Viscum: A New Light Weight Software Architecture of Mobile Edge Computing

Feng Li[1], Jae Won Chung[2], Sriram Sridhar[1], and Jamal Hadi Salim[3]

[1]Verizon Labs, 60 Sylvan Rd, Waltham, MA, 02145, USA
[2]ViaSat, 300 Nickerson Road, Marlborough, MA, 01752, USA
[3]Mojatatu, 15 Fitzgerald Rd, Nepean, ON, K2H 9G1, Canada

Multi-access edge computing(MEC) is a network architecture that originated in the cellular world to allow processing of tasks closer to the cellular customer. It has, however, evolved to be generic to apply to any network for deploying applications and services as well as to store and process content in close proximity to mobile users. Current MEC solutions require either a) deploying a specialized (often proprietary) mobile edge platform or b) modifying existing applications (known as application splitting). In this paper, we propose an alternative light weight mobile edge computing architecture which utilizes existing Linux kernel mechanisms, namely: Traffic Control (TC) utilities, and network Namespaces. Our solution is able to i) deploy real-time applications onto mobile edge device without any modification to meet their low latency requirements; ii) provide computational offloading from either battery powered mobile devices or back end services in cloud; iii) potentially convert any Linux based network devices (e.g. Wifi router, eNodeB) into application server without introducing new hardware.

I. Introduction

The users' demanding on data rates and quality of service (QoS) are exponentially increasing as the evolution of new generation of mobile technology (5G). Although new mobile devices (UEs) are more or more powerful in terms of CPU processing power, they may not be able to handle high computational applications (e.g. realtime video object detection with GPUs). On the other hand, the battery on mobile devices may often restrict the user to fully enjoy high data rate (1Gbps on 5G) to achieve longer batter life. These new challenges on future mobile network motivate the development of mobile cloud computing (MCC). With the mobile cloud computing model, users can exploit computing and storage resources inside cloud though mobile provider's core network (CN) and the Internet.

The MCC brings several advantages: 1) offloading the battery consuming computation task to the cloud; 2) allowing mobile users to access complicated service or applications; 3) provide high volume cloud storage for mobile users. On the other side, the MCC introduces additional network load on both radio and core network works and introduces higher latency since data has to be sent to server farm inside cloud. Therefore, from network topology perspective, MCC is moving the processing server far from the users.

To address the problem of extra latency introduced by MCC, industrial researchers proposed a newly emerged concept, *MEC – "Multi-Access Edge Computing or Mobile Edge Computing"*[1], which attempts to move cloud services to mobile edges, a proximity of user equipments (UEs). In this way, MEC can offer significantly lower latencies and allow more close-knit interaction between service provider and customers when compared with MCC model. In addition to low latency, MEC also potentially reduces the traffic load of the core network, and can host applications and services in an economic way [2], [4]. Application developer and content providers can also take advantages of close proximity to cellular subscribers by using realtime RAN information to develop new services [2].

However, the software development model in MEC environment splits traditional *client-server* model into three parts: client, near server, and far server [1]. As depicted in Figure 1, a MEC requires applications to be split into small tasks (e.g. micro-services) with some of the tasks performed at the local or regional clouds as long as the latency and accuracy are preserved. Therefore the current MEC architecture arises a number of challenging issues such as offload in distributing sub-tasks of an application among edge and remote clouds.

To resolve the issues with current software development model in MEC, we propose a new light weighted MEC architecture — *viscum*[2], based on existing Linux Kernel mechanism, namely Linux Traffic Control (TC) and Network Namespaces. Compared with current MEC architecture, our new approach is able to:

- deploy off-shelf realtime application on to mobile edge nodes (e.g. eNodeB and Wifi HotSpot) without any modification on application.
- provide computational offloading from either battery limited UE devices or backend cloud services to mobile edge nodes to reduce the traffic in mobile providers' core network and radio access network (RAN).

[1]In this paper, we interchangeably use Multi-Access Edge Computing and Mobile Edge Computing for the acronym MEC.

[2]We name this architecture as viscum to mimic the small service deployed on existing edge routers.

- convert any Linux based network node into application servers without using container's sky high network stacks or introducing any new hardware.

The reminder of this paper is organized as follows: Section II provides the background on MEC architecture and MEC software model; Section III describes *viscum* architecture in detail and shows the advantages of viscum with two examples; Section IV discusses the future work of this study, and Section V concludes this paper.

II. Related Work

In recent years, several MEC concepts have been proposed to move computational and storage resources out of the remote cloud (public or private) and closer to user equipments (UEs) [2], [3], such as small cell cloud (SCC), mobile micro cloud (MMC), fast moving personal cloud etc [2]. Among MEC researchers, European Telecommunications Standards Institute(ETSI) is currently deeply involved in standardization activities to integrate MEC into the mobile networks [2]. In this section, we briefly describe the ETSI's software development model.

A. ETSI Software Development Model

Because the MEC point of presence (PoP) is different from a traditional cloud PoP, MEC software development presents new challenges [1]. In the ETSI proposed a new software development mode, MEC point of presence consists of three "locations": Client, Near Server (Edge Server), and Far Server (Remote Server). The client location can be traditional smartphone or other UE devices running dedicated client applications. The "Near Servers", MEC hosts, are deployed at the network edge, providing computation, storage and network resources; and "Far Servers" acts as traditional cloud servers.

The key aspect of ETSI software development model(Figure 1) is to split application into: terminal device components, edge components and remote components. Unfortunately, this aspect creates additional tasks for developers with respect to a more traditional client-server architecture, because additional processing stage at edge nodes must be added into application workflows. Thus, application splitting may bring new challenging problems such as load balancing between edge hosts and cloud hosts in addition to simply splitting cloud services into two parts.

In addition to service splitting, ETSI MEC model requires "DNS handling" (shown in Figure 1) to redirect traffic from remote server to edge servers. It may require additional work to enhance DNS servers to support name resolution based on user's geographic location. However, the edge server geographically close to customer may not be a nearby server in network topology.

The last but not least, edge components are usually running in ISP's network realm, while cloud components are from different providers. Therefore, deploying ETSI model may require ISP to provide management interfaces to third party through a portal to support cross domain load balancing between edge nodes and cloud nodes. It may trigger numerous technical as well as non technical issues (e.g. legal issues).

B. Virtualization in MEC

As Figure 1 shows, each block may not represent physical node in the mobile network, but rather software entities running on the top of a virtualization infrastructure [4]. Thus, virtualization system for service delivery in MEC is set to play a crucial role [5].

Linux Containers (LXC) and Docker, as an operating system level virtualization technology, which allows user to define use multiple virtual instance of resources of a host and its kernel. However, because containers make extensive use of namespaces such as Cgroup, IPC, network, mount, PID, user and UTS, then eventually introduces a certain amount overheads [5]. Avio *et al.* measures the CPU usage of Docker with different type services, and they noticed that container overheads increased with the number of game servers [5]. Container based MEC implementations might be light weight for "wright box" servers [6], but Avino *et al.* still "put a Linux box behind a wifi hotspot" [5] instead of directly deploying their light game service onto the *true* edge node — a wifi router.

III. Methodology

In our practice of running the largest mobile wireless network in the world, we notice the resource usage of our edge devices (eg. eNodeB) are pretty low. For example, generally most of our edge routers' CPU usage is much less than 30%. Thus, we are thinking to *reuse* these *idle* edge nodes and convert them as computational nodes in MEC. To reduce the overhead introduced by container based MEC architecture, we propose viscum, a *real light weight*, process level virtualization based MEC architecture (shown in Figure 2) which can be deployed on low end edge routers. This section describes viscum, and gives a step-by-step instruction to build a viscum host, and shows its advantages with two use samples.

A. What is Viscum ?

Inspired by the article written by Feilner *et al.* [7], we propose a new MEC architecture based on Linux Namespace without using the "sky high" network stacks in docker containers. The key aspect of viscum is using Linux network namespaces as the application containers with a traffic redirector built with Linux Traffic Control (TC) utilities. As Figure 4 shows, viscum consists of four major components:

- **network namespace**: serves as a container provide resources for application servers. Note, the application running inside viscum's network space does not require any modification or "splitting" as "edge component" described in [1].
- **traffic director**: redirects desired traffic to from data plane to viscum application services. In current viscum implementation, we simply use Linux Traffic Control (TC) utilities to build a simplified traffic director.

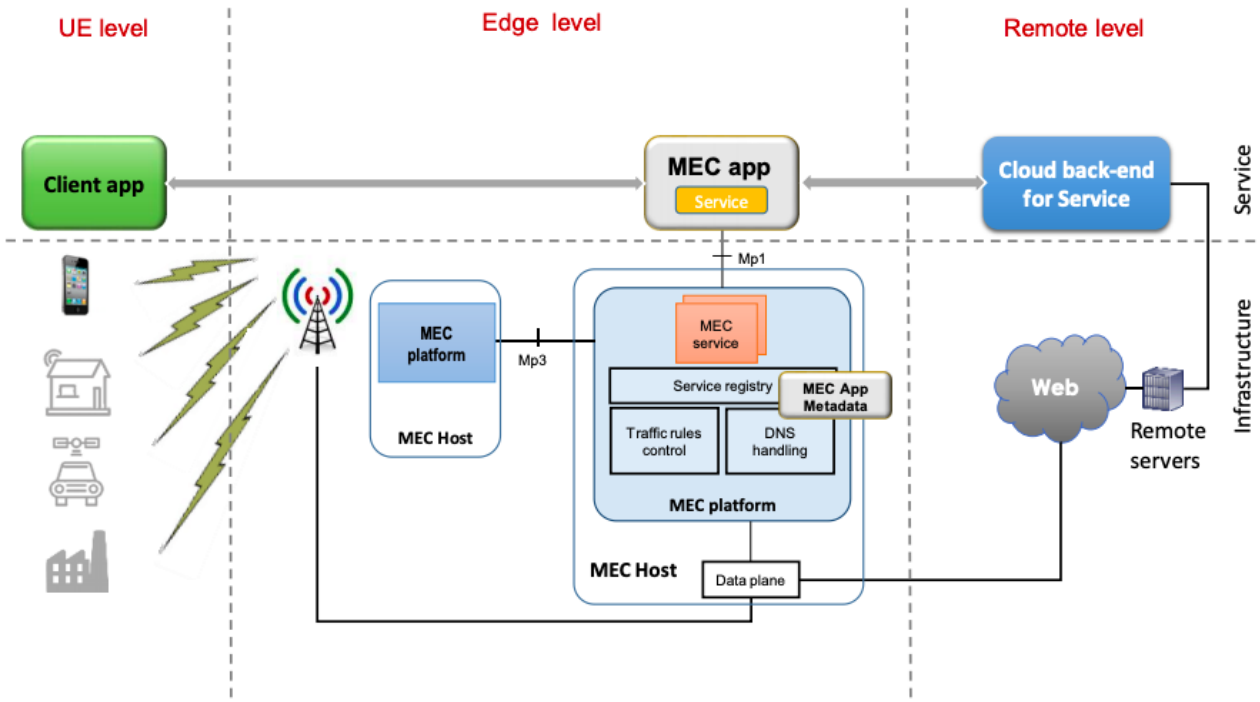


Fig. 1. ETSI MEC Software Model

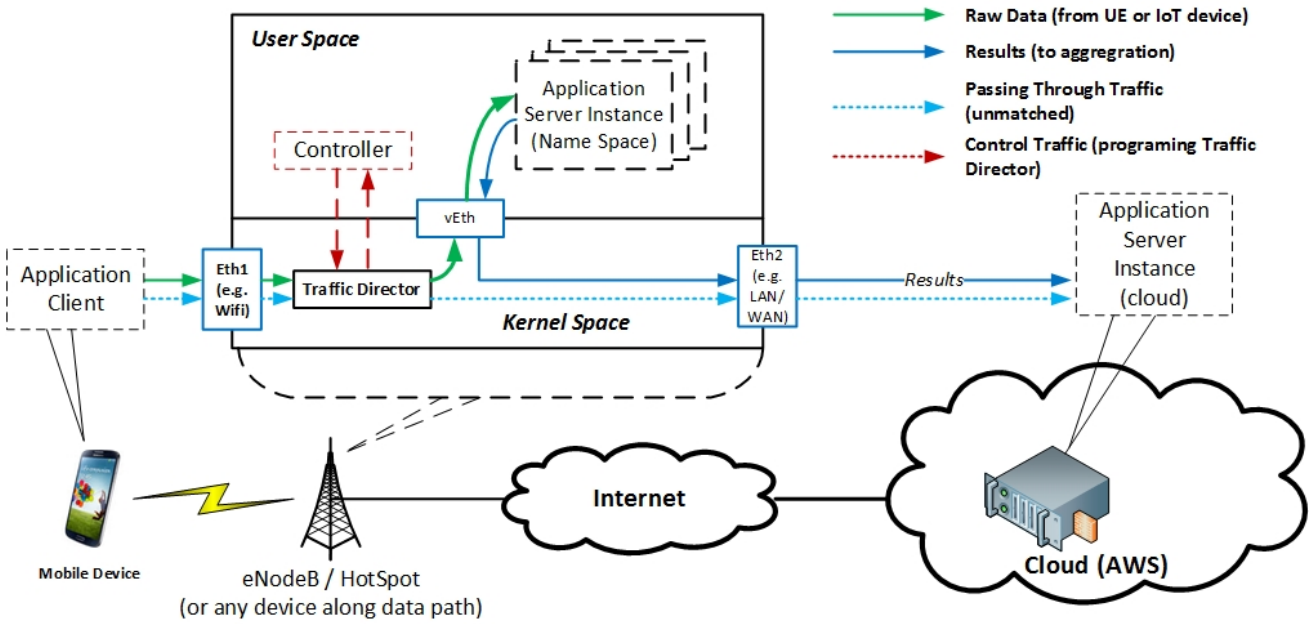


Fig. 2. Viscum Architecture

- **a pair of veths**: provides direct connection between two namespaces: the host native namespace (H_VETH) and network namespace hosting applications (NS_VETH).
- **controller**: provides management functions of viscum components, such as customizing TC rules, creating/tearing down namespaces etc. In our pilot study, we reuse the existing control plane functions in 3GPP architecture without creating any new management framework.

B. How to Create a Viscum Host?

Figure 3 gives a step-by-step guide to create a viscum host (a bash shell function create_ns_tc() on an Linux based Host

```
1   create_ns_tc() {
2         echo "============= Creating netns and tc rules ============="
3         (
4         set -x
5         sudo sysctl -w net.ipv4.ip_forward=1
6
7         # Create VETH device, configure the host side first.
8         $IP link add $H_VETH address $VETH_HOST_MAC type veth peer name $NS_VETH
9         $IP link set $H_VETH up mtu $MTU
10        $IP addr add $IP_ADDRESS_H/16 dev $H_VETH
11
12        # Create the namespace
13        $IP netns add $NAMESPACE
14        $IP -n $NAMESPACE link set dev lo up
15
16        # Create the namespace end of the veth device.
17        $IP link set dev $NS_VETH netns $NAMESPACE
18        $IP -n $NAMESPACE link set dev $NS_VETH up address $VETH_NS_MAC mtu $MTU
19        $IP -n $NAMESPACE addr add $IP_ADDRESS/16 dev $NS_VETH
20        $IP netns exec $NAMESPACE ip route add default via $IP_ADDRESS_H
21
22        # You can see the configuration from the inside of your namespace as follows:
23        sudo ip netns exec $NAMESPACE ifconfig
24
25        # Then to redirect packets to the container
26        $TC qdisc add dev $PHY_ETH ingress
27
28        $TC filter add dev $PHY_ETH parent ffff: prio 1 protocol ip \
29        handle 0x1 flower classid 1:1 dst_ip $IP_ADDRESS_T \
30        action skbmod dmac $VETH_NS_MAC index 1 \
31        action mirred egress redirect dev $H_VETH index 1
32        )
33  }
```

Fig. 3. Sample BASH Script to Create Viscum

AP[3]:

1) Line 8-10 creates the host side veth device ($H_VETH) with a given MAC address ($VETH_HOST_MAC), MTU ($MTU) and IP address ($IP_ADDRESS_H). Note, Line 8 also specifies the peer veth device ($NS_VETH) even the peer veth has not been created.
2) Line 13 and 14 create the viscum namespace ($NAMESPACE) to host application services.[4]
3) Line 17-20 create the veth inside the namespace created in step 2 with a given MAC address, IP address and MTU (similar to Line 8-10).
4) Line 23 validates the network configuration with ifconfig running inside name space created in step 2 (optional).
5) Line 26-31 create TC rules to redirect desired traffic to the namespace. In our proof of concept project, we simply redirect traffic with matched by given IP address ($IP_ADDRESS_T) to namespace while we can extend the traffic redirector to a fully functional router with a complicated TC rules

The whole process to create a viscum host is straight forward and could be done with basic ip netns commands. Note, in Step 5, it is necessary to use TC's skbmod action to modify packet's MAC addresses, otherwise Ethernet driver would drop the packets.

C. How to Use Viscum ?

We choose two representative MEC applications in this study to show how to use viscum. One is *Automatic License Plate Recognition* (openalpr)[5] as representative of services requiring high computational power and low latency, and the other is *Iperf3*, a throughput measurement service requiring to be deployed as close as possible to mobile edges.

[3]http://perform.wpi.edu/whostap/

[4]please reference linux ip man page about start/stop process inside network namespace.

[5]https://www.openalpr.com/

1) Automatic License Plate Recognition (OpenALPR): The first viscum example we chosen is a video/image analytics application (openalpr) used surveillance camera or smart phone camera. For example, the task is to detect vehicles' plate number from a camera of a toll booth. In a traditional approach, when a vehicle passes the toll booth, the camera takes a picture and sends it up to a data center where the software components for plate detection, vehicle databases are located. A lot of bandwidth can be saved by splitting the job and performing plate detection functions on the edge (e.g. eNodeB) or at the camera itself if the camera has enough computational power.

However, plate detection (or face detection) program are typical computational intensive application and may need GPU support. Running plate detection or face detection program on the battery powered cellphone or dash camera will significantly reduce their battery life. If we can run plate detection component at the edge router, we can offload the computational task from cellphone to wire powered eNodeB. Meanwhile, we can only send the possible plate number along with meta data to remote cloud, for further processing and database lookup. In this way, we can process the data close to the user with low latency, saving the battery life of UEs, and reducing traffic in core networks.

In our demo, we place *vanilla* open source version of OpenALPR as our plate detection component. OpenALPR is an automatic number-plate recognition library written in C++, which is distributed in both commercial cloud based version and open source version. OpenALPR makes use of OpenCV and Tesseract OCR libraries, and it could be run as a command-line utility, standalone library, or background process. The software also integrates with video management systems (VMS) such as Milestone XProtect[6].

We simply start an openaplr process in a viscum namespace created on a fully functional Linux based Host AP with Intel IEEE 802.11 b/g/n wireless card, 4GB memory, Ubuntu 14.04 with upgraded 4.15.0 kernel. Meanwhile, we write a simple Android application on a commercial Samsung S7 handset to take pictures of license plate and send them to a cloud based OpenALPR service through its wifi interface.

As Figure 2 shows, when the uploading data flow hits our viscum host AP, the TC based traffic redirector found its destination IP address matches the target IP address, which is the cloud openalpr server's IP address in this demo. The traffic redirector simply modifies the MAC address and sends the data flow to Namespace through the veth pairs instead of forwarding it to the Internet. The openaplr inside viscum namespace processes the picture and sends the pure text results to a cloud based mysql database along with meta data (timestamp etc). The round trip latency between Samsung S7 and viscum host AP is under 5 ms, while the RTT between host AP and cloud server is more than 60 ms. Thus, viscum version preserves the low latency nature of MEC without any development efforts. Meanwhile, when viscum host AP processes the image at the mobile edge, it reduces the data volume between the AP and data center from hundred kilobytes to hundred bytes. Reducing data volume to cloud data centers may lower customer's cloud bills, especially for customers using expensive satellite links. In this sample, the viscum openaplr service is functional same as the openalpr cloud service, and we can think the viscum node is a "part of cloud".

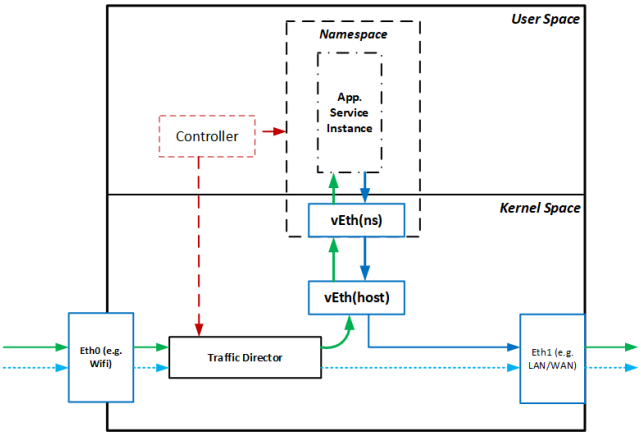


Fig. 4. Viscum MEC Host

2) Throughput Test Server on the Edge: Network connection speed is one of the most important performance metric for customers and ISPs. There are various speed test providers, such as Ookla, RootMetrics, etc. Most of them provide speed test services which measure the bandwidth and latency from end users' Internet connection against one of thousands geographically dispersed servers around the would. However, most of customers and service providers only have interests to the "last mile" link capacity which is usually the bottleneck along the Internet path. Thus, speed test service providers usually recommend customers to run tests against "the server geographically close to you."

Unfortunately, the current geographic based server selection may not work very well in future network. The future wireless network (5G) promises to provide more than 1 Gbps link speed in near future, which makes geographic based speed test server framework obsolete. Because the link capacity of 5G may be even larger than wired connection to small business users[7], the current speed test may not report "inter-domain" capacity instead of the wireless link speed [8]. Therefore, we may need a new MEC based speed test framework for 5G to accurately measure wireless link capacity of 5G.

To solve the limitation of current speed test frameworks, in our pilot study, we deploy an iperf3 server in our viscum Linux AP. In this way, we are able to accurately measure the wireless link capacity between the UE and AP without any source code or configuration changes. Note, the same test setup can still measure the end to end capacity if we simply disable the traffic redirector by clearing TC rules.

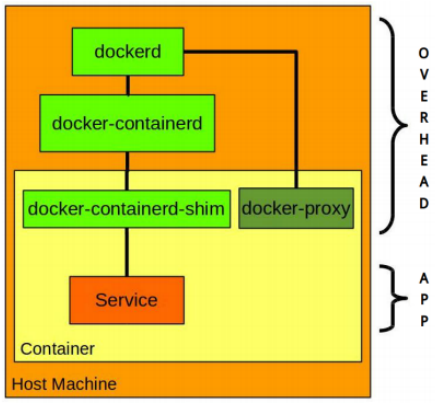
As additional benefits, our new viscum based test setup is able to reduce usage bill from cloud providers as well as traffic

[6]https://en.wikipedia.org/wiki/OpenALPR

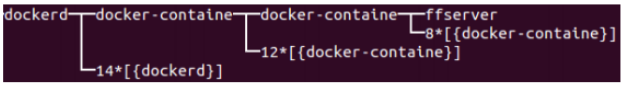
[7]Ookla currently requires to have minimum 1Gbps connection speed.

inside mobile core networks. The bandwidth measurement traffic is meaningless on non-bottleneck links. When thousands of users run speed test concurrently on 5G devices, it creates a huge amount traffic in ISP's core network which impacts core networks' performance. With the speedtest service on viscum host, there would be no such "meaningless" bandwidth measurement traffic in ISP core network while it can still provide accurate measurement on wireless links for customers. I

D. Discussion: Why We Choose Viscum ?



(a) Diagram of the Docker Processes Tree



(b) Screenshot of the output of `pstree`

Fig. 5. Docker Process Tree [5]

1) Why not using Docker in Viscum ?: The common trend of various MEC concepts including ETSI reference model is to bring cloud on to the edge of mobile network. The network virtualization exploiting NFV principles is a necessity in order to take the advantage of MEC model. The IT industry is currently investing on the use of virtual machines and lightweight containers (e.g. docker) in MEC. Although widely used docker containers framework promises to have high portability and scalability between developers and operators [5], [6]. However, virtualization comes with prices: resource duplication and overhead increase.

Figure 5 shows the docker process tree [5]. The top two processes, *dockerd* and *docker-container-rd*, are persistent, while the rest process are non-persistent. Therefore, one docker container introduces at least 4 more processes for one MEC service. Thus, operating system level virtualization may introduce too much overhead for resource limited devices such as wifi hotspots or eNodeBs.

On the contrary, viscum is built purely on Linux network namespaces, which provides network resource sharing on process level. Therefore, viscum is an even lighter solution than container based MEC solution. In this way, viscum has the potential to be deployed on wifi hotspot or eNode because it introduces less overhead than container based MEC solution.

Moreover, viscum or other process level virtualization can support a large variety of devices with different kernel builds. The only system requirement of viscum is to compile kernel with namespace support (CONFIG_NAMESPACES=y). From 2.16.32, Linux kernel supports namespaces while docker containers are relatively "new" concepts. For example Red Hat only supports docker from RHEL 7, while Linux embedded devices with older kernel could use be converted to viscum MEC hosts.

2) Why not do software splitting ?: Software splitting is an important necessity of ETSI MEC reference model. Splitting existing cloud based solution is not a trial task, while system architects need carefully splits traditional cloud based service into edge component and cloud component. It may require development and/or management overhead. Thus, the ideal way of MEC is able to deploy cloud service onto edge node directly without any modifications as viscum does.

However, viscum may not be able to host complicated services such as YouTube which requires large storage space. But viscum is good to offloading computational intensive micro services such as face detection or plate detection onto edge nodes from either cloud server or battery powered devices. Because of there is no development efforts, viscum would accelerate application development process in MEC.

3) Why not using DNS to redirect traffic ?: Different from ETSI reference model, viscum does not rely on DNS modification to redirect traffic to edge nodes. Enhancing DNS service with IP geographic database may not be a valid approach in modern mobile networks because a device from a large geographic area may share the same IP pool. Meanwhile, enhancing DNS with geo-location enhancement may give the geographically nearby edge host which may not be the nearest edge host in network topology. Last but not least, DNS with geo-location enhancement in MEC may introduce more management overhead for mobile ISPs.

Viscum host does not require updating the DNS service frequently. We can simply use the host name of original cloud service as our *target* address, and redirect the traffic towards *target* address to the service running on the viscum host. When operators want to disable edge host, they can simply clear TC filter, and the client would communicate with cloud server in its usual way. The disable/enable process could be fully automatic: e.g. a daemon process can enable/disable traffic director when observing a high CPU usage on viscum hosts. When traffic redirector is disabled, customer application still works with cloud server and there is no service interruption.

IV. Future Work

Viscum is still in its early stage as a POC project. In future, we plan to implement viscum on an off-shelf edge node, e.g. OpenWRT wireless router, or eNodeB, Amazon Echo device, etc. Because of the overhead introduced by container based MEC solution, there is less possibility to deploy MEC application onto these resource limited low end

embedded devices. Secondly, we will work on the management component. In current viscum implementation, we reply the control plane function to manage the viscum namespace and host. We may need to build a management module to allow third party business user to control their services running inside viscum namespace. Moreover, we need to measure the overhead introduced by viscum in a quantitative approach, and compared viscum's overhead with other MEC solution such as container based approach and hypervisor based approach. In these way, we may find out the performance limitation of process visualization in MEC. Last but not least, we will port more delay sensitive application onto viscum host, such as first person shoot game, face detection onto home wireless routers, etc.

V. Conclusions

This paper presents a lighter weight MEC software architecture, *viscum*, a process level virtualization MEC solution. Different from container based or hypervisor based MEC approach, viscum allows off-shelf application running inside Linux network namespace without any modification such as service splitting. Compared to container based MEC architecture, viscum introduces even less overhead. Thus, viscum might be deployed onto resource limited edge nodes such as wifi router or eNodeB without introducing new hardware. We hope viscum would accelerate the MEC development in future mobile networks.

References

[1] A. Reznik, R. Arora, M. Cannon, L. Cominardi, W. Featherstone, R. Frazao, F. Giust, S. Kekki, A. Li, D. Sabella *et al.*, "Developing Software for Multi-Access Edge Computing," European Telecommunications Standards Institute (ETSI), Tech. Rep., September 2017.

[2] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, March 2017.

[3] "Cloud RAN Architecture of 5G," Telefonica in collaboration with Ericsson, Tech. Rep., September 2017.

[4] D. Sabella, V. Sukhomlinov, L. Trang, S. Kekki, P. Paglierani, R. Rossbach, X. Li, Y. Fang, D. Druta, F. Giust, L. Cominardi, W. Featherstone, B. Pike, and S. Hadad, "Developing Software for Multi-Access Edge Computing (2nd Edition)," European Telecommunications Standards Institute (ETSI), Tech. Rep., February 2019.

[5] G. Avino, M. Malinverno, F. Malandrino, C. Casetti, and C.-F. Chiasserini, "Characterizing Docker Overhead in Mobile Edge Computing Scenarios," in *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems(HotConNet'17)*. ACM, August 2017, pp. 30–35.

[6] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An Updated Performance Comparison of Virtual Machines and Linux Containers," in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, March 2015, pp. 171–172.

[7] M. Feilner and P. Thompson, "The Practical Benefits of Network Namespace," *Admin: Network & Security*, vol. 34, September 2016.

[8] S. Sundaresan, A. Dhamdhere, M. Allman, and k. claffy, "TCP Congestion Signatures," in *Internet Measurement Conference (IMC)*, London, UK, November 2017.