Received August 24, 2022; | Revised August 24, 2022; | Accepted August 24, 2022

DOI: xxx/xxxx

RESEARCH ARTICLE

vDANE: Using Virtualization for Improving Video Quality with Server and Network Assisted DASH

Reza Shokri Kalan[1] | Stuart Clayman[2] | Müge Sayıt[3]

[1]International Computer Institute, Ege University, Izmir, Turkey
Digiturk beIN Media Group, Digiturk, Istanbul, Turkey

[2]Dept of Electronic Engineering, University College London, London, UK
[3]International Computer Institute, Ege University, Izmir, Turkey

Correspondence
Reza Shokri Kalan, Digiturk beIN Media Group, Digiturk, Istanbul, Turkey. Email: reza.shokri@hotmail.com

Summary

Network Function Virtualization (NFV) offers flexibility in traffic engineering and network resource management, by taking advantage of Software Defined Networking (SDN). By using these network technologies it is possible to enhance the performance of video streaming applications by placing network functions in suitable locations and rerouting flows. Our study addresses the "virtual cache placement" problem in dynamic networks, where traffic patterns and attachment points of the clients are changing rapidly. The cache placement is done by determining how many virtual caches are necessary to be able to provide acceptable service to the clients, as well as where to place those caches to meet demand. To this end, we provide a heuristic solution by taking advantage of NFV-SDN and having the assistance of Server and Network Assisted DASH (SAND). Experimental results show that the proposed algorithms can improve the video client re-buffering by 150% - 270% and also can provide an 8% - 12% increase in average bitrate received by the client, compared to a number of benchmark algorithms. The obtained results indicate that the co-operation between the client and the operator of an SDN enabled network, by exchanging client and network information, allows network resources to be efficiently used, and as a consequence the Quality of Experience (QoE) on the client's side is improved.

KEYWORDS:
Adaptive streaming, Virtual Cache, NFV-SDN, DASH, SAND

1 | INTRODUCTION

Streaming high-quality video is a priority yet a challenge for network providers, where the main volume of the current Internet IP traffic is video packets. *HTTP Adaptive Streaming* (HAS) has become a de facto streaming technology, which achieves the best possible quality for the user, by improving resource utilization and better adaptation to network conditions. *Dynamic Adaptive Streaming over HTTP* (DASH)[1] is a standard developed by the MPEG research group, to provide interoperability between the elements of various HAS systems, which were developed by different corporations. *Server and Network Assisted DASH* (SAND)[2] is another on-going standardization process related to DASH, which is being developed by the MPEG group. In the SAND architecture, some network elements are extended to have information about the DASH specification, and are defined in the standard as *DASH Aware Network Elements* (DANEs). The SAND architecture assists clients to achieve better *Quality of Experience* (QoE) by taking advantage of network-related information provided by DANEs.

Providing and sustaining the quality of the video is considered a challenge, since network dynamics can cause an interruption of the provided services. Caching content in a *Point of Presence* (POP), which is near to the clients, reduces both the latency and

the total amount of network traffic flows over the network, but implementing a new POP is more costly and takes time. The recent standardisation efforts and proposals indicate that there is a tendency to provide "network support" to video streaming systems. As such, the emerging networking technologies of *Software Defined Networking* (SDN) and *Network Function Virtualization* (NFV) are good options that can provide such support. Content distributors may leverage the virtualization features provided by NFV platforms in order to deploy a virtualized cache (vCache), as *Virtual Network Functions* (VNFs), instead of using physical appliances. SDN is a complementary technology, that has changed the architecture of legacy networks and makes NFV implementations more flexible and easier to manage[3],[4]. Encapuslating both of these into a solution leads to a system that has a dynamic behaviour as well as reduces the cost.

This paper addresses the problem of the placement of vCaches on forwarding devices in a highly dynamic network where the available bandwidth of links changes while clients join or leave the network. We have developed two algorithms for providing optimal cache placement and reliability. We consider the nodes in which to place instances of a vCache by taking into account: (i) the number of links connected to the nodes; (ii) the available bandwidth of these links; and (iii) the density of the connected clients. When determining and installing the first (or initial) vCache in the network, the *PressureInitialCache* algorithm is used. However, when we need additional on-demand vCache installations, we use the *PressureCache* algorithm for this purpose. Furthermore, vCache migration is essential to the efficient use of network resources, while coping with changes in network traffic patterns[5], as highly dynamic network conditions require the migration of resources when it is needed.

In order to manage the network and the vCaches, we use SDN technology and MPEG SAND features. To the best of our knowledge, this is the first study that utilizes network virtualization by using vCaches which have SAND characteristics, for improving the QoE of DASH clients. We have devised the term *vDANE* to represent this kind of SAND aware vCache that has knowledge about DASH. Using these technologies, it is possible to utilize common virtualization techniques to migrate vCache resources on-demand, to be nearer to the end-users. Considering the large amount of traffic to be delivered to video streaming clients, creating an efficient algorithm for migrating vCaches to an appropriate location requires knowledge of both network conditions and client status in a real-time manner.

In this study we utilize vDANE vCache migration by considering the changes of both the client and network conditions. The effects of vCache migration on QoE is investigated and an approach for minimizing those effects is presented. We propose an architecture which selects an almost optimal location for installing the initial vDANE vCache as a network function, migration of a vDANE, and/ or installing additional vDANE on demand. For this purpose, an adaptive approach which considers resource availability and performance requirements in terms of the end user's QoE is developed. A key problem is how to conduct effective vCache placement, while meeting the bandwidth requirement for multimedia services, yet saving as many network resources as possible. While our previous studies have only considered installing one cache on the network, the aim of this study is to find the optimal number of caches, locate more than one cache and determine their migration times to satisfy end-users QoE. Furthermore, this study has dynamic nature and is adaptable to network infrastructure changes such as traffic and client distribution updates.

The paper is organized as follows: the background and information about the tools and technologies used in this study are given next. A virtual cache placement approach, for the first vCache deployment, based on pressure score estimations and the *PressureInitialCache* algorithm are discussed in Section *3*. The test bed and experimental environment used for measuring the performance of the *PressureInitialCache* algorithm is presented in Section *4*, followed by the performance evaluation and experimental results given in Section *5*. In Section *6*, we present the *PressureCache* algorithm which is executed when additional vCache capacity is required in order to prevent a decrease in the video quality received by the clients, followed by cache migration and its effects on the received video quality. Finally, we conclude the paper and give the directions of future work in Section *7*.

2 | BACKGROUND

2.1 | Adaptive Video Streaming Technologies, DASH and SAND Standards

HTTP Adaptive Streaming (HAS) is a technology that aims to provide optimal quality to the clients given the constraints of changeable network conditions and client status. In HAS, multiple copies of the same video input are encoded with different qualities (or *representations*), and saved as different outputs. Each of these representation is split and kept on the server in a large number of same-sized small partitions, which are known as segments or chunks. The player on the client-side observes the network conditions, and adaptively makes changes to the presented quality during streaming, by requesting chunks with different qualities. The quality is determined, over time, on the basis of a rate adaptation algorithm and the available bandwidth.

Having a high-level overview of network resources and traffic patterns as feedback to clients, helps them achieve better QoE. Using the flexibility of SDN, combined with SAND features, allows for enhancing the quality of video streaming [6].

In the specification of the SAND architecture, four classes of network elements are defined in the standard [2]:

- *DASH client*: DASH clients with SAND features download segments from the server and sends DASH metrics related to their status to DANEs or a DASH metric server.
- *DANE*: DANEs have knowledge about the DASH format and the characteristics to provide improvement in the perceived quality. A DASH client may contact a DANE before requesting each segment to get information.
- *DASH metric server*: The server is responsible for gathering metrics from DASH clients.
- *Regular network elements*: Network elements which do not have information about DASH characteristics.

Messages passing between the DASH clients and the DANEs might inform DASH-aware elements about buffer level, requested quality, and required bandwidth for the delivery of the packets. As a consequence, this information triggers content caching and provides DANEs with the necessary information for estimating further requests by the clients, and making more useful caching decisions, by using these estimations. On the flip side, DANEs can inform DASH clients to trigger better adaption, by sending information such as alternative segment availability and network throughput by using *Parameters for Enhancing Reception* (PER) messages. DANEs might predict and request further segments from the origin server and pre-fetch them by sending *Parameters for Enhancing Delivery* (PED) messages to the server.

2.2 | Related Works

Edge caching in media delivery technology has a key role in terms of performance. Edge caches which are located distant from clients or in less dense popularity places increase latency and reduce the client's perceived quality, due to the high volume of requests directed to the origin servers. However, in practice, it is hard to bring content as close as possible to all clients, especially when the distribution of the locations of the clients changes rapidly. Virtualized cache (vCache) can be an intelligent and efficient solution comes to solve this challenge. The virtualization of caches can be performed by implementing cache functions as VNFs that run on top of the physical network forwarding elements within *Internet Service Provides* (ISPs).

The performance of certain Internet applications can be improved if the placement of these virtualized caches is determined by considering the requirements of those applications. There are some research efforts in the literature, that provide effective solutions which focus on the placement of the virtualized caches and by adopting NFV-SDN principles [7,8,9,10,11], and [12]. However, improving the performance of video applications, namely QoE, requires the consideration of more application specific parameters, such as the available bandwidth and the delay between the caches and the clients. Therefore, considering the video streaming applications requirements, when placing the virtual caches, is important as it directly affects QoE. Differently from some studies listed here, which have proposed approaches for optimal VNF placement by considering cost minimization, resource allocation, or content placement, in order to provide improved service to the DASH video clients, we focus on utilizing virtualization approaches for supporting video streaming and finding suitable locations for the virtual functions, in particular the placement of vCaches.

On the basis of the co-operation between ISPs and *Over-the-Top* (OTT) services, vCache instances can be deployed in several points in the network which creates a streaming service infrastructure that can deliver video packets to the clients efficiently [13]. There are several works which focus on the placement of virtualized caches for increasing QoE. In [13] and [14], virtual *Content Distribution Network* (vCDN) placement is selected by considering *Virtual Machines* (VMs) storage, RAM and vCPU capacities to increase QoS/QoE. These studies also aim to minimize the migration cost. In our previous work, we devised the term *vDANE* to represent the vCaches that have the knowledge about DASH characteristics [15]. Recently, vCache usage and placement problems in HAS systems has gained attention, and the number of the proposals in this area have been increasing. In [16], an approach was proposed for defining the placement of video streaming VNFs such as cache, transcoder, compressor, and streamer by considering the capacities of VNFs containers. The studies were proposed to use a virtualized proxy servers or caches, that selects which content from which virtualized server for HAS systems in [17,18].

Although there are several studies utilizing vCaches for increasing QoE, they mainly focus on VMs capacities for VNF and/or content placement. Our study mainly differs from these works in the literature, due to its parameters used for the placement of vCaches. These parameters are the available bandwidth between the client and the vCaches, and the density of the clients.

In our previous work[19] and[20], we focused on vCache migration and its effects on different QoE parameters such as average video bitrate and buffer fullness. More specifically, we determined the location of the initial vCache as well as the migration time of vCache in[19], and we addressed the main issues related to vCache migration in[20]. In this current study, we focus on the problem of vCache placement and migration in a more general form by enhancing our previous studies, which only considered installing one cache on the network. The aim of this study is to find the optimal number of caches, locate more than one cache and determine their migration times to satisfy end-users QoE. This study has a dynamic nature and is adaptable to network infrastructure changes, such as traffic and client distribution updates.

3 | VIRTUAL CACHE PLACEMENT AND CACHE MIGRATION APPROACH

As mentioned in the previous sections, DASH clients adapt to network conditions and request different qualities of video based on the changes in the observed network throughput. In order to minimize the total network traffic and efficiently utilize network bandwidth, the hop count between the video source and clients should be minimized. Reducing the distance between clients and the content sources also reduces the total amount of flows that travel across network paths, hence reducing the total network load. If the cache is placed behind a bottleneck link, the performance may deteriorate. To overcome this obstacle, as well as the distance factor, in this current study we take into account the total amount of bandwidth and a connectivity factor, which is the number of egress links attached to the nodes. Thus a node with more resources (e.g, bandwidth and connectivity) and the highest density is an ideal point to locate a vCache.

3.1 | Initial Virtual Cache Placement based on Pressure Score Estimations

This section presents our first algorithm which finds the optimal location for the initial cache placement to reduce the network traffic and provides high QoE for DASH clients where there are no cache instance available on the network. The algorithm, which is called as *PressureInitialCache* (PIC), calculates the pressure score of the candidate nodes and determines one of the nodes for hosting the vDANE instance, considering these pressure scores of the candidate nodes. *PressureInitialCache* was developed by following and modifying the preliminary version of the algorithm which had been proposed in our previous work[15]. In the *PressureInitialCache* algorithm, the pressure score calculation is done by considering distance in terms of hop counts and pressure in terms of client density for each node in the network. The *PressureInitialCache* algorithm, whose details are given in the next section, uses the calculated pressure scores for all candidate nodes. The node with the minimum score is selected as a location for the initial cache in the network as the output of the algorithm.

Assume that, in any network, H represents the number of online clients, d_{ij} is the distance between nodes i and j in terms of hop count of the path between these nodes. In the proposed framework, *node* refers to the network elements that can run virtualized network functions. Consider a virtual cache instance is running on node j. Suppose that the client connected to node i requests video content from the cache. By considering that, the cache sends packets to the clients which are connected to node i, and hence, the video traffic is transmitted from node j to node i over d_{ij} hops.

In order to reduce total network traffic, as well as the response time, the candidate cache node should have minimum distance with clients (d_{ij}) and maximum available bandwidth and connectivity. Generally speaking, the cache location which gives the maximum value of *bandwidth / distance* is optimal when considering the requirements. Suppose L, b_l and L_j represents the number of the links in the network, the bandwidth of the link l and the total number of connected links to the candidate node j, respectively. Here, the connected links are the links which will carry the traffic if the candidate node j is selected as a vCache location. Accordingly, formula (1) is the total network bandwidth and formula (2) is the total bandwidth of the links that directly connected to node j. While B represents the total capacity of the whole network, $B^{'}$ represents the maximum traffic amount that can be transferred by the node j.

$$B = \sum_{l=1}^{L} b_l \tag{1}$$

$$B^{'}(j) = \sum_{l=1}^{L_j} b_l \tag{2}$$

Suppose the online clients are ordered regarding to their distance to the node j in ascending order. By following this, formula (3) refers to the sum of distances between the node j and clients which are more than 1 hop away, where h represents the number of clients connected to j^{th} node through one-hop distance we can conduct to (4).

$$D(j) = \sum_{i=h+1}^{H} d_{ij} \tag{3}$$

$$D^{'}(j) = \sum_{i=1}^{h} d_{ij} = h \tag{4}$$

Locality is an important criterion which has an impact on the performance. A long-distance connection has a negative effect on the QoE due to the RTT delay and causes an increase in traffic volume in the network due to the increased number of the links used for video transmission. By considering these facts, we evaluate the suitability of all candidate locations to deploy the initial vCache by using formula (5). This formula is used for determining the *pressure score*, which is indicated by *P(j)*, for all possible candidate node locations j.

$$P(j) = \frac{\frac{B}{B^{'}(j)}}{\frac{D^{'}(j)}{D(j)}} = \frac{\sum_{l=1}^{L} b_l * \sum_{i=h+1}^{H} d_{ij}}{\sum_{i=1}^{h} d_{ij} * \sum_{l=1}^{L_j} b_l} = \frac{\sum_{l=1}^{L} b_l * \sum_{i=h+1}^{H} d_{ij}}{h * \sum_{l=1}^{L_j} b_l} \tag{5}$$

Since the value $B = \sum_{l=1}^{L} b_l$ represents the total network bandwidth and it is the same for all clients, it affects the pressure score for each candidate at the same level. Therefore, we can remove it from the formula. Hence, formula (5) can be rewritten in the form as given in (6).

$$P(j) = \frac{\sum_{i=h+1}^{H} d_{ij}}{h * \sum_{l=1}^{L_j} b_l} \tag{6}$$

Finally, the number of links connected to the candidate nodes and which carry traffic also are taken into account, in order to increase the reliability. Considering that more connection links provide more reliability, we modified the formula in (6) and obtained the formula in (7).

$$P(j) = \frac{\sum_{i=h+1}^{H} d_{ij}}{L_j * h \sum_{l=1}^{L_j} b_l} \tag{7}$$

3.2 | System Architecture Overview

SDN enables applications such as traffic engineering and load balancing to define forwarding policies that are eventually translated to southbound-specific instructions. In our system architecture, the SDN controller gathers network information from the network nodes (e.g., switches, servers, clients) by using its southbound interface. The controller utilizes some basic network service functions which tracks online clients by collecting information about their location and the time of getting online, determines the forwarding rules, and calculates available bandwidth of the paths in the network.

In addition to these basic network functions, there are three quality modules inside the SDN controller:

- *Cache Placement*: By analyzing the real-time conditions of the network and the performance of the DASH system, it is possible to determine if the current resources (e.g., origin server, or network bandwidth) have enough capacity to keep the video quality at a reasonable level. When the network size or the number of clients increases, additional caches should be installed to meet demand. If it is decided that a new cache instance is going to be initiated, it should be placed in a suitable location such that the decision will cause an increase in QoE and maintain good network utilization, due to there being less traffic in the core network links and better use of network available bandwidth.
- *Cache Migration*: Where there is a lack of resources or a cache is deployed in an inappropriate location, clients will often switch to lower video qualities and more interruptions during display could be experienced. In such a case, the cache should be moved to a suitable location.
- *SAND Messages*: Connecting a client to the appropriate cache improves both network resource utilization and client QoE. When there is more than one cache in the network, this module directs clients to connect to the appropriate caches, with the assistance of the PER messages. As a consequence, the clients will then experience better video quality.

The types and explanations of the messages used in this study, that are sent between SAND elements, are given in Table 1. The *Status messages* are sent by the clients to the DANE. These Status messages carry QoE information such as average received bitrate and buffer level information. DANE collect the Status message information and combines these to create a *PED message*. The DANE then sends these PED messages to the controller after it has collected the relevant Status messages. DANE also adds its own metrics, such as the number of links and clients connected to it into these PED messages. This combined information helps the controller decide whether to run a cache migration algorithm or install an additional cache. PED messages are also exchanged between DANEs and the media server in order to prefetch content. The controller sends PER messages to the clients to direct them to connect another DANE.

TABLE 1 SAND message types used in this study

Message Types	Description
Status messages	Dispatched from DASH clients to DANEs. Update DANE with client's last situation
PED message	Exchanged between DANEs and controller / DANEs in order to enhance delivery
PER message	Dispatched from DANEs to DASH clients. Directs clients to connect to another vDANE.

Our system architecture is illustrated in Figure. 1. In the figure, the SDN modules that were developed for this study, vDANEs and SAND messages are shown. There are two forwarding elements that run vDANE instances. Two clients in the system receive segments from two different qualities.

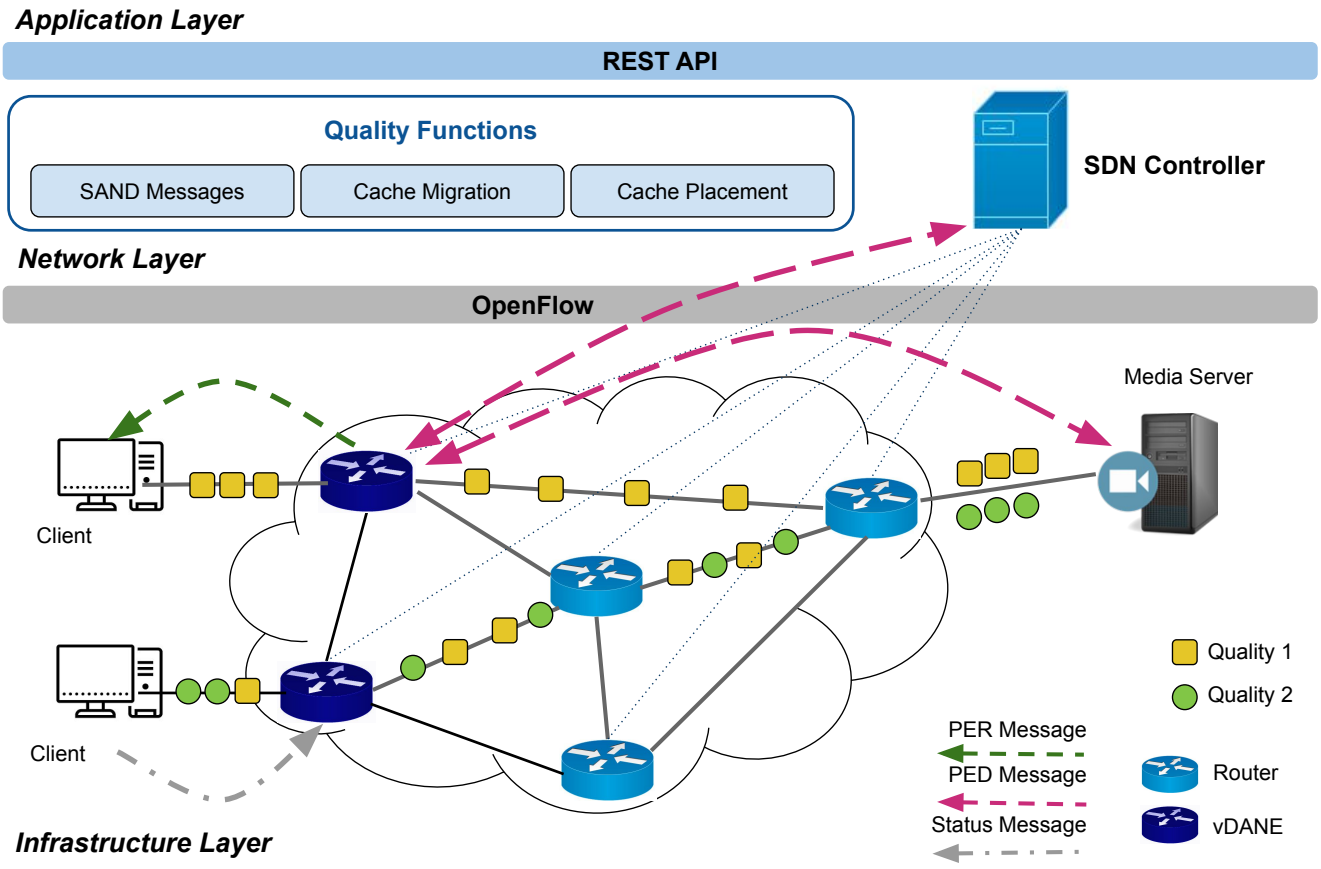


FIGURE 1 Illustration of the SDN enabled video streaming architecture

3.3 | Deploying Initial vDANE Instances

The controller triggers the *Cache Placement* module on-demand for installing the Initial vDANE. The *Cache Placement* module runs the *PressureInitialCache* algorithm given in the *Algorithm* 1. This algorithm is also used for migrating the Initial vCache. Media servers, as well as vDANEs, periodically send network information to the controller, including the average quality of the received video representations requested by the clients. The network controller retrieves this information and determines if a new cache is necessary. This decision is based on a predefined *quality threshold*, which is set by either the video streaming company or the network operator. When the average quality received by the clients, in the form of bitrate or representation, falls under the *quality threshold* the controller can decide to migrate the Initial vCache. An additional cache will be installed if the clients continue experiencing poor QoE. The installation of additional caches will be discussed in Section *6*. As previously mentioned, the *PressureInitialCache* algorithm selects the node which has the minimum pressure score, and install vCache if it is the first instance or migrates the vCache if there is a vCache in the network. This score is calculated for all potential switches that can be selected as hosting a vDANE instance. *P(j)* with the minimum score is considered to be the best location for the Initial vDANE deployment. The main objective of the algorithm is to find an optimal location for vDANE placement, by considering the number of hops and the available bandwidth between the potential location and the clients, as well as the density of the clients. The location with more connected links and more available bandwidth, but a lower hop count from the clients, has a higher chance to be selected.

In this current study, the *PressureInitialCache* algorithm only selects one switch for the Initial vDANE placement. After determining an appropriate node as a location for deploying a vDANE instance (e.g, a switch or router), the controller creates a new vDANE instance connected to the specific node. Then it forwards the client's requests to the deployed vDANE, via the shortest path. The *PressureInitialCache* is also used for deciding the migration of the Initial vCache.

3.4 | Cache Migration

The network controller continuously monitors network conditions and gathers statistical parameters related to the DASH client's traffic and to the cross-traffic pattern from underlying infrastructure elements. It was shown that network monitoring can be achieved more efficiently by using *OpenFlow* statistics[21] , compared to legacy networks. For example, instead of probing the end-to-end path, the SDN controller can keep track of the available bandwidth on each link and can estimate the end-to-end available bandwidth[22]. Any changes in the environment that affects the current configuration should be reported to the controller, and the controller consequently triggers appropriate functions.

In the case of any changes to the network conditions, such as changes in traffic pattern or even exceeding the *quality threshold*, the controller again runs the *PressureInitialCache* algorithm, given in *Algorithm* 1, for estimating a new cache location, based on the updated data. If the output of the algorithm determines a new location for a vDANE, the controller virtually migrates the vDANE to the new place and then installs it. The controller then leverages SAND technology and sends PER messages in order

Algorithm 1: *PressureInitialCache* Algorithm

```
Input: Set of switches
       Distance (hop count) of each connected client to each switch
begin
    foreach switch j do
        Compute Pressure Score (j)
    end
    Select switch with minimum score
    if there is no any vCache in network then
        Install a vCache in selected switch
    else
        Migrate vCache to selected switch
    end
end
```

to force the clients to connect to the new vDANE. After migration, the controller continues to track clients' condition. If the clients still experience low quality, which is under the *quality threshold*, the controller decides to install an additional cache.

4 | TESTING ENVIRONMENT

To measure the performance of the proposed approach and to compare it with different approaches, we implemented some simulations using the *Mininet* emulator. The agility of *Mininet* provides an easy way to prototype and evaluate SDN protocols and applications. In the context of a controller, the *FloodLight*[23] controller with the assistance of *OpenFlow* as a southbound interface is used during the simulations. The combination of *Mininet* and an *OpenFlow* switch in a virtualized container provides exactly the same semantics of software-based $OpenFlow$ switches [24].

Three different network topologies were applied to evaluate the performance of the *PressureInitialCache* algorithm, as shown in Table. 2. These include two topologies from the Internet Topology Zoo[25], namely *Compuserve* and *BellCanada*, and one *Custom* topology, which is shown in Figure.2. A $Poisson$ distribution with different mean values (λ=20 Mbps, 25 Mbps, 30 Mbps) are used for generating the network links bandwidths. Also, the number of the DASH clients are set to 30, 40, and 50 for *Custom*, *Compuserve*, and *BellCanada* topologies, respectively.

We compared the proposed algorithm with two different algorithms. The first comparison study is against the *Best effort* algorithm, in which a *vDANE* is placed in the location where forwarding devices carry more network traffic. This approach is discussed in[26] where caches are installed in high traffic switches. For the second comparison, the algorithm used is *HotSpot*, which aims to find the optimum location by taking advantage of the locality, and places the cache in the highest density location. *HotSpot* introduces the function: $F(N_i) = a^3 + b^2 + c$ to find optimal places, where a, b and c refer to the number of clients with hop count one, two, and three distance from each node N_i[19]. For transferring packets between each client and vDANEs, the shortest path algorithm is used in all approaches. Note that, it is also possible to use different routing algorithms thanks to SDN, but we prefer to use the same routing algorithm for all approaches in order to be able to provide performance results.

The video *Big Buck Bunny*[27], whose information is given in Table 3, is used for streaming during the simulation. There are six representations of this video, while the first representation (R1) has the lowest quality, the last representation (R6) has the highest quality. Each representation contains 299 video segments with equal length of 2 seconds video, producing 598 seconds of video in total. To achieve better video quality, clients should request and receive as many segments from higher bitrate representations as possible, while reducing the number of video stalls.

The attachment points of the clients to the network are randomly distributed, with the same distribution being used in all algorithms. The mean of the client's interval time is set to 1 second. During simulation, some clients leave the network and new clients join the network. Hence, the simulation scenario has a dynamic nature. DASH clients join the network based on a *Poisson* distribution, and they start requesting suitable representations after downloading the *Media Presentation Description*

TABLE 2 Network topologies

Network topologies	Network size	# Nodes	# Links	# Clients	Average available bandwidth (Mbps)
Custom	Small	7	9	30	λ=20, λ=25, λ=30
Compuserve	Medium	11	14	40	λ=20, λ=25, λ=30
BellCanada	Large	43	58	50	λ=20, λ=25, λ=30

TABLE 3 Big Buck Bunny representations and bitrates

Representation Synonym	1 #R1	2 #R2	3 #R3	4 #R4	5 #R5	6 #R6
Bitrate (Kbps)	2133	2484	3078	3526	3840	4219

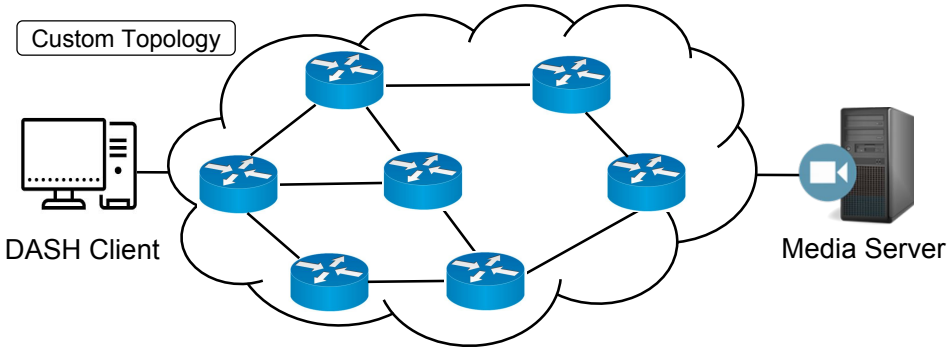


FIGURE 2 Custom network topology

(MPD) file. In this study, we used a throughput based rate adaptation algorithm on the client side, where the clients adapt the bitrate by considering only the measured available bandwidth. Each simulation is repeated 10 times and the average values are presented in the graphs and tables.

5 | INITIAL VCACHE PLACEMENT EVALUATION

For the Initial vCache placement evaluation, we implemented and evaluated the proposed approach in the simulation scenarios by using three different network topologies, each with three different configurations as explained in the previous section. Tables 4, 5 and 6 list averaged video quality parameters observed on the clients' side obtained from the simulations in the *Custom*, *Compuserve* and *BellCanada* topologies. In the tables, the *received video quality* is a measure of the bitrate of the video received on the client's side, the *startup delay* indicates the latency of starting to display video after the client triggers a request to play, and *re-buffering duration* defines the video re-buffering or freezing time during display, which has a negative impact in end-user QoE.

TABLE 4 Simulation results, Custom

Algorithm	λ=20	λ=25	λ=30
Best Effort	2513	2774	3078
HotSpot	2761	3057	3220
PressureInitialCache	2906	3160	3165

(a) *Average received video quality (Kbps)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	10	10	12
HotSpot	11	9	8
PressureInitialCache	12	13	13

(b) *Average startup delay (sec)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	121	84	12
HotSpot	205	43	14
PressureInitialCache	109	41	14

(c) *Average re-buffering duration (sec)*

TABLE 5 Simulation results, Compuserve

Algorithm	λ=20	λ=25	λ=30
Best Effort	2446	2852	3196
HotSpot	2563	2877	3083
PressureInitialCache	2471	2888	3232

(a) *Average received video quality (Kbps)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	17	16	12
HotSpot	16	14	13
PressureInitialCache	24	17	16

(b) *Average startup delay (sec)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	140	55	26
HotSpot	256	78	68
PressureInitialCache	168	112	23

(c) *Average re-buffering duration (sec)*

TABLE 6 Simulation results, BellCanada

Algorithm	λ=20	λ=25	λ=30
Best Effort	2321	2373	2397
HotSpot	2568	2701	2955
PressureInitialCache	2859	3056	3185

(a) *Average received video quality (Kbps)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	24	20	18
HotSpot	17	14	13
PressureInitialCache	24	22	20

(b) *Average startup delay (sec)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	733	683	585
HotSpot	356	235	100
PressureInitialCache	197	144	97

(c) *Average re-buffering duration (sec)*

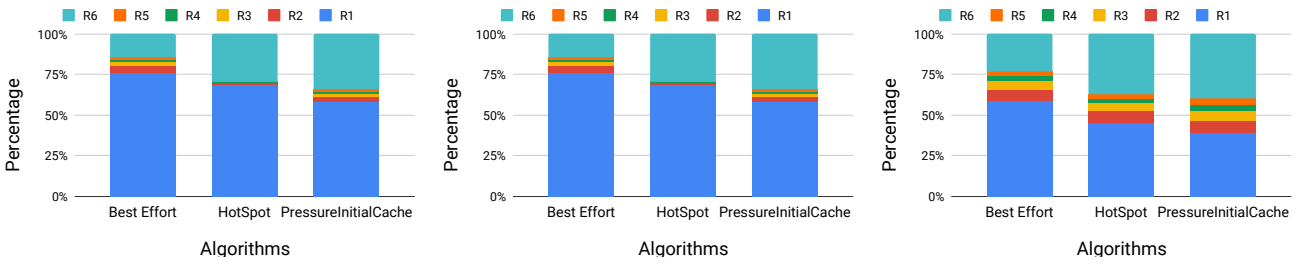
It is seen from the tables that *Best effort* has the lowest received bitrate, while *HotSpot* and *PressureInitialCache* achieve better results. The reason for that is that in the *Best effort* algorithm, cache positions are in the center of network based on the assumption of the minimum distance between clients and caches. Deploying caches in the center of the network without considering client distribution may lead to inefficient performance, especially when caches are placed behind bottleneck links. On the contrary, the *HotSpot* algorithm benefits from the locality and hence it has higher received bitrate comparing with *Best effort*, but it is still lower than the *PressureInitialCache* algorithm.

During simulation, both network traffic and client distribution changes. Since *HotSpot* and *Best effort* algorithms have a static nature, they do not change the cache location. However, the *PressureInitialCache* algorithm has a dynamic nature that migrates caches into the appropriate location when either network traffic or user distribution are changed. Since the process of migration is related to the changes in traffic patterns or client distribution, it is expected to observe re-buffering in some case (e.g., the scenarios where $lambda$ equals 20 and 25 in *Compuserve* and *BellCanada* topology) takes a little more time during migration.

It is worth emphasizing that the bandwidths of the links were very tight in these simulations, thus we expect startup delay is higher than usual in all scenarios. The startup delay also increases when the network size becomes larger. In all cases, increasing network bandwidth provides higher performance. As we expected, our proposed algorithm outperforms other algorithms in all network topologies and bandwidth settings.

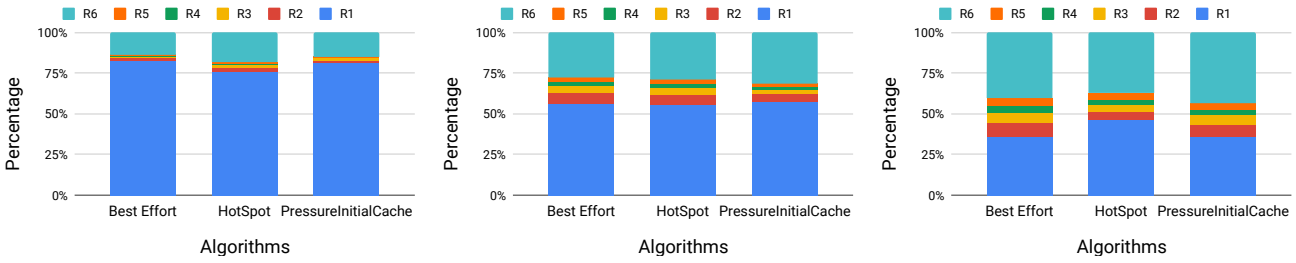
In our simulation, the clients start to display video when they have 8 seconds (or 4 segments) of buffered video. Buffering on the client-side helps to avoid re-buffering periods. The increasing startup delay may reduce re-buffering duration, but it causes the users to wait more for starting the video. As shown in the tables, the initial startup time increases with increasing network size and the number of the clients. In the *Custom* and *CompuServe* topologies, *HotSpot* shows the worse performance in terms of the re-buffering durations. The reason is that even though *HotSpot* considers the client location distribution, if the cache is behind a bottlenecked link, this causes lengthened segment download times and hence higher re-buffering duration values. Startup values in the *PressureInitialCache* algorithm are slightly higher than the other algorithms. This is due to the newly joined clients during the cache migration process. In that case, the clients starts to receive video segments after migration is completed and this process causes the starting delay to be elongated. For the clients that are already online, the client cannot make more requests and download video during the hand-off period, and so this speeds up buffer drain and may result in the buffer becoming empty, as discussed previously. We addressed this problem in *Section 6*.

Another important QoE parameter is the distribution of the received representations. Receiving more video segments from higher bitrate representations means that clients play the video with higher quality. Figure. 3, Figure. 4, and Figure. 5 show the percentage of the received video segments from each representation in our experiment. The first representation ($R1$) indicates the lowest quality, the last one ($R6$) represents the highest video quality. The observed values indicate that clients using the *PressureInitialCache* algorithm received more segments from the highest representation, which means video is displayed with a higher quality. When considering different network topology and bandwidth values (λ= 20, 25, and 30 Mbps), we can conclude that the proposed algorithm shows better performance especially in large networks and under limited bandwidth values in smaller sized networks as it can be seen in Table 4 for the values when λ is 20 Mbps. If the bandwidth is plentiful and network size is small (e.g.,*Custom* network) where the clients and content are close to each other, the performance of the proposed algorithm is similar to the other approaches.



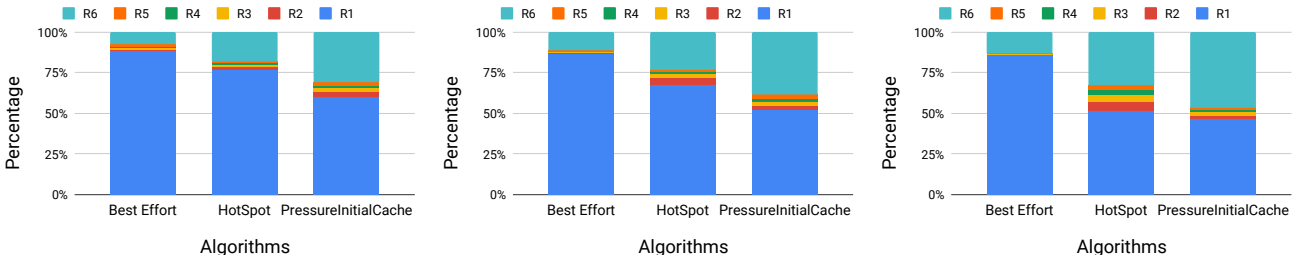
(a) λ=20 Mbps **(b)** λ=25 Mbps **(c)** λ=30 Mbps

FIGURE 3 Initial Cache. Distribution of the received Quality level – Custom



(a) λ=20 Mbps **(b)** λ=25 Mbps **(c)** λ=30 Mbps

FIGURE 4 Initial Cache. Distribution of the received Quality level – Compuserve



(a) λ=20 Mbps **(b)** λ=25 Mbps **(c)** λ=30 Mbps

FIGURE 5 Initial Cache. Distribution of the received Quality level – BellCanada

A further valuable factor to illustrate the effectiveness of our proposed algorithm is long-term received video quality. Figure. 6 illustrates the received representation bitrate values during the whole video streaming session in which 299 segments are transferred to the clients. These results show the received video quality as a fraction of segment numbers. Without a doubt, receiving high quality video with more re-buffering and interrupt during display time never provides better QoE. Therefore, receiving more segments from high quality representations without considering re-buffering duration does not make sense. By considering this fact, the observed re-buffering results given in Table 4c, Table 5c, and Table 6c we see that the proposed algorithm achieves better performance. This means that clients display video with better quality when leveraging the *PressureInitialCache* algorithm. It is clear that by increasing the $Lambda$ value, clients have more resources to receive high bitrate video segments, and therefore when $Lambda = 30$ all algorithms have good performance, but again *PressureInitialCache* has better performance when considering the number of high quality video representations as well as re-buffering duration values. Similar results were obtained with the other topologies and bandwidths values but are not reported here due to space limitation. In the *Compuserve* topology, when the $Lambda$ value is set to 20 Mbps, the *HotSpot* algorithm outperforms the other approaches. This shows that the concept of locality, with a consideration of clients' density and deploying cache as close as possible to clients, has a positive effect on the received video quality. Although the *PressureInitialCache* algorithm considers locality, the hand-off has a negative impact on it's performance as during the hand-off process, clients request video segments from lower bitrate in order to avoid re-buffering.

In addition to measuring the various QoE parameters, we also measured the overall QoE values. Received video quality and re-buffering have more impact than startup delay among QoE parameters. If a client waits a bit longer to start the video, instead of fast starting, then high re-buffering can be prevented. There are two other QoE metrics including: the number of re-buffering events and the number of quality oscillations. Quality oscillation refers to the switching between different representations during video display time, which has a negative impact on QoE. We calculate the normalized QoE, while taking into account these factors.

We used formula (8) to calculate the overall QoE given in [28], [29]. The formula calculates the QoE value at the client, where there are K segments downloaded. The first term in the formula is the bitrate of the segments and the second term is the number of quality oscillations in the QoE formula. The second term shows how much the quality was changed from $segment_i$ to $segment_{i+1}$ and the sum of them for K segments. D_r, N_r, and T_s refer to the total re-buffering duration, the total number of re-buffering events, and startup delay time, respectively. The QoE values are calculated for all network topologies, for each of the three different bandwidths. The calculated QoE values are normalized and illustrated in Figure. 7. The best normalized QoE value for any network conditions is 1. The results show how close our proposed solution is to the best value. Overall, in the small and medium-size network topologies (e.g., Custom, and Compuserve topologies), the central switches have more connectivity links which provide more bandwidth. Therefore caching content at the center of networks results in better performance. However, in a large network (e.g., BellCanada topology) caching content at the edge points leads to better results, because clients are generally distributed at these edge points. Simulation results prove the efficiency of our algorithm for the initial cache placement. In the next section, we will discuss installing extra caches based on requirements.

$$QoE = \sum_{i=1}^{K} q(R_i) - \delta * \sum_{i=1}^{K-1} |q(R_{i+1}) - q(R_i)| - \mu * D_r - \mu * N_r - \beta * T_s. \tag{8}$$

(a) λ=20 Mbps

(b) λ=25 Mbps

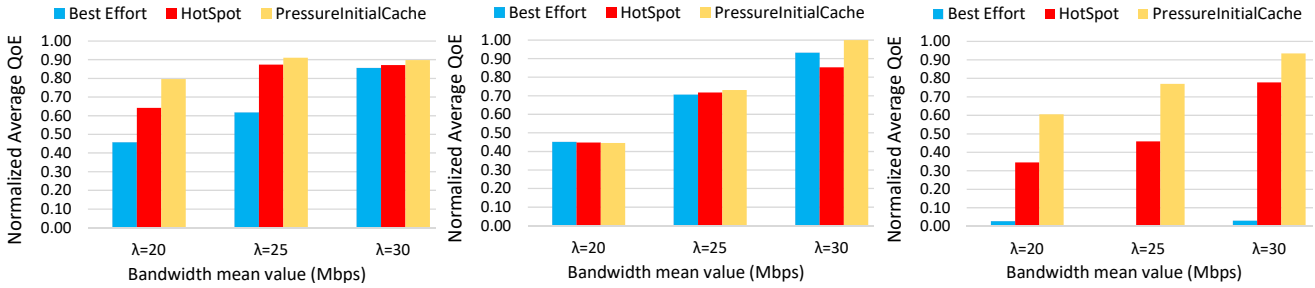
(c) λ=30 Mbps

FIGURE 6 Initial Cache. Received bitrate per segment – BellCanada

6 | EXTENDED CACHE ENHANCEMENT STRATEGY

6.1 | Installing Additional Caches

In large networks, one vCache may not provide a good service level for clients to get the best possible video quality experience. This can be observed in the performance results given in the previous section, where the results shows the performance with one vCache. If the network size is large, or the number of clients is high, one or more additional caches should be installed to meet QoE requirements [15]. During a lack of enough resources, clients experience lower video quality and more interruption during display.



(a) Custom **(b)** Compuserve **(c)** BellCanada

FIGURE 7 Initial Cache. Normalized average QoE values were calculated for all topologies and bandwidths.

When new clients join the network, network traffic patterns change, and the SDN controller triggers the *PressureInitialCache* algorithm in order to calculate the initial vCache location as explained in previous sections. After determining the location and installing the Initial vCache, the controller monitors the clients' received video parameters and can decide to install a new vCache if clients experience poor video quality or received video quality is under a predefined threshold. If the controller installs a new additional vCache in a new location, some flows will be removed from the first vCache and forwarded to the new one. Typically, the flows of the clients are rerouted towards to the closest vCache. In addition, to prevent congestion, the SDN controller can signal the clients to connect another appropriate vCache with the assistance of the SAND components.

To find a suitable location for installing further vCaches, we introduce the *PressureCache* algorithm. In order to find an optimal point for the trade-off between distance and bandwidth, we introduce a formula (9) which measures the distance between (i) each client and the first one, and (ii) each client and the new candidate one. The formula also considers the available bandwidth of the links connected to the candidate node. The candidate vCache should provide more bandwidth and should be closer to the connected clients. In this formula, $P_{j'}$ refers to the pressure score of the second vCache position, and the $P_{j'}$ with the maximum score is preferred. Distance, in terms of hop count, between client i with the first and candidate vCaches are represented by d_{ij} and $d_{ij'}$ respectively. Let $L_{j'}$ indicate the total number of connected links to the candidate node j', which will carry the video traffic, which will likely be known in advance since SDN controller manages the flow routes. Suppose b_l is the bandwidth of each connected link. The *PressureCache* algorithm uses the *formula* given in (9), and in the formula, H and h refer to the (i) total number of clients and (ii) the number of clients that are connected to the candidate node with one hop respectively. If the number of clients increases and another additional vCache installation is required, the nearest vCache is considered as the Initial vCache and the same formula is used to install the next vCache, based on this assumption.

$$P(j^{'}) = \sum_{i=h+1}^{H} (d_{ij} - d_{ij'}) * \sum_{l=1}^{L_{j'}} b_l \tag{9}$$

Algorithm 2 shows the *PressureCache* algorithm utilizing formula (9). The SDN controller monitors network conditions as well as analyses incoming status messages from the clients and messages from the DANEs periodically. Based on this information, the SDN controller decides whether or not to install a new vCache if the network traffic changes or, for instance, if clients still experience lower bitrate video even if the vCache is migrated considering current network conditions. If a new cache installation is required, the controller runs the *PressureCache* algorithm. In the algorithm, the *PressureCache* scores for each switch (line 2-5) are computed, for other than the switch which currently has a vCache installed. The switch with the highest score is selected for hosting the new cache.

If the controller decides to install a new cache, based on the output of *PressureCache* algorithm, after installing the new cache, it directs the clients to connect to the nearest vCache. This process starts by sending PER message to the client and this forces the client to connect to another vCache in order to enhance the client's received quality as well as to manage network traffic. During this period, (disconnection from one vCache and connection to another vCache), which is called *hand-off* time, the client does not receive any video segments from the vCache and if due to a lack of video segments in the buffer, it may experience re-buffering and interruption of video displaying.

Algorithm 2: *PressureCache* Algorithm

Input: N is the total number of switches with no vCache installed on them
Distance (hop count) of each connected client to each switch

```
begin
    foreach N switch do
        Compute PressureCache Score (j')
    end
    Select switch with maximum score
    Install a new vCache in selected switch
end
```

Another important issue is how fast the cache placement or cache migration process in done. In large scale networks it is difficult to take a full network snapshot showing the conditions perfectly while network traffic patterns frequently change. However, in the SDN domain, network monitoring can be achieved by using *OpenFlow* statistics. In addition, the clients periodically send their buffer fullness factor and average received bitrate values to the SDN controller. Suppose T denotes video segment duration, which is equal to 2 seconds in the *Big Buck Bunny* video, and the clients' buffer size is equal to 24 seconds of video, and that video is displayed when a client has at least 8 seconds of buffered video. Typically, in order to provide fast startup, the clients request the first 4 segments from the lower representations (R1) which have the lowest bitrate. The clients send average received bitrate values every 10 seconds -just higher than $4 * T$ seconds to the controller during simulation. The network controller checks all the online clients' average bitrate values and runs the *PressureCache* algorithm if the total average bitrate values is under the threshold level. This threshold value would be defined by the content provider. In our study, we consider that each client should be able to display video with a minimum quality without unacceptable re-buffering. If the network pattern or the client received quality is stable, the controller waits for the next 10 seconds to evaluate the clients' status messages.

In order to show the effects of the increasing number of online clients which causes the installation of an additional cache, we also implemented simulations with a varying set of online clients as shown in the Table 7. Compared with Table 2, although the number of nodes and links are the same, the number of online clients has been increased to 35, 50, and 100 for *Custom*, *Compuserve*, and *BellCanada* topologies respectively. This causes the controller to install additional cache by running the *PressureCache* algorithm instead of *PressureInitialCache*, which is used only for installing *Initial* or first cache.

To obtain fairly comparable results, we also deploy the additional caches for *HotSpot* and *Best effort* approaches. For the *HotSpot* approach, an additional cache is installed in the second highest dense location and in the *Best effort* approach, an additional cache is installed at the place where more network traffic passes through that point, which is generally near to the center of the network. In order to present a fair comparison, the additional caches are installed at the same time for all approaches. Again, each topology has been tested with three different bandwidth settings produced by a Poisson distribution having different average values (λ=20 Mbps, λ=25 Mbps, and λ=30 Mbps). While the number of online clients increased in each topology, additional vCaches need to be installed in a new location. A requirement for additional vCaches is more evident in the $BellCanada$ topology where the number of clients has increased 100%.

In practice, it is expected that the proposed algorithm significantly outperforms the two other algorithms in terms of QoE parameters. Results of the received video quality, the startup delay, and the re-buffering duration are shown in Tables 8, 9, and 10. In all cases, the received video quality, the startup delay, and the re-buffering duration tend to improve as the value

TABLE 7 Network topologies

Network topologies	Network size	# Nodes	# Links	# Clients	Average available bandwidth (Mbps)
Custom	Small	7	9	35	λ=20, λ=25, λ=30
Compuserve	Medium	11	14	50	λ=20, λ=25, λ=30
BellCanada	Large	43	58	100	λ=20, λ=25, λ=30

of λ increases. In other words, when the link bandwidths are increased, DASH clients experience better video quality. The re-buffering values are the lowest in *HotSpot* approach when λ equals to 25 Mbps in *CompuServe* topology. The reason for that is because *Hotspot* takes into account clients which have a maximum of three hops distance, and in the *Compuserve* topology the clients generally connect to the caches with maximum of 3 hops. If the bandwidth of the links used for transmission is high, the caches can allow clients to obtain a higher QoE as can be seen from these results. However, the performance gain in *HotSpot* is not observed for each case. The re-buffering values are similar for *HotSpot* and *Best effort* approaches and higher than the proposed approach where λ=30 Mbps as it can be seen in Table 9.

In the context of startup delay, it is seen that the proposed algorithm has higher values compared to the other two approaches in the small size networks (e.g., *Custom topology*). The higher values in startup delays are caused by the clients connecting to the network during hand-off. Since the *Custom* topology is a relatively small topology, the distance between clients and vCaches are limited and similar for all approaches, which does not provide any advantages to the proposed approach obtaining less startup

TABLE 8 Simulation results, Custom

Algorithm	λ=20	λ=25	λ=30
Best Effort	2791	2908	3333
HotSpot	3241	3393	3512
PressureCache	3385	3409	3547

(a) *Average received video quality (Kbps)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	10.4	12	8.6
HotSpot	10.3	9	8.1
PressureCache	13	10.3	11.9

(b) *Average startup delay (sec)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	64	45	10
HotSpot	80	45	4
PressureCache	27	23	9

(c) *Average re-buffering duration (sec)*

TABLE 9 Simulation results, Compuserve

Algorithm	λ=20	λ=25	λ=30
Best Effort	2603	2935	3067
HotSpot	3093	3157	3291
PressureCache	2967	3326	3639

(a) *Average received video quality (Kbps)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	16.7	14.6	13.4
HotSpot	14.1	14	13
PressureCache	17.4	17	15

(b) *Average startup delay (sec)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	194	109	52
HotSpot	110	37	55
PressureCache	97	48	17

(c) *Average re-buffering duration (sec)*

TABLE 10 Simulation results, BellCanada

Algorithm	λ=20	λ=25	λ=30
Best Effort	2694	2837	2810
HotSpot	3084	3416	3487
PressureCache	3152	3408	3572

(a) *Average received video quality (Kbps)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	14.7	18.5	14.7
HotSpot	14.6	15	14.5
PressureCache	14.5	15	14.5

(b) *Average startup delay (sec)*

Algorithm	λ=20	λ=25	λ=30
Best Effort	498	404	312
HotSpot	55	30	27
PressureCache	82	27	23

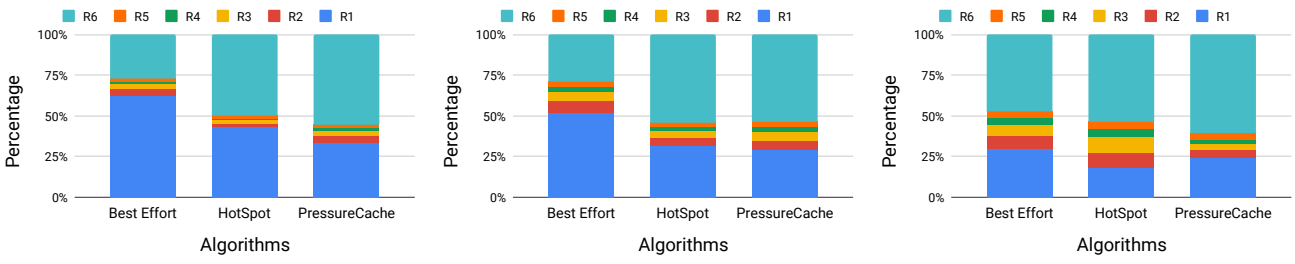
(c) *Average re-buffering duration (sec)*

delay compared to other approaches. However, in the *BellCanada* topology, which is the biggest one in terms of size and number of online clients, startup delay values in the proposed algorithm are same as *HotSpot* and better than *Best effort* approach. This is because with the proposed algorithm is deploying vCaches by considering distance causing the clients to connect in a small time, which eliminates the negative effect of hand-off process on startup delays.

Similar to the first test results given in section 5, the proposed approach obtains better performance in the second test set as well, where we install additional vCaches. Figure. 8, Figure. 9, and Figure. 10 represent the number of received segments from each representation. We found that installing vCaches in a random location, or even in the center of network topology, never leads to better results. As observed, the performance of the *Best effort* approach is low in all topologies. On the contrary, relying on locality is the main reason that *HotSpot* and *PressureCache* tend to better results. Also, the dynamic nature of the *PressureCache* algorithm gives advantages to clients to download more segments from higher representations.

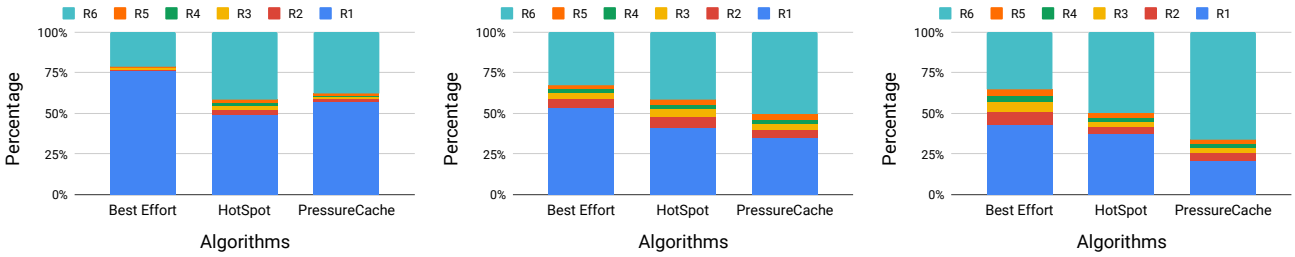
The emergence of deploying vCache in the network is more evident when the size of the network increases. It is because, in large-scale networks, either packet propagation or intermediate forwarding device delay has a negative impact on packet delivery. Network delay causes re-buffering on the client-side as well as having a negative impact on the network throughput. Therefore, in both buffer base or throughput base adaptation techniques, the client adapts to lower representation in order to avoid re-buffering. Hence, the clients receive video segments from lower representations.

The bitrate of the downloaded video segments by the clients over time, for the *Custom* topology is in Figure. 11. In all algorithms, at the beginning of the streaming, the clients request video from the lowest representation in order to fill the buffer and to experience a fast startup. Then they adaptively download video from the highest representation possible. As a result,



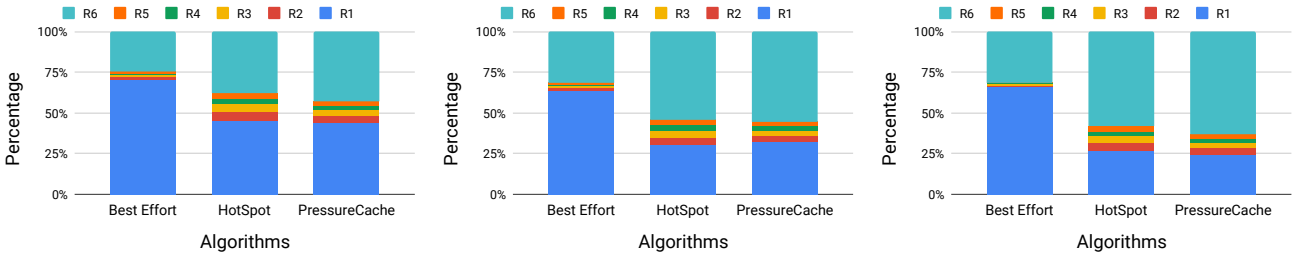
(a) λ=20 Mbps **(b)** λ=25 Mbps **(c)** λ=30 Mbps

FIGURE 8 Additional Cache. Distribution of the received Quality level – Custom Topology



(a) λ=20 Mbps **(b)** λ=25 Mbps **(c)** λ=30 Mbps

FIGURE 9 Additional Cache. Distribution of the received Quality level – Compuserve Topology



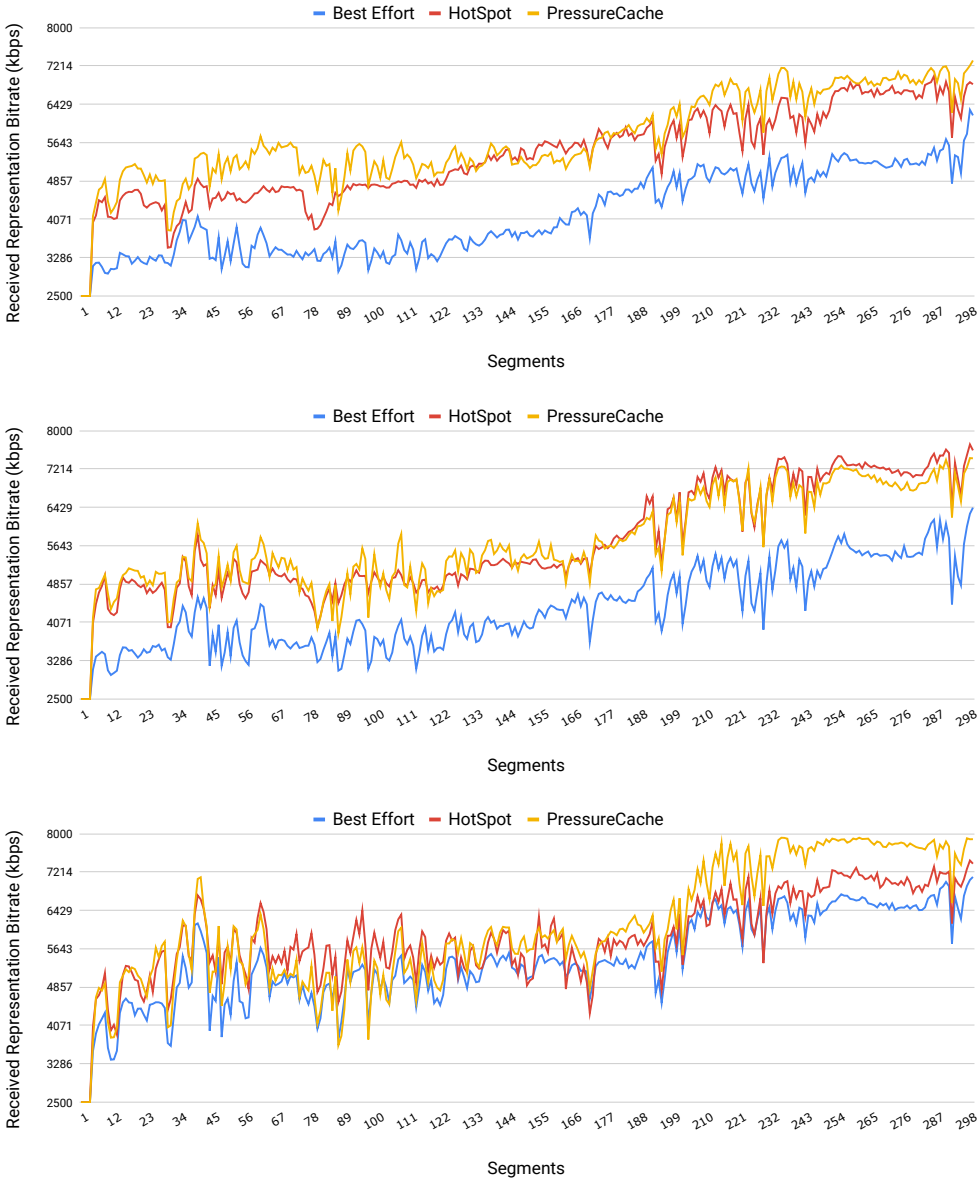
(a) λ=20 Mbps **(b)** λ=25 Mbps **(c)** λ=30 Mbps

FIGURE 10 Additional Cache. Distribution of the received Quality level – BellCanada Topology

the *PressureCache* algorithm obtains better performance in the long time. In the middle of the simulation, new clients join the network and hence change the traffic patterns. An increasing number of online clients causes the installation of new vCaches in suitable locations. As a result, in all topologies and different setup configurations with different bandwidth values, the quality of the received representation increases because more vCaches provide more resources in terms of bandwidth and reduce the distance in terms of hop counts in the proposed approach. Thus, generally in all algorithms and specifically in the proposed algorithm, clients adapt to higher representations. The reason for better adaption in the *PressureCache* algorithm mainly depends on (i) vCache placement and migration, and (ii) the mechanism of forwarding clients to the vCaches which are more appropriate to connect due to their available bandwidth.

When more than one vCache is available in the network, a client will connect to the closest cache, but sometimes, pressure on one vCache increases or cross-traffic causes paths to blocks. In such a condition, clients in both *Best effort* and *HotSpot* algorithms still continue to request video from the nearest vCaches. However, in the *PressureCache* algorithm, the controller leverages SAND technology in order to improve both client and network performance. By utilizing the SAND architecture, the controller sends PER messages to a set of the clients and forces them to request a video from specific vCache even it is located in the far distance. When clients receive *Enforcement* PER messages, they start the hand-off process and connect to the new vCache.

Figure. 12 illustrates a single sample of the requested video segments during the first 240 seconds (which is 120 segments of 2-second samples) of the streaming session obtained from a client with the proposed algorithm in the *BellCanada* topology. Recall from Table. 3, that the first representation (R1 $\rightarrow$ 2.1 Mbps) has the lowest bitrate. At the start of streaming, clients request the video from the first representation in order to fill the buffer immediately. After that, the client adapts to a suitable representation based on throughput estimation. When the client requests the 30^{th} segment (in the 60^{th} second), an increasing number of online clients or cross-traffic affects client adaptation. Hence, received video bitrate on clients' side decreases. Moving a vCache into a new location or installing additional vCaches in the appropriate location improves clients' resource availability in terms of bandwidth and distance and clients starting to request and receives video from high bitrate representations. After the 60^{th} segment, we see that the client is requesting video from the highest possible representation. By increasing the number of clients, the controller runs additional vCache algorithm and deploys new vCaches in the appropriate location. Finally, the



(a) λ=20 Mbps

(b) λ=25 Mbps

(c) λ=30 Mbps

FIGURE 11 Additional Cache. Received bitrate per segment – Custom

controller updates the forwarding device's routing tables and forwards the client requests to the suitable vCache. The effect of installing additional vCaches is clearly visible after passing segments 80 (or 160 seconds).

For the normalized QoE metrics, we followed the same overall QoE formula, given in (8), to observe the overall result of the proposed algorithm compared with two other algorithms. As seen in Figure. 13, *PressureCache* achieved better results in all network topologies and bandwidths. With the increasing number of connected users to the network, installing caches at a suitable location while considering the distribution of end-users has an effective impact on traffic flows, and consequently provides better services to clients. The results show that the *Best effort* algorithm, by installing cache at the center of the network, results in poor video quality, however, *HotSpot* and *PressureCache* achieved the best QoE by installing cache at the edge points where clients connected to the network.

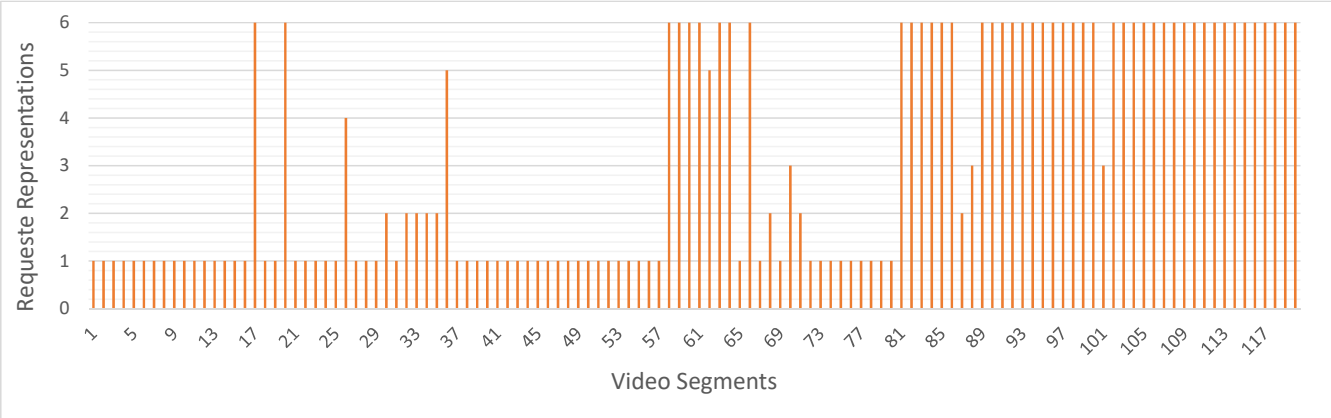
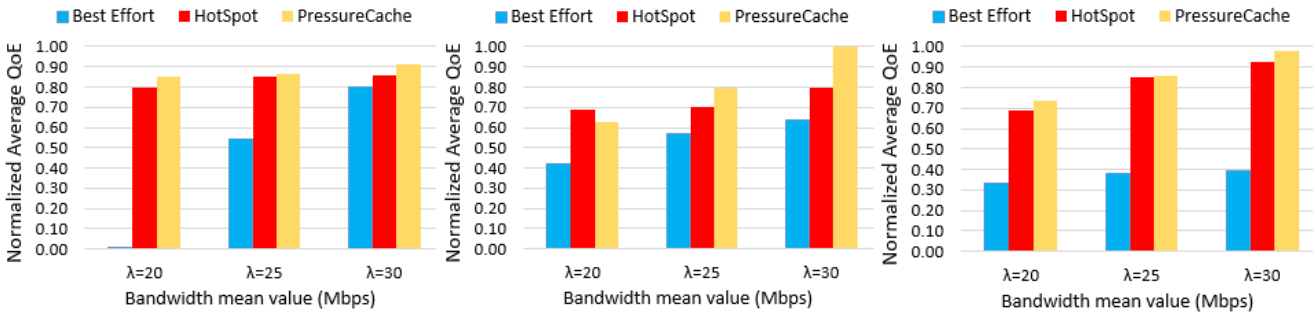


FIGURE 12 Requested representations per segments – BellCanada topology



(a) Custom **(b)** Compuserve **(c)** BellCanada

FIGURE 13 Additional Cache. Normalized average QoE values were calculated for all topologies and bandwidths.

6.2 | Cache Migration and TCP Hand-off

TCP hand-off is a set of actions that happen when a client connection is removed from one interface and connected to another. TCP hand-off occurs when a client disconnects from a cache instance and connects to another one. During this short period of time the clients stop sending and receiving data. The main reasons for this are traffic redirection, load balancing, and roaming. A hand-off also appears in CDN networks, where CDN architectures leverage a redirect mechanism to initiate a connection between a client and the most appropriate server[30].

In order to minimize the negative effects on re-buffering duration, we improve the migration process by introducing a new migration algorithm. When the controller decides to migrate a vCache, the controller also considers the buffer fullness values of the clients. According to these values, the first group of clients that are directed to the new vCache are those clients having the most buffer fullness values. A consequence of disconnecting some of the clients from that cache is that the clients still connected to the cache can increase their buffer fullness values due to the enhanced volume of resources. As the last step, the controller directs the last group of clients to the new vCache and hence, the hand-off operation is completed without causing those clients to drain their buffer. Details of this *cache migration algorithm* can be found in[20].

The quality metric values obtained from these simulations indicate that the developed migration algorithm improves the received video bitrate by 2% and decreases the re-buffering duration by 7% when compared with *PressureInitialCache*. This shows that it is possible to obtain an improvement in QoE when the clients' buffer fullness values are also being considered during hand-off.

7 | CONCLUSIONS

This study addresses the problem of finding ideal locations for the installation of virtualized caches – the vDANEs, in order to provide better video services for end users. To do this we introduced two different algorithms: *PressureInitialCache* for finding the location of the first cache deployment, and *PressureCache*, for finding additional cache locations. We evaluated those algorithms with three different network topologies, each with three different bandwidths and different clients distributions. We implemented the experiments using the *Mininet* test-bed and benefited from the advantages of using *Floodlight* as an SDN controller and *MPEG-SAND* technology and presented their comparative performance.

It has been shown that having information and a global view of the network can allow clients to make better adaptations, and such a network-assisted approach has been presented in this study. The nominated algorithms, which leverage SAND technology and the NFV-SDN architecture, have provided improvements in the QoE metrics observed on the clients' side. Experimental results have shown that this improvement can be provided by installing a virtual cache near to an optimal location, and by handling the migration of caches based on both client distribution and network traffic patterns. By using an algorithm to adapt the network resources according to the changing network and clients' conditions, the presented approach provides the clients with the ability to obtain higher QoE. Evaluating a location for installing a new vDANE vCache requires network monitoring and real-time statistic data exchange between the vDANEs and the SDN controllers, by utilizing the PED messages defined in SAND. By installing a vDANE in an appropriate location, and then redirecting some clients to a suitable vDANE, we observe a decrease in network and origin server traffic. Results show that the *PressureInitialCache* algorithm achieves a 270% and 150% reduction in re-buffering, compared to the *Best effort* and *HotSpot* schemes, and a 12% and 8% increase in the average received bitrate, compared to the *Best effort* and *HotSpot* schemes. Based on these results, we can conclude that our network-assisted adaptation approach achieves better performance in large networks where, in general, clients are apart from caches.

Conversely, this study also shows that the ability to migrate network functions can have some drawbacks for applications such as DASH. We observed that the migration of the vCaches not only affected the re-buffering duration, it also had effects on the received video quality. Utilizing SAND architecture components and managing the migrations based on the gathered statistics about clients' status was essential to reduce the negative effect of vCache migration. By taking these facts into account, the *PressureInitialCache* migration algorithm considers the buffer values of the clients and helped to reduce the re-buffering duration during hand-off. Compared with *PressureInitialCache*, the use of the *PressureInitialCache* migration algorithm improves the received video bitrate by 2% and decreases the re-buffering duration by 7%. This leads us to another observation: in order to provide the required performance levels for applications which are sensitive to changing conditions, the management framework should also be supported by some application specific tools, technologies, or standards.

In summary, we observe from our results that:

- network assisted adaptation achieves better results in terms of both client experienced quality and network resources utilization. We conclude that utilizing *MPEG-SAND* technology and having communication between vDANE elements in our algorithm gives more flexibility to clients for better adaptation.
- when considering both the received bitrate and the re-buffering time, we can conclude that the *PressureInitialCache* algorithm outperforms the other algorithms when the network has less bandwidth. The performance of the *PressureInitialCache* algorithm is more evidence in bigger networks (e,g., *BellCanada*) where more clients are widely distributed in the edge points of the network, and have more distance with caches and each other.
- installing additional caches, which is decided by using the *PressureCache* algorithm, allows clients to experience high video quality with acceptable re-buffering time.
- during hand-off, when clients disconnect from one cache and connect to another, we alleviated some re-buffering time during the migration period by using a cache migration algorithm.

This study paid considerable attention to network traffic dynamics, and for future work, we aim to improve cache efficiency with cache cooperation by implementing message exchange mechanism between caches.

ACKNOWLEDGMENTS

This work is supported by TUBITAK EEEAG under grant 115E449 and Digiturk beIN Media Group (https://www.digiturk.com.tr/), and partially supported by the EU-Brazil project: NECOS - Novel Enablers for Cloud Slicing (777067)

References

1. MPEG-DASH; . Dynamic Adaptive Streaming over HTTP; https://mpeg.chiariglione.org/standards/mpeg-dash[Online; Accessed 28-04-2022]; .
2. DASH Industry Paper, "DASH-IF Position Paper: Server and Network Assisted DASH (SAND). In: *DASH Industry Paper,version 1.0 published October 7. 2016*; 2016.
3. Kalan Reza Shokri, Sayıt Müge. SDN Assisted Codec, Path and Quality Selection for HTTP Adaptive Streaming. *IEEE Access.* 2021;9:129917-129932.
4. Mijumbi Rashid, Serrat Joan, Gorricho Juan-Luis, Latre Steven, Charalambides Marinos, Lopez Diego. Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine.* 2016;54(1):98–105.
5. Ibn-Khedher Hatem, Hadji Makhlouf, Abd-Elrahman Emad, Afifi Hossam, Kamal Ahmed E. Scalable and cost efficient algorithms for virtual CDN migration. In: *2016 IEEE 41st Conference on Local Computer Networks (LCN)*:112–120IEEE; 2016.
6. Kalan Reza Shokri. Improving Quality of HTTP Adaptive Streaming with Server and Network-Assisted DASH. In: *2021 17th International Conference on Network and Service Management (CNSM)*:244–248IEEE; 2021.
7. Han Shujun, Li Jun, Dong Qian, Ma Yuxiang, Song Liujing. Service-aware Based Virtual Network Functions Deployment Scheme in Edge Computing. In: *2020 22nd International Conference on Advanced Communication Technology (ICACT)*:562-565; 2020.
8. Hmaity Ali, Savi Marco, Musumeci Francesco, Tornatore Massimo, Pattavina Achille. Virtual network function placement for resilient service chain provisioning. In: *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*:245–252IEEE; 2016.
9. Katsaros Konstantinos V., Glykantzis Vasilis, Petropoulos George. Cache peering in multi-tenant 5G networks. In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*:1131-1134; 2017.
10. Song Youmei, Wo Tianyu, Yang Renyu, Shen Qi, Xu Jie. Joint optimization of cache placement and request routing in unreliable networks. *Journal of Parallel and Distributed Computing.* 2021;157:168-178.
11. Luizelli Marcelo Caggiani, Bays Leonardo Richter, Buriol Luciana Salete, Barcellos Marinho Pilla, Gaspary Luciano Paschoal. Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*:98–106IEEE; 2015.
12. Kiran Nahida, Liu Xuanlin, Wang Sihua, Yin Changchuan. VNF placement and resource allocation in SDN/NFV-enabled MEC networks. In: *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*:1–6IEEE; 2020.
13. Ibn-Khedher Hatem, Abd-Elrahman Emad, Kamal Ahmed E., Afifi Hossam. OPAC: An optimal placement algorithm for virtual CDN. *Computer Networks.* 2017;120:12 - 27.
14. Benkacem Ilias, Taleb Tarik, Bagaa Miloud, Flinck Hannu. Optimal VNFs Placement in CDN Slicing Over Multi-Cloud Environment. *IEEE Journal on Selected Areas in Communications.* 2018;36(3):616-627.

15. Clayman Stuart, Kalan Reza Shokri, Sayıt Müge. Virtualized Cache Placement in an SDN/NFV Assisted SAND Architecture. In: *2018 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*:1-5; 2018.

16. Cevallos Moreno Jesús Fernando, Sattler Rebecca, Caulier Cisterna Raúl P., Ricciardi Celsi Lorenzo, Sánchez Rodríguez Aminael, Mecella Massimo. Online Service Function Chain Deployment for Live-Streaming in Virtualized Content Delivery Networks: A Deep Reinforcement Learning Approach. *Future Internet.* 2021;13(11).

17. Farahani Reza, Tashtarian Farzad, Amirpour Hadi, Timmerer Christian, Ghanbari Mohammad, Hellwagner Hermann. CSDN: CDN-Aware QoE Optimization in SDN-Assisted HTTP Adaptive Video Streaming. In: *2021 IEEE 46th Conference on Local Computer Networks (LCN)*:525-532; 2021.

18. Farahani Reza, Tashtarian Farzad, Erfanian Alireza, Timmerer Christian, Ghanbari Mohammad, Hellwagner Hermann. ES-HAS: An Edge- and SDN-Assisted Framework for HTTP Adaptive Video Streaming. In: *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*NOSSDAV '21:50–57; 2021.

19. Kalan Reza Shokri, Sayıt Müge, Clayman Stuart. Optimal cache placement and migration for improving the performance of virtualized SAND. In: *2019 IEEE Conference on Network Softwarization (NetSoft)*:78–83IEEE; 2019.

20. Kalan Reza Shokri, Sayıt Müge, Clayman Stuart. Multimedia Service Management with Virtualized Cache Migration. In: *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*:171-177; 2020.

21. Singh Manmeet, Varyani Nitin, Singh Jobanpreet, Haribabu K.. Estimation of End-to-End Available Bandwidth and Link Capacity in SDN. In: Kumar Navin, Thakre Arpita, eds. *Ubiquitous Communications and Network Computing*:130–141Springer International Publishing; 2018; Cham.

22. Megyesi Péter, Botta Alessio, Aceto Giuseppe, Pescapè Antonio, Molnár Sándor. Available Bandwidth Measurement in Software Defined Networks. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*SAC '16:651–657ACM; 2016; New York, NY, USA.

23. Project FloodLight, FloodLight; http://www.projectfloodlight.org/floodlight;[Online; Accessed 28-04-2022]; .

24. Kreutz Diego, Ramos Fernando MV, Verissimo Paulo Esteves, Rothenberg Christian Esteve, Azodolmolky Siamak, Uhlig Steve. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE.* 2014;103(1):14–76.

25. Topology-Zoo; http://www.topology-zoo.org/;[Online; Accessed 28-04-2022]; .

26. Badshah Jan, Mohaia Alhaisoni Majed, Shah Nadir, Kamran Muhammad. Cache Servers Placement Based on Important Switches for SDN-Based ICN. *Electronics.* 2020;9(1):39.

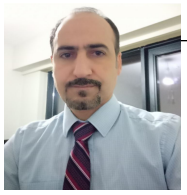
27. Big Buck Bunny; http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/BigBuckBunny/2sec[Online; Accessed 28-04-2022]; .

28. Miller Konstantin, Bethanabhotla Dilip, Caire Giuseppe, Wolisz Adam. A Control-Theoretic Approach to Adaptive Video Streaming in Dense Wireless Networks. *IEEE Transactions on Multimedia.* 2015;17(8):1309-1322.

29. Mao Hongzi, Netravali Ravi, Alizadeh Mohammad. Neural adaptive video streaming with pensieve. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*:197–210; 2017.

30. Binder Andrej, Boros Tomas, Kotuliak Ivan. A SDN Based Method of TCP Connection Handover. In: Khalil Ismail, Neuhold Erich, Tjoa A Min, Xu Li Da, You Ilsun, eds. *Information and Communication Technology*:13–19Springer International Publishing; 2015; Cham.

AUTHOR BIOGRAPHY



Reza Shokri Kalan. is a CDN specialist in Digiturk beIN Media Group. He received his B.Sc degree in Computer Engineering from IAU-Shabestar-IRAN in 2003. For a decade he worked in commercial arena. He received M.Sc degree in Computer Engineering from Eastern Mediterranean University (EMU), Cyprus in 2013. and Ph.D. degree in Information Technology from Ege University- International Computer Institute in 2020. His research interests include computer networks, Software Defined Networking, wireless communication and multimedia system.

.

Stuart Clayman received his PhD in Computer Science from University College London in 1994. He is currently a Principal Research Fellow at UCL EEE department, and worked as a Research Lecturer at Kingston University and at UCL. He co-authored over 70 conference and journal papers. His research interests and expertise lie in the areas of software engineering and programming paradigms; distributed systems; virtualised computer and network systems, network and systems management; sensor systems and smart city platforms, and artificial intelligence systems.

Müge Sayıt is an Associate Professor at International Computer Institute in Ege University in Turkey. She received a M.Sc. degree in 2005 and PhD degree in 2011 in Information Technologies from the same institute and a degree in Mathematics from Ege University in 1999. Her research interests include Software Defined Networking, P2P networks,video streaming and video codecs. She is a member of the IEEE P1916.1 Standard for Software Defined Networking and Network Function Virtualization Performance.

□