OpenCache: Distributed SDN/NFV Based In-Network Caching as a Service

Shiyam Alalmaei[1], Matthew Broadbent[2], Nicholas Race[2], Samia Chelloug[1]

[1] Information Technology Department, College of Computer and Information Sciences, Princess Nourah bint AbdulRahman University, Riyadh, Kingdom of Saudi Arabia
{smalalmaei,Sachelloug}@pnu.edu.sa

[2] School of Computing and Communications, Lancaster University, Lancaster, United Kingdom
{m.broadbent,n.race}@lancaster.ac.uk

Abstract. In-network content caching allows content to be located towards the edge of the network, closer to users. This approach addresses the challenge of exponentially increasing video traffic. We consider OpenCache: an open-source, highly configurable, efficient and transparent in-network caching that leverages Software Defined Networking (SDN) to benefit last mile environments. However, due to its reliance on a centralised OpenCache controller and SDN controller, it suffers from three issues: scalability, reliability and high availability. In this work, we build on and extend the capabilities of OpenCache as a caching solution by leveraging Network Functions Virtualisation (NFV) and using a distributed SDN controller. We discuss the architectural design and technology decisions for the caching platform distribution including the functional components and highlight the role of virtualising, orchestrating and managing the key processes of caching content and control functions. Our target is to design an open-source, distributed in-network caching platform that is highly available, reliable and with automated elasticity to enable serving the increasing VoD traffic quickly and efficiently.

Keywords: Video-on-demand (VoD), software defined networking (SDN), network functions virtualisation (NFV), open network operating system (ONOS), open source MANO (OSM), distributed in-network caching, opencache.

1 Introduction

Internet video traffic is predicted to increase to 82% of all consumer Internet traffic by 2021 [1]. End users expect the best possible quality in the video streaming experience. At the same time, high definition (HD) video streaming will continue to move to Ultra HD and 3D. Trying to meet the growing demand for Video on Demand (VoD) service, where individuals can retrieve previously recorded content at a time after it was initially broadcast or made available, and the increasing popularity of HD

content, adds a huge burden on the underlying network infrastructure, it has to provision delivery throughput in the order of tens of Mbps for just one stream. Currently, VoD requests are handled individually, using a unicast content delivery paradigm. This means naively ignoring the duplicate independent flow requests for the same content from different users at different times. Hence, a very large amount of identical contents is delivered on the same network segments repeatedly [2].
In order to address this problem, in-network caching service came up, which caches the contents locally in the edge networks very close to the end-users. In-network caching has been proved to be an efficient solution to improve the efficiency of network utilisation and the data retrieval performance.
The new paradigms of Software Defined Networks (SDN) [3] and Network Functions Virtualisation (NFV) [4] have recently redefined the vision of designing, deploying and managing networking services. Combined together, they provide network managers with a complete, programmatic and flexible control of a dynamic view of the network. These paradigms are discussed in *Section II*. Telecommunications providers and over-the-top (OTT) content providers have taken a great interest in leveraging on these technologies. In this context, this paper builds on and extends the OpenCache capabilities [2], [5] as a distributed SDN/NFV based in-network caching as a service (CaaS), focusing on the emerging capabilities of distributed SDN and the support of NFV. Our target is to design and implement a distributed in-network caching platform that is highly available, reliable and automatically elastic.
The remainder of the paper is organised as follows. *Section II* presents the background necessary to extend OpenCache as a distributed SDN/NFV based caching solution, whilst related work is presented in *Section III*. *Section IV* introduces design objectives, whereas the distributed OpenCache proposed architecture is described in *Section V*. Features of the proposed architecture are described in *Section VI*, and finally, *Section VII* concludes the paper and future work.

2 Background Information

2.1 Software Defined Networking

Software Defined Networking (SDN) is a very promising networking paradigm that facilitates decoupling the control plane in a network from the data plane and provides logical centralisation of network control, management and programmability. External applications (*i.e.* OpenCache) are provided centralised network perspective and status, and the means to programmatically manage and control the forwarding devices functionalities through the SDN controller based on the application profile and user demands quickly and cohesively.

Distributed Software Defined Networking. The standard centralised approach for SDN is based on a single controller managing all forwarding devices. However, this leads to a single point of failure and poses severe limitations to network scalability and reliability. Moreover, it may get over-loaded with large number of devices to handle, which disrupts the network functionality. Finally, in geographically large

networks, forwarding devices can be physically very far from the controller. This induces large latency in flow modifications due to the propagation delays.
Distributed SDN controllers overcome all of the above limitations [6]. Multiple instances of the controller manage the whole network jointly. The network is divided into different sub-domains, each is under the control of one controller instance. Distribution of the control functionality increases scalability and improves the reliability of the control plane. Furthermore, large networks that suffer from the big distance between the controller and the network devices can be handled, because the device control is distributed and can be balanced among the controllers, thus improving the controller reactivity.

Consistency Mechanisms for Distributed SDN. The main challenge of distributed SDN controllers is to guarantee a logical centralised global view of the network state. The same global network view must be known at each controller to make correct decisions. This behaviour should be transparent to the application layer, and requires keeping all the shared data structures synchronised through some consistency mechanisms. Some of the main consistency mechanisms that have a direct application in SDN networks are *eventual consistency* and *strong consistency* [6]. Eventual consistency provides a weak form of consistency. This implies that for some time, some controllers may read values different from the actual updated ones; but eventually after some time, all the controllers will have the updated values. In return, this mechanism is employed for superior read/write performance and high availability. On the other hand, strong consistency mechanism ensures that when a controller updates some data, then no other controller is allowed to read this specific data until it gets updated at all (or most of) the other controllers. This implies that each controller reads always the most updated version of a data. This mechanism is employed in systems that prioritise consistency over high availability.

Design Choices for Distributed SDN. There are two main design choices for distributed SDN controllers, namely, hierarchical and flat models [6]. Each model distributes the network information among controllers differently. In hierarchical model, one (or some) controllers in the cluster have the whole up-to-date global network state. This model has hierarchical controls that run from top to bottom among controllers. The bottom of the hierarchy contains local controllers that maintain the switches in each sub-domain and keep only their respective local network state. Meanwhile, the top of the hierarchy contains the root controllers that manage the coordination between local controllers. Therefore, local controllers must first query network information from the root controllers before they can execute any inter-domain operation. From the coordination perspective, the number of root controllers should be kept minimal; hence, this model tends to be more consistent since fewer controllers maintain the global network state. However, this also means that fewer backup controllers are available to take over the failed controller. Additionally, SDN controllers in this model are divided into two roles (root and local) with different capabilities. Due to this difference, the local controller may not be able to take over the root controller and vice versa.
On the other hand, in the flat model, all of the controllers in the cluster are peers and have the same privileges and maintain the whole global network state. All of them can

contact and notify each other directly to construct the global network state. Therefore, the flat model is easier to maintain. It has a robust failover mechanism; any controller in the cluster can simply take over a failed controller because they are all peers. Finally, it ensures fast access to the global topology stored at each controller. However, a main issue of this model is the overhead of the all-to-all communications between controllers to share the global state.
The survey in [6] compares between hierarchical and flat models. The hierarchical model is more scalable and efficient due to maintaing and synchronising the global network state between root controllers only. However, flat model provides a straight forward failover mechanism and robustness due to the similar roles of the controllers and direct communication between them. Additionally, high performance systems can benefit from the flat model, since a global network view is stored locally at each controller.
Two most popular distributed SDN controllers are Open Network Operating System (ONOS) [7] and OpenDaylight [8]. ONOS is principally designed for telecommunications companies, service providers and carrier networks. It is designed to support hybrid networks with a focus on scalability, high availability and performance. On the other hand, OpenDaylight is primarily designed for datacenters with the capability to support many southbound interfaces that are facing the data plane devices to bring legacy network and next generation network together.

Open Networking Operating System (ONOS). Open Networking Operating System (ONOS), is a distributed SDN controller that adopts the flat model to achieve a logically centralised SDN controller through a replicated state machine model. ONOS allows achieving high throughput, low latency, scalability, fault-tolerance and high availability. Each ONOS controller in the cluster is responsible of managing the switches under its sub-domain, and updating their state on the distributed data stores. For reliability reasons, each switch can connect to multiple ONOS controllers, but only one will be its master, whereas, the other controllers are standby. The master controller has full control over the switch it masters in terms of read/write capabilities on the switch forwarding tables. However, the standby controllers only have read capabilities on the switches connected to them and could become the new master of the switch in case the main master has failed.
Applications on top of ONOS, can view the whole network topology and may read/update the network view to decide on forwarding policies. We mention that there are many releases for ONOS. Each one of them provides some new features.

Distributed stores and consistency protocols in ONOS. The distributed data stores implement the distributed databases in ONOS. The main ones are the following:
Mastership store which keeps the mapping between each switch to its master controller and it is managed by a strongly consistent protocol, using RAFT consensus algorithm [9], A RAFT implementation requires a cluster of nodes (*i.e.* controllers) each having a database (*i.e.* mastership data store) which is replicated in all nodes. There has to be a leader of the cluster for coordinating the consistency between the nodes, which is responsible for receiving update requests from all the nodes and then relaying database updates to the other nodes. Once the majority of the follower nodes have acknowledged the update, this is actually committed. In the case of network

partitions, only the side with the majority of the nodes will be able to update the database, hence avoiding conflicting updates in two different network partitions. In ONOS, for scalability issues, multiple instances of RAFT algorithm run simultaneously. This implies that the data stores are actually partitioned (sharded) into different parts, each of them managed by a different RAFT instance.
The other main distributed data store is the *Network topology store*, which describes the network topology in terms of switches, links and hosts. It is managed in an eventually consistent protocol called anti-entropy, which is based on a simple gossip algorithm in which each controller picks at random another controller in the cluster at fixed intervals (usually 3-5 seconds), and then sends a message to compare their respective topology views. If a controller is aware of newer information that the other controller does not have yet, then they exchange that information to update their stores. This ensures that all the controllers achieve consensus on the network topology, according to an eventually consistent model. This approach quickly detects and synchronises a controller that has a slightly drifted state. Moreover, it quickly synchronises a new controller that joins the cluster with the rest.

2.2 Network Functions Virtualisation

Network Functions Virtualisation (NFV) is fundamentally changing how network services are deployed and managed by providing flexibility, agile service delivery, auto-scalability and optimal resource usage. These services are provided over the same common infrastructure. The European Telecommunications Standards Institute (ETSI) has defined a framework for Network Functions Virtualisation and Management and Orchestration Architectures (MANO) [10]. These open-source architectures are broadly defined to allow development, extension and testing in proprietary ways. *Section V* explains how OpenCache is mapped to this framework.

The ETSI MANO functional blocks are explained below:

- *VNF:* is a functional block representing the Virtualised Network Function implemented on commodity hardware.
- *Operation/Business Support System (OSS/BSS):* includes collection of operation and business applications for operators and service providers that are used to provision and operate their network services. This block is not tightly integrated into the NFV architectural framework but is expected to work in coordination with it.
- *VNF Manager (VNFM):* configures and manages single or multiple VNFs' lifecycle (*i.e.* instantiate, update, query, scale up/down, terminate) on its domain. It is responsible for Fault, Configuration, Accounting, Performance and Security Management (FCAPS) for the virtual part of the VNF.
- *Network Functions Virtualisation Infrastructure (NFVI):* consists of both virtual and physical hardware (*e.g.* compute, storage, and networking) and software (*e.g.* hypervisors) components. Together they provide the infrastructure VNF resources.

- *Virtualised Infrastructure Manager (VIM):* controls and manages the NFVI resources that are usually within one operator's infrastructure domain. There may be multiple VIMs in an NFV architecture.
- *Element Management (EM):* is not part of the MANO, however it has an important role to play. It is responsible for VNFs (FCAPS). The EM collaborates with the VNFM to perform those functions. EM is responsible for the FCAPS for the functional part of the VNF opposed to VNFM, which also manages the FCAPS of the VNF but only for the virtual part (*e.g.* reporting an issue with spinning up a VNF).
- *Network Function Virtualisation Orchestrator (NFVO):* has two primary goals; the first goal is to manage and coordinate NFVI resources across multiple VIMs. The second goal is to manage and coordinate the lifecycle of network services (*i.e.* create, update, query, delete) across multiple VNFMs. NFVO achieves its first goal by a sub component called resource orchestrator (RO), which is responsible of the coordination, authorisation, allocation and de-allocation of NFVI resources by communicating with the VIMs through their north bound APIs. To achieve the second goal of the NFVO, it uses a sub component called the Network Service Orchestrator (NSO), which creates end-to-end service between different VNFs that may be managed by possibly different VNFMs and coordinates groups of VNF instances that jointly realise more complex network functions (also called VNF Forwarding Graphs).

Authors in [11] compare between several NFV MANO projects. However, OSM [12], is an open source NFV MANO platform that is considered as the reference implementation for the NFV MANO since it is hosted by ETSI and aligned with its information models to meet the requirements of production NFV.

3 Related Work

OTT content providers (e.g. Youtube) rely on Content Delivery Networks (CDN) for content delivery, which provide a series of large number of cache server mirrors with global coverage, in order to push content to the edges of the Internet to ensure low latency. However, CDNs do not reduce the bandwidth utilisation on edge networks, as multiple duplicate flows of the same request will be created and traverse the external link to reach to the CDN. Recent reports [1] show that 77 % of all Internet video traffic will cross CDNs by 2021, compared to 67 % in 2016; this traffic can be reduced by deploying in-network caching. Therefore, OTT providers have started to deploy and remotely manage edge caching servers in telecommunication operators' networks [13]. Indeed, deploying in-network caching service that is more flexible, configurable and located closer to the end-users, would complement CDNs and truly benefit edge networks by reducing the external bandwidth and the load from the CDNs, and increasing the QoS for the end-users.

OpenCache [2, 5] is an open-source, highly configurable, efficient and transparent in-network HTTP caching service that leverages SDN technology, and supports the ETSI NFV MANO framework. It aims to improve the VoD distribution efficiency, network utilisation and increase the Quality of Experience (QoE) for the end-user, by caching video assets as close to the end-user as possible. OpenCache has three entities

namely, OpenCache Controller (OCC), OpenCache Node (OCN) and a key-value store. Network operators and service providers can program the desired caching behaviour on the OCC which communicates with the SDN controller and OCNs via a JSON-RPC interface. OCC instructs the SDN controller to add the matching redirecting rules for the cacheable content in the flow tables of the switches to redirect the user's packets appropriately to the closest OCN. OCN caches video content, so when it receives a request for accessing the cached video from a user, it delivers the video content to the user directly if there is a cache-hit; otherwise, the OCN requests the video from the original VoD server. The key-value store acts as a database, which records the metadata and caching state of user requests and video content. As a result of caching the contents very close to the user, the following benefits arise: the external link usage gets reduced 100%; it helps also in reducing the load from the VoD content provider and all transient networks along the path to the end-user. Additionally, end-users can experience higher QoE [2].
The in-network SDN based solutions presented in [2], [5], [14], [15], [16] lack for scalability, high availability, reliability and elasticity of the caching control plane due to relying on a centralised cache controller that would not be able to scale with the increasing VoD requests. It is a single point of failure that would disrupt the caching service if it fails and negatively reflect on the service availability. The work in [15], additionally doesn't leverage the NFV capabilities.

4 Design Objectives

Our objective is to extend the current OpenCache design [2], [5] to overcome the limitations mentioned above. We will present a distributed in-network caching architecture with multiple OCCs that are physically distributed but logically centralised by leveraging distributed SDN controllers to achieve high availability, elasticity and reliability even under heavy VoD requests. Moreover, this architecture is associated with NFV MANO platform, where caching functions become software applications (VNFs) leading to flexibility and agile service delivery. We choose ONOS as the distributed SDN controller because it is designed to provide high performance, high availability, reliability and scalability. These features are important to meet the exponentially growing VoD requests. On the other hand, we choose OSM as the NFV MANO because it is hosted by ETSI and considered as the reference implementation for the NFV MANO that meets the requirements of production NFV. Therefore, we expect that the proposed architecture provides support for high availability, reliability, elasticity and automation of end-to-end cache service delivery.

5 Distributed OpenCache Proposed Architecture

Considering the aforementioned objectives described above, first, we will describe how OpenCache is associated with the NFV MANO framework. Then, we will present the distributed OpenCache architecture.
Fig. 1. identifies how the functional components of the ETSI NFV framework are adapted for OpenCache as detailed as follows.

OpenCache main functional blocks are represented as VNFs, which are the Virtual OpenCache Controller (vOCC), Virtual OpenCache Node (vOCN), Virtual Backup Store and Virtual Load Balancer. VOCC has the main role in the functional lifecycle management and orchestration of the vOCNs because it has all of the required details of the caching process. It is responsible for controlling, querying, dynamically configuring and automatically scaling up/down the vOCNs. However, vOCC delegates the heavy lifting of creation and deletion of the VNFs to the VNFM. The aforementioned VNFs constitute a VNF forwarding graph that is created, coordinated and managed by the NFVO based on the network operator policies. It manages the lifecycle of the NFVI corresponding resources (possibly across multiple VIMs). Then the VNFM instantiates the services with respect to the policies that define the VNF graph.

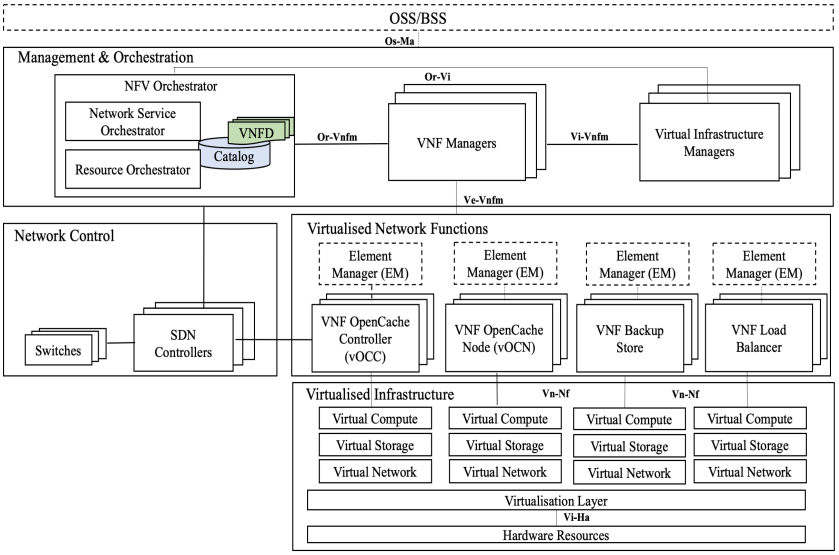

Fig. 1. OpenCache mapped to ETSI NFV framework

The proposed distributed OpenCache architecture that leverages SDN/NFV and constituent layers are illustrated in *Fig. 2.*

The top of this architecture is the application layer. It is responsible for defining the behaviour of the caches. This is where third-party developers will interact with the whole deployment through the controller API. This ensures a separation between OpenCache functionality and the cache behaviour.

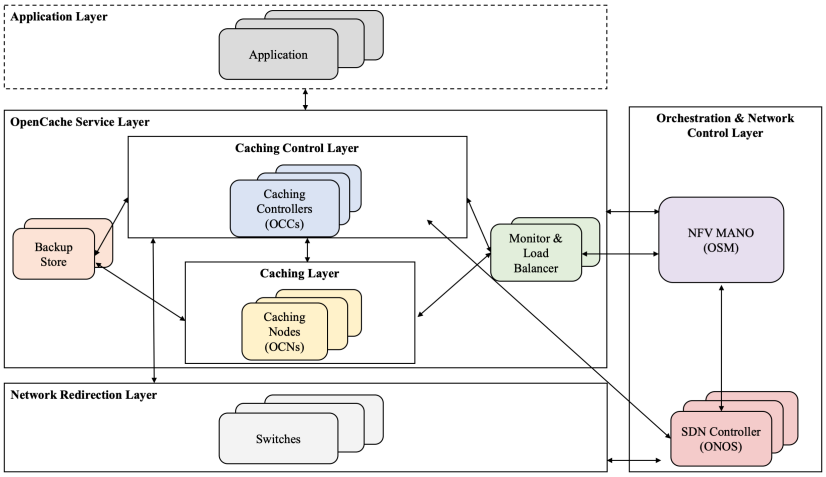

Fig. 2. OpenCache architecture

Then there is the orchestration and network control layer that is responsible for orchestrating the service and network redirection layers, and manages the underlying network. It includes the NFV MANO (*i.e.* OSM) and the SDN controllers (*i.e.* ONOS).

Below the application layer, is the service layer; this is where the core functionality of OpenCache lies. It contains two sub layers namely, *caching control layer* and *caching layer*. The caching control layer consists of multiple physically distributed but logically centralised controllers. It is responsible for the behaviour of all of its connected caching nodes. Furthermore, the caching controllers interact with the NFV MANO and SDN controllers, in order to provide flexible resource allocation and network forwarding. On the other hand, the caching layer typically consists of a number of caching services, running across multiple distributed cache nodes, which directly serve content in response to user requests. Caching node can host multiple services simultaneously, each of which is responsible for delivering a unique set of content to the client. OpenCache controllers and nodes can run on dedicated physical resources, or virtualised ones that are managed by the aforementioned NFV MANO.

Elasticity of services and resource consumption is critical. Therefore, there is a monitoring and load balancing entity for the caching control and the caching layer. This entity is responsible for monitoring the services and resources periodically and taking actions based on some thresholds and criteria in order to scale them up/down. If the monitoring and load balancing entity needs to add or remove resources, then it communicates with the MANO.

Reliability is a key feature too, so there is a backup server that is used by the caching control layer and the caching layer. The backup server stores the whole caching meta-data, state and topology, it periodically refreshes them so that any needed information could be restored back from this backup server. This server also needs to be backed up for more reliability. It provides additional reliability besides the reliability and failover mechanisms provided by ONOS.

The final and lowest layer is the network redirection layer. It redirects user requests for content towards the suitable caching nodes instead of the original VoD server. This is managed and modified by the SDN controller.

To achieve our proposed objectives mentioned in *Section IV*, OCCs and OCNs rely on the distributed SDN controller ONOS. They are implemented as ONOS applications to leverage its capabilities offered as follows:

- Mapping between OCCs and OCNs is based on the mastership mapping between ONOS controllers and the switches that connect these OCNs as illustrated in *Fig. 3*. Each switch is controlled by a single master ONOS controller and can be connected to other standby controllers that can takeover if the master fails. *Fig. 3,* illustrates a scenario of 3 ONOS controllers that have OCC running on top, and there are 4 switches connected to these controllers. The solid line depicts that a switch is mastered by the connected controller, whereas the dashed line depicts that a switch is connected to standby controllers. OCC is an ONOS application that is aware of the switches mastered by the hosting ONOS controller. Therefore, it controls the OCNs connected to these switches. According to the scenario in *Fig. 3.*, OCC (A) controls OCN 1, 2 and 3. OCC (B) controls OCN 4 and OCC (C) controls OCN 5. When the mastership relation changes either due to failover mechanism or load balancing, this reflects on the mapping between the OCC and OCNs as shown in *Fig. 4.* In this scenario, OCC (A) fails, so switch 1 and switch 2 fall back to the corresponding standby controllers, which will be their new masters.

- Fault-tolerance is one of the main features of ONOS. Once an ONOS controller failure is detected, then a new backup controller is selected which has a running OpenCache application to resume the work of the failed one.

- Accessing the global network state locally which is built by ONOS.

- Distributed primitives for managing distributed state are offered for ONOS application developers which enable creating different instances to manage their application state. They provide high availability, scalability and durability by offering strong and eventual consistencies.

- Leveraging the clustering capability that enables defining and grouping multiple ONOS instances in a cluster which are aware of the others and can communicate with them directly. Additionally, when an application is activated in one instance, this reflects on all other instances in the cluster automatically. This gives a great benefit of separating the caching control layer and the caching layer into two separate clusters, each with its own functionalities.

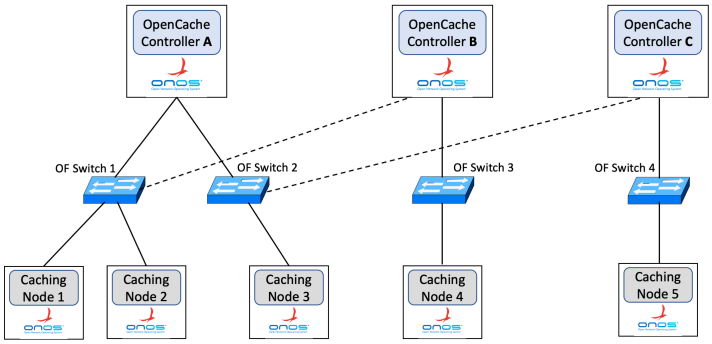

Fig. 3. Mapping OCCs to OCNs depends on masterships between ONOS controllers and switches

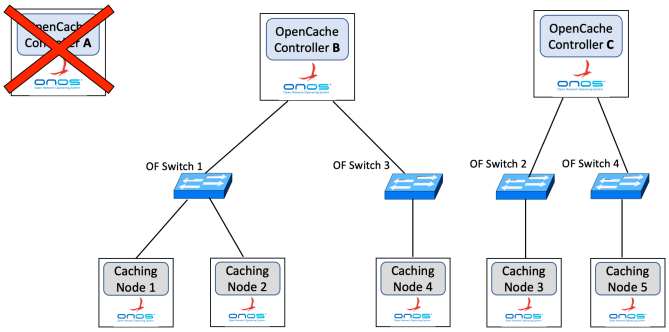

Fig. 4. Mastership changes due to ONOS controller failure or load balancing is reflected on the mapping between OCCs and OCNs

6 Features of the Proposed System

The synchronisation in this distributed architecture is achieved by sharing two types of data representations, namely, caching view and caching control state.
Caching view is a representation that depicts essential information about the existing OCCs, OCNs, mapping between them and a list of the services stored at each OCN. This information is important to be correct and consistent between all OCCs because it will be exposed to third-party application layer on top of them. So, it should be strongly consisted between OCCs to give these applications more freedom to connect to any OCC.
On the other hand, *Caching control state* is a detailed representation that contains all of the information related to caching control such as the user requests, responses, notifications, connected OCNs, contents ... etc. This caching control state is different at each OCC based on its sub-domain. The caching control state is eventually consistent between the OCC and its backups and uses the anti-entropy protocol for synchronisation. Likewise, OCNs need to synchronise their own caching states that contain all of the caching nodes related details.
The distributed architecture would have the following features:

6.1 Reliability

The flat model has an advantage in terms of robust failover mechanism because all OCCs have the same role and privileges, so any OCC in the cluster can simply take over a failed peer. Each OCC maintains its own caching control state based on its connections to OCNs. It has to periodically share its caching control state with its backup before the failure actually happens. In order to reduce the resource consumption (*i.e.* bandwidth, compute and storage), a single backup controller at least would be selected to share the caching state with. However, the degree of reliability (number of backup controllers) that share the caching state can be tuned based on the available resources. If ONOS did not fall back to the intended and expected backup controller(s) when a failure happens, this means that the new selected backup controller doesn't have a copy of the caching state of the failed controller, and it needs to retrieve this information from the dedicated backup server. Therefore, each controller sends a copy of its caching state to its backup controllers and another copy to the backup server too. This is shared in an eventually consistent fashion with anti-entropy synchronisation. Similarly, OCNs share their caching states with their backups. However, the backup nodes selection priority is based on the closeness to the node itself. OCNs also share the actual cached content.

6.2 Automated Elasticity

A monitoring and load balancing service is responsible for monitoring all of the OCCs without involving a man in the middle. It takes actions of load balancing if some thresholds (*i.e.* CPU, requests throughput) have been exceeded which indicate overloaded resource consumption. Load balancer will migrate caching nodes afterwards. If all backup controllers cannot handle the overload, then the load balancer needs to ask the VNFM to add a new vOCC to the caching control layer. The

new controller synchronises with other controllers by getting a copy of the global caching view and the caching control state of the nodes that will be migrated to it from the backup server. Additionally, monitoring and load balancing service is responsible for scaling down the control plane. So, when the controllers are underutilised, then it asks the VNFM to remove some controllers to reduce resource consumption. Similarly, caching nodes have the same elasticity feature but with the consideration of the original cache placement.

6.3 High Availability

The flat model redundancy offered by ONOS and the backup servers provide reliability and robustness that reduces the service disruption time in case of failures. Moreover, the load balancer collaborates with the VNFM to balance the load between the available resources and add/remove them as necessary to try avoiding service disruption due to overloaded resources. All of this lead to a highly available service. Additionally, since all OCCs are peers, then applications have the freedom to connect to any OCC.

7 Conclusion and Future Work

In this work, we proposed a distributed SDN/NFV based in-network content caching architecture as an extension to OpenCache, to overcome the main issues that it suffers from: scalability, reliability and high availability. We discussed the architectural design and technology decisions that we made for the caching platform distribution and functional components. We highlighted the role of the distributed SDN and NFV in virtualising, orchestrating and managing the key processes and control functions. Further, we presented the features of our platform design. The proposed platform is an open-source, distributed in-network caching that is expected to be highly available, reliable and with automated elasticity to enable serving the increasing VoD traffic quickly and efficiently. This work is still in the initial stages and under development, so in our future work, we plan to analyse the performance and measure the gains of the distributed platform opposed to the original non-distributed OpenCache. Several features would be examined and different QoE metrics would be evaluated. For example, the scalability of the distributed architecture would be measured by the increase in the VoD requests throughput. Whereas the effect of the failover mechanism would be measured by the response latency and buffer occupancy at the client side. Additionally, the placement of the OCCs and OCNs could be examined to measure its impact on the response latency and bitrate.

References

1. CISCO VNI: Cisco Visual Networking Index: Forecast and Methedology, 2016-2021. Technical report (2017)
2. Georgopoulos P., Broadbent M., Plattner B., Race N.: Cache as a Service: Leveraging SDN to Efficiently and Transparently Support Video-on-demand on the Last Mile.

In: 23rd International Conference on Computer Communication and Networks pp. 1-9 (2014)
3. Kreutz D., Ramos F. M. V., Veríssimo P. E., Rothenberg C. E., Azodolmolky S. and Uhlig S.: Software-Defined Networking: A Comprehensive Survey. In Proceedings of the IEEE, vol.103 (1), pp.14-76 (2015)
4. Mijumbi R., Serrat J., Gorricho J.-l., Bouten N., Turck F. De, and Boutaba R.: Network Function Virtualization: State-of-the-Art and Research Challenges. IEEE Communications Surveys & Tutorials, vol. 18, (1), pp. 236–262 (2016)
5. Broadbent M., King D., Baildon S., Georgalas N. and Race N.: Opencache: a Software-Defined Content Caching Platform. Proceedings of the 1st IEEE Conference on Network Softwarization, pp. 1–5 (2015)
6. Oktian Y. E., Lee S., Lee H., and Lam J.: Distributed SDN Controller System: A Survey on Design Choice. Computer Networks, vol. 121, pp. 100–111 (2017)
7. ON.LAB: ONOS, https://onosproject.org/
8. Linux Foundation: OpenDaylight, https://www. opendaylight.org/
9. Ongaro D. and Ousterhout J.: In Search of an Understandable Consensus Algorithm. In USENIX Annual Technical Conference, pp. 305–320 (2014)
10. ETSI Industry Specification Group (ISG) NFV: ETSI GS NFV 002 V1.1.1: Network Functions Virtualisation (NFV); Architectural Framework. Technical Report (2013), https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf
11. De Sousa, N. F. S., Perez, D. A. L., Rosa, R. V., Santos, M. A., and Rothenberg, C. E.: Network Service Orchestration: A Survey. Computer Communications (2019)
12. ETSI: Open Source MANO https://osm.etsi.org/.
13. Limbach F.: Cooperative Service Provisioning with OTT Players – An Explorative Analysis of Telecommunication Business Models. In 25th European Regional ITS Conference Brussels, Belgium (2014)
14. Liu Y., Point J. C., Katsaros K. V., Glykantzis V., Siddiqui M. S., and Escalona E.: SDN/NFV Based Caching Solution for Future Mobile Network (5G). In European Conference on Networks and Communications (2017)
15. Chiang W.-K. and Li T.-Y.: An Extended SDN-Based In-Network Caching Service for Video on Demand. In International Computer Symposium (2016)
16. Trajano A. F. R. and Fernandez M. P.: ContentSDN: A Content-Based Transparent Proxy Architecture in Software-Defined Networking. In IEEE 30th International Conference on Advanced Information Networking and Applications (2016)