Open Carrier Interface: An Open Source Edge Computing Framework

Marc Körner*
ICSI
koerner@icsi.berkeley.edu

Torsten M. Runge*
ICSI
runge@icsi.berkeley.edu

Aurojit Panda
NYU and ICSI
apanda@cs.nyu.edu

Sylvia Ratnasamy
UC Berkeley
sylvia@cs.berkeley.edu

Scott Shenker
UC Berkeley and ICSI
shenker@icsi.berkeley.edu

ABSTRACT

Edge computing is an emerging technology, which offers manifold performance improvements for applications with low-latency and high-bandwidth requirements. In order to lower the burden for the Network Service Provider to support edge computing, we introduce a generic and platform-agnostic open source edge computing framework implementation called Open Carrier Interface. The implemented prototype provides Application Service Providers with the opportunity to deploy software components directly at the edge of the network and without any operator intervention. We will elaborate and demonstrate that the developed framework and its interfaces (i) are easy to use for all involved parties, (ii) present a unifying abstraction layer for edge-based resource management systems and edge service architectures, (iii) and can deliver a significant performance impact on applications implementing the edge service paradigm.

CCS CONCEPTS

• **Networks → In-network processing**;

KEYWORDS

edge computing, computation offloading

ACM Reference Format:
Marc Körner, Torsten M. Runge, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2018. Open Carrier Interface: An Open Source Edge Computing Framework. In *NEAT'18: ACM SIGCOMM 2018 Workshop on Networking for Emerging Applications and Technologies , August 20, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3229574.3229579

1 INTRODUCTION

The last decade has witnessed a massive growth in Internet traffic. A large fraction of this growth can be attributed to the increasing popularity of video streaming services such as YouTube and Netflix, and due to growing adoption of networked Internet of Things (IoT) devices. Industry forecasts [1] predict that Internet traffic will continue to grow rapidly. Handling this increasing traffic volume requires upgrades to the network backbone, which ultimately adds to the cost of network services such as Netflix and Youtube. Furthermore, traditional client-server deployments, where clients connect to servers running within a remote datacenter, limits application latency and responsiveness, thus limiting the types of applications that can be supported by a network. In response to these challenges, application developers such as Netflix, Google, etc. have deployed services including cache appliances [9], search appliances [7], etc. at the edge of the network. These appliances are commonly deployed in a variety of facilities including Central Offices. We observe that the number of such appliances has been increasing steadily, and we believe network operators need to adopt a cloud provider like model to enable rapid deployment and reconfiguration of such edge services.

Thus, we propose a framework that enables network operators with the ability to open up their edge facilities to Application Service Providers, by offering edge computing. As an example, consider a third-party Application Service Provider selling an IoT device (say, a home thermostat or security camera). Whenever that device shows up at the edge of the network, the Open Carrier Interface framework starts the Application Service Provider implemented edge software at the network operator's Central Offices, which supports the IoT device with edge processing. Thus, even an early-stage Application Service Provider, who has few financial or physical resources, can offer the same level of edge support as giant companies, and therefore can compete with them on an even footing.

Two key principles dominate the proposed system: *On-demand* and *without operator intervention*. By offering edge support *on-demand*, the edge software component is started and executed at a particular network edge location when, and only when, one of the associated devices is deployed there. This means third parties Application Service Providers incur costs proportional to their resource usage, which offers a fair pay-on-use business model and lowers market barriers. By accomplishing this *without operator intervention*, the edge software component is distributed and invoked automatically, so no human assistance is required. We believe that this approach will change the application paradigm, turning client-server applications into Client-Edge-Server applications, and making the provision of these services much more competitive, while leveraging the one resource where carriers have a dominant position; their ubiquitous presence at the edge of the network.

*These authors contributed equally to this work.

NEAT'18, August 20, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5907-8/18/08…$15.00
https://doi.org/10.1145/3229574.3229579

2 BACKGROUND AND RELATED WORK

This section outlines the edge computing background and provides an overview of related frameworks.

2.1 NFV as Edge Computing Enabler

The Open Carrier Interface (OCI) concept refers to the edge computing paradigm and leverages recent developments at the carrier's Central Offices (COs). A CO is the first Network Service Provider (NSP) facility, where cables and connections from households and mobile base stations end up. COs are typically connected and served by a Broadband Access Server (BRAS), which provides the subscribers with access to the Wide Area Network (WAN). These facilities are typically dominated by proprietary signal processing and network equipment. However, recent developments in Software Defined Networking (SDN) and Network Function Virtualization (NFV) alter the hardware and software deployment and turn them into small data centers operated with general purpose server equipment. This development offers a larger flexibility and allows the carriers to use Virtualized Network Functions (VNFs) and scale them out on demand. Moreover, VNF updates can be easily deployed and stateful fail-over mechanisms enable application service operation without service disruption. AT&T [13] admitted that this process is ongoing and they have already equipped several COs with additional equipment. Their plan is to operate 75% of the COs with cloud like equipment and NFV by the year 2020. This trend is also supported by open source implementation efforts like the Central Office Re-architected as a Datacenter (CORD) [11] project. The CORD developers try to build an NFV platform for COs based on general purpose hardware.

2.2 Edge Computing Performance Analysis

Edge computing promises to increase the performance of several applications by using data locality. Moreover, it is also able to relieve the core network by addressing the increasing bandwidth demands caused by the increase of services, data volume, and IoT devices. Recent publications clearly indicate that there is a good chance of a reasonable performance gain regarding latency and bandwidth intense applications. In the area of virtual reality gaming for mobile devices, the EC+ [17] proposal verified a 10x higher frame rate with edge computing. Another Pokemon Go-like gaming application, using a framework called ENORM [15], demonstrates 20-80 % latency improvement and a data transfer reduction to the cloud by up to 95 %. In IoT sensor data aggregation some developers achieved a 2x processing time reduction on a weak embedded hardware platform and up to 10x improvement on commodity compute hardware with a framework called FADES [3]. For video analytics, the LAVEA [16] edge computing approach demonstrated an 1.2x till 4x faster processing based on the edge computing compared to the traditional client-cloud approach. The presented application examples demonstrate, that edge computing clearly improves the data processing for certain applications by utilizing data locality.

2.3 Edge Computing Frameworks

The standardization organizations IEEE and ETSI currently discuss approaches for edge computing architectures in their 5G initiatives and try to identify standardization opportunities and gaps. The IEEE established a 5G Working Group [8], which describes their vision and goals in a white paper [2]. The ETSI [5] proposes an architectural framework called Multi-access Edge Computing (MEC) [4], which offers third party Application Service Providers (ASPs) cloud computing capabilities at the network edge. These efforts are focused on mobile scenarios and the regarding support of mobile user equipment with mobile edge applications.

The Linux Foundation recently started the EdgeX Foundry [12] project for IoT edge computing. The project's mission is to develop an edge computing platform designed to facilitate hardware interoperability for IoT ecosystems. The edge platform itself is a microservice-based framework with focus on facilitating device services by standardized Application Interfaces (APIs) for IoT devices around core services. The approach is limited to microservices and has a strong focus on industrial IoT device domains. The project's execution environment is mainly focused on embedded devices like gateways or routers. Another IoT edge computing platform for embedded devices is the Everyware Software Framework [6] by Eurotech. The platform supports edge applications developed on the modular Open Service Gateway initiative (OSGi) for IoT devices. Both solutions are supposed to be deployed on site.

An approach to generalize the Content Delivery Network (CDN) concept for edge computing was introduced by Panda et al. [10], which is the basis for the OCI framework described in this paper. It introduces carrier networks as service delivery platforms for edge services. It further elaborates on the design for a client interface as well as a tenant (here referred to as ASP) interface for the deployment of third party applications. Moreover, it also provides an initial NFV-like description how edge services could look like and discusses some use cases which benefit from this approach.

The approach introduced within this paper extends the architecture by a further segreation and specification of the management entities. Furthermore, it provides an open source implementation. The presented OCI framework is designed to provide a generic solution for arbitrary Edge Service (ES) designs and underlying NSP resource management system. It can invoke dynamically application-context based ESs and presents a simple and straight forward multi-domain approach. We further advocate that the OCI is deployed in COs, since COs provide the opportunity and ability to host data center server equipment and do not come with the limitations of embedded devices. Thus, edge computing applications are not completely resource constraint and have to exist on embedded systems, like a typical home router. This provides an opportunity for a sophisticated processing beyond sensor data aggregation like the Netflix Open Connect appliances.

In a nutshell, the OCI is a platform-agnostic open source edge computing framework, which provides ASP with almost any possible freedom regarding the application architecture.

2.4 Edge vs Cloud Latency Study

Amazon Web Services (AWS) is clearly dominating the cloud market with 44.2% of total revenue [14]. For that reason, we investigated AWS as a common representative for cloud computing and conducted some latency and bandwidth performance measurements. The latency measurements are based on Round Trip Time (RTT) samples gathered on a conventional Digital Subscriber

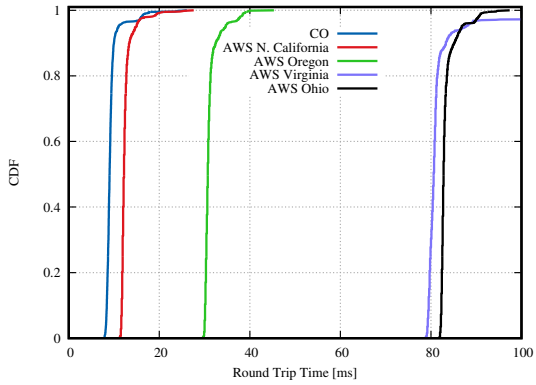

Figure 1: CDF of the RTT between a device using a typical home DSL Internet connection to different AWS locations in the USA

Line (DSL) home Internet access point providing a data rate of up to 30 Mbit/s. The samples were performed between a laptop connected to the access point in Berkeley Downtown and different Virtual Machines (VMs) in several AWS locations in the USA. The RTT was measured using the Internet Control Message Protocol (ICMP) and the corresponding throughput results were measured with *iperf* (v2.0.9). According to this samples, the throughput in general is limited by the bandwidth of the DSL connection, while the latency has a relation to the physical distance of the VM. Figure 1 depicts the collected latency measurements.

Based on the amount and distribution of AWS data centers within the USA, it is almost possible to reproduce an edge computing like behavior in Berkeley using the closest AWS cloud in the San Francisco Bay Area. The latency difference between this reference point and the local CO is only 3.5 ms on average. This might be sufficient to satisfy some low-latency application requirements, but could already jeopardize the reliable execution of some ultra-low latency applications. However, typically the latency to the next cloud is around 10x till 20x higher compared to the latency to the CO, as indicated by the results gathered with the Oregon and Virginia AWS data center.

3 ARCHITECTURE

The OCI is an open source edge computing framework solution under BSD license, which supplies ASPs with the ability to deploy edge applications at the NSP's COs. The approach aims an easy and automated deployment and lifecycle management procedure without direct interaction or negotiation with the NSP. It is designed to coexist with the NSP's NFV management system and can utilize the same compute, storage and networking resources. Thus, the OCI provides an opportunity to better utilize the existing NFV infrastructure while improving QoS. Hence, also the QoE for applications with OCI support is improved. The edge software component in this approach is further referred to as ES.

The OCI framework is composed of the following components, which are further described in detail in the next subsections:

- Global Open Carrier Interface Coordinator (GOCIC)
- Local Open Carrier Interface Coordinator (LOCIC)
- Resource and Orchestration Manager Adapter (RnOMA)
- Open Carrier Interface library (OCIlib)

As depicted in Figure 2 and already mentioned, the OCI is deployed on top of the NSP's NFV infrastructure in the CO. The architecture contains a centralized GOCIC. This component distributes all ESs which were uploaded to the LOCICs. The LOCIC is the edge wide OCI management entity and coordinates the ES instantiation. Therefore, it communicates with the RnOMA in order to request resources and to provide instructions and binaries to the NSP's Resource and Orchestration Manager (RnOM). The RnOMA also registers and de-registers ES at the LOCIC, depending on their lifecycle (e.g started, stopped) information. For the communication between the RnOMA and the RnOM the NSP needs to provide an implementation of the RnOMA's generic interface, which wraps the RnOM API and translates the commands given by the RnOMA. The RnOM is the NSP's edge resource management component. It is in charge of the NSP's VNF as well as the OCI managed ES. The RnOM is supposed to be controlled with high priority by the NSP's internal NFV management system, which has the overall control of the resource allocation and distribution. Spare resources can be used by the OCI to provide edge computing support for third-party ESs. In order to roll out and deploy an ES, the ASP uploads the Edge Service Application Package (ESAP) (see also Section 3.6) to the NSP, e.g., a portal or store, where it will been forwarded to the GOCIC. As soon as the GOCIC receives the new ESAP, it starts to distributes the ESAP to the edges.

A client application trying to connect to an ES must use the OCIlib's lookup method. This method sends an OCI name query to the LOCIC in order to resolve the corresponding ES Internet Protocol (IP) address. If the ES is already running, the LOCIC directly answers the client request with the IP address of the requested ES. Otherwise, the LOCIC starts to communicate with the RnOMA, asking for the opportunity to instantiate the ES. The RnOMA translates this query into the appropriate control messages for a specific RnOM, which eventually starts the ES if possible (ES and resources available). If a new ES was successfully started by the RnOM, the RnOMA registers the ES at the LOCIC. Thus, the LOCIC eventually answers the client query (if not timed out) with the IP address of newly instantiated ES.

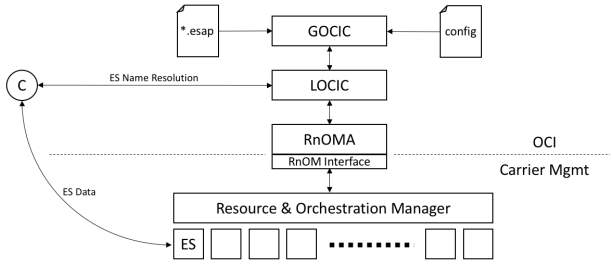

Figure 2: Open Carrier Interface Architecture Overview

3.1 Global OCI Coordinator

The GOCIC is the NSP's domain-wide OCI orchestration entity. Its main task is to distribute the ASPs's third-party ES without operator intervention or network knowledge across multiple network edges and domains. Thus, the GOCIC provides several interfaces.

The *NSP interface* is used by the NSP to manage the GOCIC and to provide all necessary configuration information. The NSP can specify and deploy a JavaScript Object Notation (JSON) configuration file, which provides the GOCIC with the IP addresses of all LOCICs within the same domain, as well as their metadata information like the geographical coded location. Optionally, this file can also specify information about peering OCI domains and their GOCICs IP addresses and metadata. The *LOCIC interface* is used to distribute the ASP's ESAPs to the LOCICs equipped COs. It is currently using secure copy to transfer these files. Future versions will provide the opportunity for the LOCICs to report status information. Thus, the GOCIC will be able to gather detailed knowledge about the current OCI utilization conditions and could also make management decisions in order to support resilience, fault tolerance, and scalability of ESs. The *ASP interface* is a REST interface, which is used to distribute the ESs. It allows to upload an ESAP to the GOCIC. Based on the included metadata, the GOCIC will afterwards distribute the ESAPs across the LOCIC equipped edges, as well as other peering OCI domains. The *inter-domain interface* synchronizes OCI domains and ESAPs exchange with other GOCICs, as further described in Section 3.7.

3.2 Local OCI Coordinator

The LOCIC is the OCI edge-wide management entity. It communicates with the GOCIC, the RnOMA and the clients. It maintains a list of all ESs, their names, IP addresses, lifecycle status, registration keys, and in future versions also potential fail-over information. The registration key is an integer value, which is generated by the LOCIC when a new ES is registered. The key serves as an access control mechanism to de-register an ES or manipulate its information. It is a simple security mechanism, which ensures that the de-registration can only be processed by the registering entity, which is in possession of the key. This prevents other RnOMAs or an intentionally manipulated component, to interact with other ESs. If the LOCIC receives an ES request from a client application, it checks its internal service list. A query for a running service is directly answered with the corresponding IP address of the ES. If the requested ES is not already executed, the LOCIC sends a request to the RnOMA to check the availability, required resources, and eventually to start the ES.

3.3 Resource & Orchestration Manager Adapter

The RnOMA serves as a driver for the underlying actual resource and orchestration manager, used by the NSP to manage the edge compute, storage and network resources. The RnOMA provides a generic interface to adapt an arbitrary resource management API. This API provides methods to check resources and sends inquiries to the RnOM to start or stop ES, or just obtain their status and network address. Future versions will also handle the ES orchestration information, e.g., for microservice-based edge services and their scalability. The interface must be implemented by the NSP in order to adapt the OCI to the RnOM's API.

3.4 OCI library

The OCIlib contains methods and templates for application developers to build Client-Edge-Server (CES) applications utilizing the CES paradigm and the OCI framework. It provides methods for the client application to obtain the ES IP addresses via LOCIC lookups. It also provides additional edge service templates, to simplify the development of OCI based CES applications and to support the integration within the OCI ecosystem, like the dynamic lifecycle management and component based stitching.

3.5 Edge Service Architecture

An OCI ES can be part of an application architecture. Depending on the ASP application design, the OCI ES is a mandatory or an optional software component, which is located between client and server components. It is executed by the OCI at the edge, in order to leverage the benefits of data locality. If an application uses an OCI ES component, it connects 1 till n clients with the centralized application server. It is a linking entity between client and server application. In order to satisfy the requirements of modern service-centric communication networks and microservice-based design patterns, the ES has its own internal complexity. The ES architecture can be distinguished into different categories. Any application design has different advantages and can be implemented depending on the application requirements. A single monolithic ES is easy to develop and might be sufficient to cache content at the edge. It can be scaled by replication of the application itself. In contrast, a microservice-based application can have scalability advantages, because every single microservice can be scaled individually. Moreover, a discovery service-based edge application is the right solution if several ES versions must be operated in parallel. Thus, a discovery service is able to connect the client with the appropriate ES version. Moreover, it could also implement an application-level load balancer. Eventually, the ES can also be a combination of all of these patterns. For instance, it can consist of a distributed structure with several microservices, which uses a discovery service.

3.6 Edge Service Application Package

An ESAP is the compiled and bundled form of an ES. It is a zip compressed file with the file extension `esap` and the following internal folder structure: `bin`, `lib`, `template`, and `meta`. The `bin` folder contains all binaries the edge service is composed of. The `meta` folder contains a JSON file, which describes the ES metadata information. This is typically the deployment location, described as a hierarchical domain name space like Domain Name Service (DNS). For instance `us.ca.berkeley`, which can be wild-carded with an asterisk like `us.ca.*` in order to deploy the edge service on every OCI facilitated edge in California. The metadata can further be used to specify Service-Level Agreement (SLA) information like required CPU, memory, or disk. At least the `meta` folder and the `bin` folder are mandatory, in order to build executable ESAP. The following folders are optional and provide the opportunity to build a sophisticated edge service architecture and deployment. The `lib` folder contains libraries, like shared objects, which might be required by the edge

service binary. In case of a microservice-based ES the ESAP must contain also the `template` folder, with JSON file describing the Directed Acyclic Graph (DAG) of the microservices and how they are connected amongst each other. It is further planned to support the developer with a tool-chain, in order to lower the hurdle to build the ESAP and keep the focus on the application business logic. Due to the possible variety of underlying bare metal platforms and resource managers we advocate to build the edge service as simple as possible.

3.7 Multi-Domain Support

The OCI multi-domain support enables inter-domain exchange of ESAPs between several GOCICs, as depicted in Figure 3. This allows to further distribute and share ESs with, e.g., other NSP. Thus, a client (application) does not notice a domain change and can proceed to communicate with its initial configuration and ES within the network of cooperating domains. The inter-domain ESAP exchange can basically be realized in a proactive or reactive manner.

- The *proactive* exchange regularly synchronizes the ESAPs between the GOCICs of different domains. This can for instance be realized based on a specific time interval or triggered by an upload of a new ESAP.
- The *reactive* exchange is triggered by a client ES request. If the available request refers to an other domain or a not available service, the LOCIC can notify the GOCIC. Thus, the request is delegated all the way up to the GOCIC and to other domains in order to retrieve the corresponding ESAP.

The reactive approach requires additional communication between the LOCIC-GOCIC and/or GOCIC-GOCIC. The current implementation uses the inbuilt proactive exchange, which is also currently used for the ESAP distribution to the LOCICs. The connections between multiple GOCICs instances for the exchange of ESAPs can be organized in many ways. For instance, they might use an overlay mesh network (e.g. Peer-to-peer (P2P)) or hierarchical deployment. The within this paper introduced implementation presents a technical proof of concept and does not yet consider any billing or SLA enforcement between NSPs. Since this issues are manifold, this approach deals only with general technical feasibility and the P2P mechanism used to also distribute an ESAP to a LOCIC.

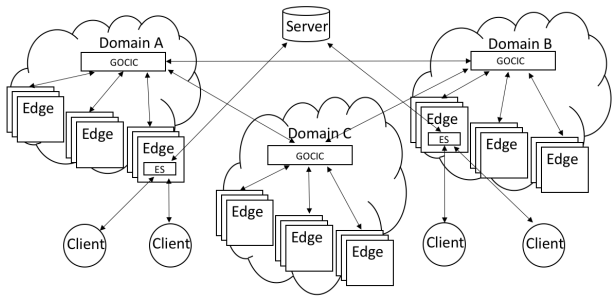

Figure 3: OCI Multi-Domain Architecture

4 EVALUATION

This section verifies the OCI implementation and demonstrates its performance capabilities.

4.1 Testbed

The OCI performance evaluation was conducted with two server nodes. Each node is equipped with 2x 4 Core Intel Xeon CPU E5-2640 v3 operating at 2.60 GHz maximum clock frequency, 128 GB DDR4 RAM, and an Intel 10 Gigabit NIC 82599ES. The nodes use a Debian Linux operating system with a 4.12.0-1-amd64 kernel. Both nodes are interconnected via a gigabit switch.

4.2 Edge Service Lookup Time

The ES lookup time is a crucial performance indicator, because in a worst case scenario, the LOCIC has to be able to serve all clients, which are subscribed to the CO. Thus, we measure the lookup time for client requests to obtain a random ES entry under different conditions.

4.2.1 Single Query Lookup Time. At first, we measured the idle system lookup time for a single ES resolution query. Figure 4 depicts the CDF lookup time for a random ES in relation to different amounts of ES entries. Each curve is composed of 1k samples, collected successively with a random pause between 1.5 and 2.5 seconds. The pause was added to force the system to an idle state, so that optimization mechanisms like caching cannot influence the measurement results. It can be observed that the average idle lookup time for up to 1k ES entries has a mean value of 0.9 ms, which is negligible compared to the sample with 100 entries. Further observations for 10k, 20k, and 50k entries indicate that the lookup time for a single query increases only minimal compared to the amount of entries. However, we assume that the support for 1k till 2k ES entries per edge might be sufficient for today's given applications with edge computing support.

4.2.2 Load Based Lookup Time. For the load based evaluation, we conducted measurements of the OCI lookup time under various load conditions. Therefore, the LOCIC and the developed benchmarking tool were located on a dedicated testbed node. This introduces an additional network delay of 0.33 ms on average. Nevertheless, this configuration is required since the LOCIC and the benchmarking tool produce almost full CPU utilization during high load conditions. The load generator of the benchmarking tool sends up to 3k lookup requests/s, simulating multiple parallel client queries. The generated requests follow a Poisson distribution, which implies that the overall amount of lookup requests might arrive in bursts. The requests are generated by multiple parallel executed threads and each thread sends an average request of one request/s. Fig. 5 shows the required time for an ES lookup in milliseconds, in relation to the offered load in requests/s (logarithmic scale). Each curve reflects a different number of ES entries. Since the mean values are influenced by outliers, the figure denotes the lookup time of the 90th percentile. For up to 1k entries, the lookup time is almost constant with approx. 1.8 ms. This means that the load in this area is independent of the amount of ES entries. Thus, each lookup request experiences the same mean waiting time before it is processed by the LOCIC. The lookup time increases at offered loads higher than

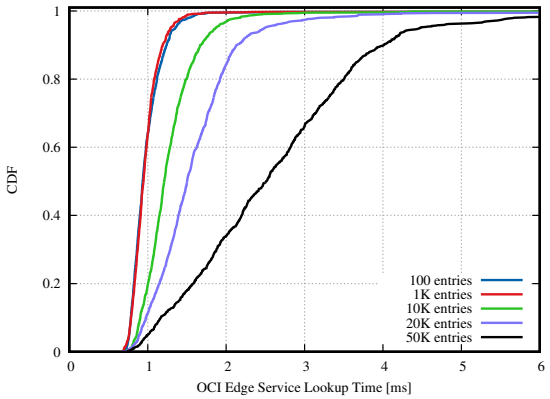

Figure 4: OCI single query edge service lookup time

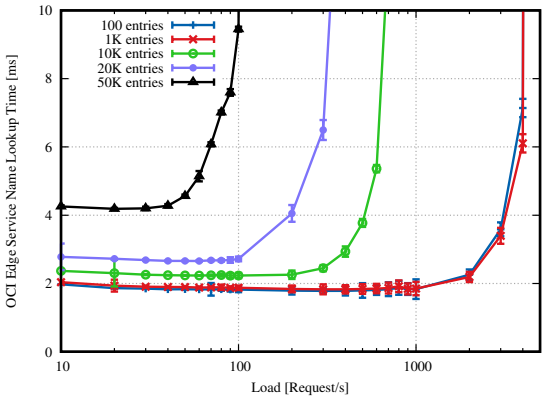

Figure 5: OCI load based edge service lookup time

1k request/s, since bursts cause backlogs and increase the waiting time for lookups. The samples for 10k, 20k, and 50k entries show an similar behaviour. They have almost constant lookup times of 2.2, 2.7 and 4.2 ms for low offered loads. With an increasing amount of lookup queries/s the load reaches a critical point at 300, 100, and 50 request/s when an increasing delay of the lookup can be observed.

Consequently, the OCI prototype performance is sufficient to cope with the usual amount of CO subscribers, which is traditionally around 10k. Let's imagine that we got 50k subscribers in a dense populated area and 90% of them try at 9 p.m. +/- 5 minutes to request the same video on-demand ES. This adds up to a total of 45k requests within 10 minutes, which is 75 requests/s. This is a challenge, which can even be addressed with 50k ES entries. Although the investigation covered a worst case scenario, the LOCIC's lookup time plus the CO network latency, as investigated in Section 2.4, comes to around 12 ms for the total ES lookup time.

5 CONCLUSION

We have implemented an open source edge computing framework. The prototype implementation contains a generic resource management interface, which allows NSPs to easily adapt and deploy the framework on their NFV enabled CO infrastructure. Thus, the framework presents an unified abstraction for edge computing by supporting several underlying resource management systems and ES architectures. It further supports third-party ASP with the ability to deploy ESs without network knowledge and in an automated manner, while simultaneously take the on-demand lifecycle management into account. The preliminary evaluation results indicate that the performance would even be sufficient to address today's amount of CO subscribers and deliver reasonable performance gains for numerous latency or bandwidth critical applications.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their work and valuable feedback. We would also like to thank Zafar Qazi, Xiaohe Hu, Henrik Klessig, Murphy McCauley, Amin Tootoonchian, and Peter Gao for their comments and suggestions on the implementation and evaluation. This work was supported by a fellowship within the FITweltweit program of the German Academic Exchange Service (DAAD).

REFERENCES

[1] Cisco Visual Networking Index. 2016. *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021.* Technical Report. Cisco, Inc.
[2] IEEE 5G Technical Community. 2017. *IEEE 5G and Beyond Technology Roadmap White Paper.* Technical Report. IEEE.
[3] Vittorio Cozzolino, Aaron Yi Ding, and Jörg Ott. 2017. FADES: Fine-Grained Edge Offloading with Unikernels. In *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems (HotConNet '17).* ACM, New York, NY, USA, 36–41. https://doi.org/10.1145/3094405.3094412
[4] MECISG ETSI. 2016. Mobile Edge Computing (MEC); Framework and Reference Architecture. *ETSI, DGS MEC* 3, 18 (2016), 1–18.
[5] ETSI MEC Initiative. 2017. Multi-access Edge Computing. http://www.etsi.org/mec.
[6] Eurotech. 2018. Edge Computing Platform. https://esf.eurotech.com/docs/edge-computing-platform.
[7] Google. 2018. Google Search Appliance. https://enterprise.google.com/search/.
[8] IEEE 5G Initiative. 2017. IEEE 5G Technology Roadmap. http://5g.ieee.org/roadmap.
[9] Netflix, Inc. 2018. Open Connect Overview. https://openconnect.netflix.com/Open-Connect-Overview.pdf.
[10] Aurojit Panda, James Murphy McCauley, Amin Tootoonchian, Justine Sherry, Teemu Koponen, Syliva Ratnasamy, and Scott Shenker. 2016. Open Network Interfaces for Carrier Networks. *ACM SIGCOMM Computer Communication Review* 46, 1 (2016), 5–11.
[11] Larry Peterson, Ali Al-Shabibi, Tom Anshutz, Scott Baker, Andy Bavier, Saurav Das, Jonathan Hart, Guru Palukar, and William Snow. 2016. Central Office Re-architected as a Data Center. *IEEE Communications Magazine* 54, 10 (2016), 96–101.
[12] The Linux Foundation Projects. 2018. EdgeX Foundry. https://www.edgexfoundry.org/.
[13] Yevgeniy Sverdlik. 2016. Telco Central Offices Get Second Life as Cloud Data Centers. http://www.datacenterknowledge.com/archives/2016/02/03/telco-central-offices-get-second-life-cloud-data-centers.
[14] Rob van der Meulen. 2017. Gartner Says Worldwide IaaS Public Cloud Services Market Grew 31 Percent in 2016. https://www.gartner.com/newsroom/id/3808563.
[15] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos. 2017. ENORM: A Framework For Edge NOde Resource Management. *IEEE Transactions on Services Computing* PP, 99 (2017), 1–1.
[16] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. 2017. LAVEA: Latency-aware Video Analytics on Edge Computing Platform. In *ACM/IEEE Symposium on Edge Computing (SEC'17).* ACM, New York, NY, USA, Article 15, 13 pages.
[17] Wuyang Zhang, Jiachen Chen, Yanyong Zhang, and Dipankar Raychaudhuri. 2017. Towards Efficient Edge Cloud Augmentation for Virtual Reality MMOGs. In *ACM/IEEE Symposium on Edge Computing (SEC'17).* ACM, New York, NY, USA, Article 8, 14 pages.