Optimal Cache Placement and Migration for Improving the Performance of Virtualized SAND

Reza Shokri Kalan
International Computer Institute
Ege University
Izmir, Turkey
reza.shokri@hotmail.com

Muge Sayit
International Computer Institute
Ege University
Izmir, Turkey
muge.sayit@ege.edu.tr

Stuart Clayman
Dept of Electronic Engineering
University College London
London, UK
s.clayman@ucl.ac.uk

***Abstract*—Nowadays, video streaming over HTTP is one of the most dominant Internet applications, using adaptive video techniques. Network assisted approaches have been proposed and are being standardized in order to provide high QoE for the end-users of such applications. SAND is a recent MPEG standard where DASH Aware Network Elements (DANEs) are introduced for this purpose. As web-caches are one of the main components of the SAND architecture, the location and the connectivity of these web-caches plays an important role in the user's QoE. The nature of SAND and DANE provides a good foundation for software controlled virtualized DASH environments, and in this paper, we propose a cache location algorithm and a cache migration algorithm for virtualized SAND deployments. The optimal locations for the virtualized DANEs is determined by an SDN controller and migrates it based on gathered statistics. The performance of the resulting system shows that, when SDN and NFV technologies are leveraged in such systems, software controlled virtualized approaches can provide an increase in QoE.**

***Index Terms*—DASH, SAND, SDN, virtualized caches.**

I. Introduction

The use of video streaming applications are dominant in the Internet, and this tendency continues to grow year-on-year according to Cisco reports [1]. Along with the popularity of these video applications, the demand for high quality video such as UHD or 3D video content causes an increase in the bitrates of the transferred videos. Recent trends in 5G have called for Network Slicing, which is a move towards segmentation of resources and deployment of NFV for the purpose of enhanced services and applications on a global shared infrastructure. Such an approach has been taken to provide assured performance for various kinds of application.

The performance of video streaming applications, such as DASH (*Dynamic Adaptive Streaming over HTTP*), can increase when ISPs provide assistance and better conditions for viewers, by increasing the network capacity or providing server availability. By considering this aspect, the MPEG group has recently created a new framework for DASH – the SAND (*Server and Network Assisted DASH*) standard [2]. The standard defines special types of network elements, namely DANE (*DASH Aware Network Elements*), as well as the types and the format of the messages which are exchanged between DANEs and other network elements. DANEs have run-time information about DASH components, such as the available representations and / or the client's status. The nature of SAND and DANE provides a good foundation for software controlled DASH environments. Our previous work [3] proposed a DANE implementation utilizing the NFV (Network Function Virtualization) concept, which we called the vDANE (*virtual DANE*). vDANEs behave as virtual caches which are managed by the ISPs, and due to the nature of vDANE, we will use the terms 'vDANE' and 'cache' interchangeably.

978-1-5386-9376-6/19/$31.00 ©2019 IEEE

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

The contributions of this work are to address: (i) an approach for selecting a node among feasible nodes for creating an instance of vDANE function; (ii) moving the vDANE instance to another node when network conditions are changed; (iii) an implementation of the SAND architecture which uses vDANEs and has the advantages of having SDN. The paper is organized as follows: Background and Related work are in Section II; the cache selection algorithm is presented in Section III; performance evaluations are in Section IV, and in Section V we conclude the paper and provide future work.

II. Background and Related Work

The performance of video streaming applications can be measured by QoE (Quality of Experience) parameters, where Video quality, outage duration, and startup delay can be given as examples to QoE parameters. QoE should not be dramatically affected by the changing network conditions in order to provide seamless service to its users. However, network conditions can be very dynamic due to the changing number of clients connected to a specific server and variation in amount of cross traffic. In order to cope with this dynamism of network conditions, adaptive HTTP streaming applications have been developed, whereby the clients can change the video quality according to the changing throughput or buffer fullness. The MPEG group standardized such adaptive video applications with the DASH proposal. In the DASH architecture, the video files are encoded multiple times, with each encoding at a different bitrate. These encoded video files are called *representations*. Each representations are divided into small sized segments so that clients can request, decode, and play segments from different representations as needed. This scheme allows clients to request video at different qualities during streaming and therefore adapt video quality over time.

In the SAND standard, the video streaming system architecture consists of DANEs, video servers, web-caches and clients. The message types defined in the standard are PER (Parameters for Enhancing Reception), PED (Parameters for Enhancing Delivery) and SAND status and metric messages. PER and PED messages are sent by the DANEs to the clients and other DANEs, respectively, and they are used for providing information about network to the clients or making content or infrastructure arrangements in DANEs for enhancing delivery. Status and metrics messages are sent by the clients to DANEs, which carries information about current status of clients' internal parameters [2].

The advantages and flexibility offered by SDN and NFV technologies have encouraged researchers utilize them in caching infrastructure for video streaming architectures. In [4], NFV based multimedia system is proposed, where the authors proposed an optimization model for finding locations of virtual datacenters by only considering of cost at datacenter. The authors introduced a caching platform called OpenCache for DASH by leveraging SDN in [5]. The communication between virtual and original CDN servers in a DASH system was studied in [6]. Cache localization was not considered in these studies [5], [6].

There are several studies on the localization of the virtualized network functions and caches in the literature. The approaches proposed for localization of the virtual functions are developed for conventional Internet applications rather than focusing on video streaming applications' requirements [7], [8]. Optimal cache localization policy which jointly considers network capacity, the load and migration of the virtual CDNs is proposed in [9]. Although these approaches provide remarkable solutions to virtual cache placement problems, they differ from our study since our implementation is developed by taking DASH and SAND into account.

In our previous work, we introduced the concept of vDANE and proposed an algorithm for selecting the placement of an additional vDANE when a new cache is required [3]. In this paper, we study the problem of selecting the location of the initial vDANE and the migration of it considering SAND characteristics. Moreover, we consider the replacement of the virtualized cache function and connectivity as well as bandwidth and hop counts of the links, where only available bandwidth is considered in [3].

III. A SAND architecture with vDANEs

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

In our scenario, the SDN controller has information about IP addresses of DASH clients as well as the bitrate of the video file's representations, by means of the cooperation between network operator and video streaming company. The system architecture and controller modules, illustrated in Figure 1, provide DANE function modules for: (i) Cache Placement and (ii) Cache Migration. The controller modules include basic network service functions which are responsible for collecting traffic statistics, monitoring hosts and flows, and uploading flow rules to the switches. The controller also sends and receives DANE Messages as part of the SAND architecture.

The Cache Placement module is triggered when the first DASH client comes online and sends a request to start video streaming, and it selects an optimal point for the vDANE instance. The Cache Migration module is triggered when the controller detect the location of vDANE should be changed. Since network capacity is dynamic due to cross traffic and a different number of DASH clients being connected / disconnected over time from various points of the network, this module changes the location of the vDANE functions to provide seamless service, under this dynamic environment. The Cache Migration module then sends DANE Messages to ensure that clients can renew their TCP connection and informing them about IP address of the new server.

A. Selecting the Initial Cache Location

Our study aims to find the optimal location for the placement of the initial vDANE instance and then to migrate it, in a such a way that mainly reduces the traffic and still provides high QoE for DASH clients. In order to minimize the total network traffic and to efficiently utilize network bandwidth, the hop count should be minimized. Reducing the distance between clients and content sources, it can eliminate

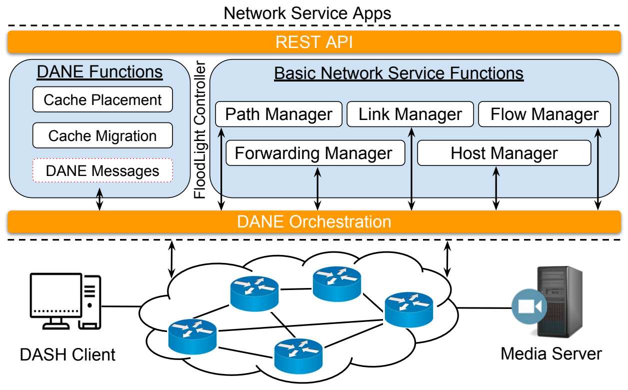


Fig. 1: SAND Architecture with vDANE Orchestration

forwarding flows over inessential links. Taking advantage of minimum hop count can improve latency and the total amount of flows that travel across network paths. Minimum hop count is considered in the $HotSpot$ algorithms [10], but this approach may result in lower performance if virtual caches are placed behind bottlenecked links. To overcome this obstacle, besides using the distance factor, in terms of hop count, we take the total amount of bandwidth and the number of connectivity into account as well.

In the proposed cache placement algorithm, vDANEs are instantiated at the network node which minimizes the distance to the clients and maximizes the bandwidth. To achieve this, we used a formula which we called $PressureInitialCache$ by modifying $PressureCache$ proposed in our previous work, which is based on the $Pressure$ formula discussed in [10] and enhanced in [11].

The Initial Cache Selection algorithm is presented here. Assume d_{ij} is the distance in terms of hop count between nodes i and j in the network and P is the set of the paths between these nodes. Here, node term denotes the switching elements that can run virtualized network functions, where there is a vDANE instance running on node j. Let's assume that clients connected to the node i request video contents from the cache. In order to reduce total network traffic and response time as well, candidate cache node should have minimum distance with clients (d_{ij}) and provide maximum available bandwidth and connectivity. It is clear that the cache location which gives the maximum value of *bandwidth / distance* is optimal. Suppose L represents number of the links in the network, $B = \sum_{l=1}^{L} b_l$ is the total network bandwidth and $B^{'} = \sum_{l=1}^{L_j} b_l$ is the total bandwidth of the links that directly connected to node j. Here, b_l refers to bandwidth of the link l and L_j refers to the number of links which j connected to. While B represent total capacity of the network, $B^{'}$ represents the maximum traffic amount that can be transferred by the node j. $D(j) = \sum_{i=h+1}^{H} d_{ij}$ refers to distance between the node j and clients > 1 hop away, where h represents the number of clients connected to j^{th} node through one hop distance, H is the number of all clients and $D^{'}(j) = \sum_{i=1}^{h} d_{ij}$. The context of locality has remarkable impact in performance. On contrary, long distance connection has negative effect due to RTT delay and consumed links bandwidth while traveling through more of them. By considering these facts, we evaluate the candidate locations to run the initial vDANE network function by using formula (1). This formula sets a pressure value, which is indicated by $P(j)$, for all possible candidate node locations denoted by j. Since client with direct connection to the cache node has only one hop distance, the d_{ij} in the denominator equals to 1, which makes $\sum_{i=1}^{h} d_{ij} = h$.

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

Finally, in order to increase reliability, the number of links connected to a candidate node also has been taken into account. Note that, more connection links provide more reliability. Lets L_j indicated the total number of connected links to the candidate node. Modified formula, which is called as $PressureInitialCache$, is given in Formula (2).

$$PressureInitialCache = \frac{\sum_{l=1}^{L} b_l * \sum_{i=h+1}^{H} d_{ij}}{\sum_{l=1}^{L_j} b_l * h * L_j} \tag{2}$$

When a DASH client request to start the video streaming application, if there is no running vDANE instance on the network, the controller triggers $Cache\ Placement$ module. A score, called the $PressureInitialCache$ score, is calculated for all potential switches that can be selected as hosting a vDANE instance. The switch which has minimum $PressureInitialCache$ value considered as the optimal place for the initial vDANE instance.

B. *Cache Migration*

After the initial vDANE instance starts, the controller continues collecting statistics of the network and monitoring DASH and cross traffic patterns. When the number of clients in the system changes or it determines changes in traffic amount, the controller calculates $PressureInitialCache$ scores for each node again. Changes in traffic amount are determined based on a predefined certain *quality threshold*, which can be assigned by either video streaming company or network operator. If the location of vDANE is changed according to the new values of pressure scores, the controller sends PER enforcement messages to the clients to change their server. Further, the shortest paths are selected for each client for streaming packets. If the average quality in terms of bitrate or representation received by the clients is under the *quality threshold*, the controller then decides to deploy additional vDANE. An algorithm which decides when a new cache instance should be installed is proposed in our previous work [3].

IV. Results

We have implemented and run simulations using the Mininet emulator in order to evaluate the performance of the proposed $PressureInitialCache$ algorithm. The agility of the Mininet [12] provides an easy way to prototype and evaluate SDN applications. The *FloodLight* SDN controller [13], with the assistance of OpenFlow as a southbound interface, was used during the simulations.

A. Simulation Setup Parameters

Three different network topologies were used to evaluate the performance of the $PressureInitialCache$ algorithm, which include a $Custom$ topology and two from the Internet Topology Zoo [14]: the *Compuserve* and *BellCanada* topologies, as shown in Table I. The number of the DASH clients are set to *10*, *12*, and *20*, for of the network topologies, respectively. A Poisson distribution with different mean values: λ=7, λ=10, λ=15 Mbps, were used for defining the network link bandwidths.

During simulation, the *Big Buck Bunny* video [15] with six different representations is used for streaming. Each representation contains 299 video segments with an equal length of 2 seconds, so the total length of each representation is 598 seconds of video. The first representation ($R1$) has the lowest bitrate, and the last representation ($R6$) provides the highest bitrate / video quality. To achieve better QoE, clients should request and receive as many segments as possible from representations with a higher bitrate. The clients run a throughput based rate adaptation algorithm, where they decide the video quality based on the observed throughput values. Clients are randomly distributed over all of the nodes, and the DASH clients join and leave the network based on a $Poisson$ distribution, which has a mean equals to 3 seconds.

B. Performance Evaluations

In order to provide comparable performance results, we also implemented two other algorithms: one of them is *Best effort*, and the other one is the $HotSpot$ approach described in [10]. In the *Best effort* approach, a cache is placed near to the center of topology. Therefore, the average distance between all the clients and the cache is minimum with this approach. The alternate $HotSpot$ algorithm, benefits from the concept of locality by introducing $F(i) = a^3 + b^2 + c$ as a pressure function. In this function, a, b and c, refer to the number of clients with one, two, and three hops away from node i, respectively. Unlike *Best effort*, $HotSpot$ places caches in a location which has highest client density. In the simulations, all the parameter settings were equal for all of the approaches. Each simulation was repeated 10 times and the average values are given in the tables.

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

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

The percentage of received video segments belonging to each representation are given in figures 2, 3, and 4. Here, $R1$ represents the lowest quality, $R2$ through to $R5$ represent the medium quality representations, and $R6$ represents the highest quality. The numbers show that the clients in the $PressureInitialCache$ approach receive the biggest share of the highest quality $R6$ segments in the Custom and Compuserve topologies, which is a good result. Since BellCanada is a relatively bigger topology, the distance between the cache and the clients affects the quality and the smallest percentage belonging to highest quality among the results related to all topologies is observed in BellCanada topology. Nevertheless, the clients with the proposed $PressureInitialCache$ approach received the highest number of segments from the top representation – again a good result.

In order to show that the improvement in the received quality provided by $PressureInitialCache$ is maintained throughout the whole run, we averaged the received quality as a function of time. These values are comparatively given for all approaches for the Bell Canada topology in Fig. 5. Average received quality values are obtained by averaging the bitrate of the representations downloaded by all clients in all simulations. When we examine the data, the clients using the $PressureInitialCache$ approach always play the video with higher quality, and it is consistent for each segment of the video. This shows that, for dynamic adaptive video streaming applications, optimizing localization by considering the density of the clients has a positive impact on the received quality. However, if the network has limited capacity, where $\lambda = 7$ Mbps, we see a small quality degradation of the proposed approach. The reason for that is that after a vDANE is migrated, and during the TCP re-establishment process, the clients stop receiving traffic and measure throughput as zero and so request the lowest quality since clients use throughput based adaptation. A buffer based rate adaptation would perform better in that case. This shows that the performance of the virtualized environment is highly connected with the rate adaptation algorithm used by DASH applications.

To demonstrate the impact of the proposed approach in larger networks where more than one cache instance is required, we also ran additional simulations by using the Bell Canada topology with 100 clients. In the first set of simulations, the first cache is placed in random locations, whereas the proposed algorithm is used in the second set of simulations. For both sets of simulations, when the number of

TABLE I: Network topologies

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

TABLE II: Performance results: Custom topology

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

Algorithm	λ=7	λ=10	λ=15
Best effort	2309	2865	3532
HotSpot	2546	3532	3440
PressureInitialCache	3013	3670	3680

(b) *Average startup delay (sec)*

Algorithm	λ=7	λ=10	λ=15
Best effort	13	7.9	5.8
HotSpot	13.4	5	6.1
PressureInitialCache	11.2	4.9	5.1

(c) *Average outage duration (sec)*

Algorithm	λ=7	λ=10	λ=15
Best effort	450	53	5
HotSpot	757	4	17
PressureInitialCache	128	2	2

TABLE III: Performance results: Compuserve topology

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

Algorithm	λ=7	λ=10	λ=15
Best effort	2799	3050	3442
HotSpot	2587	3050	2945
PressureInitialCache	3146	3066	3517

(b) *Average startup delay (sec)*

Algorithm	λ=7	λ=10	λ=15
Best effort	10.3	7.9	7.6
HotSpot	14.5	7.9	6.2
PressureInitialCache	10.1	7.9	6.1

(c) *Average outage duration (sec)*

Algorithm	λ=7	λ=10	λ=15
Best effort	351	133	34
HotSpot	615	133	42
PressureInitialCache	180	102	26

TABLE IV: Performance results: BellCanada topology

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

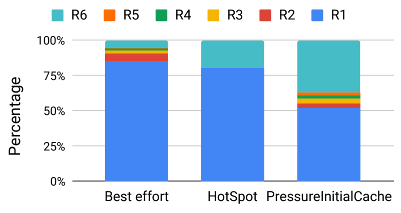
Algorithm	λ=7	λ=10	λ=15
Best effort	2304	2451	2883
HotSpot	2650	2542	2917
PressureInitialCache	2600	2607	3107

(b) *Average startup delay (sec)*

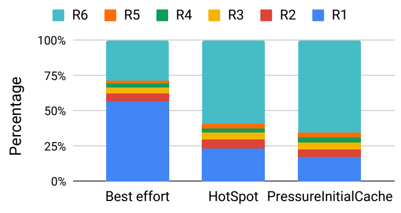
Algorithm	λ=7	λ=10	λ=15
Best effort	26	15	9
HotSpot	19	11	9
PressureInitialCache	13	11	7

(c) *Average outage duration (sec)*

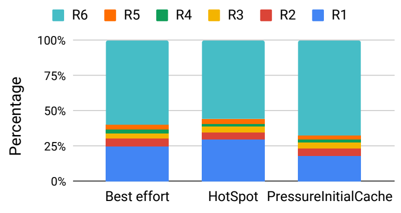
Algorithm	λ=7	λ=10	λ=15
Best effort	386	250	147
HotSpot	662	672	417
PressureInitialCache	340	233	87



(a) λ=7 Mbps

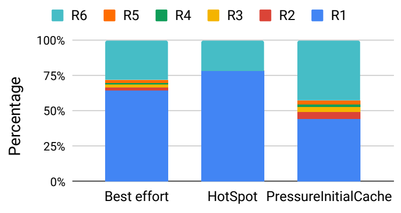


(b) λ=10 Mbps

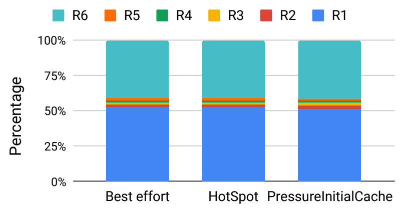


(c) λ=15 Mbps

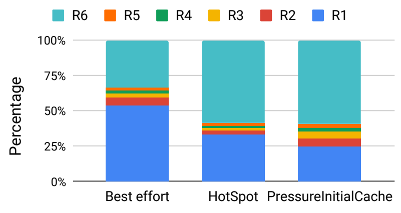
Fig. 2: Distribution of the received quality level – R1 to R6: Custom topology



(a) λ=7 Mbps

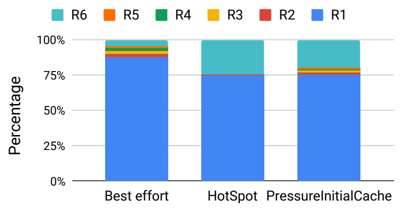


(b) λ=10 Mbps

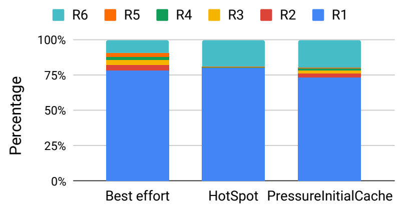


(c) λ=15 Mbps

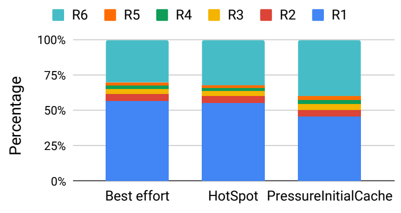
Fig. 3: Distribution of the received quality level – R1 to R6: Compuserve topology



(a) λ=7 Mbps



(b) λ=10 Mbps



(c) λ=15 Mbps

Fig. 4: Distribution of the received quality level – R1 to R6: BellCanada topology

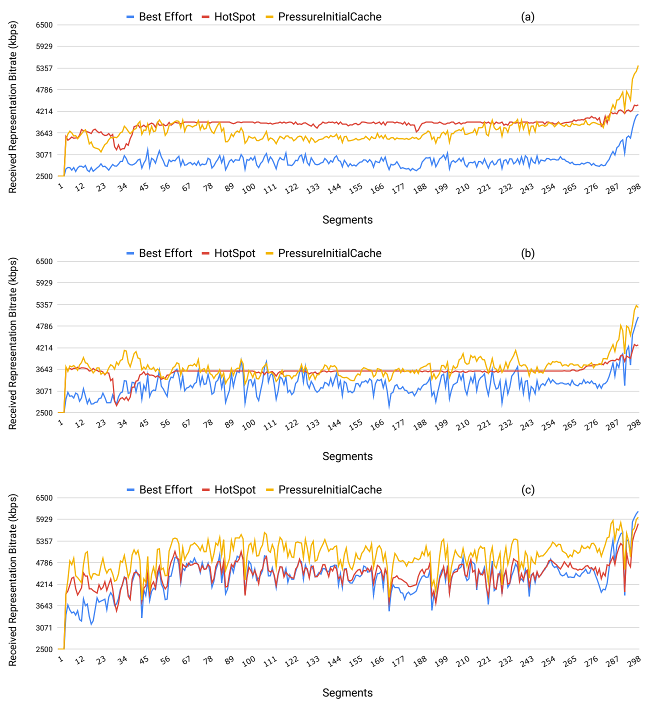


Fig. 5: Bitrate of the received video representation per segment: (a) λ=7 Mbps, (b) λ=10 Mbps, and (c) λ=15 Mbps)

online clients increases, some second caches are installed by using our algorithm given in [3]. Simulation results indicated that with the random approach, clients received 14% of the video segments from the highest quality, on average, while receiving 17% when $PressureInitialCache$ was utilized. The average video bitrate received by clients in the random approach was equal to 2466 Kbps, and was equal to 2571 Kbps when using $PressureInitialCache$. This means that end users experienced better video display with our proposed algorithm. Significant improvement is observed in the outage duration values, where the total outage duration is as low as 168 seconds for the proposed algorithm, but with the random approach results in 637 seconds of outage duration – which is totally unacceptable. For these tests we configured paths with lower bandwidth of λ=7 to show the behaviour of the clients.

V. Conclusion

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

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

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

Acknowledgment

This work was supported by Digiturk company, TUBITAK EEEAG under grant 115E449 and partially supported by the EU-Brazil project: Novel Enablers for Cloud Slicing (777067).

References

[1] Cisco, "Cisco Visual Networking Index: Forecast and Trends, 2017."
[2] MPEG's, "DASH-IF Position Paper: Server and Network Assisted DASH (SAND), ISO/IEC 23009-5, published December 2016."
[3] S. Clayman, R. S. Kalan, and M. Sayit, "Virtualized Cache Placement in an SDN/NFV Assisted SAND Architecture," in *BlackSea2018*.
[4] R. M. B. N. N. Bouten, J. Famaey, S. L. J. Serrat, and F. Turck, "Towards NFV-based multimedia delivery," in *IFIP/IEEE 2015*.
[5] P. Georgopoulos, M. Broadbent, B. Plattner, and N. Race, "Cache as a service: Leveraging SDN to efficiently and transparently support video-on-demand on the last mile," in *ICCCNb2014*.
[6] Z. Li, M. K. Sba, Y. Hadjadj-Aoul, A. Gravey, D. Alliez, J. Garnier, G. Madec, and et al, "Network friendly video distribution," in *NOF2012*.
[7] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The Dynamic Placement of Virtual Network Functions," in *IEEE/IFIP 2014*.
[8] H. Moens and F. D. Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *CNSM 2014*.
[9] H. Ibn-Khedher, E. Abd-Elrahman, A. E. Kamal, and H. Afifi, "Opac: An optimal placement algorithm for virtual cdn," *Comp. Networks, 2017*.
[10] L. Mamatas, S. Clayman, M. Charalambides, A. Galis, and G. Pavlou, "Towards an information management overlay for emerging networks," in *NOMS 2010*.
[11] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive resource management and control in software defined networks," *IEEE TNSM2015*.
[12] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *ACM SIGCOMM 2010*.
[13] FloodLight, "Project Floodlight, floodlight." [Online]. Available: http://www.projectfloodlight.org/floodlight//
[14] Topology-Zoo. [Online]. Available: http://www.topology-zoo.org/
[15] "Big Buck Bunny." [Online]. Available: http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/BigBuckBunny/2sec/