuration commands. Such process is a very complex task as they have access to very limited tools. As a result, network management and performance tuning is quite challenging. Another challenge is the unchanging structure of the Internet due to tight coupling between data and control planes. In this type of environment, the deployment of new functionalities is non-trivial as they need to be implemented directly into the physical infrastructure. Even straightforward tasks such as configuration or policy enforcement may require extreme effort since a common control interface to different network devices does not exist. Using middle boxes may be a solution, but they also increase the complexity [Raghavan et al., 2012].

Software defined networks (SDN) is a promising paradigm dealing with the network management problem. SDN is often referred to as a radical new idea in networking. It promises to dramatically simplify the network control/management and enable innovation through network programmability. SDN gives networking the benefits of programming and can reliably build more complicated functionalities [Shenker, 2011], [Nunes et al., 2014]. In SDN, the separation of the forwarding hardware from the control logic allows easier deployment of new protocols and applications. With such separation, physical middle boxes such as firewalls now consolidate into a software control. Instead of enforcing policies and running protocols on distributed devices, the network is reduced to a simple forwarding hardware and the decision making controller(s).

OpenFlow [McKeown et al., 2008] is the standard protocol defining the secure communication between forwarding devices (OpenFlow enabled switches) and controllers. Forwarding devices contain one or more flow tables. Flow tables consist of

flow entries, each of which determines how packets belonging to a flow will be processed and forwarded. Controllers are essentially operating systems of the network [OpenDaylight, 2019] and Floodlight is one example of open-source controller softwares [Floodlight, 2017]. It can be interpreted as if the forwarding devices are nearly the network resources and the controller manages the provisioning of these resources dynamically and manages flows individually. Controllers have three main software interfaces. Southbound interface allows communication with forwarders. Northbound interface enables third party applications which may implement some middle box functions in software or offer new services. East-Westbound interface enables distributed control plane.

The main benefits of logical centralization of network management are to exploit visibility of all network resources and to enable dynamic network reconfiguration and management by use of these network resources. The initial SDN idea supported only a single network domain, yet it is extended to a multi-domain SDN by a distributed, scalable inter-domain management architecture providing end-to-end dynamic path and flow management by further research [Egilmez and Tekalp, 2014], [Civanlar et al., 2015], [Bagci et al., 2016], [Gorkemli et al., 2016].

1.2.2 Multi-Domain and Multi-Operator SDN

SDN has made crucial impact on data center networks by allowing automatic network reconfiguration every time a virtual server has been moved to a different physical machine. It is expected to make a similar impact for service provider networks, called software-defined wide-area networks (SD-WAN).

A multi-domain SDN can refer to two different scenarios: i) Domains are sub-networks of a single service provider network, ii) domains are sub-networks that are operated by different service providers (multi-operator). In either case, controllers of different domains need to communicate and negotiate SLA with each other for a specific service request. There is ongoing research to enable communication and dynamic SLA negotiation among multiple controllers for real-time E2E service-aware

flow management in a fully-distributed multi-domain SDN architecture. The amount of information shared between the controllers and the specific details of the negotiation process may depend on whether all domains are operated by a single service provider or not.

A standard SDN controller has visibility of all network resources within a domain which makes traffic engineering within the domain practical. Egilmez and Tekalp proposed a fully-distributed multi-domain SDN architecture [Egilmez and Tekalp, 2014] and Civanlar *et al.* defined an extended standard SDN controller by adding inter-controller communication features [Civanlar et al., 2015]. Also, other extensions of this concept to SD-WAN have recently been proposed [Mendiola and et al., 2014], [Kotronis et al., 2014], [Phemius et al., 2014]. The first two rely on an external authority, called an orchestrator, for inter-domain path management. Our proposed SD-WAN architecture [Egilmez and Tekalp, 2014], [Civanlar et al., 2015] differs from these two in that dynamic E2E flow management across multiple SDN domains is negotiated directly between domain controllers without a need for an external authority. The last one is independent work that is a distributed architecture similar to the architecture in [Egilmez and Tekalp, 2014], [Civanlar et al., 2015], yet our proposal differs from [Phemius et al., 2014] in that our controller-to-controller messaging is OpenFlow compatible; therefore, controller-to-controller messages can be sent in-band or out-of-band depending on whether a separate control plane network is available or not.

Apart from the multi-domain communication extensions, there are also end-to-end (E2E) service management extensions for the standard SDN [Bagci et al., 2016]. These extensions include inter-controller service-level agreement (SLA) negotiation, extending functionality provided by a bandwidth broker [Bouras and Stamos, 2008] to multi-domain SDN. These functionalities enable dynamic E2E SLA negotiation and traffic engineering over a multi-domain SDN in a scalable manner. The main advantage of implementing traffic engineering and E2E SLA management using SDN framework rather than the MPLS fabric is scalability, since in the SDN framework

signaling for flow (label) management needs to be carried out only between controllers rather than all routers along a path. In addition, this work is further extended by adding the functionality to dynamically manage the control plane performance in software defined networks [Gorkemli et al., 2016].

1.2.3 Cloud Computing and Network Function Virtualization

Cloud is the state-of-the-art in building scalable services and cloud computing differs from the on-premises solutions by how they are accessed and the usage service model they utilize. On-premises solutions, as the name refers, are installed on the computers of users; however, the cloud services are generally hosted by a third party vendor and accessed via internet. Regarding the usage service model, on-premises solutions have upfront capital expenditure, while cloud services are based on on-demand usage service model. As a result, cloud computing is more beneficial when it is compared with the on-premises solutions.

Cloud computing can be considered in three different delivery groups regarding the level of responsibilities, which are infrastructure, platform and software as a service [inCloud360, 2014]. Cloud providers have the least responsibility in infrastructure as a service (IaaS) and the most responsibility in software as a service (SaaS). Yet, for all three delivery groups, the virtualization is the key concept and it is the most significant element of cloud computing.

Furthermore, containers and virtual machines can be compared as forms of virtualizations. The former is operating system virtualization while the latter one is a type of hardware virtualization [CaaS, 2019]. Containers isolate an application as if it is the only one running on a specific operating system; whereas, a virtual machine implements a hardware server which can run different operating systems simultaneously. In addition, containers such as Docker take seconds whereas virtual machines take minutes to start up [Docker, 2019]. While deciding between containers and virtual machines, the scope of the work has a significant effect. A container should be chosen if multiple copies of a single application will be run and a virtual machine makes more

sense if the flexibility of running multiple applications is necessary.

Cloud operating system is another issue regarding the cloud computing and an example of this is OpenStack [OpenStack, 2019]. OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface.

Apart from cloud computing, network function virtualization (NFV) is another key concept. NFV is software implementation of network functions , such as DNS, caching, firewalls, load balancing, path computation, routing, that are decoupled from proprietary hardware appliances, so they can run in software in whiteboxes to accelerate service innovation and provisioning, particularly within service provider environments [NFV, 2019]. There are several open source projects and platforms such as OPNFV [OPNFV, 2019] and MANO [NFV-MANO, 2019].

NFV and SDN are complementary approaches and they each offer a new way to design, deploy and manage networks and services. SDN mainly focuses on network control (Layer 2/ Layer 3/ Layer 4), while NFV addresses network services and management. These approaches are mutually beneficial, but are not dependent on one another. However, SDN makes NFV more compelling and vice-versa. SDN contributes to network automation that enables policy-based decisions to orchestrate which network traffic goes where, while NFV focuses on the services.

Central office re-architected as a datacenter (CORD) [Cord, 2017] is yet another concept which combines three related but distinct threads which are SDN, NFV and Cloud. The goal of CORD is not only to replace todays purpose-built hardware devices with their more agile software-based counterparts, but also to make the central office an integral part of every Telcos larger cloud strategy and to enable them to support more attractive and meaningful networking services. In addition, to demonstrate the opportunity of service deployment at the mobile edge, mobile CORD (M-CORD) [M-CORD, 2016] platform is developed. Lastly, residential CORD (R-CORD) [R-CORD, 2019] is an open source solution based on the CORD platform for

delivering ultra-broadband residential services.

1.2.4 Multi-Access Edge Networks

There is an incredible transition towards the edge by the industry as the datacenters are established as part of the global computing infrastructure currently. This transition creates a huge oppportunity for edge network innovations [Peterson et al., 2019]. We observe two major industry trends: i) cloud service providers are moving towards network edges for lower latency and higher bandwidth access (also known as fog computing), ii) network service providers (NSP) are replacing closed and proprietary hardware-based access technologies with disaggregated and virtualized software running on edge clouds to manage their edge access networks.

When we consider the first trend, which is the transition of the cloud service providers to the edge, we can definitely see that cloud service providers are aware that cloud computing is not enough despite its many advantages such as agility, massive centralization and full automation [Bittman, 2017], [Levine, 2016]. Weight of data is still a critical concern. It is not logical for the clients sitting at the edge to wait data from the datacenters. Internet-of-Things (IoT), immersive UIs and autonomous vehicles are some applications that will create the edge explosion. Hence, edge computing will play a very crucial part in the future Internet.

The second trend that we observe is that network service providers (NSP) are replacing closed and proprietary hardware-based access technologies with disaggregated and virtualized software running on edge clouds to manage their edge access networks. IHS denoted that %85 of operators plan to deploy central office re-architected as a data center (CORD) in smart central offices [Longwell, 2018], which will eventually bring the end-user closer to the operators. The next step from here is the R-CORD extension SDN Enabled Broadband Access (SEBA), which generalizes all access technologies [ONF, 2019].

1.2.5 Video Service Models

Video service models can be investigated in four groups which are one-way streaming, live streaming, peer-to-peer streaming and bi-directional streaming.

Dynamic adaptive streaming over HTTP (DASH) is the state-of-the-art regarding one-way streaming video services. It is a video streaming solution where small pieces of video streams or files are requested with HTTP and spliced together by the client [DASH, 2019]. The DASH client entirely controls the delivery. There are several works utilizing DASH over SDN in the literature [Bagci et al., 2016], [Cetinkaya et al., 2015]. In addition, Netflix and Youtube are some provider services that utilize DASH.

Live streaming, which refers to content delivered live over the Internet, requires a form of source media (e.g. a video camera, an audio interface, screen capture software), an encoder to digitize the content, a media publisher, and a content delivery network to distribute and deliver the content [Wikipedia, 2019]. Regarding live server-client video streaming, multicast is a technique that decreases congestion in the service provider network and can be combined with the SDN concept [Noghani and Sunay, 2014].

The third video service model is peer-to-peer (P2P) streaming. In P2P, the task of servers in conventional client-server systems, which is to provide data to multiple clients, is distributed among peers where each peer acts as both the server and the client at the same time so that upload capacity of each peer is utilized. This approach is highly attractive for server owners since each peer contributes to the available bandwidth [P2PTV, 2019]. If a user wishes to view a certain channel, a tracker server is contacted for that channel in order to obtain addresses of peers who distribute that channel; it then contacts these peers to receive the feed. The tracker records the user's address, so that it can be given to other users who wish to view the same channel.

Lastly, there is the bi-directional video streaming model. As the name suggests, both ends of the video service both download and upload streams. Some examples of bi-directional video service providers are Skype and Google Hangouts

[WebRTC, 2019]. In addition, WebRTC is an open source project for browser-based real-time communication started by Google in May 2011. This has been followed by ongoing work to standardize the relevant protocols in the IETF and browser APIs in the W3C [WebRTC, 2019].

1.2.6 End-to-End Concept

In the literature, there are three definitions of the end-to-end concept regarding the communication network.

(i) End-to-end concept in CORD [Cord, 2017] is as following. As explained earlier, CORD tries to decrease edge network congestion and it provides the user connection links to the core network. In other words, CORD is the blackbox through which the edge networks connect to internet. As a result, the end-to-end concept in CORD is just about providing internet access to the edge networks but there is no quality of service guarantees.

(ii) In some studies such as [Lopez et al., 2017], end-to-end concept is just about connecting the core and edge network in a single operator network.

(iii) The best definition of end-to-end concept is the one that connects the edge and core network while considering multiple operator networks as it is a more realistic approach. For example, studies [Bagci et al., 2016], [Phemius et al., 2014] consider end-to-end concept in a multi-operator environment with quality of service guarantees.

1.2.7 Quality of Service Models

In order to enable end-to-end quality of service (QoS), both intra and inter domain routing protocols have to be QoS-aware. For intra-domain QoS, there is QoS extension of OSPF. Although there exist some QoS extensions of BGP, it is hard to predict end-to-end QoS parameters due to the hop-by-hop nature of BGP. In order to enable

some QoS, IETF published standards, such as IntServ [Braden et al., 1994], DiffServ [Blake et al., 1998] and MPLS [Farrel et al., 2008]. MPLS has found some use; however, it is difficult to configure an IP network according to predefined policies and to reconfigure it dynamically to respond to faults, load and changes.

The studies [Civanlar et al., 2010], [Egilmez et al., 2011], [Egilmez et al., 2012a], [Egilmez et al., 2012b], [Egilmez et al., 2012c], [Egilmez et al., 2013] are examining end-to-end QoS for the software-defined networks. Apart from these, there are just a few studies about this issue. For instance, in [Kim et al., 2010], separate reservations are made dynamically for each network slice on each switch by an application running on the SDN controller.

Network traffic routing problem can be formulized as a constrained path selection problem, which is known as NP-complete and can be solved by heuristic and approximation algorithms. The literature review till the year 2006 can be seen in [Masip et al., 2006], [Kuipers et al., 2002]. A better and developed approximation algorithm can be found in [Xue et al., 2008]. In addition, [Juttner et al., 2001] presents a solution using LARAC algorithm to find the best path with the delay constraint and this algorithm works on real-time and also gives the down limit that can be reached theoretically to compare the result. Then, two other approximation algorithms that can be alternatives to LARAC were presented in [Chen et al., 2008]. Regarding the QoS in inter-domain routing, the extensions done to the BGP method can be seen in [Li Xiao et al., 2002]. Lastly, a literature review on network topology aggregation and choosing metric can be seen in [Uludag et al., 2007].

As for the QoS for peer-to-peer streaming services, IETF ALTO group suggested a network information gathering system [Alimi et al., 2014] that will be run by a service provider. However, there is no solution that financially optimizes the benefits of all the shareholders in peer-to-peer services.

1.2.8 Peer-to-Peer Architectures

There are two basic peer-to-peer architectures in the literature. These are mesh-based and tree-based peer-to-peer solutions.

In mesh-based solutions, data is distributed over an unstructured network in which each peer can connect to multiple neighbors with no parent-child relation. Mesh-based solutions require an initiation interval in which multiple connections are built dynamically. Therefore, mesh-based solutions introduce some initiation delay. However, they are robust against peer exit events as in case of a sudden interruption in the data transfer a peer can immediately connect to any other neighbor and so the data transfer continues. The second type of peer-to-peer architectures is the tree-based solutions. In a tree-based solution, the peers are connected to each other in a parent-child fashion and the server remains at the top of the tree. Content is pushed from the top to the leaves of the tree and therefore tree-based solutions introduce lower latency in data dissemination. However, there is a major problem regarding the tree-based solutions which is the ungraceful peer exits. When a peer exits, its descendants lose their parent and therefore get disconnected from the tree. Besides on-the-fly tree construction, there are two alternative solutions to this problem, which are to use backup parent pools [Fesci-Sayit et al., 2012] and to use multiple trees [Castro et al., 2003].

1.2.9 Managed Video Services

Managed video services are examined in three parts which are: i) Managed DASH video services, ii)Managed WebRTC video services, and (iii) Managed P2P video services.

Managed DASH Video Services

During a DASH video services, several clients share the same bottleneck links and compete with each other over them. This results in QoE fluctuations and unfairness while the network is underutilized. Managed DASH video services can be investigated in two parts which are NSP-managed [Bagci et al., 2016], [Bagci et al., 2017],

[Bagci and Tekalp, 2018] and VSP-managed [Sahin et al., 2017], [Bagci et al., 2017].

NSP-managed DASH video services can guarantee end-to-end quality of service in two alternative ways. These are to allocate queues per flow or to write rules to the switches per flow during the video streaming session. The former will introduce queue processing delay and also for each flow the queues will be allocated or deleted. Though, it will not be a big problem as this procedure will only be done in the edges [Bagci et al., 2016]. The latter approach, which is to write rules to the switches, will introduce rule processing time. In addition, there is another study [Bagci et al., 2017] which proposes an NSP-managed DASH video service. In this architecture, the NSP, VSP and the clients collaborate with each other and the fairness between the DASH clients are managed by the NSP itself. Next, this work is extended to enable the NSP to offer value-added video services [Bagci and Tekalp, 2018].

VSP-managed DASH video services are proposed in studies [Sahin et al., 2017], [Bagci et al., 2017]. In this architecture, the network SDN controller collects the network information and shares it with the video service provider management server. This information is used to make a grouping based on the congested link and traffic information between the clients. The clients that share some congested links are grouped together to serve them in a fairer and more efficient way. After the clients are grouped, the fair share bitrates are calculated either by the VSP in a centralized manner [Bagci et al., 2017] or by the clients themselves in a distributed fashion [Sahin et al., 2017]. This architecture enables a fairer and more efficient video streaming session for each client.

Managed WebRTC Video Services

At present, multi-party WebRTC videoconferencing between peers with heterogenous network resources and terminals is enabled over the best-effort Internet using a central selective forwarding unit (SFU), where each peer sends a scalable encoded video stream to the SFU. This connection model avoids the upload bandwidth bottleneck associated with mesh connections; however, it increases peer delay and overall network

load (resource consumption) in addition to requiring investment in servers since all video traffic must go through SFU servers. To this effect, a new multi-party WebRTC service model over future 5G networks, where a video service provider (VSP) collaborates with a network service providers (NSP) to offer an NSP-managed service to stream scalable video layers using software-defined networking (SDN)-assisted Internet protocol (IP) multicasting between peers using NSP infrastructure is proposed in our earlier studies [Kirmizioglu and Tekalp, 2019].

Managed P2P Video Services

Despite its many advantages, the uncontrollable behaviour of P2P video services still remains as a critical problem. There are some proposals in the literature aiming to solve the problems arising from the uncontrollable behavior of P2P service. There are many P2P studies aided by the ALTO protocol [Alimi et al., 2014]. The P2P trackers communicate with ALTO servers that have the network information (e.g., basic network location structure and preferences of network paths) with the goal of modifying network resource consumption patterns while improving application performance. In addition, the grouping of peers from different operator networks results in a heavy cross-ISP network traffic. This cross-ISP traffic is reduced in [James and Crowley, 2010] via an ISP-friendly, transparent, peer-discovery service. There is also P4P [Xie et al., 2008] study introducing p-distances and iTrackers. In this study peer selection is based on location thus achieving more efficient network resource usage and higher performance. Apart from these, there are managed P2P architectures for IPTV that increase the client streaming quality [Jeon et al., 2009] and provide rewind function through a P2P network [Cha et al., 2008]. There is also a hybrid architecture integrating P2P and cloud that aims to increase the QoE of the clients [Trajkovska et al., 2010]. SDN-based P2P implementation [Shibasaki et al., 2016] increase the streaming quality with the integration of SDN; however, the drawbacks of P2P service on NSPs and VSPs are not considered.

Several recent studies focus on integrating P2P video streaming with WebRTC

technology for streaming and sharing objectives, instead of relying on a third-party plug-in or proprietary software. Some of these studies are SwarmCDN, PeerCDN and Sharefest [Huang and Zhang, 2014]. They all utilize the WebRTC data channel protocol, which is designed to allow browsers to exchange data other than audio or video. There are also commercialized hybrid CDN-P2P [Barbosa and Soares, 2014] products by Streamroot [Streamroot, 2020] and Peer5 [Peer5, 2017]. The hybrid architecture helps decrease the load on CDN servers and transmission cost while taking advantage of P2P distributed computing to improve user experience and system scalability.

1.2.10 Modeling Internet Topology

There are many graph models such as mesh, ring or randomly generated ones that researchers can utilize in their studies. However, none of these models help the researchers to generate valid conclusions as these graph models are not applicable to real internetworks. Therefore, modeling real internetworks is a critical issue for the study of all network related problems such as routing or resource reservation.

Firstly, physical distance based probabilistic model was proposed [Waxman, 1988]. According to this model, the edge probability between nodes u and v is given in Equation 1.1 where $d(u, v)$ is the euclidean distance between nodes u and v, L is the maximum distance between any two nodes and α and β are parameters affecting the density of short and long edges.

$$P(\{u, v\}) = \beta exp(\frac{-d(u, v)}{L\alpha}) \tag{1.1}$$

Next, Transit-Stub model is proposed [Zegura et al., 1996]. Today's Internet comprises of many interconnected routing domains, which can be defined as either a stub domain or a transit domain. Stub domains are like the leaves of a tree, they are the edge networks of the real Internet structure. Transit domains are the domains connecting stubs to each other; therefore, we can say that the transit domains are emulating the backbone network. Then, this work is further extended and the Transit-Stub generation software GT-ITM [Calvert et al., 1997] is proposed. This tool is being

utilized to generate real internetwork models since then. There is a multi-domain topology model [Bagci et al., 2016] for Mininet [Mininet, 2017] and it is generated in the same way as in the GT-ITM tool.

1.3 Contributions of the Thesis

The main contributions of this thesis are listed below:

- A novel content delivery service architecture integrating CDN-P2P hybrid content distribution model with the emerging multi-access edge clouds is proposed.

- An important feature of the proposed hybrid service architecture is that both the CDN access by clients and P2P video streaming between clients are controlled by the network service provider within each SDN-based edge access network domain to optimize video service key performance indicators (KPI).

- A novel SDN-assisted controllable P2P application with the following features is proposed:

  (i) The proposed CDN PoP location aware neighborhood formation is an extension of the K-means clustering algorithm, which brings the groups closer to the CDN PoP.

  (ii) The proposed centralized chunk scheduling method favors all the peers receiving the live video content and schedules while prioritizing the peers with high incentive points.

  (iii) We developed ISP accountant and content accountant for fair charging and incentive policy.

- The desired QoE is supplied within the edge domains leveraging SDN controller of multi-access edge clouds.

- The shareholders of the controllable P2P service are the content providers, service providers and the users. We propose a model that financially optimizes the benefits of all the shareholders to be accepted and supported by them.

- P2P piracy is avoided by means of WebRTC and CDN encryption.

1.4 Organization of the Thesis

The organization of the rest of this thesis is as following:

- Chapter 2 introduces the proposed architecture which is SDN-based controllable P2P-assisted CDN for HTTP adaptive live video streaming over edge access networks. This architecture combines SDN, WebRTC and edge computing technologies not only to reduce the load on CDN servers but also to overcome QoE fluctuations per flow and unfairness between clients. The proposed architecture is given in Section 2.1. The components of this architecture, which are P2P APP, CDN APP, traffic engineering manager (TEM) and accountants are explained in Sections 2.2, 2.3, 2.4 and 2.5, respectively. The service provision information flow is introduced in Section 2.6. Lastly, Section 2.7 mentions the security issues.

- Chapter 3 presents the implementation method of the proposed architecture, which is introduced in Chapter 2. The implementation details of group formation and related algorithms are given in Section 3.1. The centralized chunk scheduling algorithm is introduced in Section 3.2. Section 3.3 explains the details of traffic engineering. The chunk transmission details with the sender and receiver side transmission algorithms are presented in Section 3.4. Lastly, Section 3.5 concludes this chapter by introducing the implementation of accounting.

- Chapter 4 includes the evaluation of the proposed architecture. Firstly, the experimental setup is presented in Section 4.1. Next, the experimental scenarios

are explained in Section 4.2. Lastly, the results obtained are given in Section 4.3.

- Chapter 5 presents the concluding remarks.

Chapter 2

CONTROLLABLE-P2P-ASSISTED CDN OVER EDGE ACCESS NETWORKS

2.1 Proposed Service Architecture

An e2e-managed video service should not only manage the traffic at the NSP backbone and edge networks but also consider the server-side congestion. To this effect, we propose a managed P2P-assisted hybrid video service architecture, where clients request (pull) some chunks from the CDN, while remaining chunks are sent (pushed) P2P using the WebRTC data channel under the supervision of SDN controller. Our proposed formulation jointly optimizes routing in the edge networks as well as the load of CDN servers by means of optimal peer selection and P2P chunk scheduling. The optimization criteria include i) maximization of video quality at clients, ii) minimization of CDN server load, iii) minimization of cross-ISP traffic, and iv) minimization of bandwidth consumption. The proposed formulation successfully addresses the substantial problem of "uncontrolled P2P video", where the P2P content sharing and P2P traffic routing is now under full control of the SDN controller of the edge access network to eliminate unauthorized sharing.

The proposed service architecture is depicted in Figure 2.1 for the case of one edge-access network; however, it can be generalized to the case of multiple edge-access networks connected through the core network via their aggregation switches. The edge access network consists of a group of peers connected through the switches in the edge network and a multi-access edge computing unit. This multi-access edge computing unit includes the aggregation switch, SDN controller, CDN PoP, and five web servers namely, traffic engineering manager (TEM), ISP accountant, Peer-to-Peer

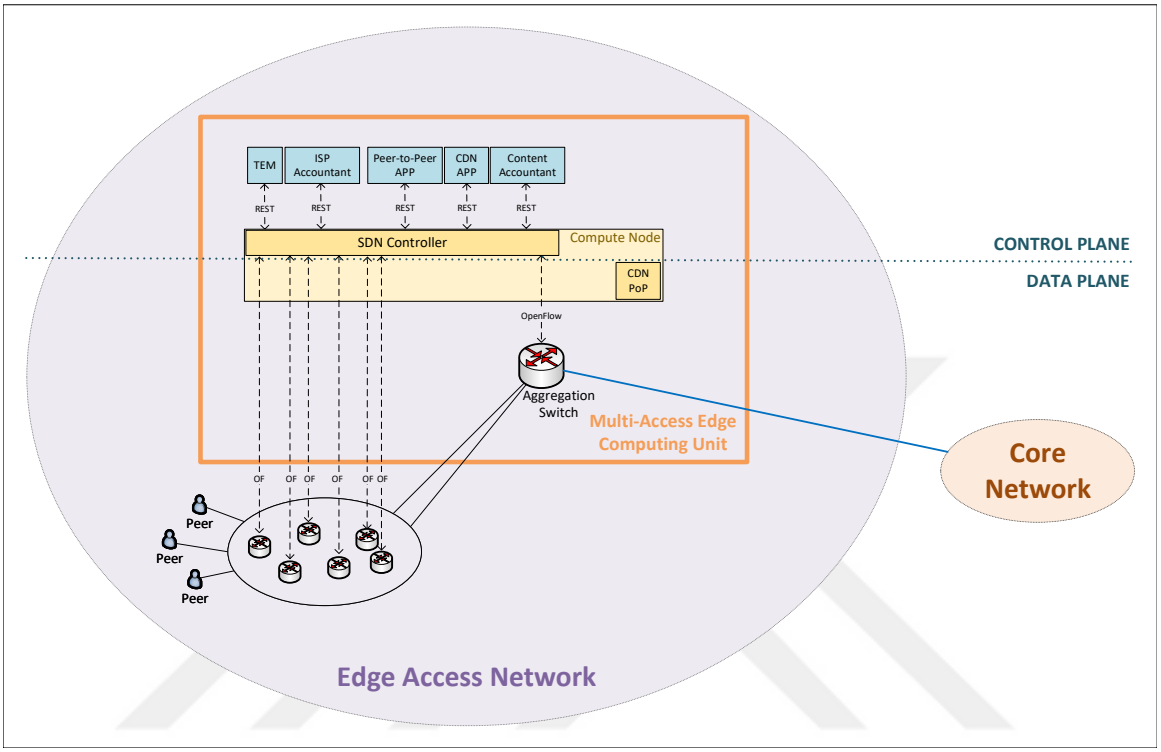


Figure 2.1: System architecture.

APP, content accountant, and CDN APP which communicate with the SDN controller through REST API. In addition, all the switches in the edge network communicate with the SDN controller through OpenFlow protocol including the aggregation switch, which is in the multi-access edge computing unit.

2.2 Peer-to-Peer APP

Peer-to-peer APP entity is placed in a web server by the VSP and it is able to communicate with the edge access network SDN controller via REST API. It is essential that the full control is given to the NSP controller because topology related information is needed while forming the groups and these information should be kept in the NSP for security and privacy issues.

This module has two main jobs, which are the formation of the neighborhood groups and the centralized chunk scheduling, in the proposed architecture. The neigh-

borhood groups consist of multiple peers having same uplink/downlink bandwidths that are subscribed to the same ISP. The groups are subject to change dynamically in case of peer churn events according to the decisions of this module. Next, centralized chunk scheduling is proposed in this thesis and peer-to-peer APP module handles this job. In standard P2P chunk scheduling systems, the peers themselves do the scheduling as a result they end up favoring themselves. However, centralized scheduling enables a fairer scheduler algorithm considering buffer status and incentive points of all the peers.

2.3 CDN APP

CDN APP entity is placed in a web server by the VSP and it is able to communicate with the edge access network SDN controller via REST API. This module is responsible from starting the controllable-P2P-assisted service, pulling live video content from the datacenter to CDN PoP and charging the peers according to their download amounts.

2.4 Traffic Engineering Manager (TEM)

Traffic engineering manager entity is placed in a web server by the NSP and it is able to communicate with the edge access network SDN controller via REST API. The task of this module is to formulate and solve an optimization problem to determine the routes the packets will use.

2.5 Accountants

The proposed architecture deploys the accountant modules in order to provide a fair charging and incentive policy for P2P video streaming services and includes two types of accountants in the SDN-enabled cloud platform of NSP:

(i) ISP accountant placed by the NSP

(ii) content accountant placed by the VSP

The video service provider is not able to charge the P2P clients fairly in a standard P2P video service. To solve this issue, the VSP should be notified about the download amounts of each client and the accountant modules become essential for this reason. The ISP accountants keep track of the download and upload amount of each peer in their own edge access network and then these ISP accountants feed the session information to the content accountant.

The second problem that the accountant modules aim to solve is providing a fair incentive policy. The peers download the chunks from the P2P service, yet they do not contribute to the upload process adequately in the standard P2P video services. This thesis proposes a managed P2P video service that involves an incentive policy for peers that contribute more to the upload process by the help of accountant modules.

2.6 Service Provision Information Flow

The information flow in the edge access network for the controllable-P2P-assisted CDN service consists of the following steps:

1. The client requests the video with a required specific QoE from the CDN APP.

2. CDN APP passes client IP and the parameters of the requested video to the edge access network SDN controller.

3. SDN controller transfers these lists and the topology information with link capacities to its P2P APP through REST API.

4. TEM calculates the shortest paths from each peer and CDN PoP and to each peer receiving the video service.

5. P2P APP forms the P2P neighborhood groups including multiple peers, which are able to reach CDN PoP, using the topology information received from the SDN controller as described in Section 3.1.

6. SDN controller, clients and the CDN APP are notified about the neighborhoods.

7. CDN APP start pulling content from the VSP datacenter to CDN PoP, which is the only event that introduces cross-ISP traffic in the proposed architecture.

8. TEM calculates the multicast paths for the P2P startup phase.

9. Peers are notified about the paths and they open WebRTC data channel between themselves.

10. SDN controller make resource reservation per group.

11. Until all the peers received at least a pre-determined number of chunks, startup phase continues.

12. During the startup phase, the ISP accountant keeps track of the download and upload information of each peer as introduced in Section 3.5.

13. When startup phase ends, SDN controller releases the resources reserved for the startup phase and the peers close the WebRTC data channels.

14. After startup phase, post-play P2P scheduling starts.

15. Peers are notified about the previously found shortest paths and they open WebRTC data channel between themselves.

16. SDN controller make resource reservation per group.

17. P2P APP starts post-play P2P chunk scheduling for each peer in intervals of time as explained in Section 3.2.

18. For each new scheduling decision, peers continue using the service.

19. During the post-play phase, the ISP accountant keeps track of the download and upload information of each peer as introduced in Section 3.5.

20. In case of rerouting event described in Section 3.2, TEM recalculates the path and WebRTC data channel path between sender and receiver is updated. When the rerouted chunk and the ones sent with it in the same interval are received by the receiver peer, rerouting is made again to continue giving service to that peer from the shortest path.

21. When all the peers have downloaded all the chunks, P2P APP notifies the SDN controller to release the resources reserved for the P2P post-play session and to instruct CDN APP to stop requesting the content from the datacenter to CDN PoP. Also, the peers are notified so that they close the WebRTC data channels.

22. SDN controller takes the download and upload information from ISP accountant and passes it to content accountant.

2.7 Security Issues

The proposed architecture is a secure system and P2P piracy is avoided by the following means. WebRTC data channel being used for the chunk transmission enables the secure communication between peers. In addition, all the peers pull several of their first chunks from the CDN node. This enables the P2P video service to be notified about all the legal users of the service and to deny the illegal users by not sharing the encryption keys.

Chapter 3

IMPLEMENTATION

3.1 Group Formation

Peer-to-peer APP web server entity implements the neighborhood group formation. This entity has the information of client lists and the topology information with available link capacities. Using these information cross-ISP traffic and the total cost of each group is minimized by the neighborhood formation algorithms proposed in this thesis.

Firstly, bandwidth aware graph simplification for each session of P2P service is made. The details can be seen in Algorithm 1. The topology graph of the edge access

Algorithm 1: Bandwidth-Aware Graph Simplification

Input : $G, session_i$

Output: G_i

`/* Topology graph is simplified so that all the links in the new graph has available bandwidth more than the minimum bandwidth requirement of the session. */`

1 **foreach** $session_i$ **do**

2 $packetSize = session_i.minPacketSizeinBits$;

3 $time = session_i.maxHopTimeinSeconds$;

4 $numberofChunks = session_i.minNumberofChunksonLinkatSametime$;

5 $reqBW_i = (packetSize * numberofChunks)/time$;

6 Create graph G_i such that $\forall$ Link $\in G \geq reqBW_i$;

7 Return G_i;

network is simplified so that all the links in the new graph has available bandwidth more than the minimum bandwidth requirement of the session. As it can be seen from the algorithm supplied, the minimum and maximum requirements are obtained between lines 2-4, then the required bandwidth is calculated accordingly in line 5. The bandwidth aware graph simplification is done by excluding the links with less available bandwidth than the required bandwidth from the graph (line 6). This procedure is repeated for each session and lastly the simplified topology graphs are returned. This simplification guarantees that all the links in a P2P session has the minimum required available bandwidth.

Next, the CDN PoP location is calculated and its details are shown in Algorithm

Algorithm 2: CDN PoP Location Calculation

Input : G

Output: $nodeID$

1 Find $midPoint$ of the edge graph;

2 $r = (min(midPoint_x, midPoint_y))/2$;

`/* Nodes located in the circle centered in the middle with a radius of` r `are found. */`

3 $possibleNodes = \{nodes_i | (||nodes_i - midPoint|| < r)\}$;

`/* The node with maximum number of direct links is chosen as the CDN PoP node. */`

4 $linkCount = 0$;

5 $nodeID = -1$;

6 **for** $i \leftarrow 0$ **to** $len(possibleNodes)$ **do**

7 **if** $len(G.get(possibleNodes_i)) > linkCount$ **then**

8 $linkCount = len(G.get(possibleNodes_i))$;

9 $nodeID = possibleNodes_i$;

10 Return $nodeID$;

2. By definition, CDN PoP should be able to reach all the peers in the edge access network with ease as CDN PoP will be serving chunks to the peers in case of CDN pull event. For this reason, we propose to choose the switch which is close to the center of the edge access network and has the most direct links to other switches in the network as CDN PoP. After the midpoint and the half of the minimum of its coordinates (r) are found, the switches inside the circle centered at the middle with a radius of r are chosen to be the possible nodes for the location of CDN PoP in line 3. Then, the switch with the most direct links is chosen as CDN PoP location as seen in lines 6-9.

Neighborhood group formation algorithm is depicted in Algorithm 3. This algorithm takes two inputs: (i) G, which is the bandwidth aware simplified topology graph of peers with same $uplinkBW$, $downlinkBW$, ISP and $reqBW$, and (ii) $node_{CDN}$, which is CDN PoP. The nodes to be grouped are the ones with direct links to peers and they are found by the SDN controller examining the devices connected to the OpenFlow switches and their attachment points (line 1). We propose neighborhood group formation as an extension of K-means clustering algorithm; therefore the required k value is chosen in line 2 according to the size of the edge access network. Then, in line 3, the shortest paths from $node_{CDN}$ to all peers and the paths between all peers are calculated with Dijkstra's shortest path algorithm so that the path calculations are not repeated in each iteration. The path information is taken from the memory in neighborhood formation and scheduling iterations. Next, the extended K-means algorithm is called to make the group formations. According to peer churn events, which are peer arrivals and exits, the groups can change dynamically during the P2P video service. Peer arrival event can be seen in lines 6-13 of Algorithm 3. If peer p joins P2P service, then it is placed into the group with whose nodes peer p has the minimum average hop count. In addition, peer exits may reduce the number of peers in group i below a threshold. This event is depicted in lines 14-15 of Algorithm 3. Each peer in group i is considered as a peer arrival event and after the peers are placed into their new groups, group i is deleted.

Algorithm 3: Neighborhood Group Formation

Input : $G, node_{CDN}$

Output: $groups, spPaths$

/* Input G is the bandwidth aware simplified topology graph of peers with same $uplinkBW$, $downlinkBW$, ISP and $reqBW$. */

/* Find nodes with direct peer connections. */

$nodes_N = findNodesToGroup()$;

Choose k according to edge size;

/* Find the shortest paths both from CDN PoP to all peers and between all the peers. */

$spPaths = getspPaths(peers, node_{CDN})$;

/* Run CDN PoP location aware K-means with hop count algorithm to group peers. */

$groups = CDNpopLocationAware\_Kmeans(k, G, nodes_N, node_{CDN})$;

Return $groups, spPaths$;

/* Event: Peer p joins P2P service. */

$id = 0$;

$cost = avgHopCount(p, groups_0)$;

for $i \leftarrow 0$ **to** $len(groups)$ **do**

    $temp = avgHopCount(p, groups_i)$;

    if $temp < cost$ **then**

        $cost = temp$;

        $id = i$;

Add p to $groups_{id}$;

/* Event: Number of Peers in $groups_i$ is below Threshold */

Treat each peer p in $groups_i$ as a peer arrival event;

Delete $groups_i$;

There are studies extending K-means clustering method with hop count which is given in Algorithm 4 [Datta et al., 2009]. We further extend it as CDN PoP location aware K-means with hop count and it is given in Algorithm 5.

K-means clustering algorithm with hop count calculates the distance as the num-

ber of hops in the shortest path between nodes. Also instead of the mean of the sum of euclidean distances, the node with the minimum total hop count to the rest of the nodes in its cluster is chosen as the new centroid in each iteration.

Algorithm 4: Kmeans with Hop Count

Input : $k, G, nodes_N$

Output: *groups*

Select a random subset $C1 = c0_0, c0_1, ..., c0_{K-1}$ of $nodes_N$ as the initial set of centroids;

$C0 = 0$;

$notGrouped = TRUE$;

while *notGrouped* **do**

  $C0 = C1$;

  /* Find cluster of each node in $nodes_N$. */

  for $i \leftarrow 0$ **to** $len(nodes_N)$ **do**

    $cluster_i = \underset{k \in 0,...,K-1}{\operatorname{argmin}} \ hopCount(nodes_i, c0_k)$;

  /* Recalculate centroids. */

  for $k \leftarrow 0$ **to** K **do**

    /* $groups_k$ contains the set of $nodes_i$ with minimum hop count to $c0_k$. */

    $groups_k = \{nodes_i | cluster_i = k\}$;

    /* Recalculate centroids as the minimum hop $nodes_i$ to the rest in $groups_k$. */

    $hopCount = Integer.MAX\_VALUE$;

    for $i \leftarrow 0$ **to** $len(groups_k)$ **do**

      $count = 0$;

      for $j \leftarrow 0$ **to** $len(groups_k)$ **do**

        $count+ = hopCount(groups_i, groups_j)$;

      if $count < hopCount$ **then**

        $hopCount = count$;

        $c1_k = i$;

  if $C0 == C1$ **then**

    $notGrouped = FALSE$;

Return *groups*;

Algorithm 5: CDN PoP Location aware Kmeans with Hop Count

Input : $k, G, nodes_N, node_{CDN}$

Output: *groups*

Select a random subset $C1 = c0_0, c0_1, ..., c0_{K-1}$ of $nodes_N$ as the initial set of centroids;

$C0 = 0$;

$notGrouped = TRUE$;

while *notGrouped* **do**

  $C0 = C1$;

  /* Find cluster of each node in $nodes_N$. */

  for $i \leftarrow 0$ **to** $len(nodes_N)$ **do**

    $cluster_i = \underset{k \in 0,...,K-1}{\operatorname{argmin}} (hopCount(nodes_i, c0_k) + hopCount(nodes_i, node_{CDN}))$;

  /* Recalculate centroids. */

  for $k \leftarrow 0$ **to** K **do**

    /* $groups_k$ contains the set of $nodes_i$ with minimum total hop count to $c0_k$ and CDN PoP. */

    $groups_k = \{nodes_i | cluster_i = k\}$;

    /* Recalculate centroids as the minimum hop $nodes_i$ to the rest in $groups_k$ and CDN PoP. */

    $hopCount = Integer.MAX\_VALUE$;

    for $i \leftarrow 0$ **to** $len(groups_k)$ **do**

      $count = 0$;

      for $j \leftarrow 0$ **to** $len(groups_k)$ **do**

        $count = count + (hopCount(groups_i, groups_j)) + hopCount(groups_i, node_{CDN}))$;

      if $count < hopCount$ **then**

        $hopCount = count$;

        $c1_k = i$;

  if $C0 == C1$ **then**

    $notGrouped = FALSE$;

Return *groups*;

Algorithm 5 further extends Algorithm 4 in order to improve the performance metrics of the controllable-P2P-assisted video service, which are the reduction of the CDN load and the bandwidth consumption. K-means clustering with hop count forms nicely separated groups; however, some groups are far from CDN PoP. This is considered as a drawback as it will reduce the performance of the service. Therefore, we propose CDN PoP location aware K-means clustering with hop count depicted in Algorithm 5. The distance metric used here is the sum of the hop counts to the centroid and CDN PoP. The cluster of each node is found according to this distance metric as it can be seen in lines 6-7. After the nodes are assigned to their new clusters (lines 8-9), the new centroids are calculated in lines 10-17. The proximity of the centroid to CDN PoP is as important as the mean of the total hop count to the nodes in its cluster. Therefore, while calculating the new centroid, the hop count to CDN PoP is added to the summation for each node in the cluster. When the algorithm converges (lines 18-19), the final centroids and the final clusters are found.

3.1.1 Complexity Analysis of Neighborhood Formation

Firstly, let us analyse the complexity of finding the shortest paths. The complexity of Dijkstra's shortest path calculation on a graph with edges E and vertices V can be expressed as a function of the number of edges, denoted $|E|$, and the number of vertices, denoted $|V|$ and it is given as $\mathcal{O}(|E| + |V| \log |V|)$.

Next, the complexity of the K-means clustering algorithm is $\mathcal{O}(nki)$, where n is the number of nodes to be grouped, k is the number of clusters, and i is the number of iterations until convergence. [Arthur and Vassilvitskii, 2006] proposed that i is often small, and results only improve slightly after the first dozen iterations. Lloyd's algorithm is therefore often considered to be of "linear" complexity in practice, although it is in the worst case superpolynomial when performed until convergence.

The total complexity of the neighborhood formation algorithm is $\mathcal{O}((nki) + (|E| + |V| \log |V|))$.

3.2 Chunk Scheduling

P2P APP web server entity implements the centralized chunk scheduling. This entity has the information of client lists and the topology information with available link capacities. After the neighborhoods are formed, P2P APP starts chunk scheduling. We propose a centralized chunk scheduling which favors all the peers at the same time, thus performing better than the typical distributed scheduling algorithms that state of the art P2P services implement.

The general flow of the centralized chunk scheduling can be seen in Algorithm 6. Centralized chunk scheduling algorithm schedules each peer and CDN PoP to push each chunk to its leaf peers from the multicast path in the P2P startup phase to decrease the startup delay. Therefore, the multicast tree is constructed and the leaf peers of each peer and CDN PoP is found in lines 3-6.

The startup phase of the controllable-P2P-assisted CDN service continues until all the peers have at least the minimum number of chunks to save them from low buffer status. After the scheduling is done, the peers and the CDN APP are notified about the decisions for each time interval in the startup phase. When all the peers have at least normal buffer status, the centralized scheduling algorithm waits for any packet sent earlier to reach its destination (lines 12-13). *startupDelay* is calculated at the end of the startup phase. Then, the peers are notified and they start playing the video. This phase is called the post-play P2P scheduling. We propose *schedulingInterval* to decrease the number of signalling between P2P APP and the peers. Until all chunks are received by all the peers, post-play P2P scheduling continues in intervals of *schedulingInterval*. At the end of each interval the peers are notified about the scheduling decisions, which can be (i) the chunk number information of the chunks (given in order) they will pull from CDN PoP, (ii) the destination and chunk number information of the chunks they will upload, and (iii) the chunk number information of the chunks (given in order) they will receive from the other peers in their neighborhood group. Also, the total values of bandwidth consumption, CDN load, rerouting and freeze event counts are calculated for each neighborhood group.

Algorithm 6: Centralized Chunk Scheduling - General Flow

Input : $videoPacketCount, reqBW, peers, peer_{CDN}, G, spPaths$

Output: $startupDelay, bwConsumption, CDNload, reroutingCount, freezeCount$

$timeIntervalMS$;

$schedulingInterval$;

`/* The optimum multicast tree from CDN PoP to peers is found. */`

$multicastTree = findMulticastTree(peers, peer_{CDN})$;

`/* Find leaf peers of CDN PoP and each peer in` $peers$`. */`

$CDNleafPeers = findleafPeers(multicastTree, peer_{CDN})$;

for $i \leftarrow 0$ **to** $len(peers)$ **do**

  $leafPeers_i = findleafPeers(multicastTree, peers_i)$;

`/* P2P startup phase starts. */`

$time = 0$;

while $atLeastOneLowBuffer()$ **do**

  $[receiving_i, sending_i, CDNsending] = startupScheduling()$;

  Send $peers$ and CDN PoP related $receiving_i, sending_i, CDNsending$ information;

  $time++$;

while *anyPacketinReceiving()* **do**

  $time++$;

$startupDelay = time * timeIntervalMS$;

`/* Peers start playing the video and P2P scheduling starts in intervals of` $schedulingInterval$`. */`

$playTime = 0$;

while $(!(allReceived()))\&\&(playTime\%schedulingInterval == 0)$ **do**

  $[receiving_i, sending_i, receivingCDN_i] = postPlayScheduling(playTime)$;

  Send $peers$ related $receiving_i, sending_i, receivingCDN_i$ information;

  $playTime++$;

Calculate total values of $bwConsumption, CDNload, reroutingCount, freezeCount$;

Return $startupDelay, bwConsumption, CDNload, reroutingCount, freezeCount$;

Peer-to-peer startup scheduling is explained in details in Algorithm 7. As stated earlier, this function is called in each time interval during the startup phase. As denoted in lines 1-10, CDN PoP pushes chunks while there is available bandwidth on its multicast path to its leaf peers for each time iteration. During the startup phase, each peer adds the chunk to its packets to send list as soon as it receives a chunk. For any time interval, if a peer has a packet to send, it pushes the chunk to its leaves, as shown in lines 11-21. Therefore, the chunks are distributed from the CDN PoP to all the peers through their parent peers. In both cases, the chunk is removed from the list of packets to send after it is pushed on the link.

Next, we included the general flow of P2P post-play scheduling in Algorithm 8. Firstly, the peers are grouped according to their buffer status and sorted accordingly. The scheduling starts from the low buffer status group. After all buffer status groups are examined, the algorithm sorts peers according to their proximity to CDN PoP. Starting from the closest peer to CDN PoP, if a peer has any chunk in its window that no peer has and the shortest path between that peer and CDN PoP has available bandwidth, CDN pull scheduling is done for that peer as it can be seen in lines 7-13.

Now, let us look into post-play P2P scheduling in details. The buffer and incentive based grouping and sorting of the peers can be seen in Algorithm 9. The buffer status of each peer for $playTime + schedulingInterval$ is checked firstly. According to predetermined threshold values, each peer is assigned a group, which can be low, normal, high and very high. Low buffer peers are also added to the normal buffer group in case there is more available bandwidth for them after the urgent chunk scheduling (lines 2-3). Then, between lines 4-8, the group sortings are made. For the case of low buffer group, the peers are sorted according to their buffer status in an increasing order, giving priority to peers with lower buffer status. For the rest of the groups, peers are sorted according to their incentive points in a decreasing order, giving priority to peers with higher incentive points.

Algorithm 7: Peer-to-Peer Startup Scheduling

Input : All values from Algorithm6

Output: $receiving_i, sending_i, CDNsending$

```
/* CDN PoP sends chunks to its leaves through multicast paths.  */
```

while $((len(peer_{CDN}.getPacketstosend())) != 0)$
    $\&\&(multicastPath(peer_{CDN}, CDNleafPeers).hasBW(reqBW))$ **do**
  foreach $peer \in CDNleafPeers$ **do**
    $packet = peer_{CDN}.getPacketstosend().get(0);$
    $sender = peer_{CDN};$
    $receiver = peer;$
    $path = multicastPath(peer_{CDN}, peer);$
    $receiving_{peer}.add([sender, receiver, packet, path]);$
    $CDNsending.add([sender, receiver, packet, path]);$
  $peer_{CDN}.getPacketstosend().remove(0);$

```
/* Each peer sends chunks(earlier received) to its leaves through
   multicast paths.                                              */
```

for $i \leftarrow 0$ **to** $len(peers)$ **do**
  while $(len(peers_i.getPacketstosend())) != 0$
      $\&\&(multicastPath(peers_i, leafPeers_i).hasBW(reqBW))$ **do**
    foreach $peer \in leafPeers_i$ **do**
      $packet = peers_i.getPacketstosend().get(0);$
      $sender = peers_i;$
      $receiver = peer;$
      $path = multicastPath(peers_i, peer);$
      $receiving_peer.add([sender, receiver, packet, path]);$
      $sending_i.add([sender, receiver, packet, path]);$
    $peers_i.getPacketstosend().remove(0);$

Return $receiving_i, sending_i, CDNsending;$

Algorithm 8: Peer-to-Peer Post-Play Scheduling - General Flow

Input : All values from Algorithm6

Output: $peerReceiving_i, peerSending_i, receivingCDN_i$

```
/* Group peers according to buffer and sort with buffer(bufferLow)
   or incentive(other cases).                                    */
```

1 $bufferGroups_i = bufferandincentivebasedGroupSort(peers, playTime)$

```
   /* Low buffer schedules                                       */
```

2 $lowBufferP2P(bufferGroups_0)$;

```
/* Normal buffer schedules                                       */
```

3 $incentiveAwareP2P(bufferGroups_1)$;

```
/* High buffer schedules                                         */
```

4 $incentiveAwareP2P(bufferGroups_2)$;

```
/* Very high buffer schedules                                    */
```

5 $incentiveAwareP2P(bufferGroups_3)$;

```
/* Pull from CDN PoP if no peer has a packet                     */
```

6 Sort $peers : spPaths[peer_{CDN}, peers_i] < spPaths[peer_{CDN}, peers_{i+1}]$;

7 **for** $i \leftarrow 0$ **to** $len(peers)$ **do**

8 $packets = nopeerhas(windowPackets(peers_i))$;

9 **while** $(peer_{CDN}.hasUplinkBW(reqBW))\&\&(peers_i.hasDownlnkBW(reqBW))$ **do**

10 **if** $len(packets) == 0$ **then**

11 break;

12 **if** $(spPaths[peer_{CDN}, peers_i]).hasBW(reqBW)$ **then**

13 $assignChunks(peer_{CDN}, peers_i, packets, -1)$;

14 Return $receiving_i, sending_i, receivingCDN_i$;

Algorithm 9: Buffer and Incentive Based Grouping and Sorting

Input : $peers, time$;

Output: $bufferGroups_n$;

`/* Group peers according to buffer. */`

1 $bufferGroups_n =$
$\{peers_i | checkBufferatTimeT(i, time + schedulingInterval) = n\}$;

2 **foreach** $peers_i \in bufferGroups_0$ **do**

3 $\quad bufferGroups_1.add(peers_i)$;

`/* Sort buffer low peers according to buffer. */`

4 Sort $BufferGroups_0$:
$BufferGroups_0[i].buffer < BufferGroups_0[i+1].buffer$;

`/* Sort others according to incentive. */`

5 Sort $BufferGroups_1$:
$BufferGroups_1[i].incentive > BufferGroups_1[i+1].incentive$;

6 Sort $BufferGroups_2$:
$BufferGroups_2[i].incentive > BufferGroups_2[i+1].incentive$;

7 Sort $BufferGroups_3$:
$BufferGroups_3[i].incentive > BufferGroups_3[i+1].incentive$;

8 return $bufferGroups_n$;

Next, we explain the low buffer P2P scheduling in details in Algorithm 10. For each peer i in low buffer status group, if its downlink bandwidth has available bandwidth, the *urgentChunk* is defined to be the chunk first in window. If this chunk is not being received by any peer that is directly connected to the same switch as peer i, we find the closest peer with *urgentChunk* and available uplink bandwidth. If there is no such peer, the algorithm checks CDN PoP between lines 6-8. If a peer is able to send the chunk, chunks are assigned from all the chunks in the window. However, if the sender is CDN PoP, just the urgent chunk is scheduled. The methodology of chunk

assigning can be seen in Algorithm 11. If there is an urgent packet, it is scheduled firstly. Otherwise, the chunk is chosen randomly from the window. The probability of the chunk first in window is the highest, while the last one has the lowest probability. Scheduling decisions are made for the interval of $schedulingInterval$ one by one. If neither a peer nor the CDN PoP can send the chunk, rerouting is made as indicated in line 16. The details of rerouting is given in Algorithm 12. Firstly, the links whose available bandwidth is less than $reqBW$ are excluded from the edge graph. Then, we check if any peer or CDN PoP can find a shortest path to $receiver$ from the simplified graph and if we can find a sender, chunks are scheduled accordingly. However, if there is no such sender again, it means that all the links connecting $receiver$ to the CDN PoP or the possible senders are full. Therefore, peer i will be tried in the next interval, which may cause buffering and freeze event for peer i.

Algorithm 10: Low Buffer P2P Scheduling

```
   Input : peers, G;
1  foreach peers_i ∈ bufferGroups_0 do
2      if peers_i.hasDownlinkBW(reqBW) then
3          urgentChunk = peers_i.buffer;
4          if !checkSameNodePeerReceiving(peers_i, urgentChunk) then
5              sender = closestPeerwPacketwUpBW(peers_i, urgentChunk, G);
6              if (sender == −1)&&(peer_CDN.hasUplinkBW(reqBW)) then
7                  if spPaths[peer_CDN, peers_i].hasBW(reqBW) then
8                      sender = peer_CDN;
9              if sender! = −1 then
10                 if sender! = peer_CDN then
11                     packets = packetsin(sender, windowPackets(peers_i));
12                 else
13                     packets.add(urgentChunk);
14                 assignChunks(sender, peers_i, packets, urgentChunk);
15             else
16                 rerouting(G, reqBW);
```

Rerouting means that (i) the WebRTC data channel between the sender and the receiver will be closed and a new data channel with another path will be created, (ii) SDN controller will change the rules in the switches in both paths, which we consider as signalling overhead, (iii) after receiving event, the same procedure will be repeated to go back to the shortest path case. Because of these drawbacks, rerouting is an undesirable event.

Algorithm 11: Chunk Assigning

Input : $sender, receiver, packets, urgentChunk$;

```
1  interval = 0;
2  path = spPaths[sender, receiver];
3  while len(packets)! = 0 do
       /* If there is an urgent packet, it is scheduled firstly.
          Otherwise, packet is chosen randomly from window packets  */
4      if (interval == 0)&&(urgentChunk! = −1) then
5          packet = urgentChunk;
6      else
7          packet = selectRandomly(packets);
8      if interval + len(path) < schedulingInterval then
9          sending_sender.add([sender, receiver, packet, path]);
10         if sender == peer_CDN then
11             receivingCDN_receiver.add([sender, receiver, packet, path]);
12         else
13             receiving_receiver.add([sender, receiver, packet, path]);
14         interval + +;
15         packets.remove(packet);
16     else
17         break;
```

Algorithm 12: Rerouting

Input : $G, reqBW$;

`/* Exclude links whose available bandwidth is less than` $reqBW$`. */`

Create graph G_{new} such that $\forall$ Link $\in G_{new} \geq reqBW$;
$sender = closestPeerwPacketwUpBW(peers_i, urgentChunk, G_{new})$;
if $(sender == -1)\&\&(peer_{CDN}.hasBW(reqBW))$ **then**
    if $bfs(peer_{CDN}, peers_i).hasBW(reqBW)$ **then**
        $sender = peer_{CDN}$;
if $sender! = -1$ **then**
    if $sender! = peer_{CDN}$ **then**
        $packets = packetsin(sender, windowPackets(peers_i))$;
    else
        $packets.add(urgentChunk)$;
    $assignChunks(sender, peers_i, packets, urgentChunk)$;
    $reroutingCount++$;
else
    No possible sender is found. This peer will be tried in the next interval.

Algorithm 13: Incentive Aware P2P Scheduling

Input : $peers$;

foreach $peers_i \in buffGroups_0$ **do**
    while $peers_i.hasDownlinkBW(reqBW)$ **do**
        $packets = windowPackets(peers_i)$;
        $sender =$
          $findOptimumSender(peers_i, packets, G, schedulingInterval)$;
        if $sender! = -1$ **then**
            $assignChunks(sender, peers_i, packets, -1)$;
        else
            break;

Algorithm 14: Calculation of the Optimum Sender Peer for Window Packets

Input : $receiver, packets, G, schedulingInterval$;

Output: $sender$;

/* This function calculates the closest peer (with upload capacity and available bandwidth on the path) that can send the maximum number of packets, which are included in $receiver$'s window, in $schedulingInterval$. */

```
1  totalPacket = 0;
2  sender = −1;
3  for i ← 0 to len(peers) do
4      if !sameNodePeers(peers_i, receiver) then
5          path = spPaths[peers_i, receiver];
6          if (len(path) <= schedulingInterval)&&(path.hasBW(reqBW))
7                  &&(peers_i.hasUplinkBW(reqBW)) then
8              maxPacket = floor(schedulingInterval/len(path));
9              packetList = packetsin(peers_i, packets);
               /* Choose peers_i if it is the peer that can send the
                  maximum number of packets.                          */
10             if (len(packetList) > totalPacket)&&(maxPacket > totalPacket)
                then
11                 if len(packetList) > maxPacket then
12                     totalPacket = maxPacket;
13                     sender = peers_i;
14                 else
15                     totalPacket = len(packetList);
16                     sender = peers_i;
               /* Choose peers_i if packet number is the same; however,
                  hop count is less.                                  */
17             else if (len(packetList)! = 0)&&(len(packetList) ==
                totalPacket)&&((maxPacket > totalPacket) then
18                 prevHop = len(spPath[sender, receiver]);
19                 if prevHop > len(path) then
20                     sender = peers_i;
21 Return sender;
```

We include the details of incentive aware P2P scheduling in Algorithm 13. This scheduling is performed for the cases of normal, high and very high buffer status groups. As the peers are sorted according to their incentive points in a decreasing order, the peers with high incentive points are prioritized. For each peer in group, while it has available downlink bandwidth, the optimum sender for window packets is found as described in Algorithm 14 and the scheduling is made accordingly.

The function supplied in Algorithm 14 calculates the closest peer with upload capacity and available bandwidth on the path that can send the maximum number of chunks, which are included in *receiver*'s window, in *schedulingInterval*. For each peer i that is not directly connected to the switch *receiver* is directly connected to and has available uplink bandwidth, shortest path is checked if its size is less than *schedulingInterval* and if it has available bandwidth (lines 4-7). If this condition holds, the maximum number of chunks peer i can send in *schedulingInterval* is found in line 8 and the chunks in peer i from window of *receiver* are found in line 9. Between lines 10-16, the algorithm checks if peer i is the peer that can send the maximum number of chunks in *schedulingInterval* from *packetList*, it is assigned as the sender peer. In addition, if the maximum number of packets from *packetList* that peer i can send is the same, yet the total hop count to the *receiver* is less than the previously assigned sender peer, peer i is chosen as the sender peer. Lastly, the sender information is returned.

3.2.1 Complexity Analysis of Centralized Chunk Scheduling

The complexity of forming the multicast tree is found as following. As the shortest path values between all nodes have already been found in the the neighborhood formation phase, Dijkstra is not solved at each iteration of the multicast tree formation. Therefore, the complexity is found as $\mathcal{O}(|V||E|)$, where $|V|$ denotes the vertices and $|E|$ denotes the edges in the graph.

The complexity of the startup phase of the P2P scheduling depends on the time needed for all the peers to have a predetermined number of chunks in their buffer

denoted as t_0, and the number of peers in the service denoted as p. The complexity is found as $\mathcal{O}(t_0 p \log p)$.

The complexity of the post-play phase of the P2P scheduling depends on the time needed for all the peers to receive all the chunks denoted as t_1, the number of peers in the service denoted as p. Grouping and sorting the peers is of complexity $\mathcal{O}(p + (4p \log p))$ as p number of peers are being grouped and then four sorting operations are being made. Each buffer group scheduling operation introduces a complexity of $\mathcal{O}(p^2)$ as the algorithm tries to find a sender peer for each receiving peer. Sorting peers according to their proximity to CDN PoP is of complexity $\mathcal{O}(p \log p)$. Lastly, CDN pull scheduling operations introduce $\mathcal{O}(p^2)$ because the rest of the peers are checked if they do not have the window packets of receiving peer. Totally, the complexity of post-play P2P scheduling has a time complexity of $\mathcal{O}(t_1((4p \log p)+(4p^2)+(p \log p)+(p^2)))$, which can be approximated as $\mathcal{O}(t_1 p^2)$.

Totally, the complexity of P2P centralized chunk scheduling can be formulated as $\mathcal{O}((|V||E|) + (t_0 p \log p) + (t_1 p^2))$. Knowing that t_0 is smaller than t_1, we can approximate the total complexity as $\mathcal{O}((|V||E|) + (t_1 p^2))$.

3.3 Traffic Engineering

There are three different jobs of TEM module. First, SDN controller instructs TEM module to find the shortest paths between each peer and also the shortest paths between CDN PoP and the peers at the beginning of the live video service and these path information are used from the memory in each iteration of the proposed methods to decrease the time complexity of the procedures. Second, SDN controller instructs TEM module to find the multicast paths for each neighborhood group after the neighborhoods are formed. After TEM constructs the multicast tree and finds the leaf peers of each peer and CDN PoP, it notifies the SDN controller, which makes the related resource reservation. The startup phase of the centralized chunk scheduling continues on these multicast paths determined. The third task of TEM module is in an event of rerouting. When rerouting is needed, P2P APP notifies the SDN controller, which

in return notifies the TEM module to solve a constrained path problem to obtain a sender for the peer in buffer low status.

Resource reservation is not done per peer as it is not efficient to set queues between all peers in a group. There is a group reservation for group of clients to CDN PoP, which guarantees QoE requests between CDN and peers.

3.4 Chunk Transmission

There are two different ways a chunk can be transmitted in the proposed controllable-P2P-assisted CDN video service: i) pull method from a CDN node with DASH, ii) push method between the peers through WebRTC data channel. According to the scheduler decision, each peer is instructed to upload scheduled chunks to the destination peers and to download the scheduled chunks with the determined download method. In addition, each peer is instructed about the peers that are directly connected to the same switch as itself earlier and for each received chunk, each peer sends it to the peers in its list of same node peers.

Algorithm 15: WebRTC data channel chunk transmission: Sender side

Input: $sending, dataChannel$

```
1  maxUnit = 15000;
2  foreach sending message do
3      for i ← 0 to len(sending) do
4          channel = dataChannel[this.id, sending_i.receiver];
5          chunk = sending_i.packet;
6          channel.send('chunk.ID chunk.size');
7          count = 0;
8          while count < chunk.size do
9              end = count + maxUnit − 1;
10             channel.send(chunk[count : end];
11             count = count + maxUnit;
```

Algorithm 16: WebRTC data channel chunk transmission: Receiver side

Input : $receiving, receivingCDN, dataChannel$

Output: $receivedChunks$

```
1  chunkName = 'none';
2  chunkSize = 0;
3  foreach receiving message do
4      for i ← 0 to len(receiving) do
5          channel = dataChannel[receiving_i.sender, this.id];
6          if channel.onmessage then
7              if message.type == STRING then
8                  chunkID = message[0];
9                  chunkSize = message[1];
10             else
11                 received = received + message.size;
12                 if received == chunkSize then
13                     receivedChunks[chunkID] = message;
14                     chunkName = 'none';
15                     chunkSize = 0;
16                     received = 0;
17     send(sameNodePeers_this.id, message);
18 foreach receivingCDN message do
19     for i ← 0 to len(receivingCDN) do
20         receivedChunks[receivingCDN_i.packet] =
           dashRequest(receivingCDN_i.packet);
21     send(sameNodePeers_this.id, receivedChunks[receivingCDN_i.packet]);
22 return receivedChunks;
```

The chunk transmissions between the peers are implemented through the WebRTC data channel. The sender side of WebRTC data channel chunk transmission is explained in Algorithm 15. Each peer with a *sending* message behaves as the sender side for chunk transmission. After the chunk and receiver information is gathered from the $sending_i$ message, the chunk ID and chunk size is sent through the data channel. Then, the chunk is put to the path in MTU units.

The receiver side of the WebRTC data channel chunk transmission is depicted in Algorithm 16. For each $receiving_i$ message, the receiver side starts listening to the related data channel. After obtaining the chunk ID and the chunk size, it receives the chunk from the data channel as explained in lines 3-16. For each $receivingCDN$ message, dash request is made to the CDN PoP and the chunk is sent by the CDN PoP (lines 18-21).

3.5 Accounting

The proposed architecture includes accounting procedure in order to provide a fair charging and incentive policy for P2P video streaming services.

The detailed information flow consists of the following steps:

1. P2P APP gives scheduling decisions.

2. For each scheduling decision, P2P APP passes the downloading and uploading peer information with the amounts to the ISP accountant.

3. After the session ends, each ISP accountant calculates the incentive points of each peer and passes the total downloaded chunk size information of each peer to the content accountant through the SDN controller.

4. CDN APP gets the information from content accountant and charges all the peers that joined the P2P video service.

As indicated in the information flow, ISP accountant has the information of the amount of uploads and downloads of each client in its edge access network. While

making scheduling decisions, P2P APP requests these information from the ISP accountant and uses them to prioritize the high contributor peers, thus providing an incentive policy.

Chapter 4

EVALUATION

4.1 Experimental Setup

We first implemented the proposed SDN-based controllable-P2P-assisted CDN service in an emulation environment as a proof of concept. Then, we conducted large-scale experiments in a simulation environment.

4.1.1 Emulation Environment

The emulation of the proposed architecture is done over an SDN testbed to prove that the P2P chunks are transmitted through the WebRTC data channel according to scheduling decisions. The topology of the edge access domain is realized as a partial mesh topology using Mininet [Mininet, 2017] and OVS switches, which are connected to the Floodlight SDN controller [Floodlight, 2017]. Mininet configuration is done such that the Mininet hosts have access to the Internet through NAT as this access is necessary for the configuration of WebRTC data channels. Also, one of the nodes is denoted as CDN PoP in the edge access network. We implemented ISP accountant, TEM, P2P APP, CDN APP and content accountant as web services running on separate servers. These modules communicate with data-plane entities such as the peers through JSON based APIs. The WebRTC signaling server is implemented with node modules and we distribute HTML, CSS and JS codes to the browsers with node-static command [Node, 2017]. WebSockets are used for the communication between the browsers of the peers and the signaling server [Websocket, 2017].

4.1.2 Simulation Environment

Performance of the proposed architecture is evaluated over a testbed featuring large-scale SDN-based edge access network. As it is not possible to emulate hundreds of hosts in Mininet [Mininet, 2017], we simulated this edge access network. In order to obtain valid and applicable results, we used Waxman edge probability equation (Equation 1.1) explained in Section 1.2.10 to generate the edge access network topology graph. The peers are assigned to randomly chosen switches that are in the edge access network. Some peers end up being connected to the same switches, which is suitable to the real network conditions. The location of the CDN PoP is chosen according to Algorithm 2. In addition, all the entities are implemented as described in Section 4.1.1.

4.2 Experimental Scenarios

Key performance indicators (KPI) of the proposed managed hybrid CDN-P2P video streaming services are as follows:

i) Network KPI

- rate of server load reduction
- rate of cross-ISP traffic reduction
- rate of overall bandwidth consumption reduction in the edge network
- number of rerouting events

ii) Video Quality KPI

- number of freeze events
- average and maximum startup delay, also known as time to first bite (TTFB) in CDN delivery services

4.2.1 Comparison of Client-Server vs. Managed P2P

In traditional DASH video services, peers pull all content directly from CDN nodes, which results in higher overall bandwidth consumption, server load and possibly cross-ISP traffic. In the proposed architecture, P2P service will reduce the load on CDN

nodes. In addition, as the peers will be streaming from other peers which are in close proximity to themselves, the bandwidth consumption will also decrease. Furthermore, in a typical client-server video service, many of the peers may be outside the ISP of the distributed CDN node, which will result in higher cross-ISP traffic when compared to our proposed architecture, which only introduces a baseline of cross-ISP traffic.

4.2.2 Comparison of Multicast vs. Managed P2P

Multicast video services decrease the bandwidth consumption substantially. Therefore we do not expect to see bandwidth consumption reduction when compared with our proposed architecture. However, we expect to see cross-ISP traffic and server load reduction in the proposed architecture. CDN node may have multiple leaves, which will increase server load, according to the constructed multicast tree. Therefore, we expect to see higher CDN utilization for multicast operations when compared with the proposed architecture.

4.2.3 Comparison of Unmanaged vs. Managed P2P

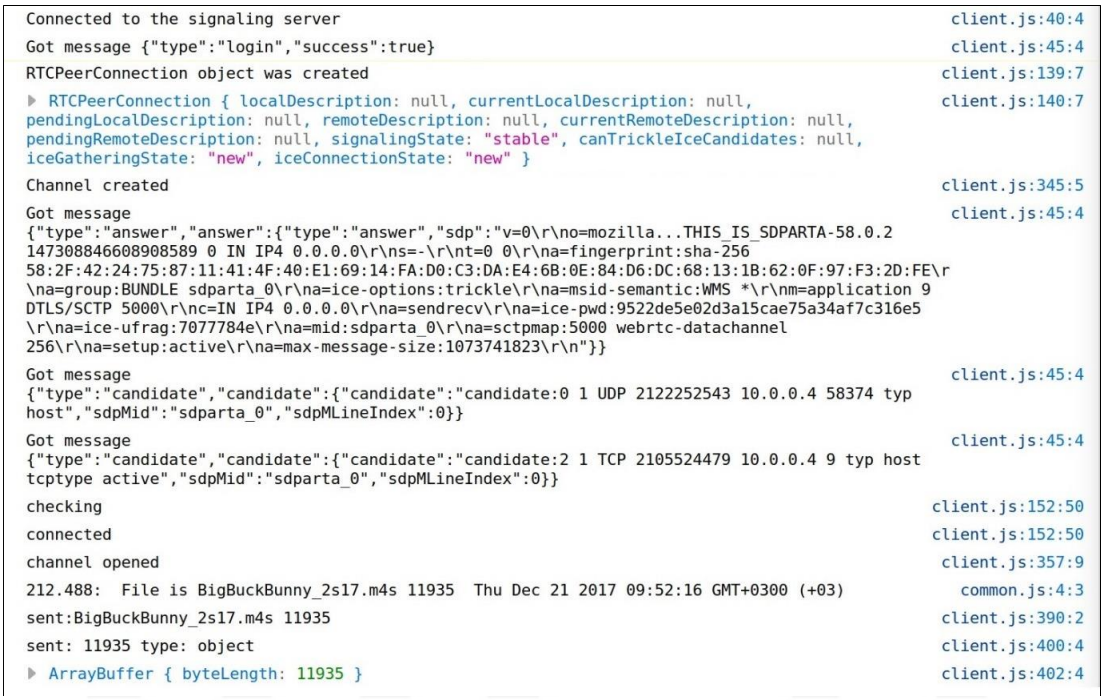
The proposed managed P2P video service architecture avoids illegal and unauthorized usage that is seen in unmanaged P2P services by the help of accountant modules. In addition, the proposed controllable-P2P-assisted CDN service will only create a baseline of cross-ISP traffic, while decreasing it when compared to the unmanaged P2P video services.

4.3 Results

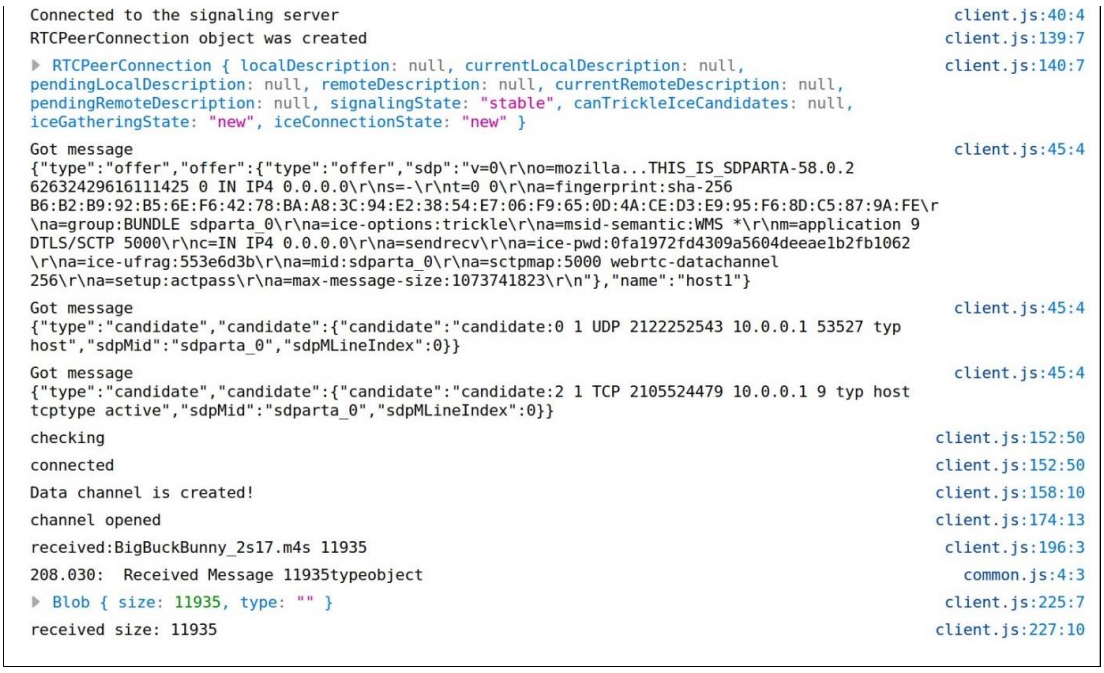
The results of this thesis is given in this section.

4.3.1 Emulation Results

The results of WebRTC transmission, whose algorithm is given in Section 3.4, is given in this section. According to the centralized chunk scheduling decisions, the chunks are transmitted via WebRTC data channel. Figures 4.1(a) and 4.1(b) depict



(a)



(b)

Figure 4.1: Emulation results: (a) console output for sender, and (b) console output for receiver.

Table 4.1: α and β values according to the number of nodes.

Total Number of Nodes	100	150	200	250	300	350	400	450	500
α	0.05	0.04	0.035	0.03	0.025	0.02	0.0175	0.0165	0.015
β	5	5	5	5	8	10	10	10	10

the console outputs for sender and receiver, respectively. From this figure, we can conclude that the chunks are being transmitted by the WebRTC data channel.

Firstly, both the sender and the receiver connect to the signalling server. The data channel between the sender and the receiver is opened after all the messaging necessary is completed. Then, the sender puts the chunk on the WebRTC data channel and the receiver end outputs that the chunk arrived.

4.3.2 Topology Results

We included the results regarding the edge access network topology creation in this section. As indicated Waxman edge probability equation (Equation 1.1) explained in Section 1.2.10 is utilized.

The parameters α and β change the density of short and long edges. Figure 4.2 show different topology graphs for 100 nodes for different α and β values. One can observe that the best graph is generated in Figure 4.2(e). Figures 4.2(a) and 4.2(c) does not have enough edges, while the edge density is too heavy in Figures 4.2(b) and 4.2(d). Conducting this experiment for several cases with different number of nodes, we obtained α and β values according to the number of nodes ranging from 100 to 500. Table 4.1 demonstrates these results.

Figure 4.3 depicts generated topology graphs for 100,300 and 500 nodes with the appropriate α and β values.

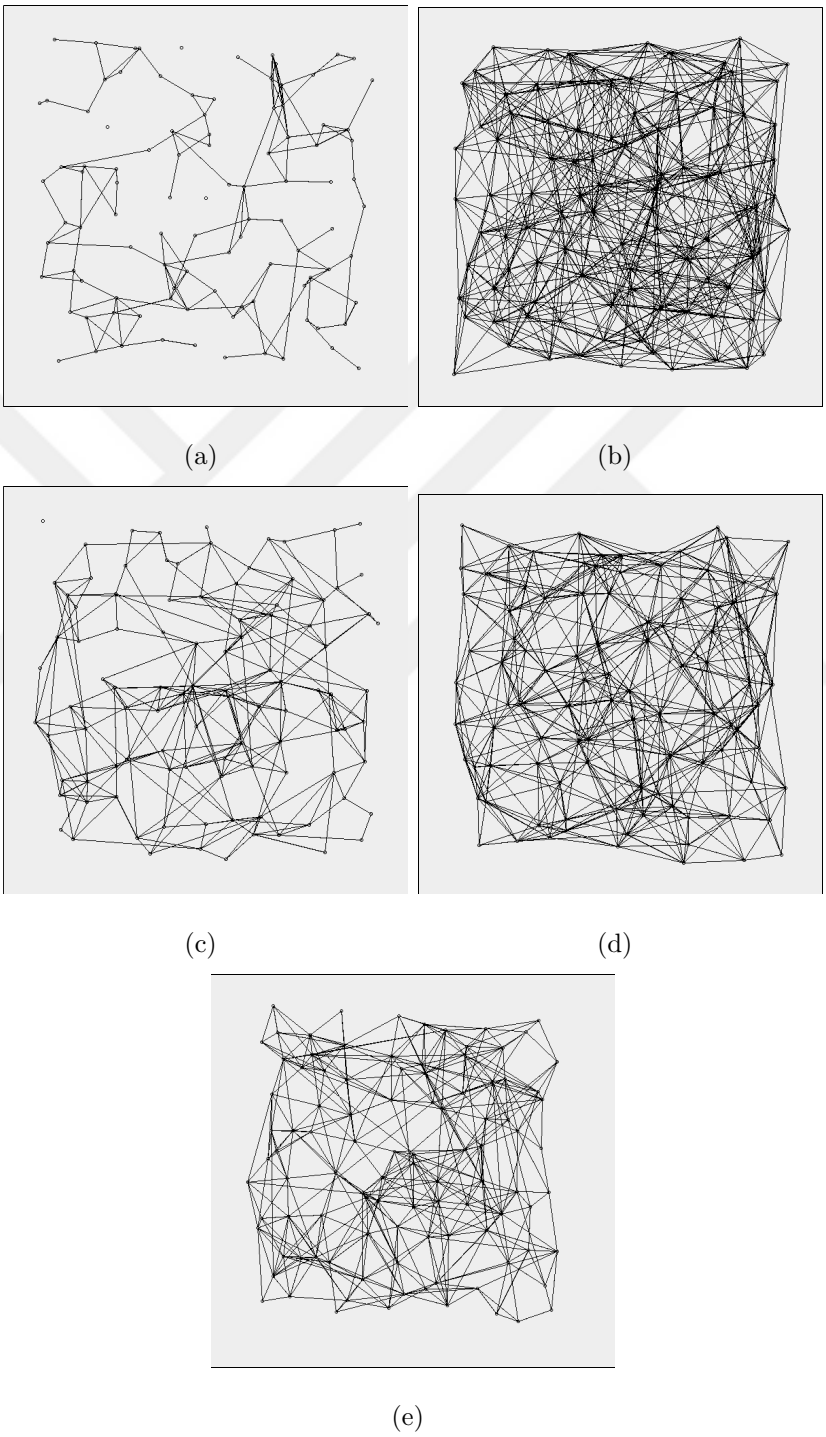


Figure 4.2: Generated topology graphs for 100 nodes: (a) $\alpha = 0.03$ and $\beta = 5$, (b) $\alpha = 0.07$ and $\beta = 5$, (c) $\alpha = 0.05$ and $\beta = 3$, (d) $\alpha = 0.05$ and $\beta = 10$, and (e) $\alpha = 0.05$ and $\beta = 5$.

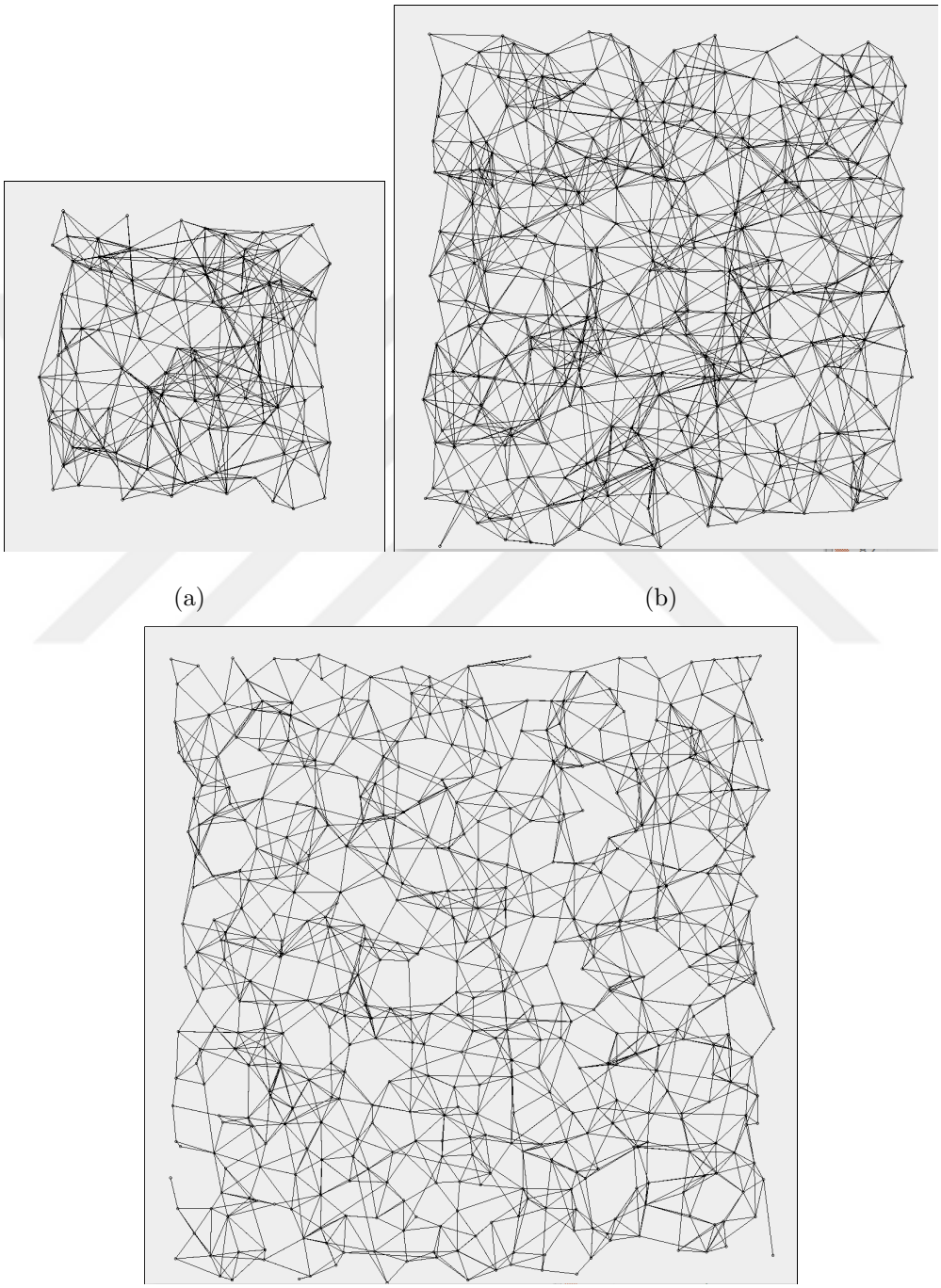


Figure 4.3: Generated topology graphs with appropriate α and β values: (a) 100 nodes with $\alpha = 0.05$ and $\beta = 5$, (b) 300 nodes with $\alpha = 0.025$ and $\beta = 8$, and (c) 500 nodes with $\alpha = 0.015$ and $\beta = 10$.

4.3.3 Group Formation Results

Neighborhood formation results are taken with both of the methods introduced in Section 3.1, in Algorithms 4 and 5 for edge access networks of 5 different sizes for 3 different k values, which are as following:

i) Topology 1: 100 switch - 100 peer, $k = 2, 3, 4$,

ii) Topology 2: 200 switch - 200 peer, $k = 2, 3, 4$,

iii) Topology 3: 300 switch - 300 peer, $k = 3, 4, 5$,

iv) Topology 4: 400 switch - 400 peer, $k = 4, 5, 6$,

v) Topology 5: 500 switch - 500 peer, $k = 5, 6, 7$.

Results for topology 1,2,3,4 and 5 can be seen in Figures 4.4, 4.5, 4.6, 4.7, and 4.8, respectively. CDN PoP is indicated by a big black dot and as it can be seen it is in close proximity to the center and has multiple direct links. Each neighborhood group is indicated by a different color and the switches with a direct peer connection are shown with colored little dots.

The graphs on the left are taken with the K-means with hop count algorithm explained in Section 3.1, in Algorithm 4, and the graphs on the right are taken with CDN PoP location aware K-means with hop count algorithm explained in Section 3.1, in Algorithm 5. The results for 100 switch and 100 peer is the worst as the total number of peers receiving the service is already small and also most of the switches are in close proximity to each other. For the bigger topology sizes, K-means with hop count algorithm creates nicely separated groups; however, some of the groups are very far from CDN PoP which will introduce an increase in both the delay and the bandwidth consumption. As delay increases more peers will have low buffer status, which will in return increase the CDN load, too. When we observe the results of CDN PoP location aware K-means with hop count, we can conclude that each group is getting closer to CDN PoP diminishing the number of outages which will increase the performance of the proposed architecture.

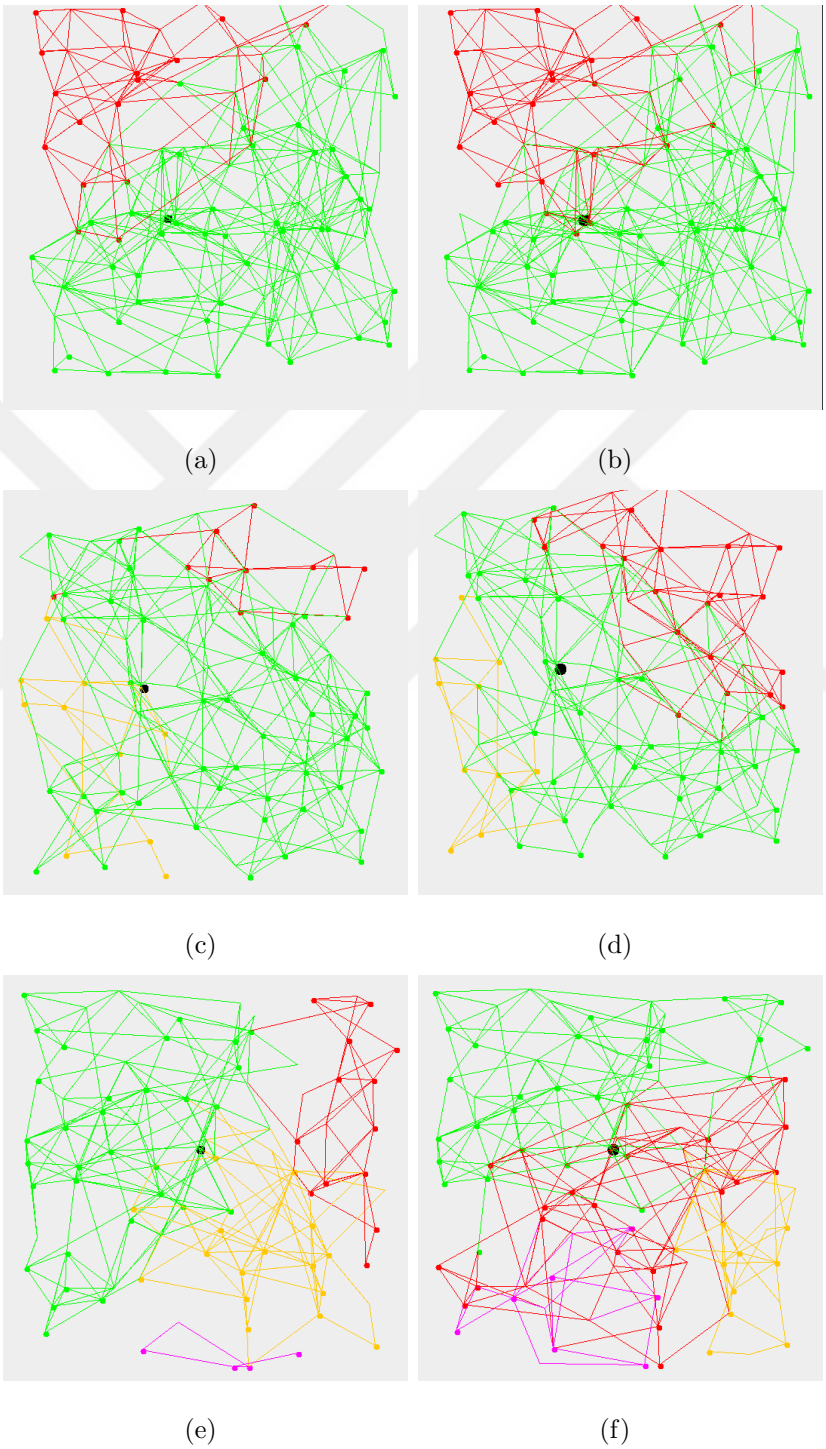


Figure 4.4: Neighborhood group formation results for 100 switches and 100 peers: (a) $k = 2, Algorithm\ 4$, (b) $k = 2, Algorithm\ 5$, (c) $k = 3, Algorithm\ 4$, (d) $k = 3, Algorithm\ 5$, (e) $k = 4, Algorithm\ 4$, and (f) $k = 4, Algorithm\ 5$.

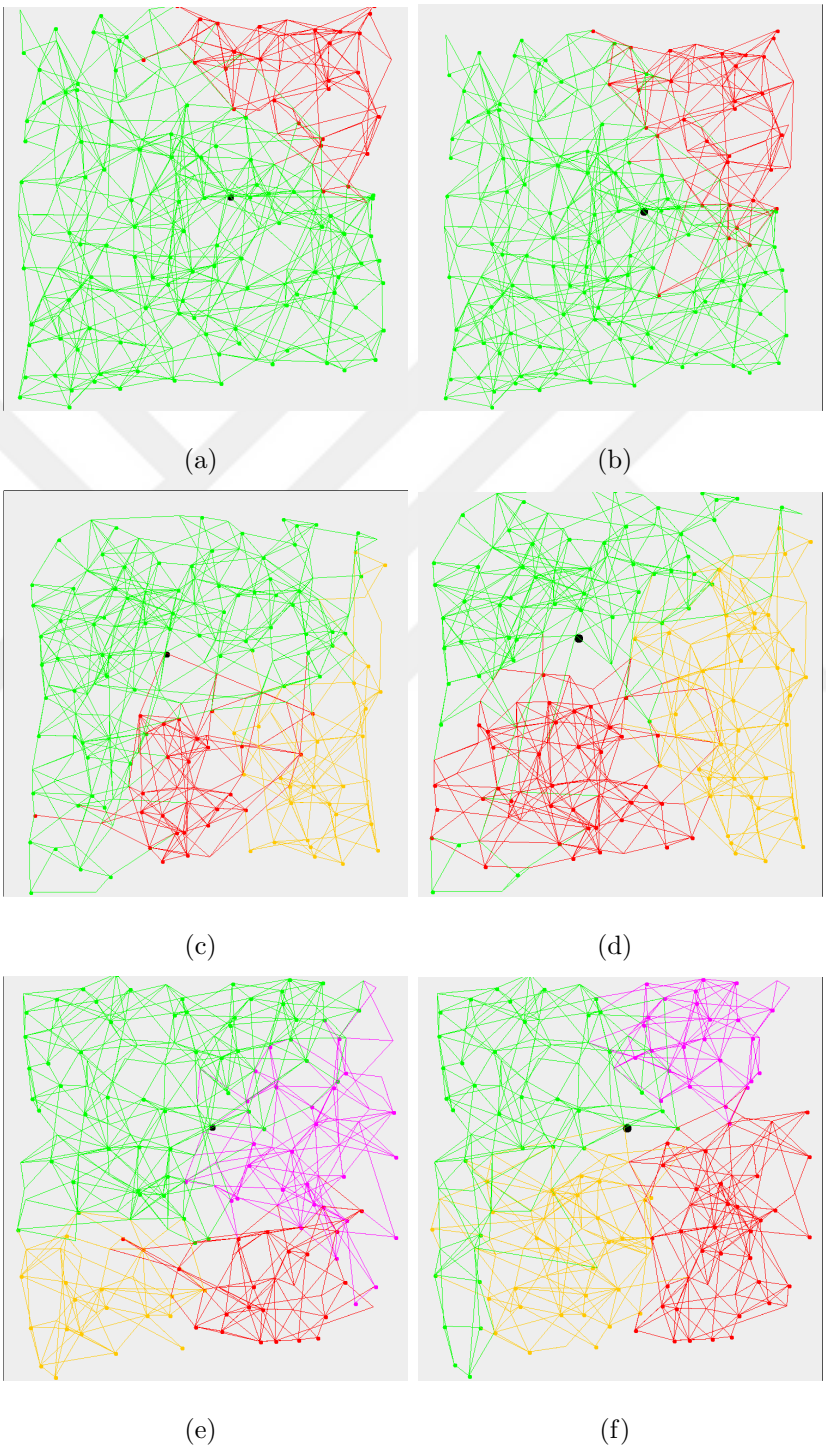


Figure 4.5: Neighborhood group formation results for 200 switches and 200 peers: (a) $k = 2, Algorithm\ 4$, (b) $k = 2, Algorithm\ 5$, (c) $k = 3, Algorithm\ 4$, (d) $k = 3, Algorithm\ 5$, (e) $k = 4, Algorithm\ 4$, and (f) $k = 4, Algorithm\ 5$.

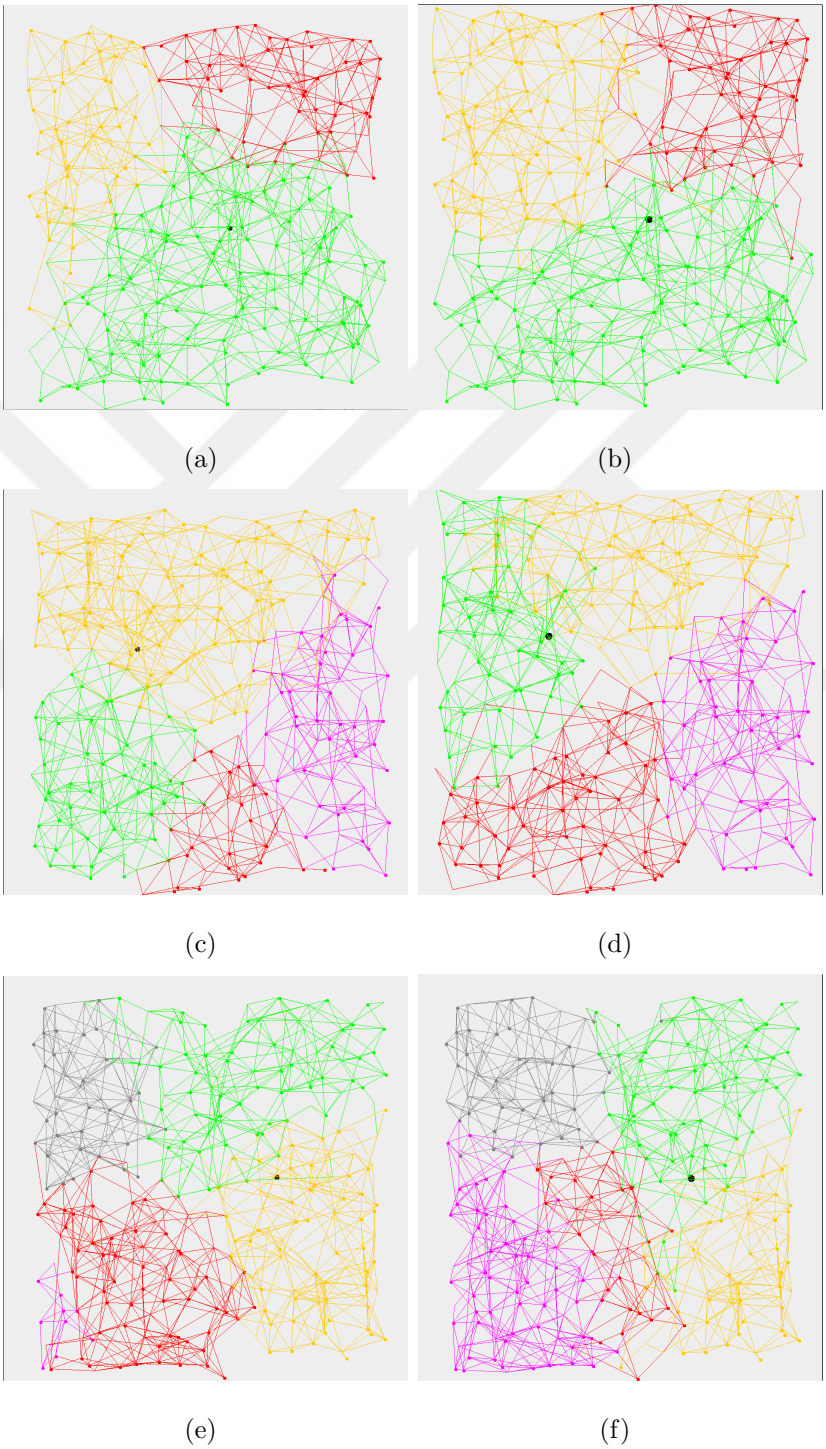


Figure 4.6: Neighborhood group formation results for 300 switches and 300 peers: (a) $k = 3, Algorithm\ 4$, (b) $k = 3, Algorithm\ 5$, (c) $k = 4, Algorithm\ 4$, (d) $k = 4, Algorithm\ 5$, (e) $k = 5, Algorithm\ 4$, and (f) $k = 5, Algorithm\ 5$.

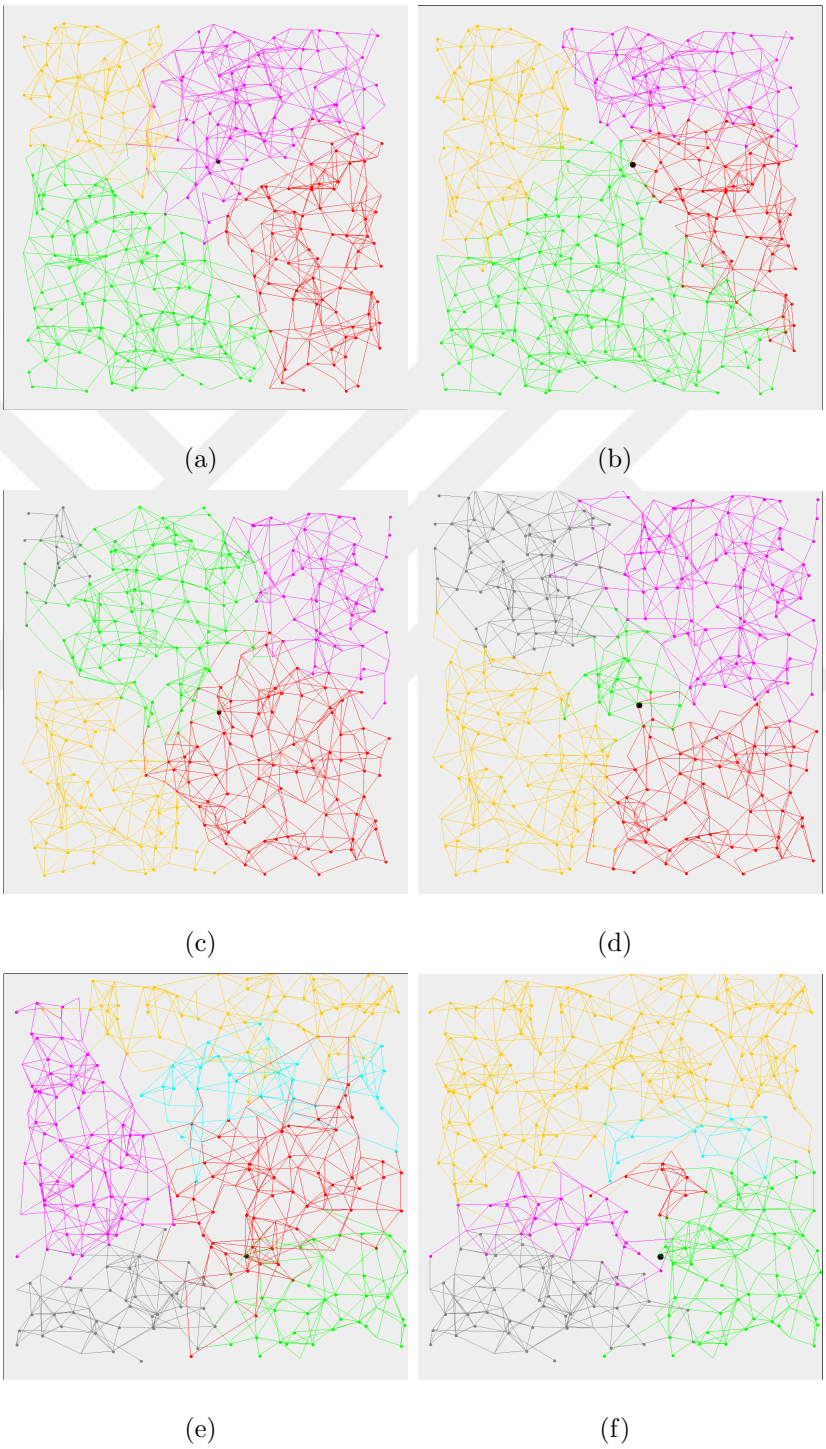


Figure 4.7: Neighborhood group formation results for 400 switches and 400 peers: (a) $k = 4, Algorithm\ 4$, (b) $k = 4, Algorithm\ 5$, (c) $k = 5, Algorithm\ 4$, (d) $k = 5, Algorithm\ 5$, (e) $k = 6, Algorithm\ 4$, and (f) $k = 6, Algorithm\ 5$.

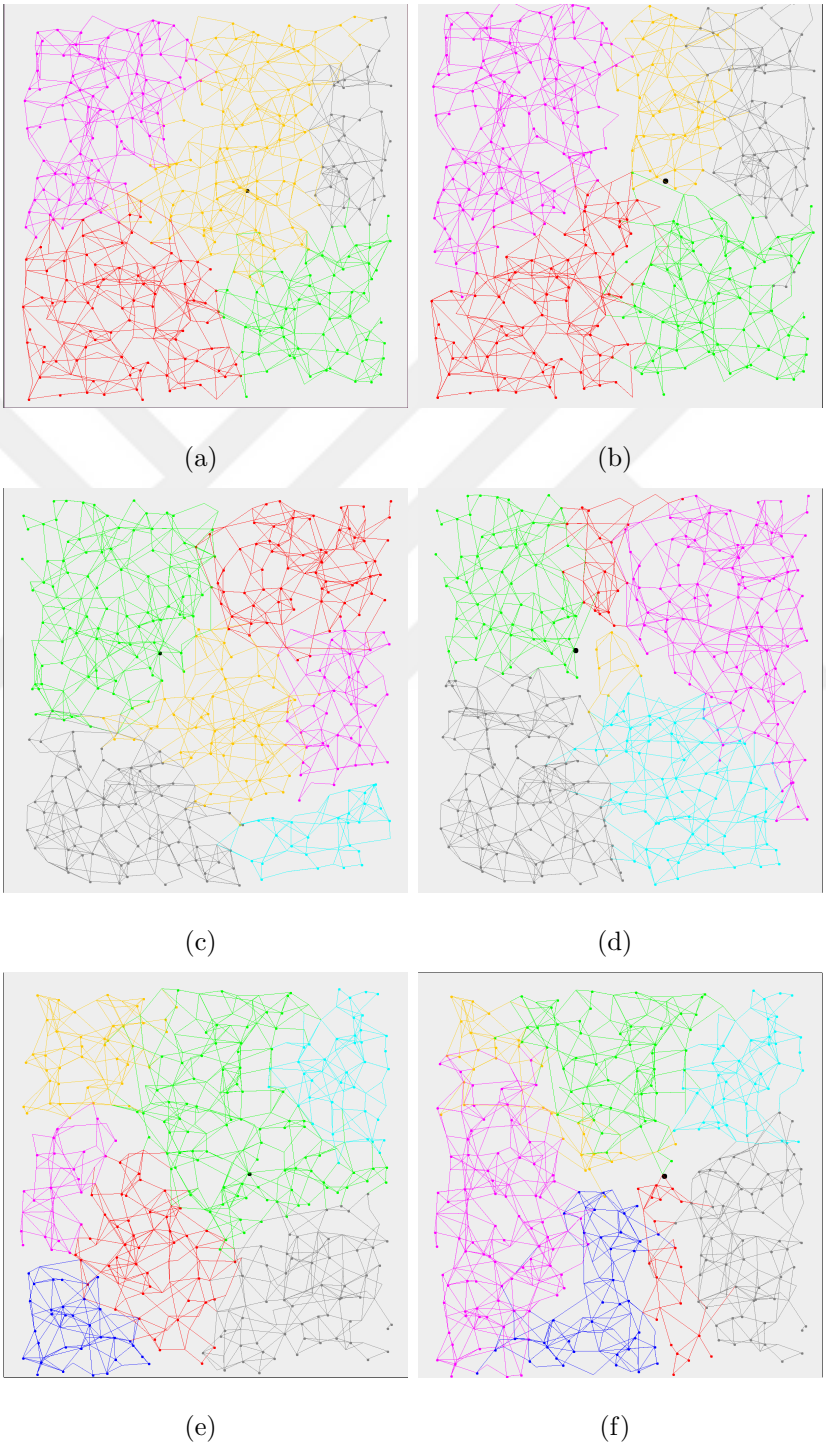


Figure 4.8: Neighborhood group formation results for 500 switches and 500 peers: (a) $k = 5, Algorithm\ 4$, (b) $k = 5, Algorithm\ 5$, (c) $k = 6, Algorithm\ 4$, (d) $k = 6, Algorithm\ 5$, (e) $k = 7, Algorithm\ 4$, and (f) $k = 7, Algorithm\ 5$.

4.3.4 Video Streaming Results

Video streaming results are taken for the experimental scenarios introduced in Section 4.2 for edge access networks of 5 different sizes, which are as following:

i) 100 switch - 100 peer, $k = 2, 3, 4$, $max\ link\ BW(Mbits/s) = 3, 6, 15, 30, 60, 120$,

ii) 200 switch - 200 peer, $k = 2, 3, 4$, $max\ link\ BW(Mbits/s) = 3, 6, 15, 30, 60, 120$,

iii) 300 switch - 300 peer, $k = 3, 4, 5$, $max\ link\ BW(Mbits/s) = 3, 6, 15, 30, 60, 120$,

iv) 400 switch - 400 peer, $k = 4, 5, 6$, $max\ link\ BW(Mbits/s) = 3, 6, 15, 30, 60, 120$,

v) 500 switch - 500 peer, $k = 5, 6, 7$, $max\ link\ BW(Mbits/s) = 3, 6, 15, 30, 60, 120$,

Each experimental scenario is tested with both of the neighborhood formation algorithms. The animation video 'Big Buck Bunny' found in [ITEC, 2020] is used in the video streaming tests and for the HD tests we used chunk duration of 6 seconds with (i) 2,5 Mbit/s, 1920×1080, average chunk size of 1.5Mbit, and (ii) 8,0 Mbit/s, 1920×1080, average chunk size of 2.5Mbit. The former will be denoted as HD1 and the latter will be denoted as HD2 from here on. Each test run is repeated 20 times and then the mean and the standard variation values are found.

Rerouting Event Count Results

Rerouting introduces signalling overhead for the SDN controller because the rules in the switches are updated with each rerouting event. As the signalling overhead increases, the deployment of the proposed architecture cannot be realised. Therefore, we checked the rerouting counts for each of our tests. Total of 3600 test runs are done and the results including both the mean and the standard deviation for HD1 and HD2 are given in Tables 4.2 and 4.3, respectively.

The rerouting count values are mostly zero as it can be seen from the given tables. For both HD1 and HD2, we observe a few tolerable rerouting count results which are in the range of 0 to 5 with small standard variation values given in green colored cells. These are denoted as tolerable because such small values will not create a big signalling overhead for the SDN controller. However, we observed a high rerouting

Table 4.2: Rerouting results for HD1.

Topology	k	Grouping Algorithm	Maximum Bandwidth per Link											
			3Mbit/s		6Mbit/s		15Mbit/s		30Mbit/s		60Mbit/s		120Mbit/s	
			Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
(1) 100 switch 100 peer	2	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	3	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	4	4	0.2	0.6	0.2	0.6	0	0	0	0	0	0	0	0
		5	0.2	0.6	0	0	0	0	0	0	0	0	0	0
(2) 200 switch 200 peer	2	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	3	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	4	4	0	0	0	0	0	0	0.2	0.6	0.2	0.6	0	0
		5	0.2	0.6	0	0	0	0	0	0	0	0	0	0
(3) 300 switch 300 peer	3	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	4	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0.2	0.6	0	0	0	0
	5	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0.2	0.6	0	0	0.2	0.6	0	0	0	0	0	0
(4) 400 switch 400 peer	4	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	5	4	0	0	0	0	0	0	0.15	0.53	0	0	1.45	4.6
		5	0	0	0	0	0.27	0.68	0.15	0.53	0.17	0.55	0	0
	6	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0.2	0.6	0	0	0	0
(5) 500 switch 500 peer	5	4	0	0	746.7	2793.8	0.18	0.57	0.2	0.6	0	0	0	0
		5	0.15	0.53	0	0	0.18	0.57	0	0	0	0	0	0
	6	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	7	4	0	0	0	0	0.55	1.72	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.3: Rerouting results for HD2.

Topology	k	Grouping Algorithm	Maximum Bandwidth per Link											
			3Mbit/s		6Mbit/s		15Mbit/s		30Mbit/s		60Mbit/s		120Mbit/s	
			Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
(1) 100 switch 100 peer	2	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	3	4	1.6	3.56	0	0	0	0	0	0	0	0	0	0
		5	1.6	4.8	0	0	0	0	0	0	0	0	0	0
	4	4	4	5.87	0	0	0	0	0	0	0	0	0	0
		5	1.6	1.96	0.6	1.8	0	0	0	0	0	0	0	0
(2) 200 switch 200 peer	2	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	3	4	2.6	7.16	0	0	0	0	0	0	0	0	0	0
		5	39.2	117.6	0	0	0	0	0	0	0	0	0	0
	4	4	1	3	0	0	0	0	0	0	0	0	0	0
		5	2.6	6.07	0	0	0	0	0	0	0	0	0	0
(3) 300 switch 300 peer	3	4	0	0	0.4	1.2	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	4	4	0	0	0	0	0.6	1.8	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	5	4	0.4	1.2	0.4	1.2	0.4	1.2	0	0	0	0	0.2	0.6
		5	0.2	0.6	0	0	0	0	0	0	0	0	0	0
(4) 400 switch 400 peer	4	4	0	0	0	0	0	0	0.2	0.6	0	0	0	0
		5	0	0	0.2	0.6	0	0	0	0	0	0	0	0
	5	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0.2	0.6
	6	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0.4	1.2	0	0	0	0	0	0	0	0
(5) 500 switch 500 peer	5	4	1	3	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0.8	2.4	0	0	0	0
	6	4	0.6	1.28	0	0	0	0	0	0	0.2	0.6	0	0
		5	0.8	2.4	0	0	0	0	0	0	0.2	0.6	0.2	0.6
	7	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	1	3	0	0	0.2	0.6	0	0	0	0	0.2	0.6

value for topology 5, k=5, Algorithm 4, and 6Mbit/s maximum link bandwidth for HD1 tests. Also for HD2, we observed a relatively smaller yet high value of rerouting count for topology 2, k=3, Algorithm 5 and 3 Mbit/s of maximum link bandwidth test run. We can say that there is signalling overhead for these two cases denoted by the purple colored cells. The reason behind these two cases can be the insufficient neighborhood formation method for the former and the insufficient link bandwidth for the latter. For most of the cases, we observe zero rerouting count which means that the centralized chunk scheduling algorithm makes the schedulings so that low buffer status decreases amongst the peers and available paths are found even when they are suffering from low buffer status. In addition, we can state that 3Mbit/s is a very low bandwidth for HD2 case as we mostly observe nonzero rerouting counts. Furthermore, the results show that Algorithm 5 performs better than Algorithm 4 as we see more zeros for the case of Algorithm 5.

Looking at the results, we can conclude that the proposed architecture does not cause heavy signalling traffic for the SDN controller and the edge network.

Freeze Event Count Results

Freeze event means that the peers receive chunks with delay. This is an undesirable event for P2P video services. Therefore, it is checked at each run and the final results for HD1 and HD2 including both the mean and the standard deviation can be seen in Tables 4.4 nad 4.5, respectively. The purple colored cells denote the cases with nonzero freeze event counts.

As it can be seen the mean of the freeze event count is mostly zero for HD1. The only nonzero values obtained from the HD1 test are the ones with Algorithm 4, that is why we can state that the neighborhood formation method was not good for those cases. However for the tests with HD2, we observe more freeze events compared to HD1. The results for maximum link bandwidth of 3Mbit/s are mostly nonzero and in an unacceptable range. However, this is an expected result as 3Mbit/s is a very low bandwidth for HD2 video. For the rest of the maximum link bandwidth values, we

Table 4.4: Freeze event count results for HD1 (in seconds).

Topology	k	Grouping Algorithm	Maximum Bandwidth per Link											
			3Mbit/s		6Mbit/s		15Mbit/s		30Mbit/s		60Mbit/s		120Mbit/s	
			Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
(1) 100 switch 100 peer	2	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	3	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	4	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
(2) 200 switch 200 peer	2	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	3	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	4	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
(3) 300 switch 300 peer	3	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	4	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	5	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
(4) 400 switch 400 peer	4	4	0	0	4	12	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	5	4	0	0	7.1	22.4	0	0	0	0	0	0	3.67	11.6
		5	0	0	0	0	0	0	0	0	0	0	0	0
	6	4	0	0	0	0	0	0	0.4	1.1	0	0	0.4	1.3
		5	0	0	0	0	0	0	0	0	0	0	0	0
(5) 500 switch 500 peer	5	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	6	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	7	4	0	0	0	0	1.9	5.98	0	0	0	0	0.5	1.7
		5	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.5: Freeze event count results for HD2 (in seconds).

Topology	k	Grouping Algorithm	Maximum Bandwidth per Link											
			3Mbit/s		6Mbit/s		15Mbit/s		30Mbit/s		60Mbit/s		120Mbit/s	
			Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
(1) 100 switch 100 peer	2	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	3	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	4	4	2.2	6.6	0	0	0	0	0	0	0	0	0	0
		5	1.5	2.92	0	0	0	0	0	0	0	0	0	0
(2) 200 switch 200 peer	2	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	3	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	2220	6660	0	0	0	0	0	0	0	0	0	0
	4	4	7.3	22	0	0	0	0	0	0	0	0	0	0
		5	4.4	9.4	0	0	0	0	0	0	0	0	0	0
(3) 300 switch 300 peer	3	4	3.7	7.5	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	4	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	5	4	0.7	2.2	0	0	0	0	0	0	0	0	0	0
		5	0.7	2.2	0	0	0	0	0	0	0	0	0	0
(4) 400 switch 400 peer	4	4	11.7	35.1	0	0	0	0	0	0	0	0	0	0
		5	0.7	2.2	0	0	0	0	0	0	0	0	0	0
	5	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	0	0	0	0	0	0	0	0	0	0	0	0
	6	4	0	0	0	0	0	0	0	0	0	0	0	0
		5	2.9	8.8	0	0	0	0	0	0	0	0	0	0
(5) 500 switch 500 peer	5	4	2.92	8.8	0	0	0	0	0	0	0	0	0	0
		5	10.2	30.7	0	0	0	0	0	0	0	0	0	0
	6	4	33.7	91.5	0	0	0	0	0.8	2.4	0.7	2	0	0
		5	44.7	129	0	0	0	0	0	0	0	0	0	0
	7	4	0	0	13.9	41.8	1.3	3.97	0	0	0	0	0	0
		5	1092	3276	0	0	0	0	0	0	0	0	0	0

observe nonzero values for the cases with Algorithm 4 as in HD1. In addition, as the topology size increases, the freeze event occurrence increases. The reason behind this is that Algorithm 4 ignores the location of CDN PoP and the groups may end up far from it, which hardens the process of pulling chunks from CDN PoP. Furthermore, we can say that rerouting prevents freeze events because when the algorithm was able to find a sender via rerouting, the receiver peer did not encounter freeze event according to the results of rerouting and freeze event count given in related tables.

Startup Delay Results

Startup delay results are given separately for each edge access network size in Figures 4.9 and 4.10. The graphs on the left are for HD1, while the ones on the right are for HD2. Also, the bars denote the average startup delay while the error bars indicate the maximum startup delay in milliseconds.

For live video services, the startup delays less than 10 seconds are acceptable. Therefore, we can easily say that maximum link bandwidth of 3 Mbit/s is not acceptable for HD1 with topology size of 300, 400 and 500 peers and for HD2 with any topology size. In addition, maximum link bandwidth of 6 Mbit/s is not acceptable for HD2 with topology size of 400 and 500 peers. Changing the number of neighborhood groups also has an effect on the results; however, there is no significant difference between the results with different k values. In addition, it is easily seen that when the neighborhoods are formed with CDN PoP location aware K-means with hop count method that is proposed in this thesis, both the average and the maximum startup delays decrease. As for the maximum startup delays we observe very high results for topologies with 400 and 500 peers as Figure 4.10 depicts. The reason of these high peaks is the neighborhood formation method. When the topology size is big and the method in Algorithm 4 is used for the neighborhood formation, some groups end up very far from CDN PoP, which increases the maximum startup delay results.

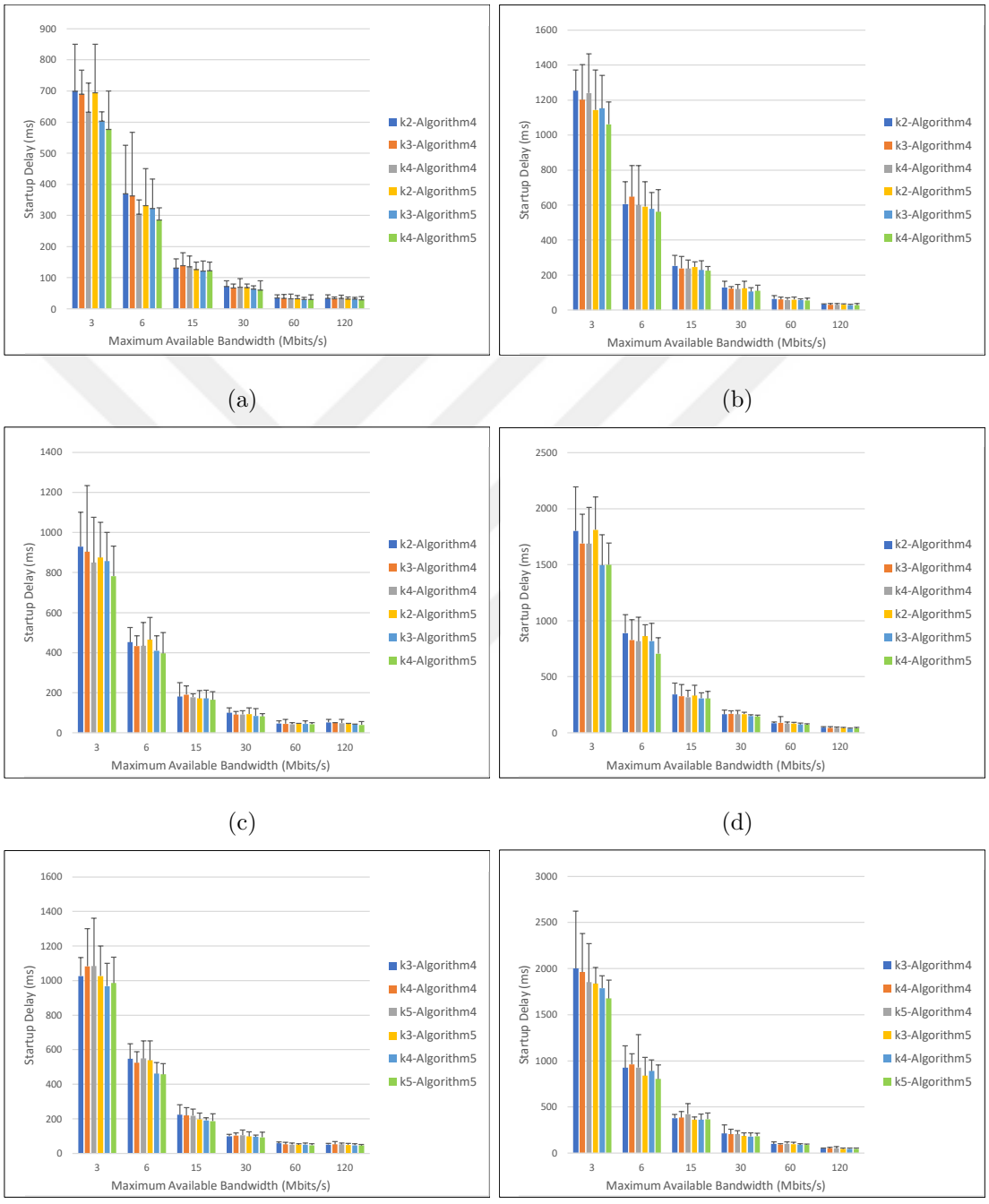


Figure 4.9: Average and maximum startup delay results: (a) topology 1, HD1, (b) topology 1, HD2, (c) topology 2, HD1, (d) topology 2, HD2, (e) topology 3, HD1, and (f) topology 3, HD2.

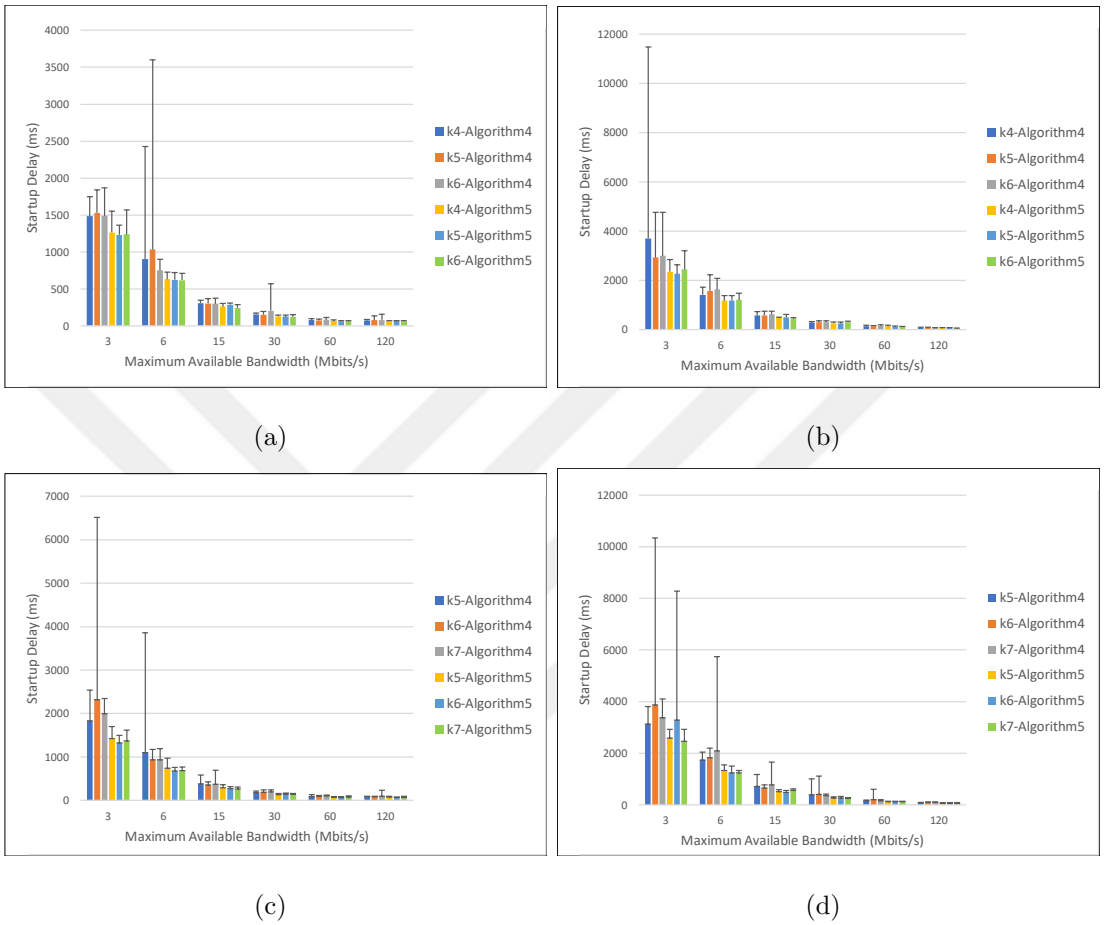


Figure 4.10: Average and maximum startup delay results: (a) topology 4, HD1, (b) topology 4 , HD2, (c) topology 5, HD1, and (d) topology 5, HD2.

Further details about the comparison of Algorithm 4 and 5 are supplied in Figure 4.11, which for each topology size. This figure shows the reduction percentage of both the average and the maximum startup delay when CDN PoP location aware neighborhood formation method is preferred over the other method introduced. As it can be seen Algorithm 5 performs better than Algorithm 4 and it performs even better as the topology size increases. For smaller topologies, the probability of a group being far from CDN PoP is smaller; therefore, the reduction percentage between two methods is not as high as in bigger topologies. In addition, we observe that the reduction percentage of maximum startup delay in comparison to that of average startup delay increases as the topology size increases because of the reasons stated.

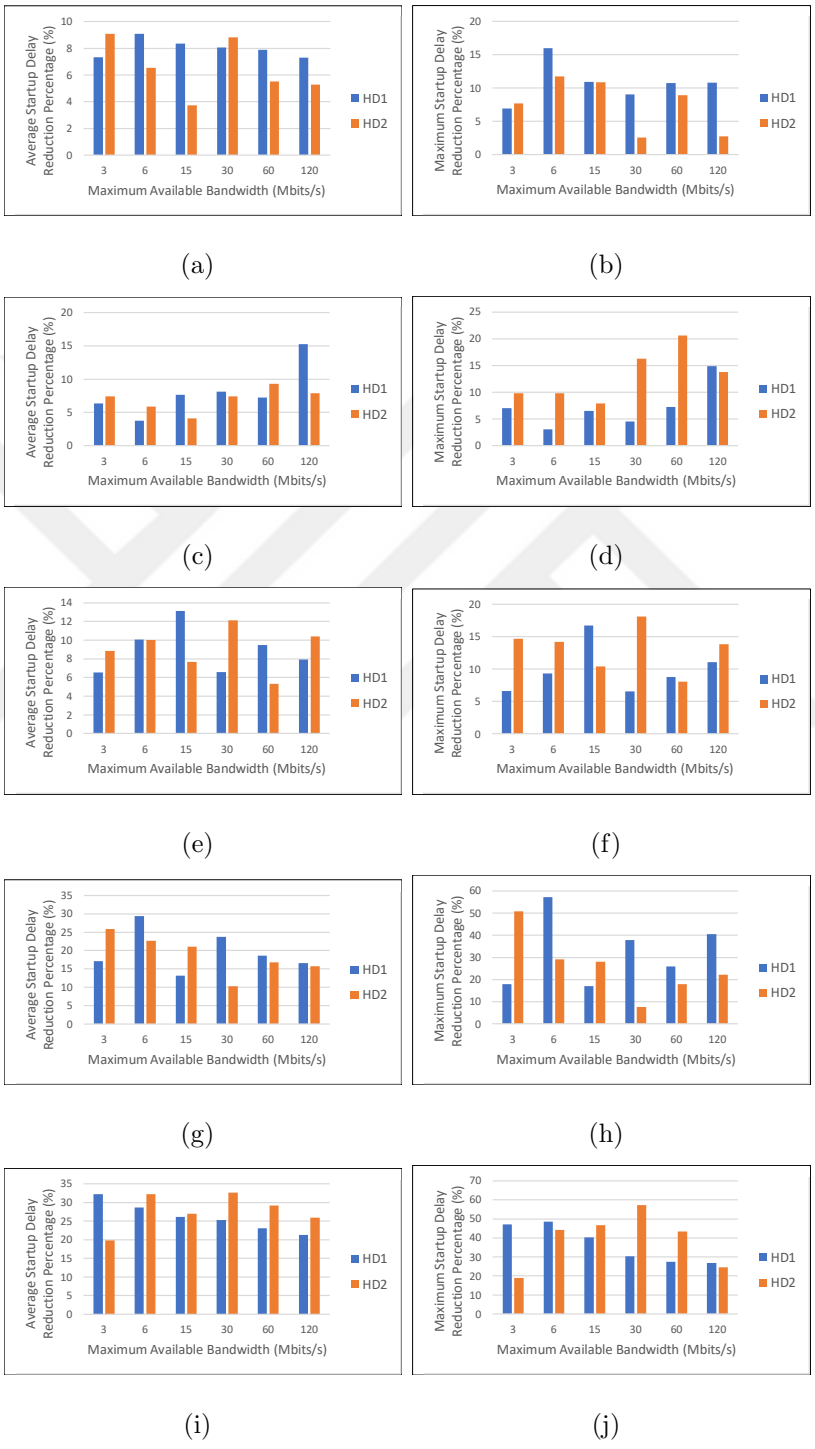
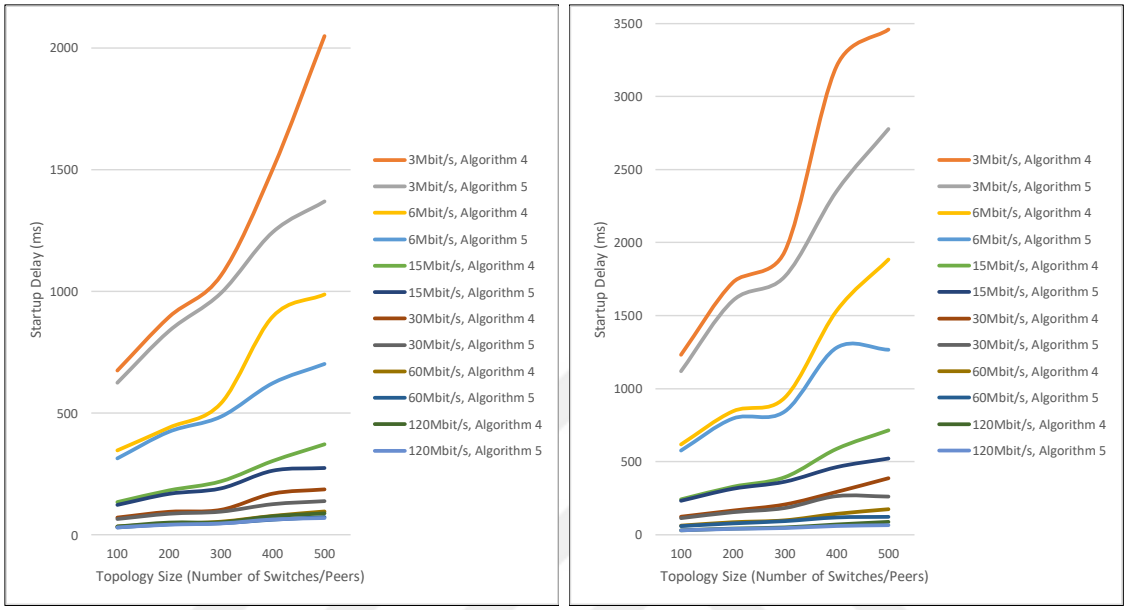
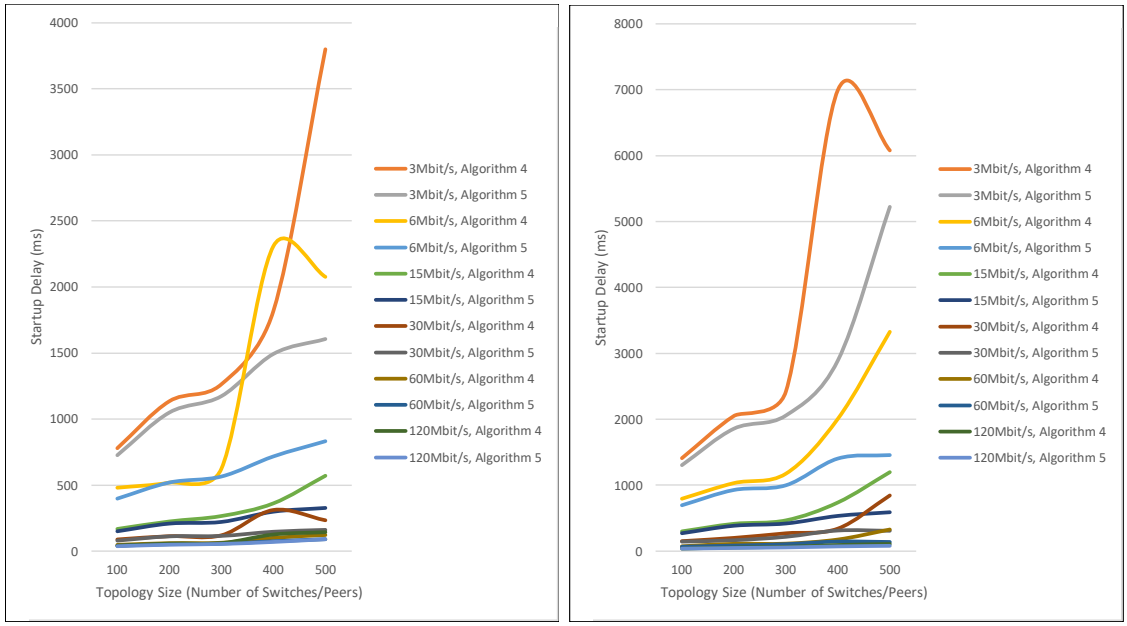


Figure 4.11: Startup delay reduction percentage with Algorithm 5 in comparison to Algorithm 4: (a) topology 1 - average value, (b) topology 1 - maximum value, (c) topology 2 - average value, (d) topology 2 - maximum value, (e) topology 3 - average value, (f) topology 3 - maximum value, (g) topology 4 - average value, (h) topology 4 - maximum value, (i) topology 5 - average value, (j) topology 5 - maximum value.



(a) (b)

Figure 4.12: Average startup delay vs. topology size: (a) HD1, and (b) HD2.



(a) (b)

Figure 4.13: Maximum startup delay vs. topology size: (a) HD1, and (b) HD2.

Finally, the average and the maximum startup delay versus topology size graphics are given in Figures 4.12 and 4.13, respectively. For each bandwidth value, we took the mean value of the results with different k values. These figures state the fact that increasing the topology size, increases the startup delay. Also, both figures show that Algorithm 5 performs better than Algorithm 4 and we can state that the difference between these two methods is much higher for the maximum startup delay results because of the outage neighborhood groups in the edge access network.

Determination of the Number of Neighborhood Groups (K)

Changing the parameter k, which denotes the number of neighborhood groups in the edge access network, ends up changing the startup delay and the total CDN load of the edge while making an insignificant difference in average bandwidth consumption.

Figure 4.14 depicts the average startup delay for maximum link bandwidth of 120Mbit/s for each topology size. It can be seen that increasing the number of groups, creates a tendency to decrease in the average startup delay for CDN PoP location aware grouping method as the groups come closer to CDN PoP and the average number of peers in a group decrease. However; for the method without CDN PoP location awareness, increasing the number of groups bring some groups further away from CDN PoP which ends up increasing the average startup delay.

The results of total CDN load for maximum link bandwidth of 120Mbit/s for each topology size is shown in Figure 4.15. As it can be seen increasing the number of groups increase the total CDN load for each case. The reason behind this situation is that the CDN PoP needs to send all the chunks to each group, which directly increases the total CDN load.

Examining the results given in Figure 4.14 and 4.15, we can conclude that the determination of the number of groups in an edge access network depends on the service requirements. If a lower startup delay is needed than the k value should be increased with the cost of increasing the total CDN load and vice versa.

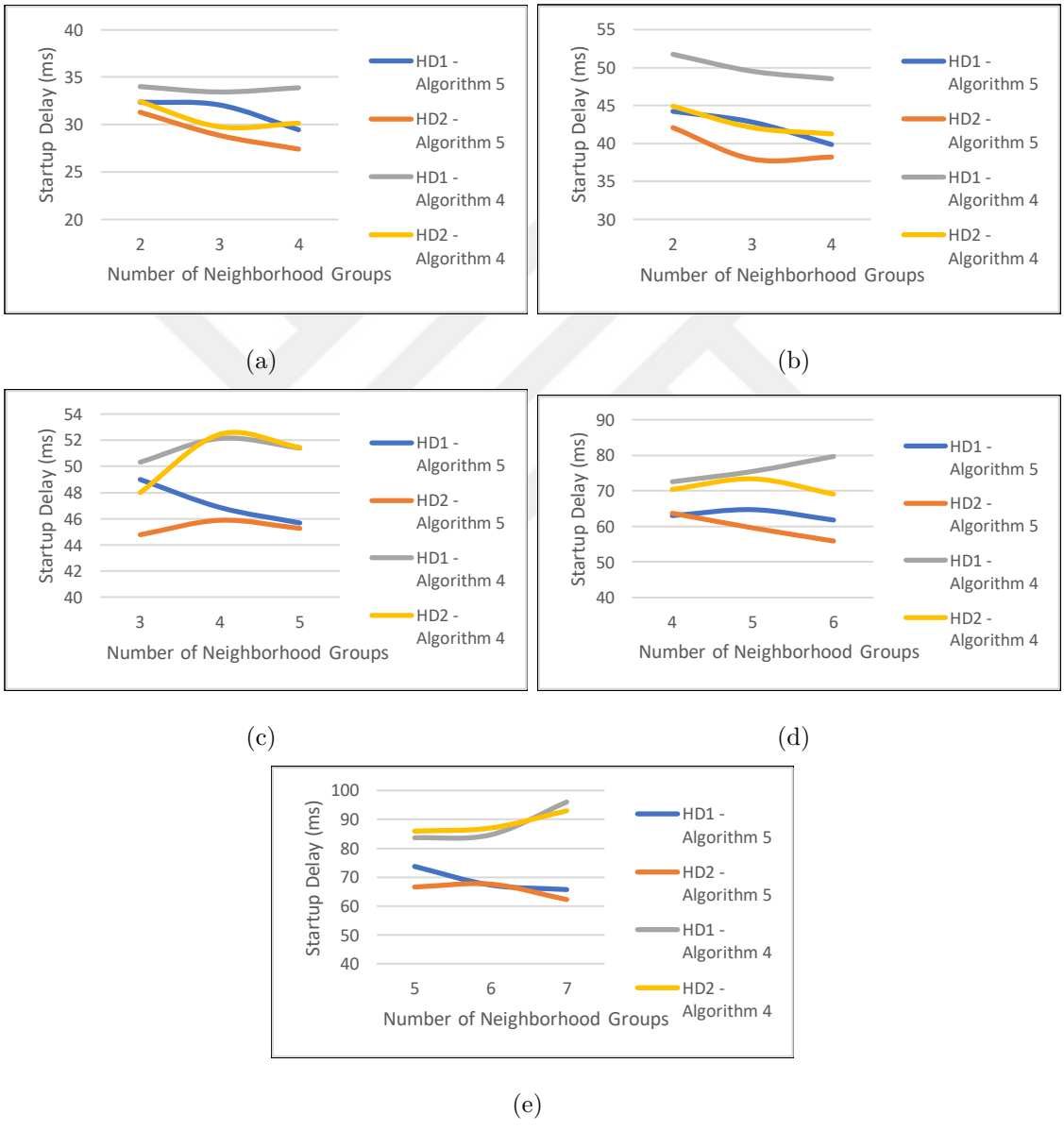


Figure 4.14: Average startup delay vs. number of neighborhood groups (k) for maximum link bandwidth of 120 Mbit/s: (a) topology 1, (b) topology 2, (c) topology 3, (d) topology 4, and (e) topology 5.

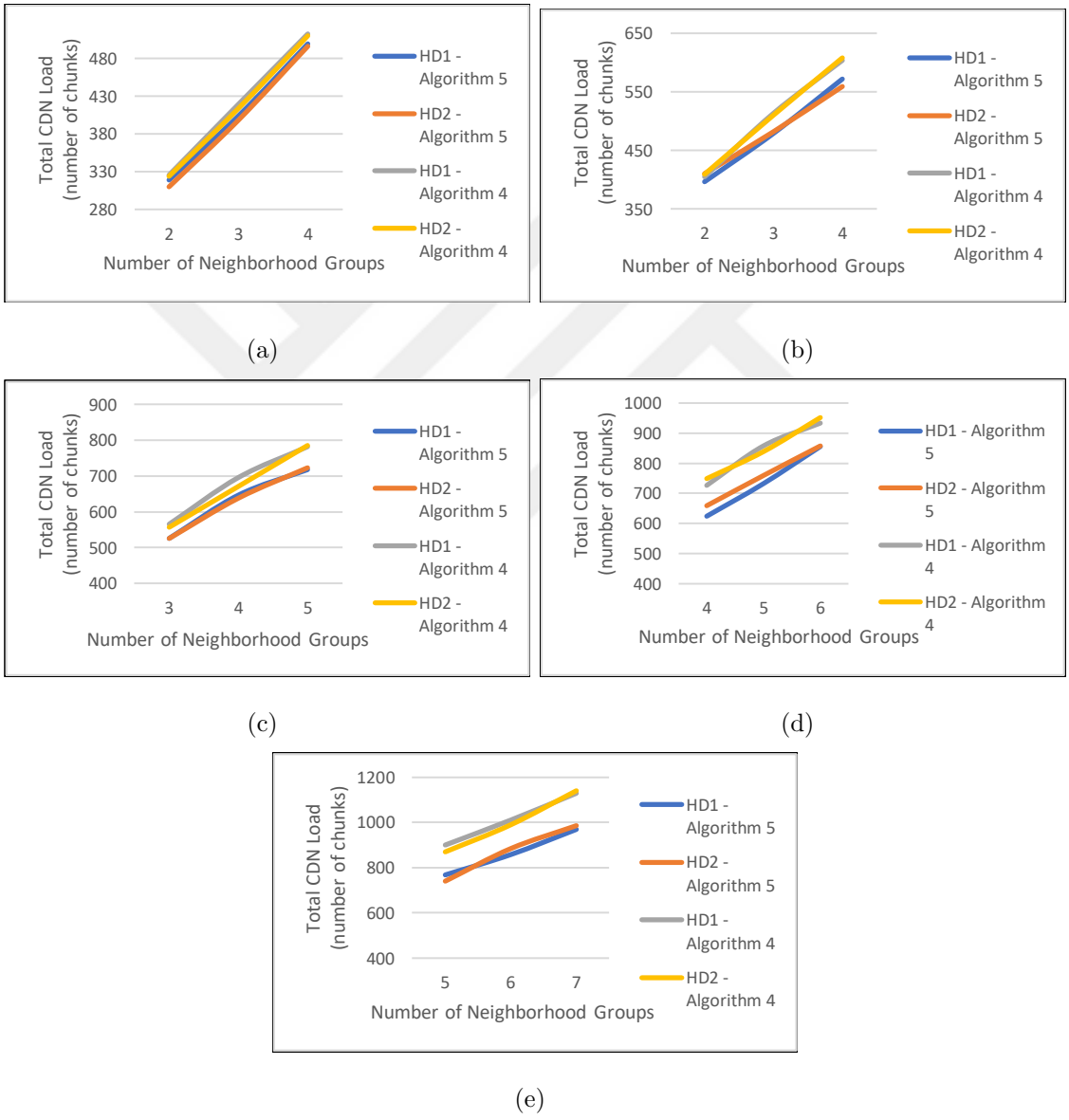


Figure 4.15: Total CDN load vs. number of neighborhood groups (k) for maximum link bandwidth of 120 Mbit/s: (a) topology 1, (b) topology 2, (c) topology 3, (d) topology 4, and (e) topology 5.

Bandwidth Consumption Results

The results obtained for the bandwidth consumption reduction can be seen in Figure 4.16 for all topology sizes. We compared our proposed architecture with the client-server approach and also we took some results regarding the comparison between multicast and client-server to observe how close is the results of our proposed architecture to the reduction multicast creates.

Firstly, we can state that the standard variation values for these results are close to zero and changing the k value did not have a significant effect on the results. Therefore, we gave the results as the mean values of the results with different k values for both methods introduced in Algorithms 4 and 5. We can easily see that the neighborhood formation algorithm does not have a big effect on the results as both algorithms try to decrease the total hop counts between peers in a group. In addition, it can be seen that the bandwidth consumption reduction percentage of multicast compared with the client-server approach is always higher than the bandwidth consumption reduction percentage of the proposed P2P architecture compared to the client-server approach. This is an expected result as multicast is known to decrease the bandwidth consumption substantially. Also, the results for different maximum link bandwidth values does not differ a lot as it can be seen from Figure 4.16. Therefore, to observe the approximation to multicast approach, we also include the bandwidth consumption reduction percentage versus topology size for 120 Mbit/s in Figure 4.17. As it can be seen from the figure, as the topology size increases, the bandwidth reduction percentage of the proposed P2P architecture approximates to that of multicast approach more. For smaller topologies, the multicast tree has less branches, which increases its bandwidth reduction more. As the topology size increases, the number of branches in the multicast tree increase and therefore our proposed architecture approximates to multicast approach more. As a result, we can conclude that the bandwidth reduction percentage of the controllable-P2P-assisted video service architecture is in a desired range.

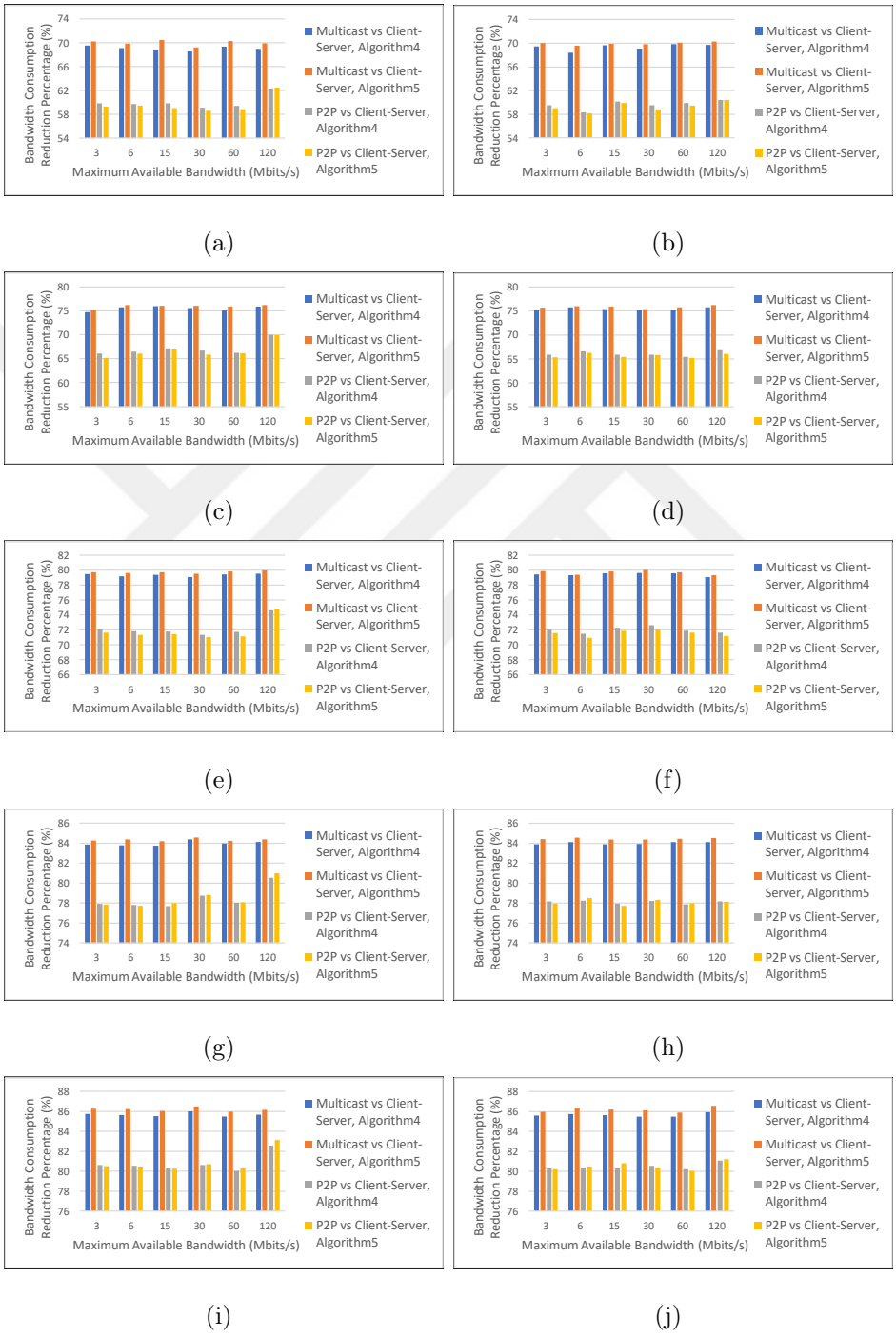
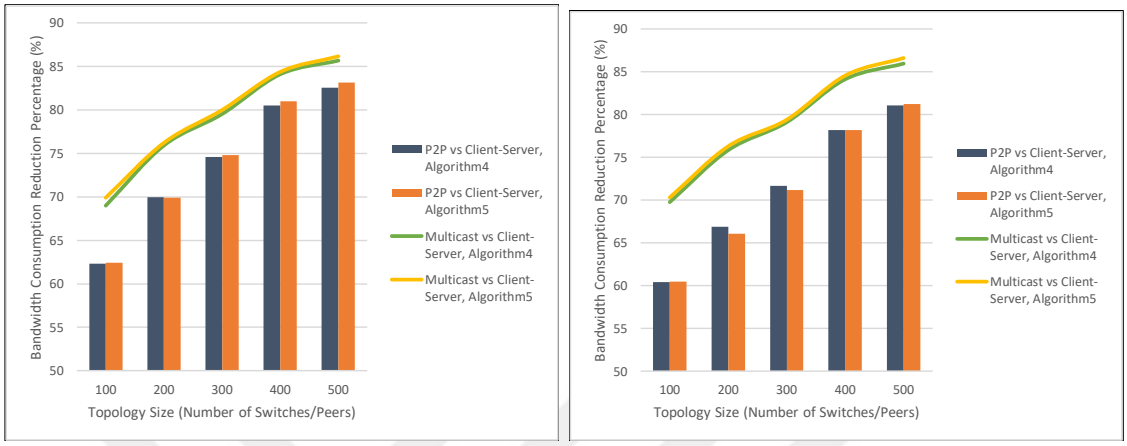


Figure 4.16: Bandwidth consumption reduction percentage results: (a) topology 1, HD1, (b) topology 1, HD2, (c) topology 2, HD1, (d) topology 2, HD2, (e) topology 3, HD1, and (f) topology 3, HD2, (g) topology 4, HD1, (h) topology 4 , HD2, (i) topology 5, HD1, and (j) topology 5, HD2.



(a) (b)

Figure 4.17: Bandwidth consumption reduction percentage vs. topology size for maximum link bandwidth of 120 Mbit/s: (a) HD1, and (b) HD2.

CDN Load Results

One of the main goals of the P2P architectures is to decrease the server (CDN) load. Therefore, we compared the CDN load reduction of our proposed architecture with both client-server and multicast. The results are supplied in Figure 4.18 for each topology size.

The results are given as the mean values of the results with different k values for both the Algorithm 4 and 5 as we did not observe significant differences between the CDN load reduction percentage results with different k values. Figure 4.18 shows that the proposed architecture creates a CDN load reduction percentage of nearly %100 for both HD1 and HD2. This shows that only %1 - %2 of the chunks are pulled from the CDN PoP when the proposed architecture is deployed. In addition, we compared the proposed architecture with multicast, too. Results show that there is at least %20 reduction on the CDN load when compared with multicast.

When we compare the results taken with Algorithm 4 and 5, it may seem the results are not being affected by the method used for neighborhood formation. However, CDN PoP location aware neighborhood formation outperforms the other group-

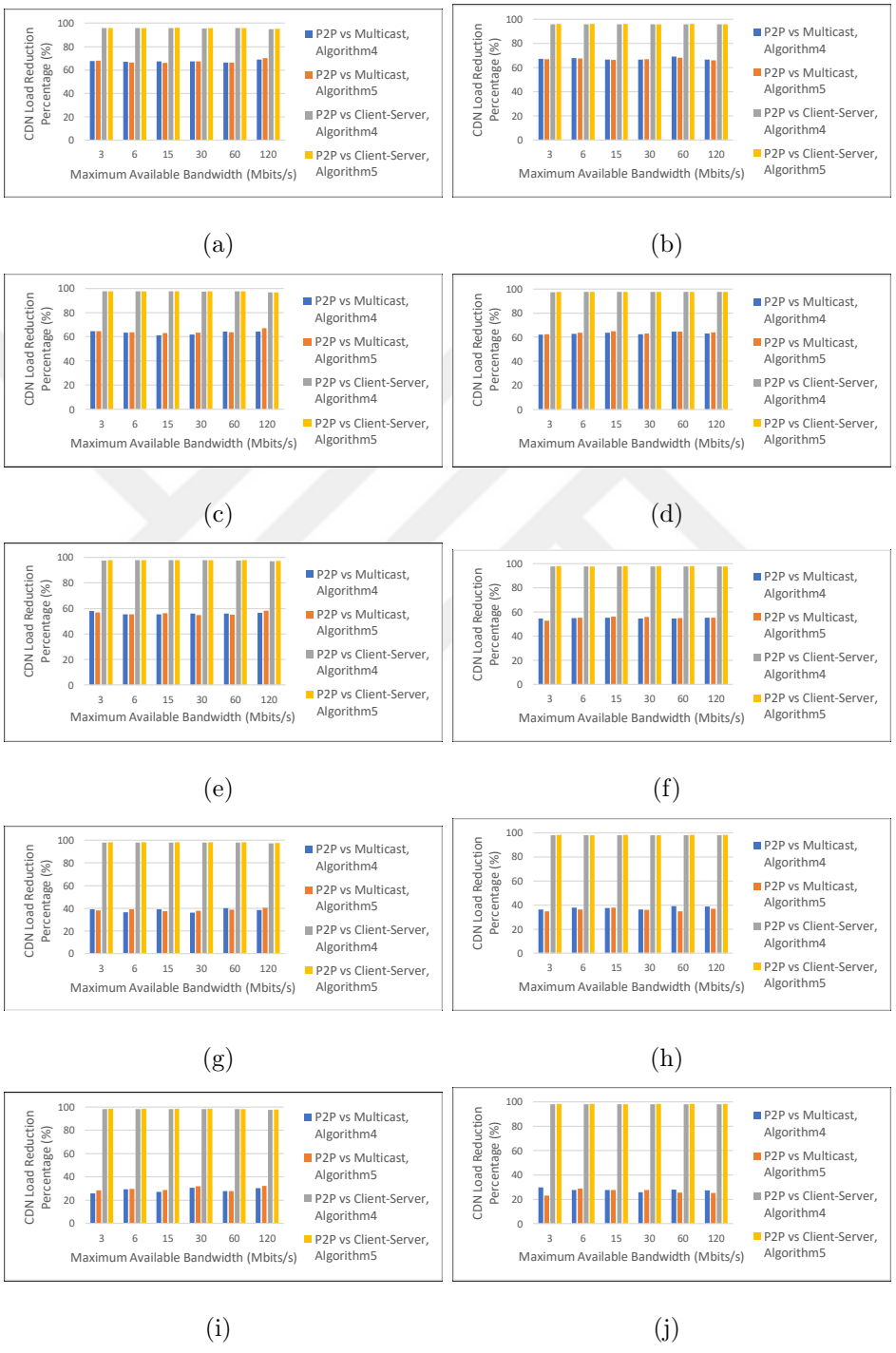


Figure 4.18: CDN load reduction percentage results: (a) topology 1, HD1, (b) topology 1, HD2, (c) topology 2, HD1, (d) topology 2, HD2, (e) topology 3, HD1, and (f) topology 3, HD2, (g) topology 4, HD1, (h) topology 4 , HD2, (i) topology 5, HD1, and (j) topology 5, HD2.

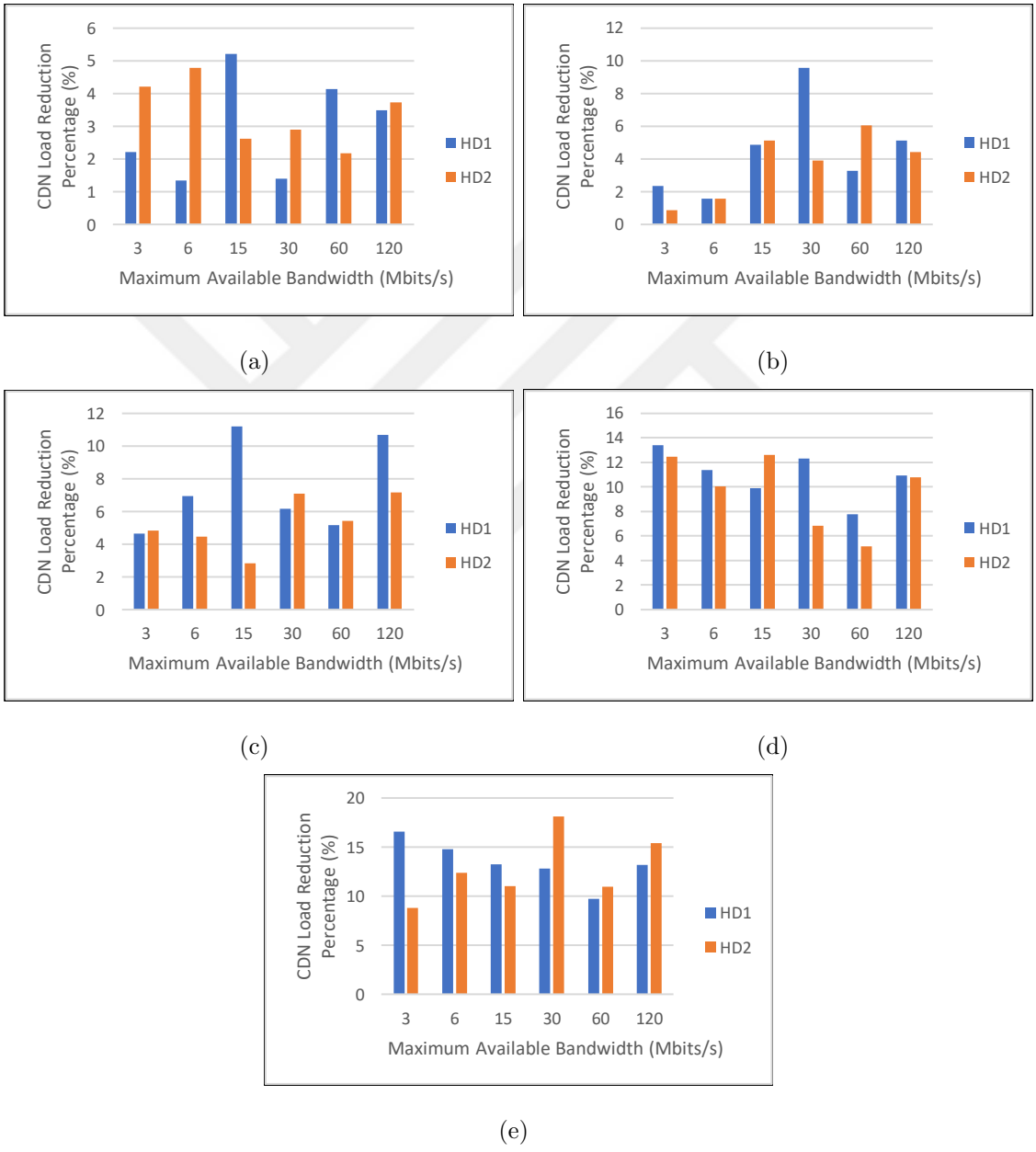
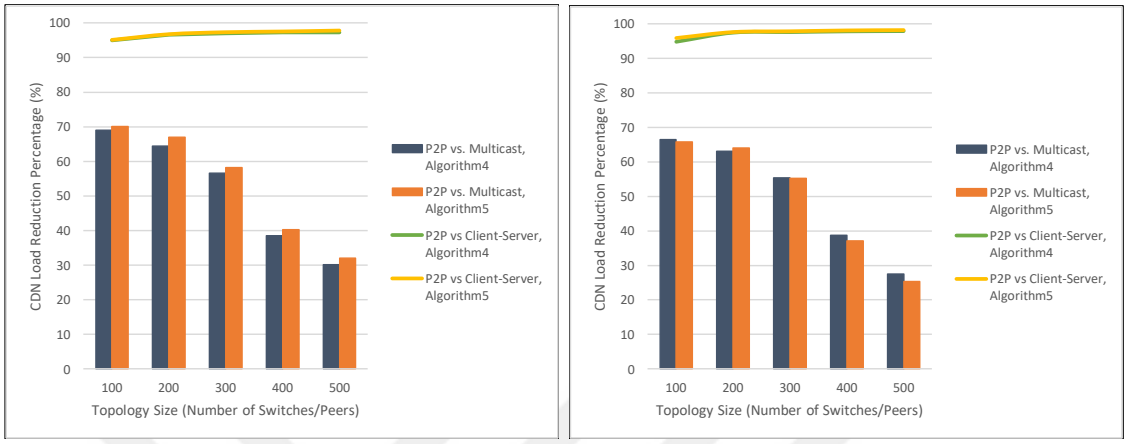


Figure 4.19: CDN load reduction percentage with Algorithm 5 in comparison to Algorithm 4: (a) topology 1, (b) topology 2, (c) topology 3, (d) topology 4, and (e) topology 5.



(a) (b)

Figure 4.20: CDN load reduction percentage vs. topology size for maximum link bandwidth of 120 Mbit/s: (a) HD1, and (b) HD2.

ing method as it is shown in Figure 4.19. It seems as if there are small changes or that Algorithm 4 has better results than Algorithm 5 in Figure 4.18. However, the neighborhood formation method improves the performance of the multicast method, too. Therefore, the percent reduction for Algorithm 5 seems less at some points. The comparison is made clear with Figure 4.19, which compares the CDN load reduction percentage results of the two methods while all the other parameters are kept constant. Figure 4.19 shows that the performance of Algorithm 5 is better in any case.

Lastly, we supplied the CDN load reduction percentage versus topology size for maximum link bandwidth of 120 Mbit/s for HD1 and HD2 in Figure 4.20. The proposed P2P architecture shows a better performance than both the client-server and the multicast approach. However, as the topology size increases, the superiority of the proposed architecture compared to the multicast approach is decreasing. The reason behind this fact is that multicast starts to perform better as the topology size increases. Although the CDN load reduction percentage is decreasing as the topology size increases when the proposed P2P architecture is compared with the multicast approach, the proposed controllable P2P-assisted CDN architecture performs better

than the multicast approach in each and every one of the tests conducted which is the critical point.

Cross-ISP Traffic Results

In the proposed architecture, the groupings are made so that all the peers in a group are subscribed to the same ISP and as the CDN PoP is placed in the edge network, only a baseline of cross-ISP traffic is introduced which is the amount of chunks pulled from the video server datacenter by the CDN PoP. However; for typical P2P video services, peers with different ISP subscriptions may belong to the same neighborhood group.

In order to show the results, we devised a two scenarios: (i) Scenario 1: only 1 peer from each group will create cross-ISP traffic and then distribute the chunks to its group, and (ii) Scenario 2: only 2 peers from each group will create cross-ISP traffic and then distribute the chunks to their groups. Let us assume that the peers in the same neighborhood group are distributed to the ISPs uniformly and each ISP is connected to each other in a mesh structure. Also, CDN PoP is placed in one of these ISPs and it is directly connected to the video server domain. Therefore, each exit for a chunk costs 1 cross-ISP traffic in terms of number of chunks.

The results are given in Figure 4.21 for these two scenarios and the proposed

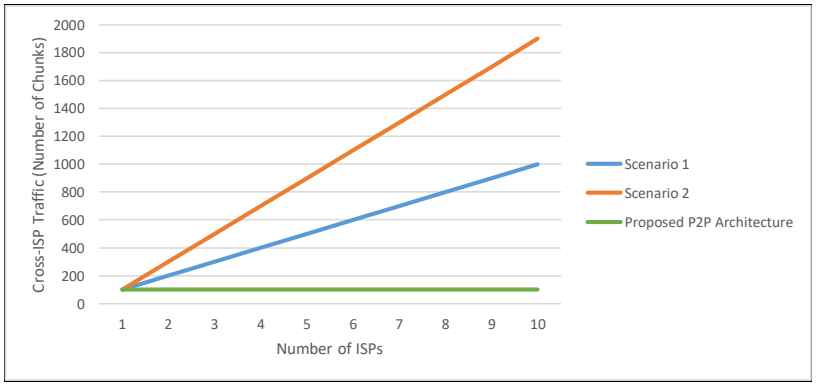


Figure 4.21: Number of chunks creating cross-ISP traffic vs. number of ISPs.

architecture. As it can be seen the proposed architecture causes a baseline of cross-ISP traffic of 100, where the number of chunks is 100. The results for the other two scenarios increases linearly with the number of peers exiting its ISP. The results are same for the case of one ISP as all the peers and the CDN PoP are in the same ISP. However, as the number of ISPs in a group increase, the cross-ISP traffic created by both Scenario 1 and Scenario 2 increases.

Chapter 5

CONCLUSION

In this thesis we have proposed an SDN-based controllable-P2P-assisted CDN for HTTP adaptive live video streaming over edge access networks recognizing the shortcomings of current CDN and P2P video solutions and the potential of SDN-based edge cloud and edge-access network technologies.

The main contributions of this thesis are listed in Chapter 1. Mainly, integration of hybrid P2P-CDN architecture with NSP edge clouds is a novel concept. Uncontrollable nature of P2P services is a critical problem and this architecture delivers a solution to this issue, which in return makes it possible for all the parties involved in P2P service to benefit. The proposed neighborhood group formation and the centralized chunk scheduling methods are both novel and they improve the performance of the P2P video services. In addition, SDN-assisted cloud allows managing video delivery; hence, video quality loss due to congestion within edge network domains is avoided by means of this architecture. Therefore, it is possible to offer video services with end-to-end quality guarantees. There are several other contributions explained throughout this dissertation.

The proposed architecture and its components are explained in Chapter 2 in detail. In this novel architecture, the edge access network consists of a group of peers connected through the switches in the edge network and a multi-access edge computing unit, which includes the aggregation switch, SDN controller, CDN PoP, and five web servers namely, traffic engineering manager (TEM), ISP accountant, Peer-to-Peer APP, content accountant, and CDN APP which communicate with the SDN controller through REST API. An important feature of the proposed hybrid service architecture is that both the CDN access by clients and P2P video streaming be-

tween clients are controlled by the network service provider within each SDN-based edge network domain to optimize video service key performance indicators.

Next, Chapter 3 gives all the details of the implementation method of the proposed architecture. Neighborhood group formation is a dynamic procedure, which i) minimizes cross-ISP traffic, ii) minimizes total cost of all groups, and iii) brings each group closer to CDN PoP. Next, chunk scheduling is implemented in a centralized manner unlike typical P2P services in which the peers themselves give the scheduling decisions. The centralized chunk scheduling avoids self-favoring and gives decisions which are beneficial for all the peers. For the chunk transmission between peers, WebRTC data channel is used as it introduces less delay. Lastly, accounting is done in order to keep incentive scores and to charge each peer fairly.

Experimental details and the results of the thesis are given in Chapter 4. We firstly implemented an emulation environment as a proof of concept. Then, the simulation environment details and the experimental scenarios are supplied with the key performance indicators. Random network topologies are created according to the Waxman edge probability and the results show that appropriate α and β values ensure a good density of edges in the topology graphs.

The neighborhood formation methods for the P2P component of the proposed system, which are K-means with hop count and CDN PoP location aware K-means with hop count, are tested with five different topology sizes with three different k (number of neighborhood groups) values. The results show that the K-means with hop count method nicely separates the groups according to the hop counts between peers. This method does not take the location of CDN PoP into account; therefore, some of the groups are formed far from CDN PoP which is an unwanted result. Therefore, we further extended the K-means clustering method to make it CDN PoP location aware. Results show that this method formed groups which are both close to CDN PoP and also consisting of peers that are in close proximity to each other.

Finally, video streaming results are taken with two different qualities of the animation video 'Big Buck Bunny', both of which are HD. Chunk duration is chosen

to be 6 seconds as the tests are for live video streaming. Video streaming tests are also conducted with five different topology sizes with three different k values and with six different values of maximum bandwidth per link and the results are obtained according to the KPI given.

In terms of video quality KPI, the results of freeze event count show that CDN PoP location aware K-means with hop count formation method improves the centralized chunk scheduling by avoiding any freeze event for any peer except the case for very low bandwidth values, which are insufficient. Startup delay is another measure for P2P video services and delays below 10 seconds are tolerated. Accordingly, we stated the acceptable ranges for each test case in this thesis. Our results show that as the maximum bandwidth per link increases or the topology size decreases, the startup delay decreases as expected. Also, the CDN PoP location aware neighborhood formation method improves the results for startup delay KPI, too.

In terms of network KPI, overall bandwidth consumption reduction results are taken in comparison to the client-server video delivery service and we observed that the results approximate the reduction multicast approach creates over client-server, which is considered a good result as multicast is known to reduce the bandwidth consumption in huge amounts. CDN load reduction percentage results are taken in comparison to both client-server and multicast. The results indicate that the proposed architecture performs better than both client-server and multicast in terms of CDN load reduction. Regarding the rerouting event count results, we mostly did not observe any rerouting or observed insignificant amounts of rerouting. However, when the K-means with hop count algorithm is used for the neighborhood formation or when there is not enough bandwidth to serve the requested quality, high rerouting event counts are observed. Lastly, the results regarding the cross-ISP traffic are included and they show that the proposed architecture creates only a baseline of cross-ISP traffic, while the cross-ISP traffic introduced to the network increases with the number of peers receiving chunks from different ISPs and the number of ISPs in the edge domain in typical P2P video services.

Changing the number of neighborhood groups (k) has an effect on the startup delay and the total CDN load. Increasing the k value creates a tendency to decrease in startup delay and an exact increase in total CDN load. Therefore, we concluded that k parameter should be chosen according to the criteria of the service knowing these two facts.

To conclude, proposed controllable-P2P-assisted CDN for HTTP adaptive live video streaming over edge access networks performs better than the state of the art CDN or P2P video delivery services alone according to our KPI as the results suggest. In addition, this architecture is the first integrating P2P-assisted services with SDN, WebRTC and edge computing technologies to the best of our knowledge.

BIBLIOGRAPHY

[Alimi et al., 2014] Alimi, R., Penno, R., Yang, Y., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and Woundy, R. (2014). Application-layer traffic optimization (alto) protocol. RFC 7285, RFC Editor. `http://www.rfc-editor.org/rfc/rfc7285.txt`.

[Arthur and Vassilvitskii, 2006] Arthur, D. and Vassilvitskii, S. (2006). How slow is the k-means method? In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, SCG 06, page 144153, New York, NY, USA. Association for Computing Machinery.

[Bagci et al., 2016] Bagci, K. T., Sahin, K. E., and Tekalp, A. M. (2016). Queue-allocation optimization for adaptive video streaming over software defined networks with multiple service-levels. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1519–1523.

[Bagci et al., 2017] 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.

[Bagci and Tekalp, 2018] Bagci, K. T. and Tekalp, A. M. (2018). Dynamic resource allocation by batch optimization for value-added video services over sdn. *IEEE Transactions on Multimedia*, 20(11):3084–3096.

[Bagci et al., 2016] Bagci, K. T., Yilmaz, S., Sahin, K. E., and Tekalp, A. M. (2016). Dynamic end-to-end service-level negotiation over multi-domain software defined networks. In *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, pages 33–39.

[Barbosa and Soares, 2014] Barbosa, F. R. N. and Soares, L. F. G. (2014). Towards the application of webrtc peer-to-peer to scale live video streaming over the internet. In *Simposio Brasileiro de Redes de Computadores (SBRC)*.

[Bittman, 2017] Bittman (2017). The Edge Will Eat The Cloud. `https://blogs.gartner.com/thomas_bittman/2017/03/06/the-edge-will-eat-the-cloud/`. Accessed: 2019-07-05.

[Blake et al., 1998] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W. (1998). [RFC 2475] An Architecture for Differentiated Service.

[Bouras and Stamos, 2008] Bouras, C. and Stamos, K. (2008). An efficient architecture for bandwidth brokers in diffserv networks. *Int. J. Netw. Manag.*, 18(1):27–46.

[Braden et al., 1994] Braden, R., Clark, D., and Shenker, S. (1994). Integrated services in the Internet architecture: an overview. RFC 1633.

[CaaS, 2019] CaaS (2019). Containers as a Service (CaaS). `https://searchitoperations.techtarget.com/definition/Containers-as-a-Service-CaaS`. Accessed: 2019-07-05.

[Calvert et al., 1997] Calvert, K. L., Doar, M. B., and Zegura, E. W. (1997). Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163.

[Castro et al., 2003] Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., I. T. Rowstron, A., and Singh, A. (2003). Splitstream: High-bandwidth content distribution in cooperative environments. volume 2735, pages 292–303.

[Cetinkaya et al., 2015] Cetinkaya, C., Ozveren, Y., and Sayit, M. (2015). Route optimization for dash over software defined networks. In *2015 23nd Signal Processing and Communications Applications Conference (SIU)*, pages 2454–2457.

[Cha et al., 2008] Cha, M., Rodriguez, P., Moon, S., and Crowcroft, J. (2008). On next-generation telco-managed p2p tv architectures. In *Proc. of the 7th International Conference on Peer-to-peer Systems*, IPTPS'08, pages 5–5, Berkeley, CA, USA. USENIX Association.

[Chen et al., 2008] Chen, S., Song, M., and Sahni, S. (2008). Two techniques for fast computation of constrained shortest paths. *IEEE/ACM Transactions on Networking*, 16(1):105–115.

[Civanlar et al., 2015] Civanlar, S., Lokman, E., Kaytaz, B., and Murat Tekalp, A. (2015). Distributed management of service-enabled flow-paths across multiple sdn domains. In *2015 European Conference on Networks and Communications (EuCNC)*, pages 360–364.

[Civanlar et al., 2010] 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*, pages 351–356.

[Cord, 2017] Cord (2017). CORD: Reinventing Central Offices for Efficiency and Agility. `https://opencord.org/`. Accessed: 2017-10-12.

[DASH, 2019] DASH (2019). Dynamic Adaptive Streaming over HTTP. `https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP`. Accessed: 2019-07-05.

[Datta et al., 2009] Datta, S., Giannella, C., and Kargupta, H. (2009). Approximate distributed k-means clustering over a peer-to-peer network. *IEEE Trans. on Knowledge and Data Eng.*, 21(10):1372–1388.

[Docker, 2019] Docker (2019). Docker: The Modern Platform for High-Velocity Innovation. `https://www.docker.com/why-docker`. Accessed: 2019-07-05.

[Egilmez et al., 2012a] Egilmez, H. E., Civanlar, S., and Tekalp, A. M. (2012a). A distributed qos routing architecture for scalable video streaming over multi-domain openflow networks. In *2012 19th IEEE International Conference on Image Processing*, pages 2237–2240.

[Egilmez et al., 2013] 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 et al., 2012b] Egilmez, H. E., Dane, S. T., Bagci, K. T., and Tekalp, A. M. (2012b). 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*, pages 1–8.

[Egilmez et al., 2012c] Egilmez, H. E., Dane, S. T., Bagci, K. T., and Tekalp, A. M. (2012c). 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*, pages 1–8.

[Egilmez et al., 2011] Egilmez, H. E., Gorkemli, B., Tekalp, A. M., and Civanlar, S. (2011). Scalable video streaming over openflow networks: An optimization framework for qos routing. In *2011 18th IEEE International Conference on Image Processing*, pages 2241–2244.

[Egilmez and Tekalp, 2014] Egilmez, H. E. and Tekalp, A. M. (2014). Distributed qos architectures for multimedia streaming over software defined networks. *IEEE Transactions on Multimedia*, 16(6):1597–1609.

[Farrel et al., 2008] Farrel, A., Vasseur, J., and Ayyangar, A. (2008). Inter-Domain

MPLS and GMPLS Traffic Engineering – Resource Reservation Protocol-Traffic Engineering (RSVP-TE) Extensions. RFC 5151.

[Fesci-Sayit et al., 2012] Fesci-Sayit, M., Tunali, E. T., and Tekalp, A. M. (2012). Resilient peer-to-peer streaming of scalable video over hierarchical multicast trees with backup parent pools. *Image Commun.*, 27(2):113–125.

[Floodlight, 2017] Floodlight (2017). Project Floodlight: Open Source Software for Building Software-Defined Networks. `http://www.projectfloodlight.org/floodlight/`. Accessed: 2017-10-26.

[Gorkemli et al., 2016] Gorkemli, B., Parlakisik, A. M., Civanlar, S., Ulas, A., and Tekalp, A. M. (2016). Dynamic management of control plane performance in software-defined networks. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 68–72.

[Huang and Zhang, 2014] Huang, R. and Zhang, Y. (2014). Survey of webrtc based p2p streaming. Internet-Draft draft-huang-ppsp-p2p-webrtc-survey-00, Internet Engineering Task Force. `http://www.ietf.org/internet-drafts/draft-huang-ppsp-p2p-webrtc-survey-00.txt`, Accessed: 2017-10-27.

[inCloud360, 2014] inCloud360 (2014). Data Security: Considerations, Responsibilities, and Control. `https://www.incloud360.com/data-security-considerations-responsibilities-and-control/#.XUYV--gzY2w`. Accessed: 2019-07-05.

[ITEC, 2020] ITEC (2020). Dynamic Adaptive Streaming over HTTP - Datasets. `https://dash.itec.aau.at/dash-dataset/`. Accessed: 2020-01-02.

[James and Crowley, 2010] James, S. and Crowley, P. (2010). Imp: Isp-managed p2p. In *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, pages 1–9.

[Jeon et al., 2009] Jeon, S., Kim, Y., Yi, J., and Kang, S. (2009). Isp-driven managed p2p framework for effective real-time iptv service. In *NEW2AN*.

[Juttner et al., 2001] Juttner, A., Szviatovski, B., Mecs, I., and Rajko, Z. (2001). Lagrange relaxation based method for the qos routing problem. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 2, pages 859–868 vol.2.

[Kim et al., 2010] Kim, W., Sharma, P., Lee, J., Banerjee, S., Tourrilhes, J., Lee, S.-J., and Yalagandula, P. (2010). Automated and scalable qos control for network convergence. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 1–1, Berkeley, CA, USA. USENIX Association.

[Kirmizioglu and Tekalp, 2019] Kirmizioglu, R. A. and Tekalp, A. M. (2019). Multiparty webrtc services using delay and bandwidth aware sdn-assisted ip multicasting of scalable video over 5g networks. *IEEE Transactions on Multimedia*, pages 1–1.

[Kotronis et al., 2014] Kotronis, V., Dimitropoulos, X., Klöti, R., Ager, B., Georgopoulos, P., and Schmid, S. (2014). Control exchange points: providing qos-enabled end-to-end services via sdn-based inter-domain routing orchestration. In *Presented as part of the Open Networking Summit 2014 (ONS 2014)*, Santa Clara, CA. USENIX.

[Kuipers et al., 2002] Kuipers, F., Van Mieghem, P., Korkmaz, T., and Krunz, M. (2002). An overview of constraint-based path selection algorithms for qos routing. *IEEE Communications Magazine*, 40(12):50–55.

[Levine, 2016] Levine (2016). Return to the Edge and The End of Cloud Computing. `https://www.youtube.com/watch?v=l9tOd6fHR-U`. Accessed: 2019-07-05.

[Li Xiao et al., 2002] Li Xiao, King-Shan Lui, Jun Wang, and Nahrstedt, K. (2002). Qos extension to bgp. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 100–109.

[Longwell, 2018] Longwell (2018). IHS Markit: %85 of Operators Plan to Deploy Smart Central Offices. `https://www.sdxcentral.com/articles/news/ihs-markit-85-of-operators-plan-to-create-smart-central-offices/2018/01/`. Accessed: 2019-07-05.

[Lopez et al., 2017] Lopez, V., Gran Josa, J. M., Uceda, V., Slyne, F., Ruffini, M., Vilalta, R., Mayoral, A., Munoz, R., Casellas, R., and Martinez, R. (2017). End-to-end service orchestration from access to backbone. *IEEE/OSA Journal of Optical Communications and Networking*, 9(6):B137–B147.

[M-CORD, 2016] M-CORD (2016). Rethinking the Mobile Edge Network with CORD (Mobile-CORD). `https://sdn.ieee.org/newsletter/march-2016/rethinking-the-mobile-edge-network-with-cord-mobile-cord`. Accessed: 2019-07-05.

[Masip et al., 2006] Masip, X., Yannuzzi, M., Domingo-Pascual, J., Fonte, A., Curado, M., Antonio Kuipers, F., and Van Mieghem, P. (2006). Research challenges in qos routing. *Computer Communications*.

[McKeown et al., 2008] 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.

[Mendiola and et al., 2014] Mendiola, A. and et al. (2014). Multi-domain software defined network: exploring possibilities. In *TERENA Networking Conference (TNC 2014)*, Dublin, Ireland.

[Mininet, 2017] Mininet (2017). Mininet: An Instant Virtual Network on your Laptop (or other PC). `http://mininet.org/`. Accessed: 2017-10-26.

[NFV, 2019] NFV (2019). network functions virtualization (NFV). `https://searchnetworking.techtarget.com/definition/network-functions-virtualization-NFV`. Accessed: 2019-07-05.

[NFV-MANO, 2019] NFV-MANO (2019). What is NFV MANO? `https://www.sdxcentral.com/networking/nfv/mano-lso/definitions/nfv-mano/`. Accessed: 2019-07-05.

[Node, 2017] Node (2017). Node.js. `https://nodejs.org/en/`. Accessed: 2017-12-26.

[Noghani and Sunay, 2014] 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*, pages 551–556.

[Nunes et al., 2014] 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.

[ONF, 2019] ONF (2019). SEBA. `https://www.opennetworking.org/seba/`. Accessed: 2019-07-05.

[OpenDaylight, 2019] OpenDaylight (2019). OpenDaylight: A Linux Foundation Collaborative Project. `https://www.opendaylight.org/`. Accessed: 2019-07-05.

[OpenStack, 2019] OpenStack (2019). What is OpenStack? `https://www.openstack.org/software`. Accessed: 2019-07-05.

[OPNFV, 2019] OPNFV (2019). OPNFV. `https://www.opnfv.org/`. Accessed: 2019-07-05.

[P2PTV, 2019] P2PTV (2019). P2PTV. `https://en.wikipedia.org/wiki/P2PTV`. Accessed: 2019-07-05.

[Peer5, 2017] Peer5 (2017). Peer5. `https://www.peer5.com/cdn`. Accessed: 2017-12-27.

[Peterson et al., 2019] Peterson, L., Anderson, T., Katti, S., McKeown, N., Parulkar, G., Rexford, J., Satyanarayanan, M., Sunay, O., and Vahdat, A. (2019). Democratizing the network edge. *SIGCOMM Comput. Commun. Rev.*, 49(2):31–36.

[Phemius et al., 2014] Phemius, K., Bouet, M., and Leguay, J. (2014). DISCO: Distributed SDN controllers in a multi-domain environment. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–2.

[R-CORD, 2019] R-CORD (2019). R-CORD. `https://www.opennetworking.org/r-cord/`. Accessed: 2019-07-05.

[Raghavan et al., 2012] Raghavan, B., Casado, M., Koponen, T., Ratnasamy, S., Ghodsi, A., and Shenker, S. (2012). Software-defined internet architecture: Decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 43–48, New York, NY, USA. ACM.

[Sahin et al., 2017] Sahin, K. E., Bagci, K. T., and Tekalp, A. M. (2017). Distributed-collaborative managed dash video services. In *Proc. of the Int. Conf. on Network and Service Management (CNSM)*.

[Shenker, 2011] Shenker (2011). The future of networking, and the past of protocols. `https://docs.huihoo.com/open-networking-summit/2011/the-future-of-networking-and-the-past-of-protocols.pdf`. Accessed: 2019-07-05.

[Shibasaki et al., 2016] Shibasaki, R., Matsumoto, N., and Yoshida, N. (2016). Sdn-based implementation of p2p streaming networks with dynamic reconfiguration. In *The Tenth International Conference on Digital Society and eGovernments*.

[Streamroot, 2020] Streamroot (2020). The most trusted hybrid P2P CDN for live streaming and video on-demand. `https://streamroot.io/streamroot-dna/`. Accessed: 2020-2-1.

[Trajkovska et al., 2010] Trajkovska, I., Salvachua Rodriguez, J., and Mozo Velasco, A. (2010). A novel p2p and cloud computing hybrid architecture for multimedia streaming with qos cost functions. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, pages 1227–1230, New York, NY, USA. ACM.

[Uludag et al., 2007] Uludag, S., Lui, K.-S., Nahrstedt, K., and Brewster, G. (2007). Analysis of topology aggregation techniques for qos routing. *ACM Comput. Surv.*, 39(3).

[Waxman, 1988] Waxman, B. M. (1988). Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622.

[WebRTC, 2019] WebRTC (2019). Getting Started with WebRTC. `https://www.html5rocks.com/en/tutorials/webrtc/basics/`. Accessed: 2019-07-05.

[Websocket, 2017] Websocket (2017). WebSockets. `https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API`. Accessed: 2017-12-26.

[Wikipedia, 2019] Wikipedia (2019). Streaming Media. `https://en.wikipedia.org/wiki/Streaming_media`. Accessed: 2019-07-05.

[Xie et al., 2008] Xie, H., Yang, Y. R., Krishnamurthy, A., Liu, Y. G., and Silberschatz, A. (2008). P4p: Provider portal for applications. *SIGCOMM Comput. Commun. Rev.*, 38(4):351–362.

[Xue et al., 2008] Xue, G., Zhang, W., Tang, J., and Thulasiraman, K. (2008). Polynomial time approximation algorithms for multi-constrained qos routing. *IEEE/ACM Transactions on Networking*, 16(3):656–669.

[Zegura et al., 1996] Zegura, E. W., Calvert, K. L., and Bhattacharjee, S. (1996). How to model an internetwork. In *Proceedings of IEEE INFOCOM '96. Conference on Computer Communications*, volume 2, pages 594–602 vol.2.