Multi-Cloud Connectivity for Kubernetes in 5G Networks

Lirim Osmani[1] Tero Kauppinen[2] Miika Komu[2] Sasu Tarkoma[1]

The state of the art in designing scalable services relies on microservices with Kubernetes being the de facto platform for deploying them. While the Cloud Native Computing Foundation has standardized and improved several features in Kubernetes, one of the challenges remaining is that using multiple clouds simultaneously and dynamically, for example in a hybrid-cloud setting, is not possible. We address this crucial challenge by integrating the multi-cloud compute solution Federated Kubernetes with the emerging multi-cloud networking tool Network Service Mesh (NSM). This is the first solution for enabling Kubernetes-based multi-cloud workloads with seamless connectivity meeting the requirements for multi-cloud services and the Telco industry in 5G and beyond systems.

Index Terms—**Kubernetes, K8s, NSM, Network Service Mesh, hybrid cloud, multi cloud**

Introduction

Hardware virtualization has been a success factor for cloud computing. During the evolution of cloud computing, hypervised virtual machines have been a crucial steppingstone to success. Since then, Linux containers present a further evolution in virtualization by providing a "slimmer" virtualization layer than virtual machines by sharing the host kernel between the containers. The success of Linux containers can further be attributed to Docker software that simplified the creation and running of container operating system images.

The Docker corporation provides its own orchestration tools to manage containers in a datacenter environment. However, Kubernetes is another competing alternative to manage containerized workloads, originating from Google. Hyper-scale providers such as Google, Amazon and Microsoft, offer commercial support for Kubernetes and it can be argued that Kubernetes has become the de facto platform for the IT industry to run containerized workloads.

Kubernetes has further paved the way for scalable microservice-based service architectures where a monolithic service is split into smaller entities, each running on separate containers. While this can facilitate faster software development, testing and deployment, the main drawbacks of microservices are increased inter-process communication overhead and management complexity due to a larger number of containers. Different service mesh frameworks such as Istio [1] have alleviated the management burden of large and complex microservices by automating traffic management and security between microservices, and by supporting microservice monitoring for easier diagnostics. However, we argue that two overlooked challenges remain unaddressed in Kubernetes networking for the Telco industry as described in the remainder of this section, and in this article we argue that Kubernetes can be tailored to meet these key challenges.

As the first challenge, Kubernetes networking is provided by so-called plug-ins and while many of them are fit for IT services, the Telco industry has special requirements unmet by many of the plug-ins. For instance, containers need to have support for multiple network interfaces [2] for improved security for Containerized Network Functions. In addition network separation is crucial in separating control and data plane traffic. While some plug-ins solve this immediate need, they are still missing other requirements such as support for User Datagram Protocol and some special protocols mandated by 3GPP specifications. In this article our contribution to this challenge is a qualitative analysis of how different Kubernetes networking tools support Telco requirements.

Related to the first challenge, we also introduce a new networking framework for Kubernetes called Network Service Mesh (NSM) that we analyze quantitatively. We believe it can be a fair alternative as a networking framework to be used in reimplementing networking-intensive components in 5G, such as User Plane Functions based on microservice architectures. Such architectures can incur a high inter-process communication overhead which is further aggravated when utilizing service meshes(e.g.Istio), due to their extra performance penalty stemming from their proxy-based design. NSM can avoid this overhead as its dataplane is proxyless.

The second challenge in Kubernetes is related to its multi-cloud support, or to use the more technical term, multi-cluster support. We believe that it is necessary because a single Kubernetes cluster spanning numerous edge sites could be problematic because Kubernetes has limits on how many compute nodes it can encompass,

[1]University of Helsinki, Department of Computer Science
[2]Ericsson Research

and the potential "blast radius" is large in the case of orchestration faults or network partitioning. Multi-cluster capability is a fundamental requirement for global services and service optimization that enables for example hybrid cloud workload offloading and coping with national and regional data protection regulations and privacy requirements. Networking slicing in 5G could also benefit from additional isolation guarantees provided by separate clusters. Testing in production with canary tests that are tightly coupled with hardware could be separated into their own clusters to minimize interference. In a multi-vendor environment, integration of 3rd-party applications and different Kubernetes platforms along with their extensions can be a complex and challenging task, especially when the service availability requirements in cellular network are typically higher than in many other IT services, and this may involve vendor-specific clusters in the early phases of development. In the context of this article, we focus on two technical challenges related to multi-cloud Kubernetes: how to launch and destroy workloads, and how the workloads can communicate with each other across cluster boundaries.

Background on Kubernetes Networking

All Kubernetes resources including application containers are instantiated and terminated using configuration files called manifests that specify the desired state and type of the resource and other details. Once instantiated using deployment manifests, the containers need connectivity between each other and towards services external to the cluster. Kubernetes does not provide a default networking solution but relies on external network plugins, or Container Network Interfaces (CNIs) to implement networking capabilities. Some of the most popular CNIs include Calico, Weave and Cilium among others. Kubernetes imposes certain networking requirements for the plugins [3]. For instance, containers bundled into the same "pod" within the same worker node share the same (virtual) networking stack. Kubernetes networking can be implemented in different ways. Next, we present three common ways utilized by different CNIs.

Linux kernel-based forwarding is the least performant method to forward traffic into and out from containers. It is a software-based solution utilizing CPU resources for packet processing, meaning that Kubernetes can allocate less CPU time for workloads.

Vector Packet Processing (VPP) is a packet-processing framework based on Data Plane Development Kit (DPDK). Contrary to the conventional way of processing packets one at a time, VPP instead introduces a vector-based approach where packets can be processed in parallel, thus enabling faster processing of packets [4].

Single-root input/output virtualization (SR-IOV) is an extension to the PCI Express (PCIe) specification that allows a physical NIC to be partitioned into several Virtual Functions (VFs) accessible from within a virtualized environment. The VFs can be used to implement lightweight network functionality, such as encapsulation and de-capsulation of VxLAN tunnels directly on the NIC hardware operating near line speed [4], [5].

Related Work

In this section, we introduce some relevant open-source tools that can be used for implementing multi-cluster compute and networking. We focus on the latter aspect because it is critical for Telco use cases. Proprietary, hybrid-cloud solutions such as Google Anthos, Azure Stack and AWS Outposts are excluded from the comparison. We start with the compute part by introducing two of the multi-cluster management solutions. In the remainder of this section, we focus on the networking solutions: secondary networking and multi-cloud connectivity. Finally, we compare the networking solutions from the viewpoint of Telco requirements to justify our choice of Federated Kubernetes and Network Service Mesh to be used for a solution prototype.

It is worth noting that we have blatantly excluded service meshes, such as Istio and Linkerd [1], that operate on layers 4 and above. In this article we focus on solutions that operate on layers 2 and 3 and can be offloaded to hardware for faster processing.

Multi-cluster Management

The 5G architecture is aligned with Network Function Virtualization (NFV) specifications from the European Telecommunications Standards Institute (ETSI). ETSI NFV and its "Management and Network Orchestration" (NFV-MANO) architecture was originally specified for virtual machines but recent versions even include support for containers. While NFV-MANO supports multiple administrative domains [6], we believe NFV-MANO could benefit from cloud native implementation methods to manage multiple clusters, such as Federated Kubernetes (KubeFed). In addition, we introduce also Karmada which is an alternative to KubeFed.

Federated Kubernetes [7], or KubeFed, aims to achieve a unified management mechanism for multiple Kubernetes clusters, despite these clusters being distributed across regions, cloud providers or private clouds. Once a cluster is federated, the Federation API can be used to manage resources in multiple clusters

based on the standard Kubernetes API. This includes defining how the applications are deployed to different clusters and how they are scaled out. The Federation addresses several issues, including supporting multi/hybrid-cloud deployment, simplifying management of multiple clusters, scaling services across multiple clusters and cross-cluster service discovery. Once deployed, KubeFed enables a centralized way of launching and destroying containers spread across multiple clusters.

An alternative to KubeFed is to have a single master node that controls all the clusters, and the main benefit of such an approach would be to pool all the worker nodes under a single cluster to simplify operations and management. A problem with this solution is that the master node becomes a single point of failure: if it crashes or experiences network outage, no new workloads can be deployed. In contrast, KubeFed allows each cluster to have their own local master so that workloads can be started locally even if a cluster is partitioned from the others, so it could be stated that clusters retain a sense of local autonomy when managed by KubeFed. As a drawback, some redundant software resources in each cluster are needed to support autonomy in KubeFed. Also synchronization of, e.g.identity management and storage resources may be required across KubeFed-managed clusters, which can introduce more complexity.

Karmada is a multi-cloud management solution originating from Huawei. It is similar to KubeFed but supports some advanced features such as high-availability, failure recovery and traffic scheduling.

Secondary Networking Solutions

Kubernetes network plugins create so-called primary network interface to a pod. However, in some cases additional network interfaces are needed in a container, for instance some Containerized Network Functions in 5G require the different traffic types to be separated into their own interfaces. To support such functionality secondary network interface(s) can be provided by some of the networking solutions. It is worth noting that all of the solutions described here allow the creation of multiple network interfaces for containers and operate on lower layers of the networking stack (L2-L3).

Network Service Mesh (NSM) [8] is a programmable container networking framework for Kubernetes. NSM provides secondary network connectivity between containers and supports service chaining [9]. The NSM network interface in a container is always a point-to-point link connected directly to another container. The NSM link is meant only for reaching a single host, so the links are relatively free from namespace collisions assuming the address ranges allocated to the NSM links do not overlap with primary network interfaces. Internally, an NSM link incurs no network address translation due to the point-to-point nature of the link, even though the addresses are typically allocated from a private address range. It is also worth noting that a container can be allocated multiple NSM network interfaces when the container needs to establish NSM based connectivity with multiple other peers.

When utilizing NSM, a container is required to act in a "client" or "endpoint" role, or to act in both roles. The role names can be a bit misleading: typically a client could be a “service process” running, e.g., a web server, and the endpoint would be a middlebox, such as a router, firewall or load balancer. To connect multiple middleboxes into a service chain the services in the middle act in both roles, with each of them having one interface facing a client and the other facing an endpoint. The order for the containers in the chains is defined in a separate network service manifest.

Before NSM can be enabled, the NSM software infrastructure needs to be deployed to each worker node. The NSM infrastructure consists of two components: NSM managers are responsible for control plane communications using the gRPC protocol, and NSM forwarders are responsible for setting up the data-plane connectivity between the containers. NSM forwarders can set up an NSM connection using different configurable ways, with VxLAN as the default.

After the NSM infrastructure is up and running, two or more containers can be interconnected using NSM interfaces by annotating their deployment manifests using NSM labels. However, Kubernetes interrupts the normal starting process of the containers because their deployment labels match the NSM web admission hooks. This triggers the launching of two additional control-plane containers in the client pod. The responsibility of these so-called "init" containers is to configure the creation of the NSM network interface and other networking configuration details of the associated pod, such as routes. One of the init containers can configure DNS-related details, and the other init container requests the local NSM manager to contact the remote NSM manager to set up connectivity to the endpoint using the NSM forwarders. Upon completion the init containers exit and the client container is set to running state. At the side of the endpoint container init containers are not utilized, but the endpoint pod includes an extra "sidecar" container that handles the creation of NSM network interface and other network configuration details. The difference between an init and sidecar container is that the latter is kept running throughout the lifetime of the pod, so that

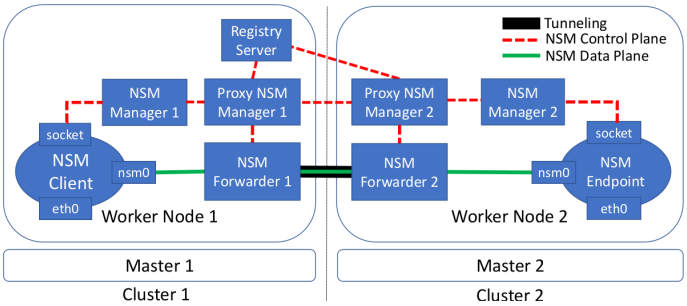

Fig. 1. Inter-domain connectivity with NSM

the endpoint container can be dynamically connected to new client containers.

NSM supports connecting a client and endpoint located in different Kubernetes clusters using the so-called "NSM interdomain" functionality that is illustrated in Figure 1. The use of this functionality requires deploying an extra registry service that centrally collects information about the clients and endpoints from different clusters, so that containers can be located and connected. In addition to this, each worker node includes a proxy NSM manager that is used only when the traffic passes cluster boundaries.

Multus [10] is a container network interface (CNI) plugin for Kubernetes that enables the attachment of multiple network interfaces to pods. It differs from the commonly used CNIs (Flannel, Calico, Weave, etc) because it acts as a meta-plugin of other CNI plugins and invokes other CNI plugins to create additional network interfaces, rather than implementing the networking functionality itself. While the support for Multus can be considered stable in Kubernetes, it has some drawbacks when compared to NSM. For example, the deployment of containers between different Kubernetes distributions may require some manual configuration depending on the local cluster set up that may not result in very portable set ups. Also, Multus is missing a programmable API and does not include built-in service chaining support.

DANM [11] from Nokia is somewhat similar to Multus, as DANM is also a meta-plugin for other CNIs and provides secondary network interfaces to containers. In contrast to Multus, it can support DNS-based service discovery of individual DANM-based containers (i.e., so called "headless" services), but L3/L4 load balancing is unsupported. Compared to NSM, built-in support for multi-clusters and service chaining is also missing from DANM.

Kube-OVN [12] is a Software-Defined Networking solution for Kubernetes. Kube-OVN is capable of supporting SR-IOV and service function chaining. Kube-OVN supports primary networking across clusters, but secondary networking is Multus based which by default does not support connectivity across clusters. Kube-OVN provides limited support for connectivity without NATs for its Virtual Private Cloud (VPC) networking and multi-cluster connectivity. While SCTP is supported, NAT-based connectivity can influence SCTP negatively because its NAT extensions are still being standardized. Kube-OVN is adopted for instance by Edge Multi-Cluster Orchestrator (EMCO) [13] project, which is an open-source framework to support networking for applications for the edge.

Multi-cloud Connectivity Solutions: Submariner

Submariner [14] from Rancher provides transparent multi-cluster connectivity for containers using IPsec based gateways and can provide DNS-based service discovery across clusters. Submariner supports networking across multiple clusters without introducing NATs but can optionally insert a NAT between two clusters upon IP address conflicts. It is worth noting that Submariner supports only primary networking.

A Comparison of Networking Solutions

In this section we summarize the different networking extensions to Kubernetes that can potentially be used to implement a solution to the challenges introduced in Section . The extensions must support both multi-cloud compute and networking. For the multi-cloud compute any extension supporting starting and stopping workloads across cluster boundaries is acceptable, but we have chosen KubeFed here for further experimentation. For multi-cloud networking the extension must support workload connectivity across cluster boundaries, and must also fulfill Telco requirements for Kubernetes networking. Table I lists the Telco requirements on the first column: support for secondary networking (2nd n/w), service function chaining (SFC), multi-cluster connectivity support (multi-cl.), support for TLS/VPN connectivity for containers (TLS/VPN), support for connectivity without NATs - especially in load balancers and multi-clusters (no-NAT), Stream Control Transmission Protocol support (SCTP), UDP support (UDP) and support for hardware accelerated networking (h/w accl.). To our understanding only NSM can meet the networking Kubernetes requirements for Telco, so we have chosen it for the prototype solution described in the following section.

Proposed Solution

We propose a solution for multi-cluster Kubernetes. The solution prototype builds on two key components:

	Multus	danm	Subm.	NSM	Kube-OVN
2nd n/w	✓	✓		✓	✓
SFC				✓	✓
multi-clu.			✓	✓	(✓)
TLS/VPN			✓	✓	✓
no-NAT		✓	✓	✓	(✓)
SCTP		✓		✓	✓
UDP	✓	✓	✓	✓	✓
h/w accl.	✓	✓		✓	✓

TABLE I
A COMPARISON OF THE RELATED WORK

KubeFed to manage multi-cluster compute and NSM to manage multi-cluster networking. To our knowledge this combination is novel and unexplored both from the viewpoint of design and implementation. An overview of the implementation architecture is illustrated in Fig. 2.

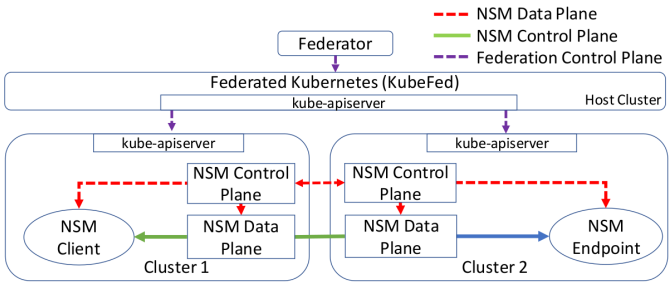

Fig. 2. Our proposed solution based on NSM and KubeFed

Starting from the top, Fig. 2 shows a new wrapper service for KubeFed called Federator. The use of Federator is optional, but it simplifies and automates the otherwise manual cluster (un)joining process under a web-based API. The Federator relays API calls to an unmodified KubeFed that communicates with the clusters using their standard Kubernetes API servers. The relayed calls include a manifest file, and the calls are either related to (un)joining of clusters, or to starting or terminating workloads in different clusters. The Federator can automatically add KubeFed and NSM related annotations in the associated manifests. The Federator can also be used for visualizing the location of the containers in the clusters. Other than this, the clusters have been predeployed with unmodified NSM that is triggered automatically based on Kubernetes webhooks as described in Section . NSM can establish connectivity automatically according to the annotations in the user provided manifests, even across cluster boundaries.

PERFORMANCE EVALUATION AND DISCUSSION

The performance evaluation was conducted in an environment where the host machines were equipped with 10 Gbit physical network interfaces, with each master/worker node equipped with 4 CPUs and 8 Gb of RAM. Kubernetes was installed on top of OpenStack infrastructure (Train), and the physical servers were connected using VLANs in OpenStack. The Kubernetes environment consisted of two separate clusters, with each cluster consisting of one master and two workers. Each Kubernetes node was assigned to its own virtual machine.

We benchmarked basic data-plane performance between two containers (a client and endpoint) using NSM (v0.2) because no such results have been published yet, and our kernel-forwarder measurements will act as a baseline for the upcoming SR-IOV support in NSM. The two communicating containers were ensured to run always on two different physical machines.

The measured containers were admitted administrative privileges for network capabilities for MTU tuning because NSM v0.2 does not yet support changing this declaratively, so the MTU needs to be changed manually from within the container. We tested NSM with the default Maximum Transfer Unit (MTU) with the typical size of 1500 bytes and with the maximum the underlying system could offer, that is, 8950 bytes. NSM was measured both in single and multi-cluster scenarios, but the other baseline measurements were measured only in single cluster scenario due to lack of multi-cluster support. VxLAN was used for tunneling the traffic for NSM. NSM was configured with kernel forwarder. VPP measurements were omitted because VPP integration in NSM was performing sub-optimally according to the developer community.

We measured dataplane throughput using using *iperf3* and latency with *ping* using IPv4. As transport protocols we mainly utilized Transport Control Protocol (TCP) and User Datagram Protocl (UDP), but we also measured Stream Control Transmission Protocol (SCTP) performance. UDP was measured at the server side with unlimited bandwidth and buffer length set to 60 kB. All measurements were repeated 10 times over intervals of 60 seconds.

Figure 3 depicts the results of TCP and UDP throughput performance in a single-cluster and multi-cluster environments with NSM. For reference, we also show the throughput performance of Calico and physical machine-to-machine (no virtualization) based communications. It is worth noting that Calico automatically adjusts the MTU, so the actual MTU values for Calico are 1440 and 8940 which is the same configuration as utilized by Ducastel et al [15] for measuring Calico. The MTU values are slightly different in the machine-to-machine and NSM cases, 1500 and 8900, due to different tunneling overheads. Our Calico measurements are otherwise

aligned with Ducastel et al except in the case of UDP using the higher MTU value. They report such a case performing as well as TCP but we achieved a higher UDP throughput (7 Gbit/s) only by placing two measured containers into the same worker node. The measurements we illustrate in this section were conducted always between two separate worker nodes.

As shown in Fig. 3, increasing the MTU from 1500 to 8900 resulted in 2-4 times better performance in all the cases. NSM was observed to have comparable performance to Calico both in single-cluster and multi-cluster cases. In general single-cluster NSM performance was similar to the multi-cluster NSM because the underlying host machines were located in the same datacenter within the same rack. The standard error values were relatively small (between 0.01 and 0.08 Gbit/s). It is also worth noting that the nearly equal performance of TCP and UDP has also been reported by Ducastel et al [15].

Latency measurements are depicted in Fig. 4. As a base line we show the latency observed in the physical machines hosting the infrastructure. The average NSM latency is 845 microseconds in the single cluster case and 1039 microseconds in the multi-cluster case. This is marginally better compared to Calico with an average of 1146 microseconds. The standard error values were relatively very small: 0.69 μs for the physical machine-to-machine case, 61 for μs Calico, 17 μs for NSM single-cluster, 9 μs for NSM multi-cluster.

We also measured the performance of the Stream Control Transmission Protocol (SCTP). SCTP performed poorly with NSM: the throughput was 139.1 *Mbit/s* with a MTU of 1500, and 486.4 *Mbit/s* with a MTU of 8900, and the largest standard error of 8.48 *Mbit/s*; also Calico performed similarly (126.8 Mbit/s and 391.4 Mbit/s) with a standard error of 8.12 *Mbit/s*.

While running the throughput measurements we retrieved load values from the Kubernetes metrics server for NSM and KubeFed control components across the cluster. The results show the total CPU load in the range between 0,2 percent and 2,3 percent for NSM management components and around 0,2 percent for KubeFed components on a master node. For the NSM kernel forwarding data plane components running on worker nodes, we observed fairly stable values in the range of 0.1 percent and 0.2 percent (total CPU load).

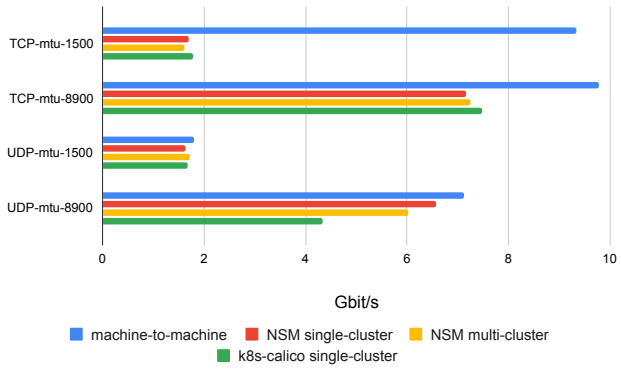

Fig. 3. Throughput Performance. It should be noted that the MTU of Calico deviates from the other MTUs

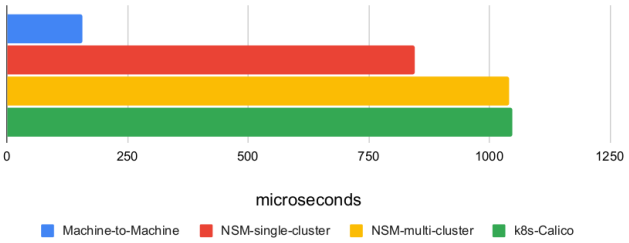

Fig. 4. Latency Performance

Conclusion

We presented two challenges in Kubernetes. The first challenge was that Kubernetes networking does not meet all Telco industry requirements. Based on surveying the state of the art, we did not find any other framework for Kubernetes other than NSM to meet all the Telco requirements. The second challenge was related to multi-cluster support. Our proposed solution integrates two open-source tools: KubeFed for deploying workloads in multiple clusters and NSM for providing connectivity for containers across cluster boundaries. KubeFed maintains the autonomy of each cluster so that a local cluster can still serve local workload requests even upon network partitioning, which can be useful especially in edge cloud scenarios.

We provided some basic benchmarking results for NSM, including latency and throughput measurements of the dataplane which are on par with the popular Calico network plug-in for Kubernetes. Kernel-based forwarding has the lowest performance. VPP would provide medium-level performance, but we did not measure it because it is not optimized for performance in NSM. While waiting for the SR-IOV-based hardware acceleration support for SmartNICs to mature in NSM, it is expected to perform in near line-speed performance as indicated by other performance evaluations [4].

We discovered some areas of improvement for NSM. For example, the network MTU value had to be tuned manually for NSM and the workaround of granting extra privileges to containers can introduce unnecessary security vulnerabilities. Based on our feedback, the NSM community has enabled automated setting of the MTU

in the upcoming 1.0 release. As another example, SCTP performance can further be improved not only for NSM but also our baseline reference, Calico.

As future work we are evaluating moving of stateful containers across cluster boundaries to support 5G service mobility. Rather than employing container migration, our goal is to achieve the same using cloud-native methodology, that is, by starting new containers in the target cluster, transferring the of the state of the containers to be moved and finally terminating the unnecessary containers in the originating cluster. Based on our initial experiments, the rough time scale to move a single stateless container between two clusters using unoptimized KubeFed and NSM is a bit less than one minute. This gives room for further optimizations and requires further analysis: moving the state of containers between clusters will add further overhead, and the process must be orchestrated smartly to avoid service unavailability. We believe such service mobility could be used as a building block for 5G edge services that are automatically moved to the proximity of terminals, or for services that move between core network and edge cloud. Our approach could also be used for realizing mobility for network slices. Finally, we envision that our work could be further used to extend 5G-related frameworks, such as NFV-MANO and Akraino to manage Kubernetes and workload connectivity in multi-cluster environments.

ACKNOWLEDGMENT

In addition to the IEEE ComSoc reviewers, the authors would like to thank Tomas Mecklin, Jimmy Kjällman, Andrew Williams, Miljenko Opsenica, Alexandra Duque, Marina Kurten, Ed Warnicke, and Jan Scheurich for their insightful feedback and input.

REFERENCES

[1] W. Li *et al.*, "Service Mesh: Challenges, State of the Art, and Future Research Opportunities," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2019, pp. 122–1225.

[2] 5G-PPP Software Networking Group, "From Webscale to Telco, the Cloud-Native Journey," Whitepaper, July 2018

[3] "Kubernetes," http://kubernetes.io, accessed August 2021.

[4] N. Pitaev *et al.*, "Characterizing the Performance of Concurrent Virtualized Network Functions with OVS-DPDK, FD.IO VPP and SR-IOV," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 285–292.

[5] J. Li *et al.*, "When I/O Interrupt Becomes System Bottleneck: Efficiency and Scalability Enhancement for SR-IOV Network Virtualization," *IEEE Transactions on Cloud Computing*, vol. 7, no. 4, 2019, pp. 1183–1196.

[6] ETSI Group specification, "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Multiple Administrative Domain Aspect Interfaces Specification", GS NFV-IFA 030 V4.2.1, May 2021

[7] "Federated Kubernetes," https://github.com/kubernetes-sigs/kubefed, accessed August 2021.

[8] "Network Service Mesh," https://networkservicemesh.io/, accessed August 2021.

[9] B. Dab *et al.*, "Cloud-native Service Function Chaining for 5G based on Network Service Mesh," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–7.

[10] "Multus," https://github.com/intel/multus-cni, accessed August 2021.

[11] "DANM", https://github.com/nokia/danm/, accessed August 2021.

[12] "Kube-OVN – The Most Advanced Kubernetes Network Fabric for Enterprises" https://kubeovn.io, accessed August 2021.

[13] "EMCO", https://github.com/open-ness/EMCO, accessed June 2021.

[14] "Submariner," https://submariner.io/, accessed August 2021.

[15] "Benchmark results of Kubernetes network plugins (CNI) over 10 Gbit/s network," August 2020; https://itnext.io/benchmark-results-of-kubernetes-network-plugins-cni-over-10gbit-s-network-updated-august-2020-6e1b757b9e49,accessed September 2021.

Lirim Osmani is a Ph.D. candidate in Computer Science at University of Helsinki, Finland. He is currently conducting research on datacenter management,optimization and virtualization technologies for intensive on-demand computing. He has industry engineering background in virtualizing enterprise systems.

Tero Kauppinen M.Sc. Tero Kauppinen has been working over 20 years on various networking technologies at Ericsson Research, ranging from IETF protocols to cloud networking stacks.

Miika Komu D.Sc. Miika Komu has been working in Helsinki Institute of Technology and Aalto University before starting at Ericsson Research. He has been working a decade on IETF standardization and five years on cloud software.

Sasu Tarkoma (SMIEEE'12) is a Professor of Computer Science at the University of Helsinki, and Head of the Department of Computer Science. His research interests include mobile computing, Internet technologies, and AI.