A Novel Framework for Cross-Cluster Scaling in Cloud-Native 5G NextGen Core

Dumitru-Guzu, Oana-Mihaela ; Vlădeanu, Călin; Kooij, Robert

DOI
10.3390/fi16090325

Publication date
2024

Document Version
Final published version

Published in
Future Internet

future internet

Article

A Novel Framework for Cross-Cluster Scaling in Cloud-Native 5G NextGen Core

Oana-Mihaela Dumitru-Guzu [1,†], Vlădeanu Călin [1,*,†] and Robert Kooij [2]

[1] Department of Telecommunications, Faculty of Electronics and Telecommunications Systems, Polytechnic University of Bucharest, 061071 Bucharest, Romania; oana.ungureanu30@gmail.com
[2] Department of Network Architectures and Services, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2628 CD Delft, The Netherlands; r.e.kooij@tudelft.nl
* Correspondence: calin@comm.pub.ro
† These authors contributed equally to this work.

Abstract: Cloud-native technologies are widely considered the ideal candidates for the future of vertical application development due to their boost in flexibility, scalability, and especially cost efficiency. Since multi-site support is paramount for 5G, we employ a multi-cluster model that scales on demand, shifting the boundaries of both horizontal and vertical scaling for shared resources. Our approach is based on the liquid computing paradigm, which has the benefit of adapting to the changing environment. Despite being a decentralized deployment shared across data centers, the 5G mobile core can be managed as a single cluster entity running in a public cloud. We achieve this by following the cloud-native patterns for declarative configuration based on Kubernetes APIs and on-demand resource allocation. Moreover, in our setup, we analyze the offloading of both the Open5GS user and control plane functions under two different peering scenarios. A significant improvement in terms of latency and throughput is achieved for the in-band peering, considering the traffic between clusters is ensured by the Liqo control plane through a VPN tunnel. We also validate three end-to-end network slicing use cases, showcasing the full 5G core automation and leveraging the capabilities of Kubernetes multi-cluster deployments and inter-service monitoring through the applied service mesh solution.

Keywords: multi-cluster; liquid computing; multi-cloud; network slicing; vertical and horizontal scaling

check for updates

Citation: Dumitru-Guzu, O.-M.; Călin, V.; Kooij, R. A Novel Framework for Cross-Cluster Scaling in Cloud-Native 5G NextGen Core. *Future Internet* **2024**, *16*, 325. https://doi.org/10.3390/fi16090325

Academic Editor: Paolo Bellavista

Received: 7 August 2024
Revised: 25 August 2024
Accepted: 2 September 2024
Published: 6 September 2024

1. Introduction

Network slicing architecture has emerged in 5G as a key feature to efficiently support dynamic resource allocation in a multi-tenant environment, as it offers solutions for isolating the network for various industries and use cases. In this manner, the mobile network operators (MNOs) can configure and manage the control plane and user plane network functions along with the corresponding resources (e.g., access, transport, and core networks) to support various slice/service types (SST). A network slice can be defined as a logical end-to-end network that is dynamically created, and a user may access multiple slices over the same gNB. Each slice may serve a particular service type with an agreed-upon service-level agreement (SLA) [1].

In the technical specification, TS 23.501 v16.4.0 [2], the 3rd generation partnership project (3GPP) provides a standardized classification to classify different services: enhanced mobile broadband (eMBB) services, ultra-reliable low-latency communications (URLLC) services, massive internet of things (IoT) services (MIoT), and vehicular-to-everything communications (vehicles, infrastructure, pedestrians, etc.). Network slicing is mostly tackled in the existing solutions through network virtualization, which is encompassed by the NFV MANO (network functions virtualization management and orchestration) framework to manage and orchestrate VNFs (virtualized network functions). An important pillar addressed by NFV MANO is service availability and systems capacity to deal with failover, whereas the operation support system (OSS) and business support system (BSS) do not fulfill the recovery and failover role for network slicing. Moreover, the existing NFV MANO systems such as ONAP [3] and OSM [4] are dedicated to onboarding and instantiating VNFs, but regarding application runtime orchestrations, it has been observed that they lack the ability to scale operations properly [5]. Especially due to the increase in traffic reflected on both the user and control plane, the scalability becomes a virtualization deployment challenge [6,7]. Another challenge that network slicing imposes in terms of intensive computational power is the efficient management and utilization of the resources [8,9]. Most of the deployments are limited in terms of physical or virtual resources, whereas network slicing for different verticals requires consistent amounts of networking, computing, or storage resources to be allocated on demand. Security is a mandatory requirement when it comes to the operational aspect of the telco network, therefore any open-source implementation of the mobile stack needs to adhere to the security standards in terms of multi-site deployments [10]. Service monitoring is also a recurrent concern in the multi-layered architecture of 5G, which might require data aggregation from multiple sources to be able to ensure SLA monitoring at each level of the mobile stack [11,12].

The GSMA [13] paper handles the end-to-end network slicing through network automation, which is an important requirement in supporting the deployment of slices through APIs provided by a network slice provider. The network automation technology is expected to roll out network slices and modify them in an agile and automated fashion. In contrast to the limited capabilities of virtualization for scaling of network slicing, the container orchestration, specifically Kubernetes standalone, which is the de facto standard, leverages automation for cloud-native life cycle management such as automatic scaling, scheduling, and self-healing independently of the lack of resources at the edge of the network [14]. In contrast to the classical cloud applications, telco applications are geographically distributed across multiple data centers. The main challenge in cross-location deployments is the service performance, especially for delay-sensitive services, which might experience delay, jitter, or packet loss. Nevertheless, a drawback of the current cloud-native solutions for 5G mobile core deployment is that they are centered around Kubernetes standalone clusters and lack a broader approach for multi-clusters and multi-cloud scenarios in terms of automation of network slicing.

Unlike other standalone Kubernetes deployments, this paper addresses the Kubernetes full-stack capabilities in terms of multi-cluster scalability for a multi-site 5G deployment by introducing a new framework to scale both horizontally and vertically the resources shared across isolated Kubernetes clusters. Moreover, due to the Kubernetes APIs that work over the internet, the connectivity can be extended to different cloud providers as well as to on-premises sites. This emphasizes the idea of unlimited and on-demand utilization of resources for a 5G mobile core instance running as cloud-native.

We consider our contributions fourfold below:

- Our main contribution depicted in Figure 1 covers all layers of the MANO framework since it addresses the multiple concerns in terms of self-management, scheduling, scalability, end-to-end service monitoring, and foremost the multi-site deployment performance challenge. We also compare the two frameworks and map the correspondence between the functional blocks. Another important aspect of this approach is the network isolation provided by the cloud ecosystem, which uses the concept of tenants running in separate data centers. In our proposed design, we benefit from the Kubernetes capabilities in terms of operators, which are software extensions; in our case, the Liqo operator [15] runs inside the cluster. Moreover, by peering multiple Kubernetes clusters using Liqo, we ensure unlimited capacity and resources for the scalability and deployment of mobile core, which unblocks the barriers for network slicing and on-demand resource consumption.

- A secondary contribution concerns the comparison between different mechanisms for Liqo offloading. In our setup, we use Liqo to peer the clusters and employ the Kuelet [16] component, which is part of the Kubernetes architecture to operate at the level of eachb node. In this manner, we can offload the workload between clusters that reside in different tenants as well as separate data centers and regions in the public cloud. We test and analyze three scenarios for offloading the control and user planes of the 5G mobile core. To demonstrate the flexibility of the cloud-native approach, we run the Open5GS [17], which is an open-source simulator for virtualized 5G mobile cores that we run in containers orchestrated by Kubernetes.
- The third contribution consists of the system's performance validation, which is achieved by configuring and deploying end-to-end network slicing corresponding to three verticals, which is analyzed in terms of latency and throughput for the proposed offloading scenarios. In order to configure the slices, we use *UERANSIM* [18], which is an emulator for the RAN (radio access network).
- The fourth contribution looks at the telemetry solutions; hence, we endow our setup with the monitoring capabilities for inter-service communication by employing Istio [19] specific to service-mesh topologies.

Furthermore, this paper is structured as follows: Section 2 gives an overview of the existing body of literature for existing ETSI MANO (network functions virtualization management and orchestration) open-source projects as well as cloud-native testbeds that validate end-to-end network slicing scenarios. Additionally, in Section 3, we provide a thorough correlation between the MANO framework and Kubernetes orchestration, assessing scalability for multi-cluster and multi-cloud deployments. In Section 4, we present our setup and configuration for the three proposed offloading scenarios of the 5G user and control planes. The following two Sections, Sections 5 and 6, reveal an in-depth view of the functionality of the testing environment as well as highlight the differences between out-of-band and in-band peering in terms of configuration and deployment model. Moreover, Section 7 presents the evaluation of our results in terms of throughput and latency for the three instantiated network slices. In our final Section 8, we present our findings along with the proposed future work.

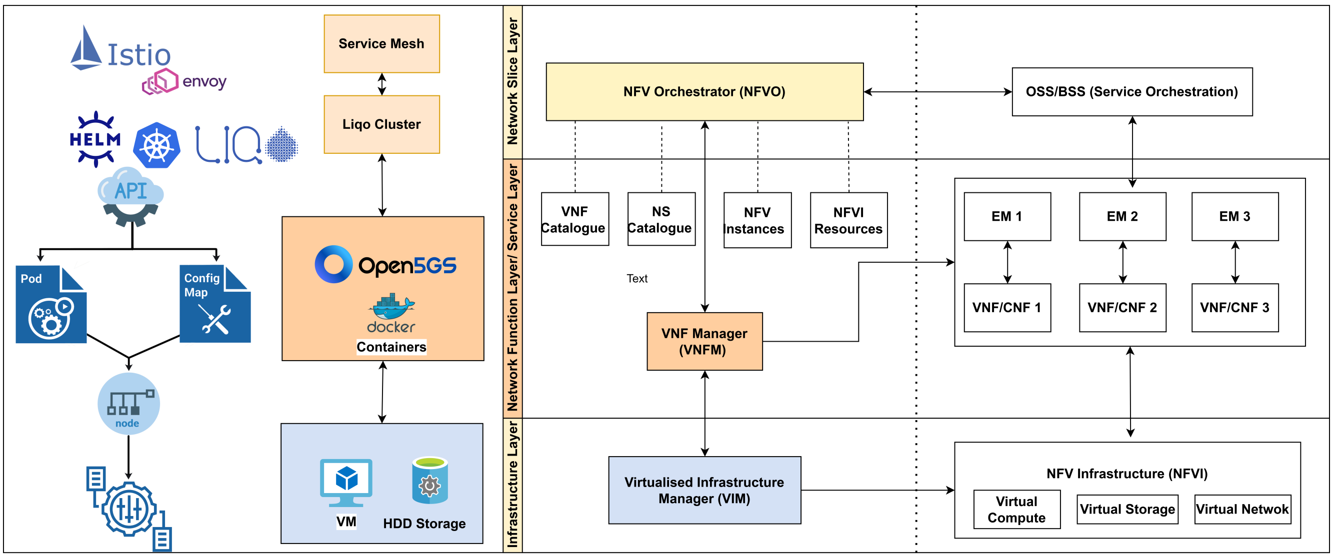


Figure 1. Proposed framework for network slicing in cloud-native 5G core.

2. Related Work

The existing body of literature around network slicing is divided between virtualization achieved in the MANO framework with a dedicated orchestration function and Kubernetes deployments that cover the full stack, including the flexibility of network functions, service recovery, and monitoring. The MANO framework handles dynamically the onboarding, instantiating, scaling, and terminating of network slices, while OSS/BSS (Operations and Business Support Systems) is responsible for the static provisioning and de-provisioning of network slices in a virtualized environment.

From a virtualization point of view, a classification encountered in the literature consists of two categories of testbeds that address the deployment of network slices: the first category is limited to a set of functionalities; the second one addresses the end-to-end capabilities. Initiation of the network slicing is present in many of the existing open-source systems, such as OSM [4] and ONAP [3], SONATA [20], OpenStack Tracker [21], and Cloudify [22]. However, both OSM and ONAP systems offer mechanisms to track and restore the services. The authors Yilma et al. are addressing in paper [23] the challenges and gaps of the OSM and ONAP in terms of missing full stack functionality. The authors propose several benchmarks for both functional and operational KPIs that are validated in their test scenarios.

The first category we address does not tackle the E2E network slicing [14,24–26]. For instance, Arampatzis et al. are deploying in paper [24] the network function instantiation and management based on OPNFV [27], with OpenStack [28] as VIM (Virtualized Infrastructure Manager) and VMware [29] as a hypervisor. This approach shows the capability of NFVO (NFV Orchestrator) to serve as a network service and resource orchestrator.

The second kind of virtualization testbeds encompasses the full stack capabilities of MANO for management of E2E network slicking [30–39]. In this regard, the authors Chang et al. introduce in paper [33] a slightly different architecture where the network slices are coordinated at both MANO and OSS/BSS levels. A qualitative comparison of three systems is conducted: Tracker, standalone OpenStack, and free5GMANO [40] in terms of configuration and fault management of the network slices.

In regards to Kubernetes deployments, there are also several tools available that leverage the DevOps capabilities of Kubernetes. An open-source tool of this kind in which network slicing initiation occurs at both MANO and OSS/BSS layers is Aether [37]. The design consists of a set of centralized components named *Aether Central*, which manages several edge sites. Other tools such as 5G-Berlin [38] and 5Genesis [39] present a 5G network to deploy and run applications of verticals on top of NFV infrastructure. Network slicing capabilities as well as mobile edge computing are addressed in a cloud-native NFV MANO implementation.

The cloud-native deployments can also be categorized in two different directions: the first one presents the standalone deployments enhanced with monitoring capabilities, whereas the second category looks at multi-cluster deployments that can be monitored and observed through service-mesh solutions. Currently, no Kubernetes-native standard is defined, but some popular standards have emerged (such as Prometheus [41] and OpenTelemetry [42]) for monitoring and collecting metrics purposes, where KPI solutions can be configured.

A significant amount of papers envision the mobile 5G core as a cloud-native application orchestrated in a standalone Kubernetes cluster [43–46]. In paper [47], the authors Arouk et al. describe in depth the CI/CD provisioning workflow of the 5G mobile core and introduce a customized Kubernetes operator called Kube5G-Operator. This operator aims to provide both reconfiguration and reconciliation in case of service failure based on the CR (custom resources). In this manner, the performance in terms of both UDP and TCP traffic is improved significantly compared to the bare-metal setup. Paper from Barrachina-Muñoz et al. [48] employs the Open5GS tool, where all the network functions are running in containers. For the RAN, Amarisoft is used, whereas in terms of monitoring, Prometheus and Grafana [49]. The same authors are concentrating in paper [50] on the monitoring capabilities for a cloud-native 5G deployment with Open5GS. The connectivity for RAN is facilitated by using the physical installation of Amarisoft Callbox [51].

Multi-cluster Kubernetes deployments are present in several papers that deal with scalability challenges as well as service-mesh solutions. One of the proposed solutions for scaling the control plane is presented in [52]. In this setup, RedHat makes use of the HPA

(HorizontalPodAutoscaler) to scale the 5G CNFs and validates the results by employing service monitoring using the Istio service mesh. Ungureanu et al. present in paper [53] a novel design for a declarative abstraction for cloud-edge communication using CAPI (Cluster API) [54] for infrastructure automation. In this manner, it is possible to deploy, scale, and manage Kubernetes multi-clusters in on-premise and multi-cloud environments using a Kubernetes lightweight distribution, K3s [55]. Also, Mfula et al. [56] emphasize the need for scaling the MEC (multi-access edge computing) architecture across environments in a declarative fashion through CAPI. An extensive survey among stakeholders with an analysis of the results is also conducted. A multi-cloud federated Kubernetes solution is presented by Osmani et al. in paper [57] that makes extensive use of a network Service Mesh (NSM) tool taking into account real-time and geographically distributed workloads. In terms of inter-service communications, Ungureanu et al. [58] employ Linkerd [59] as a solution for the service mesh.

3. Comparison between ETSI MANO and Kubernetes

The traditional ETSI MANO framework relies on virtualization at its core. In Kubernetes, the applications are containerized on top of virtualization and referred to as containerized network functions (CNFs). Nevertheless, an accurate mapping between ETSI requirements and Kubernetes can be established. The ETSI IFA029 [60] specification translates VIM and VNFM architecture into a container framework by creating new concepts such as Container Infrastructure Service (CIS), Container Infrastructure Service Management (CISM), and Container Infrastructure Service Instance (CISI).

MANO framework has three main components: NFV Orchestrator (NFO), VNF Manager (VNF), and Virtualized Infrastructure Manager (VIM). The main characteristics of ETSI MANO lay in the use of lifecycle operating procedures between NFVO VNFM and VIM, which might cause duplicates or bottlenecks in the operating model, whereas the Kubernetes model promotes intent-driven operations on both southbound and northbound interfaces using manifests and APIs. In our proposed architecture, the infrastructure layer is ensured through the cloud infrastructure, i.e., VMs on top of which Open5GS is installed inside Kubernetes clusters. For the Open5GS deployment, we employed the Kubernetes built-in object called ConfigMap, which allows a declarative configuration of the NFVs through APIs to specify the desired state of the operation.

Therefore, it is essential to analyze all layers of the MANO architecture in order not to offload the details of the lower layers, i.e., virtual machines (VMs), to the upper layers (see Figure 1). In this regard, the ETSI MANO differs from the Kubernetes model since the VNFM (VNF Manager) holds a detailed view of deployed associated VNFs and exposes it northbound to NFVO. In Kubernetes, this information is not exported to the upper layers since Kubernetes offers a better way to control it by defining the intent through object definitions (such as labels, tags, selectors, taints, etc.). In Kubernetes, the concept of *pods* exists to define where the containerized applications reside. Kubernetes manages the lifecycle of those pods by scaling up and down using changes in deployments and defining requirements in the configurations for the minimum, maximum, and desired number of replicas. In terms of operation, the NFVO can define the desired end state in a declarative manner using artifacts, such as deployment and service YAML files, and then the Kubernetes scheduler ensures the resource provisioning. Additionally, Kubernetes can use the automatic cluster scaler to request extra resources from an underlying cloud infrastructure to meet the additional needs of the Kubernetes workload [61].

In the context of 5G SBA (Service-Based Architecture), network slicing involves the virtualization and replication of the entire service graph that implements the mobile core. A slice can be considered a system abstraction that keeps a record of the set of interconnected microservices running per slice and gives clear instructions to the underlying schedulers to allocate the required network bandwidth according to the service demands as well as the CPU scheduler to allocate enough cores to the running containers [62].

3.1. Horizontal vs. Vertical Cloud Scaling

In cloud environments, vertical scaling makes use of the existing infrastructure in terms of adding more computational power, such as RAM, CPU, etc., whereas horizontal scaling, also called "scaling out," relies on the deployment of new infrastructure by adding more instances of the same kind.

Horizontal scaling serves the purpose of organizations that need high availability and near-zero downtime or other disruptions, and it is faster and easier compared to vertical scaling. At a basic level, scaling out can mean adding new computing nodes or machines to enhance the data processing and storage capabilities. On the other hand, when we refer to vertical scaling, we address the optimization of data processing and multi-threading for one instance. Both horizontal and vertical cloud scaling aim to leverage processing capabilities and storage, increase flexibility, and reduce costs [63].

In Kubernetes, horizontal scaling means to expand the number of pods as a response to the increased workload. The HorizontalPodAutoscaler (HPA) feature automatically updates a workload resource (such as Deployment or StatefulSet), with the aim of automatically scaling the workload to match capacity demands [64].

In terms of vertical scaling, the Kubernetes scheduler assigns more resources (i.e., memory or CPU) to the pods that are already running as part of the workload. The key role of the vertical pod autoscaler (VPA) [65] is to automatically set the requests for the containers based on pod usage and utilize the optimal scheduling mechanism to allocate on-demand the necessary computational resources for each pod.

3.2. Scaling in Kubernetes Multi-Cluster and Multi-Cloud Deployments

From security and compliance considerations in large environments such as a telecom network, one Kubernetes cluster is generally not sufficient; hence, a multi-cluster infra management solution is needed. There are many existing open-source and vendor-provided solutions to manage multiple Kubernetes clusters, i.e., Rancher, OpenShift [66], Crossplane [67], ClusterAPI, etc.

Liqo [15] is an open-source project enabling dynamic Kubernetes multi-cluster topologies. Liqo leverages the Liquid computing paradigm [68] for scalability purposes in both hybrid-cloud (i.e., the combination of on-premise and public cloud) and multi-cloud approaches, which aim for high availability, geographical distribution, and cost-effectiveness, enabling telcos to become vendor agnostic in terms of cloud provider. Liqo makes use of the virtual node abstraction as an extension of the *Virtual Kubelet* project [16]. In Kubernetes, the kubelet is the primary node agent, responsible for registering the node with the control plane and handling the scheduling of the pods. The virtual ubelet replaces a traditional kubelet for a physical node through the standard Kubernetes APIs with both the local and the remote clusters [69].

Moreover, the virtual node summarizes and abstracts the amount of resources shared by a given remote cluster. In terms of vertical scaling, one key role that the Virtual Kubelet has is to offload the local pods scheduled on the virtual node to the remote cluster and to allocate the amount of resources (e.g., CPU, memory) shared by the remote cluster. It also automatically propagates the negotiated configuration (Services, ConfigMaps, Secrets, Storage) into the capacity required for the proper execution of the offloaded workloads, a mechanism that is called *resource reflection*. All the available resources that exist in a particular namespace and are selected for offloading are automatically propagated into the corresponding twin namespaces created in the selected remote clusters.

Liqo also has an incorporated failover mechanism for the custom resource, i.e., *ShadowPod* to ensure remote pod resiliency even in case of temporary connectivity loss between the local and remote clusters. The local copy of each resource is considered the source of trust leveraged to synchronize the content of the reflected remote shadow copy. The Virtual Kubelet also ensures a remapping of certain information (e.g., endpoint IPs) that guarantees the uniqueness of different configurations for different clusters.

4. Open5Gs and UERANSIM Configuration

Proposed Test Scenario

In our setup, the 5G network services deployment follows the Open5GS implementation, whereas, for the emulation of the gNB and user configuration, we employ the RAN simulator, UERANSIM [18]. In addition, we run the 5G network functions as CNFs in the Open5GS deployment, which are hosted in separate VMs across two different data centers (DCs) (each VM has allocated 4 cores and 8 GB RAM) in a public cloud environment running in different data centers of a European hyperscaler public cloud provider. The first DC is located in Berlin, and the capacity for transmitting data to and from the internet is 200 Gbps, while the second DC is in the UK and has a capacity of ten times lower than the first DC [70].

In our example, we display in Figure 2 three established network slices corresponding to three RAN schedulers for which the 5G mobile core functions are required: one instance of AMF, SMF, UPF, etc. microservices running on behalf of the first slice and the other two instances of each of the Open5GS network functions running on behalf of the other two slices. These three deployments are able to scale independently based on their respective workloads and QoS service guarantees.

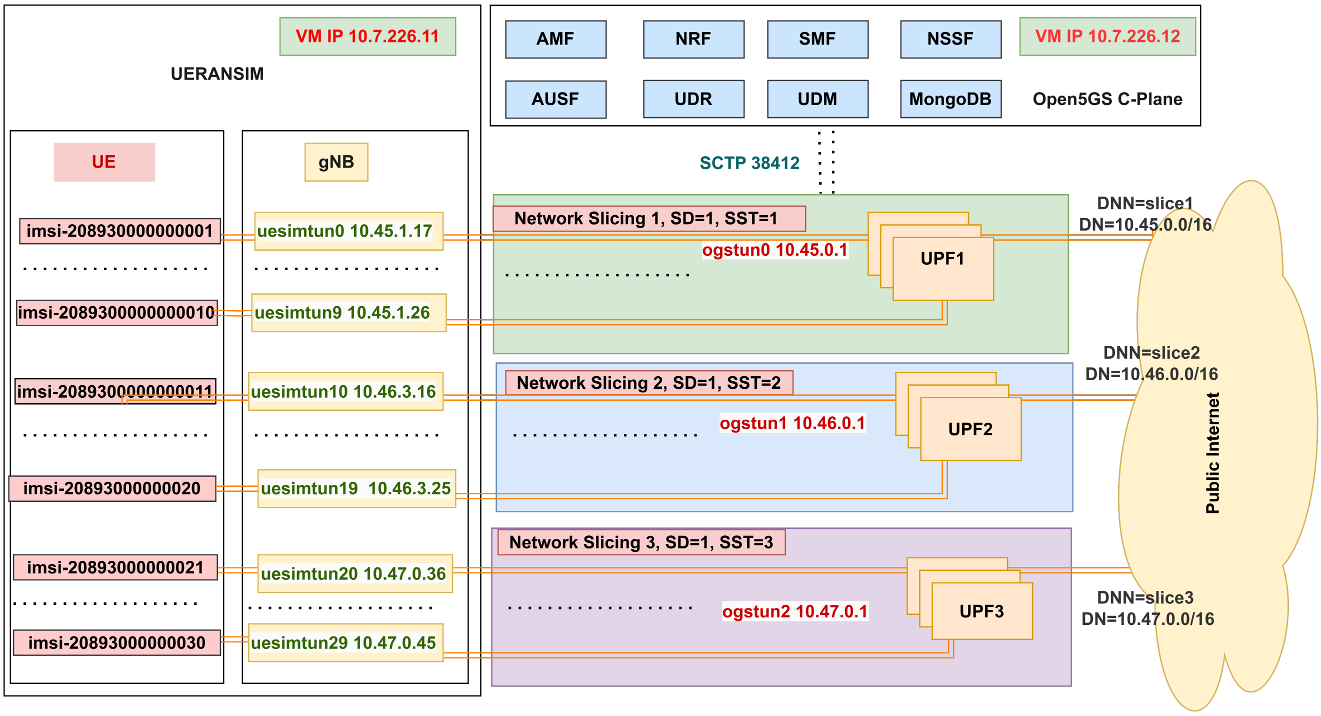


Figure 2. Deployment scenario for end-to-end network slicing.

The three network slices serve different application demands, for instance, based on the configured SSD values. The 3GPP specifies a standard set of network slices, called standardized slice type (SST) values. The value of 1 for SST corresponds to eMBB, which is suitable for the handling of 5G enhanced mobile broadband applications such as streaming of high-quality video, large file transfers, etc. The SST value of 2 is suitable for the handling of URLLC communications for applications including industrial automation and remote control systems, whereas the SST value of 3 is suitable for handling massive IoT devices efficiently and cost-effectively [71]. These values dictate the use cases for verticals; for instance, the values for latency (<100 ms) can cover a multitude of performance requirements that require real-time applications (for instance, smart energy applications [1,72].

Firstly, we create a 5G mobile network (internet reachable) for simulation to set up an environment in which packets can be sent end-to-end with different DNs (data networks) for each DNN (data network name), which is the equivalent of access point name (APN) in LTE. For example, APN does not have any control over the radio access network, whereas network slicing can control the configurations not only for the APN path but for the radio access path as well. In our configuration, the control plane serves multiple user planes, whereas the user plane is connected to multiple DNs.

Secondly, in our scenario, we distribute the traffic as follows: we connect 10 users to DN slice 1, another 10 users are connected to the DN corresponding to slice 2, and the last 10 users are connected to the DN configured for slice 3. This information is also registered in the Open5GS WebUI.

Finally, the deployment of Open5GS is achieved in a declarative manner using Helm charts [73], specifically using YAML templates. In Kubernetes, the configmap represents an API object that stores the key-value pairs, which follows a similar pattern for environment variables. For the configmap of UPF and SMF, we specify the three DN subnet addresses and the corresponding DNN values. We also set three ReplicaSet for the UPF deployment. On each UFP, we need to configure the three ogstun interfaces along with the DN values. In the configmap for AMF, we specify the SST and SD values.

The output of the two cluster provisioning processes consists of two kubeconfig files that contain credentials used to interact with the Kubernetes API server of the workload cluster. Another tool that we employ in our deployment is kubectl, which is an SDK client used to consume access to the Kubernetes API server.

In addition to this, for the current deployment, we employ a service mesh solution, where we inject the Envoy [74] proxy, which is an add-on for Istio [19], at the lever of each service. Both tools operate at the control plane of our deployed application and provide monitoring capabilities for networking in service-to-service communication.

5. Out-of-Band Peering and Network Slicing

The standard peering approach is called *out-of-band control plane* (see Figure 3) because two API servers are mutually communicating to each other over the Liqo control plane traffic, and the user plane is completed via the VPN tunnel interconnecting the two clusters.

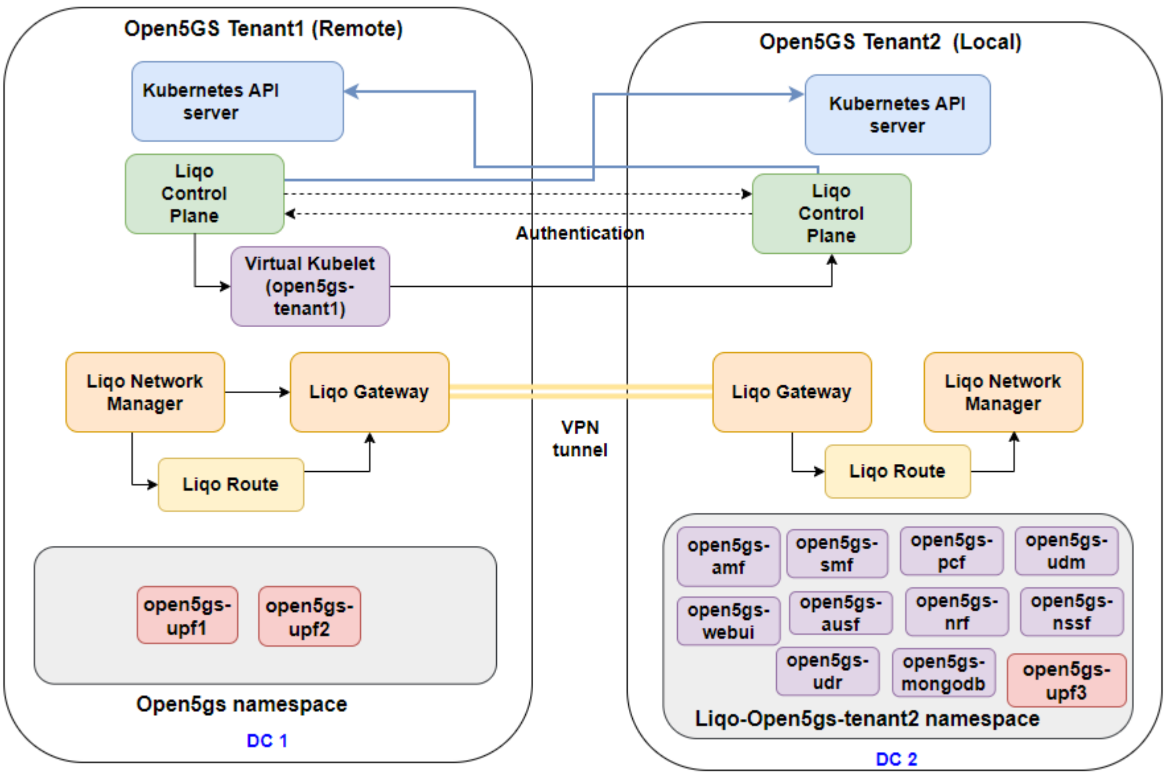


Figure 3. Offloaded user plane in the out-of-band peering.

In the first scenario, we initiate the peering between two K3S clusters that are isolated from each other because they are hosted in different cloud tenants. The Open5GS control plane is deployed on the local cluster, which is Tenant 2, whereas the user plane is offloaded to the remote cluster on Tenant 1. The authentication is performed using a generated authentication token and the K3S API server URL endpoint along with the cluster ID.

The Liqo Network Manager represents the control plane of the Liqo network fabric. It is executed as a pod, and it is responsible for the negotiation of the connection parameters with each remote cluster during the peering process. The overlay network is leveraged to forward all traffic originating from local pods/nodes and redirected to a remote cluster via a gateway that serves as a termination/entry point for the VPN tunnel.

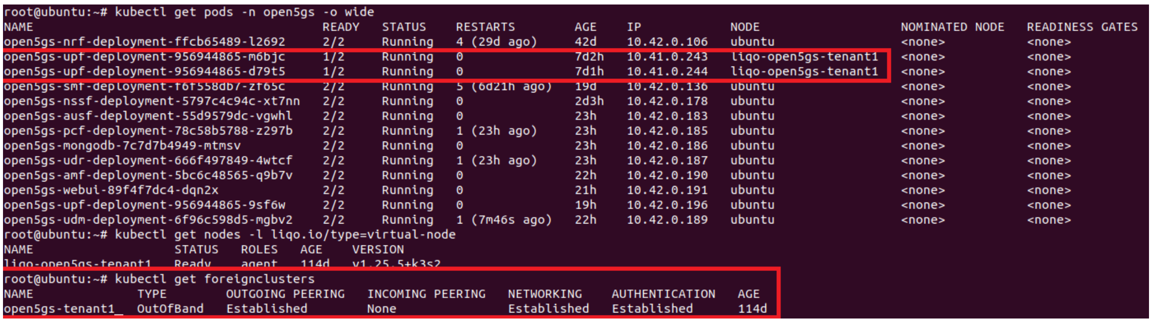
In our Kubernetes clusters, the overlay networking is provided by default CNI (container network interface) for K3S which is Flannel [75]. For each remote cluster, a different instance of the Liqo Virtual Kubelet is started in the local cluster, ensuring the isolation and segregation of the different authentication tokens.

Liqo extends Kubernetes namespaces across the cluster boundaries by creating twin namespaces in the subset of selected remote clusters whenever a namespace is selected for offloading. Remote namespaces host the actual pods offloaded to the corresponding cluster, as well as the additional resources propagated by the resource reflection process.

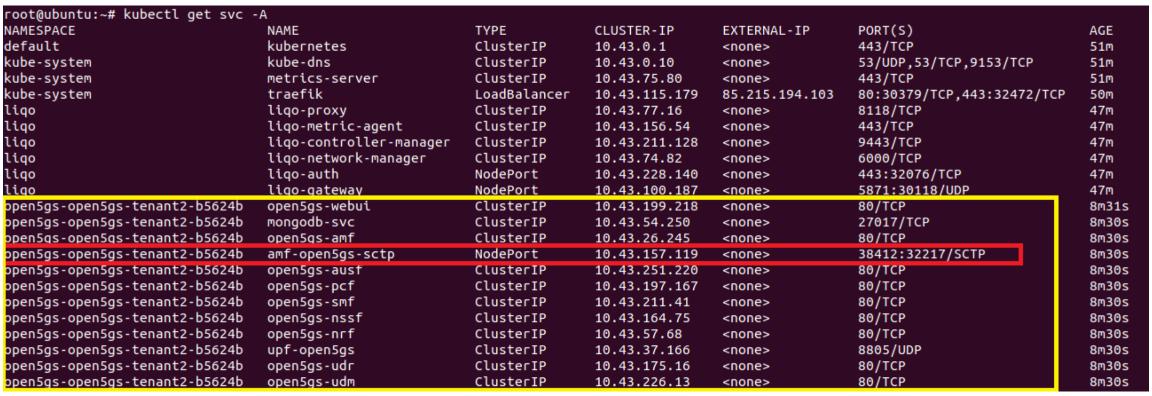
5.1. Out-of-Band Peering—User Plane Offloaded to Foreign Cluster

In the first scenario, we offload the two UPF replicas to the remote peered cluster running on node *liqo-open5gs-tenant1* (see Figure 4a. The AMF pods are running on the local cluster (Tenant2), and the SCTP port 38412 is exposed via *NodePort* on port 30412.

In UERANSIM, we start the connection between the gNB and the 30 users to successfully establish the PDU sessions. In Figure 5, we display the established PDU sessions for slice 1 configuration along with the *uesimtun* interfaces created in the UERANSIM. Figures 6 and 7 show that the other *uesimtun* interfaces created correspond to the second slice and third slice configurations.

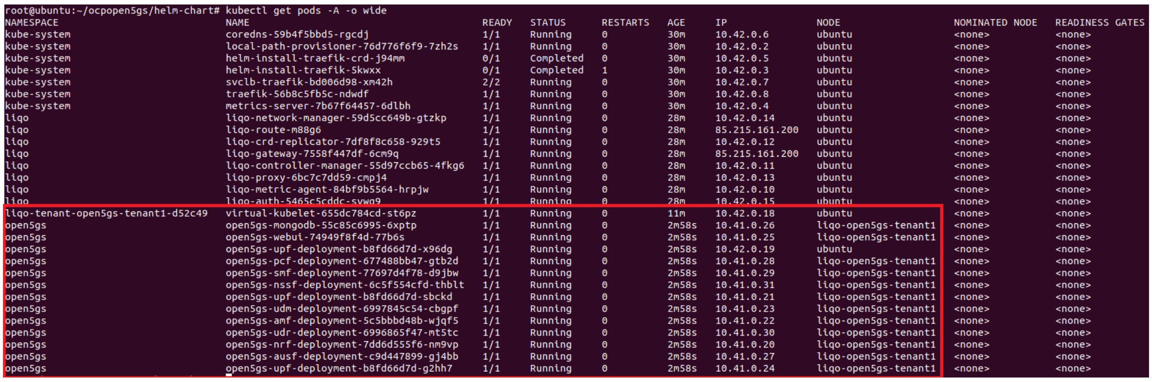


(**a**)



(**b**)

Figure 4. *Cont.*



(**c**)

Figure 4. Proposed Liqo peering scenarios. (**a**) Open5GS user plane services offloaded to foreign cluster in the out-of-band peering scenario. (**b**) Open5GS control plane services offloaded to foreign cluster in the out-of-band peering scenario. (**c**) Open5GS control plane services offloaded to foreign cluster in the in-band peering scenario.

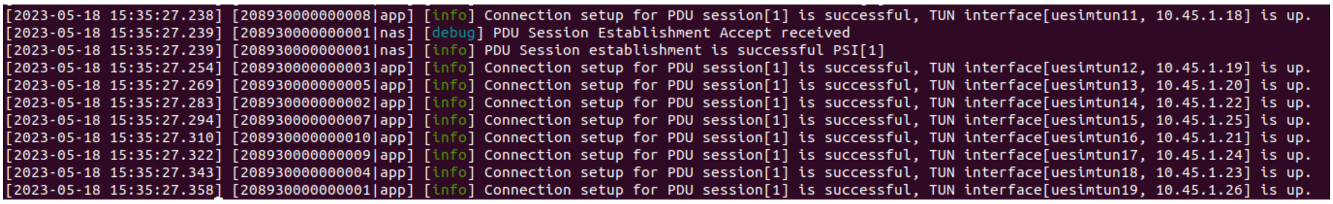


Figure 5. PDU Session establishment for the first slice.

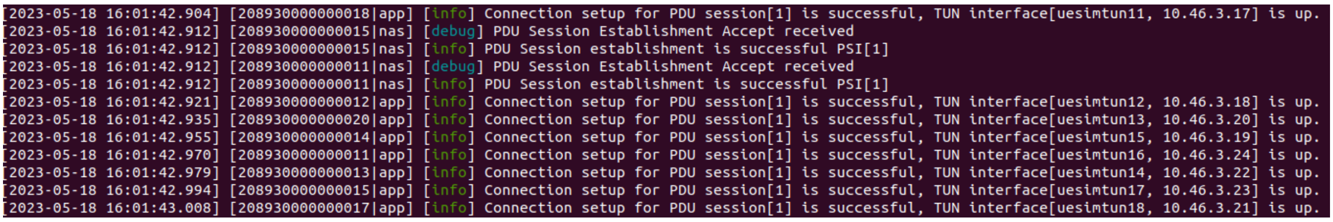


Figure 6. PDU Session establishment for the second slice.

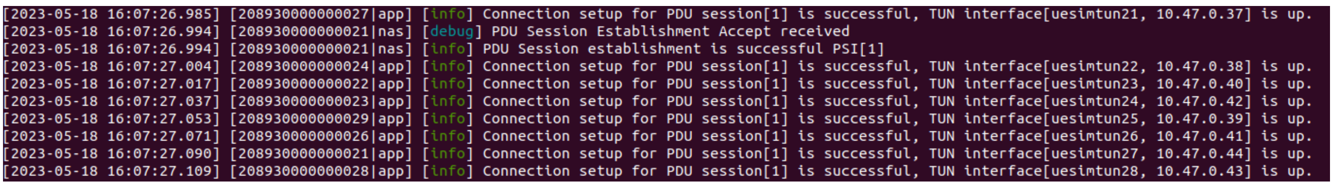


Figure 7. PDU Session establishment for the third slice.

5.2. Out-of-Band Peering—Control Plane Offloaded to Foreign Cluster

In the second scenario, we analyze the Open5GS control plane services offloaded to the remote cluster running on node *liqo-open5gs-tenant2-b5624b* displayed in Figure 4b.

The remote cluster resides in different isolated tenants and regions (Berlin DC vs. UK DC). The authentication service and the network gateway are exposed through a dedicated *NodePort* service configured with private IP addresses as displayed in Figure 4b. In contrast to the first scenario illustrated in Figure 4a, when the AMF service is exposed as *NodePort* on port 30412, now the AMF service is exposed as running on a different port, *32217* (highlighted in yellow frame) corresponding to the SCTP service.

6. In-Band Peering and Network Slicing

The particularity of the in-band peering for the two clusters is that the entire control plane traffic (including authentication services) is going through the VPN tunnel. Figure 4c shows the corresponding control plane functions running on the remote cluster *liqo-open5gs-tenant1*. On the local clusters, the services are running in the namespace open5gs, whereas on the remote cluster, the namespace was created automatically according to the Virtual Kubelet node naming *liqo-open5gs-tenant1-d52c49*.

In Figure 8, we display the logic diagram for the in-band peering connectivity. The prerequisite for establishing a successful connection is that both network gateways should be reachable. Additionally, the Liqo VPN endpoint is reachable from the pods running in the remote cluster. In this scenario, the Kubernetes API service is not to be exposed outside the cluster. The in-band peering involves several steps for the authentication and the VPN establishment using the WireGuard [76] client (see Figure 9). When we display the list of virtual nodes configured, we can see an outgoing peering is established to the foreign cluster *open5gs-tenant1*.

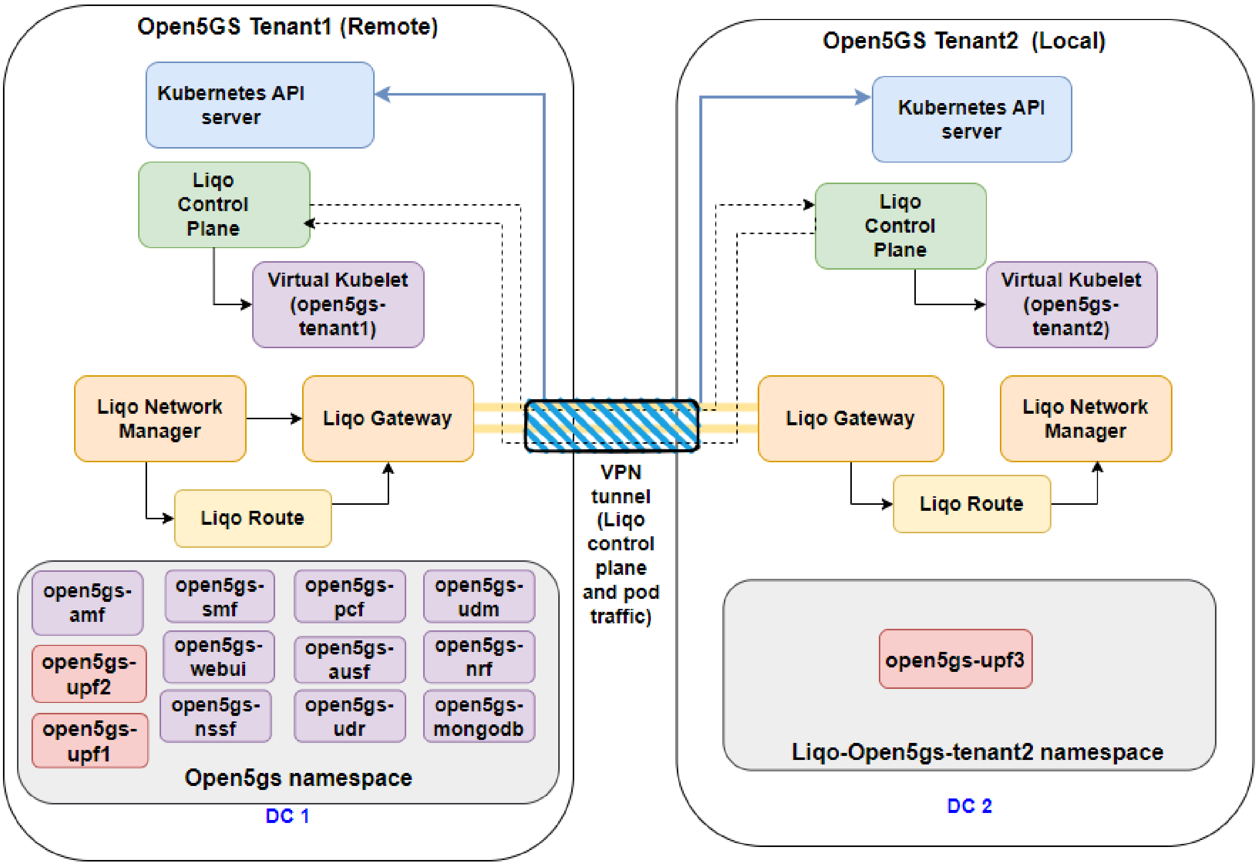


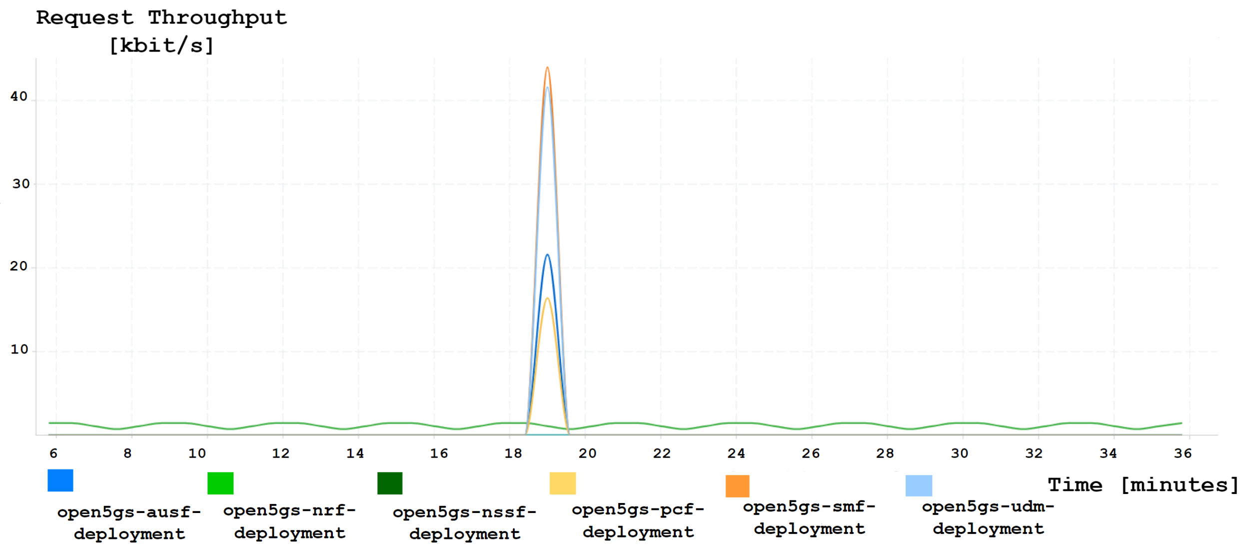
Figure 8. In-band peering offloaded control plane.

```
INFO  (local) tenant namespace "liqo-tenant-open5gs-tenant1-d52c49" created for remote cluster "open5gs-tenant1"
INFO  (remote) tenant namespace "liqo-tenant-open5gs-tenant2-b5624b" created for remote cluster "open5gs-tenant2"
INFO  (local) network configuration created in local cluster "open5gs-tenant2"
INFO  (local) network configuration created in remote cluster "open5gs-tenant1"
INFO  (local) network configuration status correctly reflected from cluster "open5gs-tenant1"
INFO  (remote) network configuration created in local cluster "open5gs-tenant1"
INFO  (remote) network configuration created in remote cluster "open5gs-tenant2"
INFO  (remote) network configuration status correctly reflected from cluster "open5gs-tenant2"
INFO  (local) IPAM service correctly port-forwarded "37243:6000"
INFO  (remote) IPAM service correctly port-forwarded "34643:6000"
INFO  (local) proxy address "10.43.76.247" remapped to "10.44.0.2" for remote cluster "open5gs-tenant1"
INFO  (remote) proxy address "10.43.77.16" remapped to "10.44.0.2" for remote cluster "open5gs-tenant2"
INFO  (local) auth address "10.43.142.25" remapped to "10.44.0.3" for remote cluster "open5gs-tenant1"
INFO  (remote) auth address "10.43.228.140" remapped to "10.44.0.3" for remote cluster "open5gs-tenant2"
INFO  (local) foreign cluster for remote cluster "open5gs-tenant1" correctly configured
INFO  (remote) foreign cluster for remote cluster "open5gs-tenant2" correctly configured
INFO  (local) Network established to the remote cluster "open5gs-tenant1"
INFO  (remote) Network established to the remote cluster "open5gs-tenant2"
INFO  (local) Authenticated to cluster "open5gs-tenant1"
INFO  (remote) Authenticated to cluster "open5gs-tenant2"
INFO  (local) Outgoing peering activated to the remote cluster "open5gs-tenant1"
INFO  (local) Node created for remote cluster "open5gs-tenant1"
INFO  (remote) IPAM service port-forward correctly stopped "34643:6000"
INFO  (local) IPAM service port-forward correctly stopped "37243:6000"
root@ubuntu:~# kubectl get foreignclusters
NAME              TYPE     OUTGOING PEERING   INCOMING PEERING   NETWORKING    AUTHENTICATION   AGE
open5gs-tenant1   InBand   Established        None               Established   Established      2m52s
root@ubuntu:~#  kubectl get nodes -l liqo.io/type=virtual-node
NAME                   STATUS   ROLES   AGE     VERSION
liqo-open5gs-tenant1   Ready    agent   2m56s   v1.26.4+k3s1
```

Figure 9. VPN configuration for in-band peering.

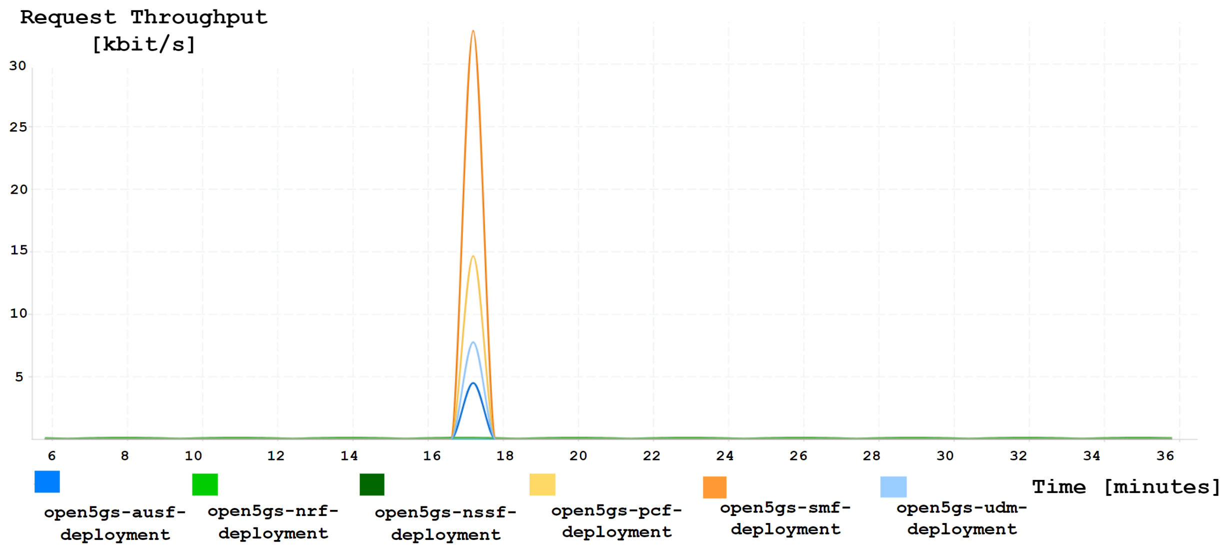
7. Results Analysis

In Figure 10a, we measure the AMF service **request throughput** on the outbound interface for the out-of-band peering scenario, which is the traffic captured at the source for the offloaded Open5GS user plane during the PDU session establishment. We measure a higher maximum throughput for the outgoing AMF request traffic in the offloaded Open5GS user plane compared to the offloaded Open5GS control plane shown in Figure 10b. The reason is mainly due to the PDU establishment traffic generated along with the creation of user interfaces, which is done across tenants for the out-of-band peering.

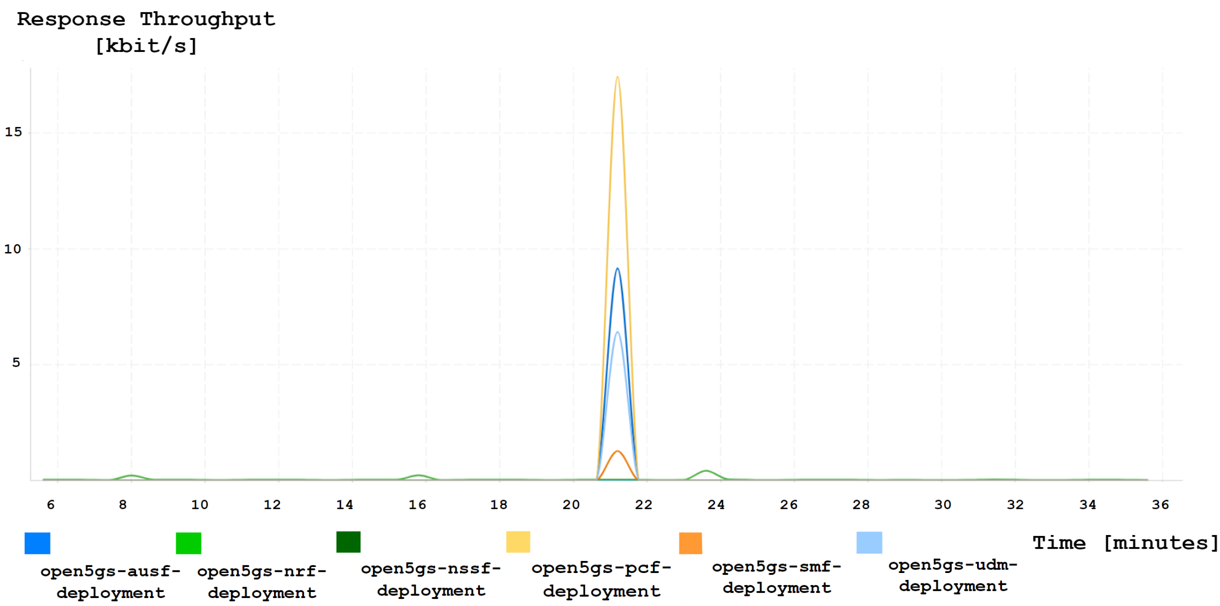


(a)

Figure 10. *Cont.*



(**b**)



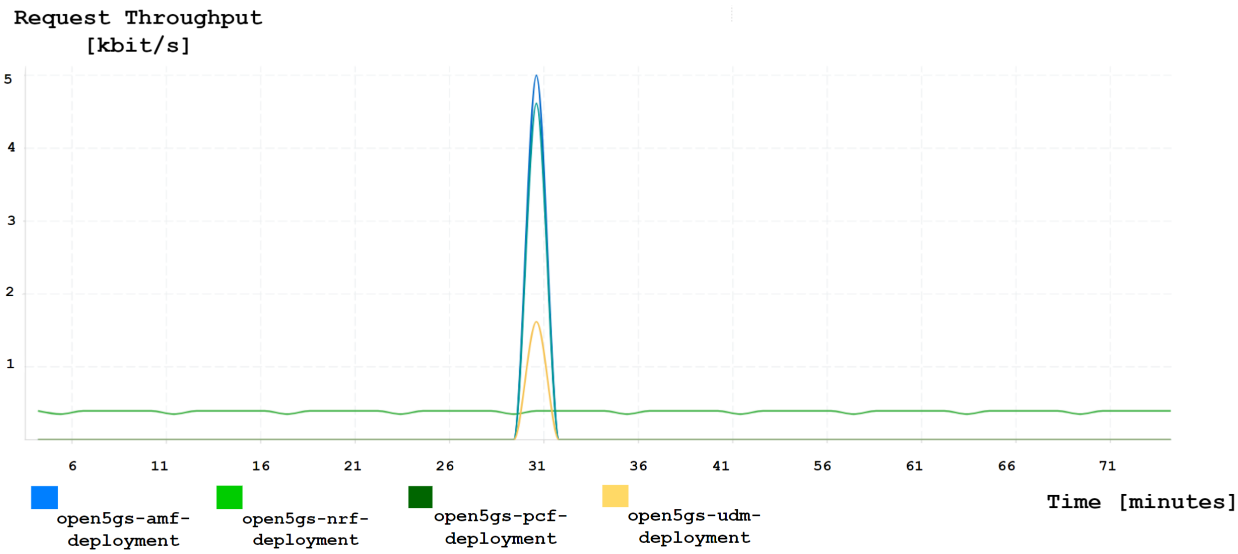
(**c**)

Figure 10. Request throughput on AMF for the out-of-band peering scenario. (**a**) Request throughput on AMF source for offloaded user plane. (**b**) Request throughput on AMF outbound source for offloaded control plane. (**c**) Response throughput on AMF destination for offloaded control plane.

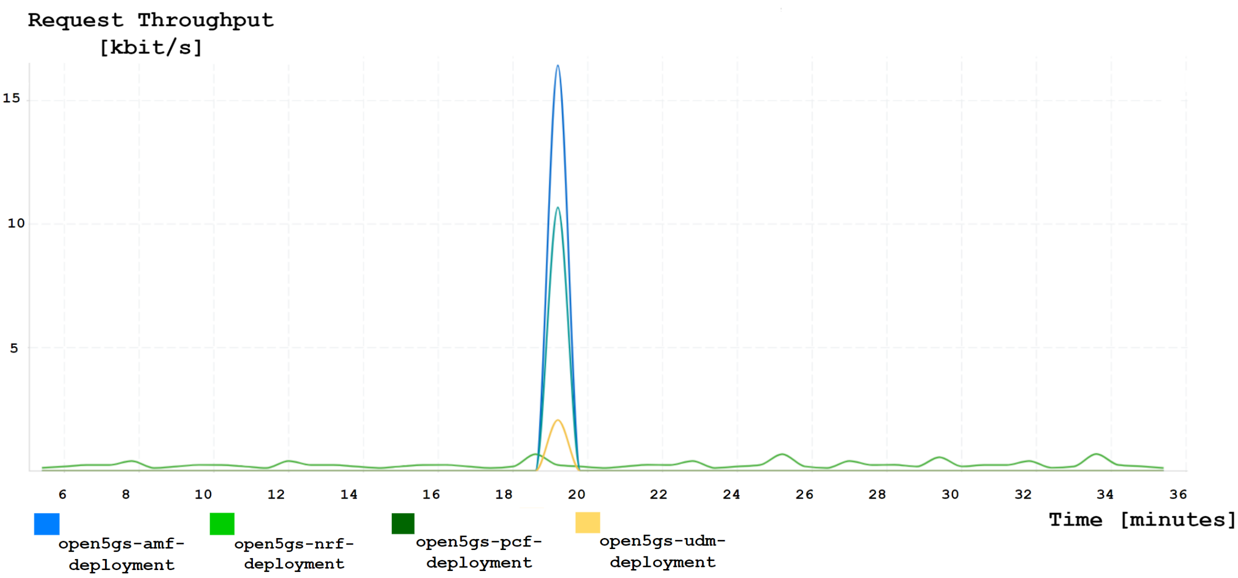
In Figure 10, we can also observe that the AMF maximum **request throughput** on the outgoing interface for the offloaded control plane is twice the **response throughput** value measured at the destination for the same scenario displayed in Figure 10c. For both source and destination outbound traffic, the peak values for the request throughput are generated by the SMF that requires the performance of an AMF selection.

The same set of measurements is obtained for the SMF response and request throughput for both out-of-band peering scenarios. The maximum **request throughput** for the SMF service on the outbound destination for the Open5GS offloaded control plane (Figure 11b) is three times higher than in the case of the offloaded user plane; see Figure 11a. On the other hand, for the **response throughput** at the source in the case of the offloaded user plane on the remote cluster displayed in Figure 11c, we register half the value of the **request throughput** measured at the destination for the same scenario (see Figure 11a).

In Figure 12, we display the latency expressed in milliseconds for each of the user plane functions in the out-of-band scenario where two UPFs are offloaded to the remote cluster. To measure the latency, we used the *ping* tool that gives us the round trip time (RTT) of transactions [77]. Ping uses ICMP packets; for instance, for an interval of 10 ms, ping sends one ICMP packet per second to the specified IP address until it has sent 100 packets.

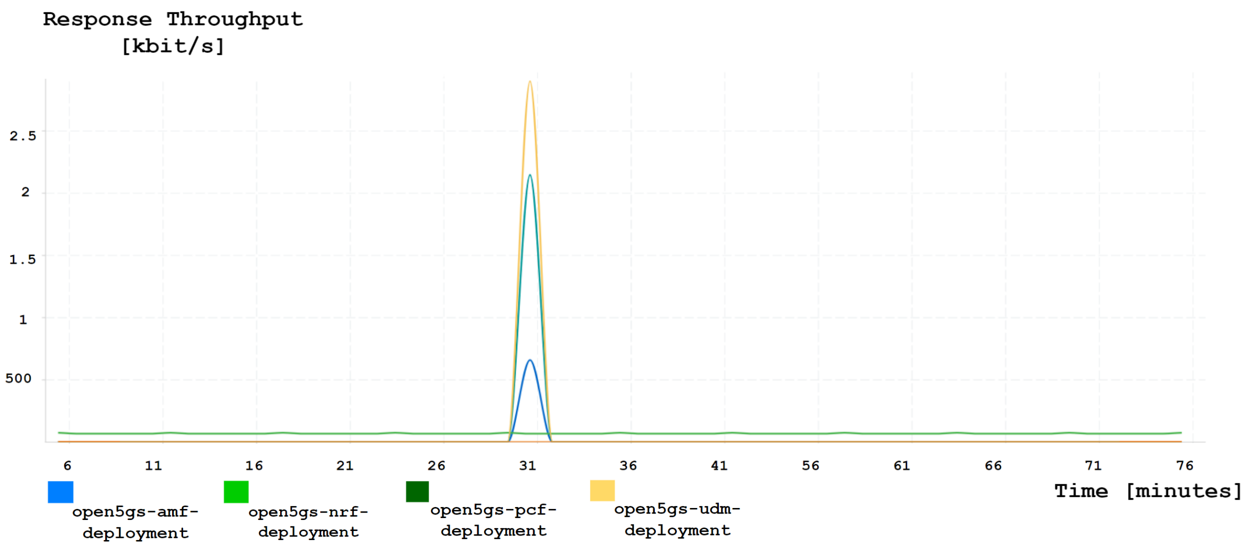


(**a**)



(**b**)

Figure 11. *Cont.*



(**c**)

Figure 11. Request throughput on SMF for the out-of-band peering scenario. (**a**) Request throughput on SMF destination for offloaded user plane. (**b**) Request throughput on SMF destination for offloaded control plane. (**c**) Response throughput on SMF source for offloaded user plane.

In the out-of-band scenario, we observe that the latency for UPF1, which represents the user plane running on the local cluster, is smaller (around 1 ms) in comparison to the other two user functions hosted on the remote cluster, where the latency is above 2 ms as displayed in Figure 12. In the case of the in-band peering, the values in terms of latency for the three UPF functions are similar since the user plane traffic between the two clusters is flowing through the VPN tunnel (see Figure 13).The end-to-end latency when we ping google.com (accessed on 9 April 2024) is displayed in both Figures 14 and 15. In both out-of-band and in-band peering scenarios, the registered end-to-end latency is around 11 ms for all three network slices when running simultaneously 10 user connections for each network slice. Since the obtained values for end-to-end latency measured for all three user plane functions are less than 100 ms when running all 30 user connections, a wide range of vertical use cases can be covered, from streaming to smart energy applications.

Figure 12. User plane latency for out-of-band peering scenario.

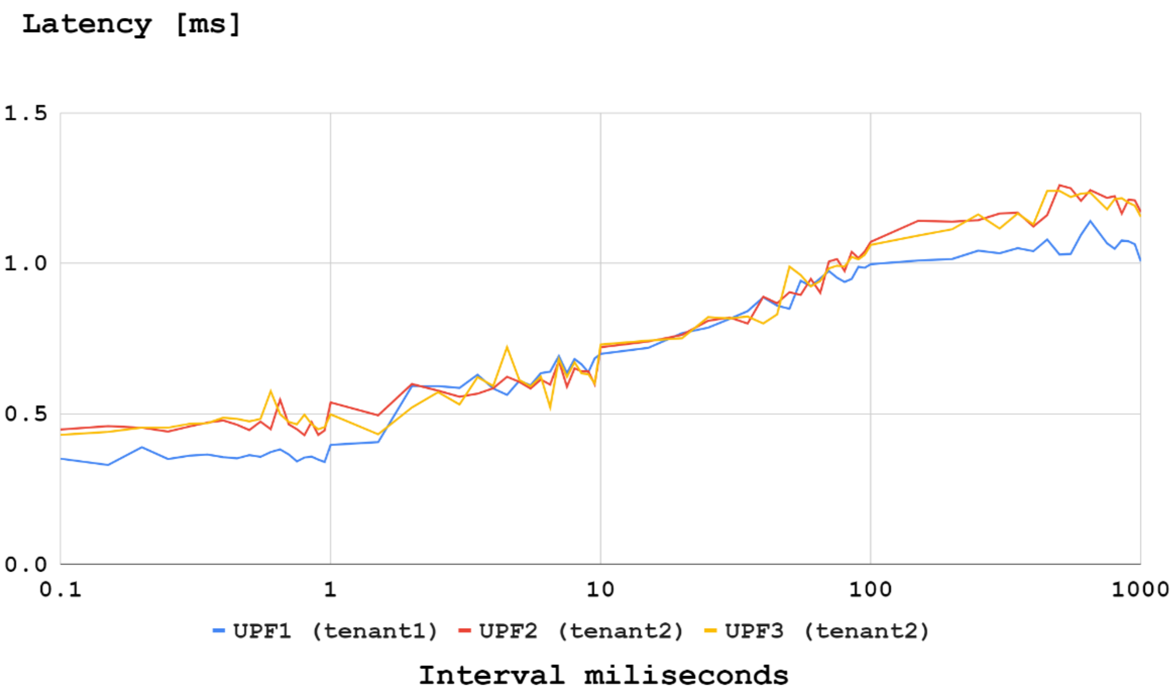


Figure 13. User plane latency for in-band peering scenario.

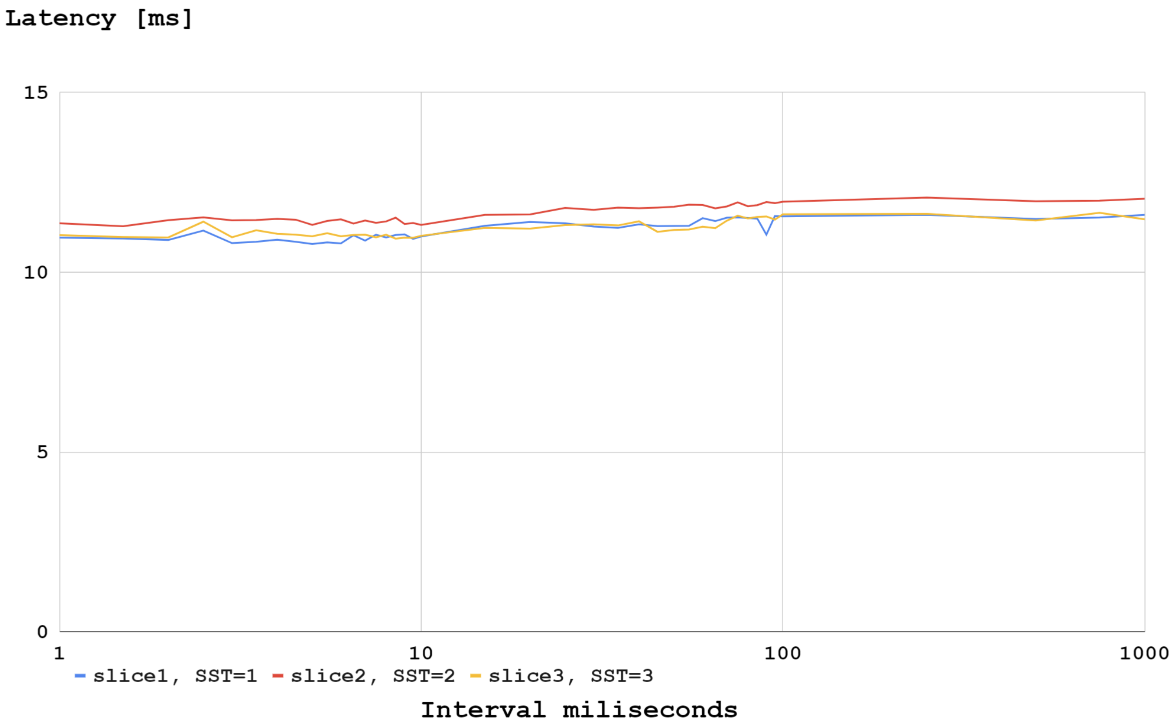


Figure 14. End-to-end user latency for different slices in the out-of-band peering scenario.

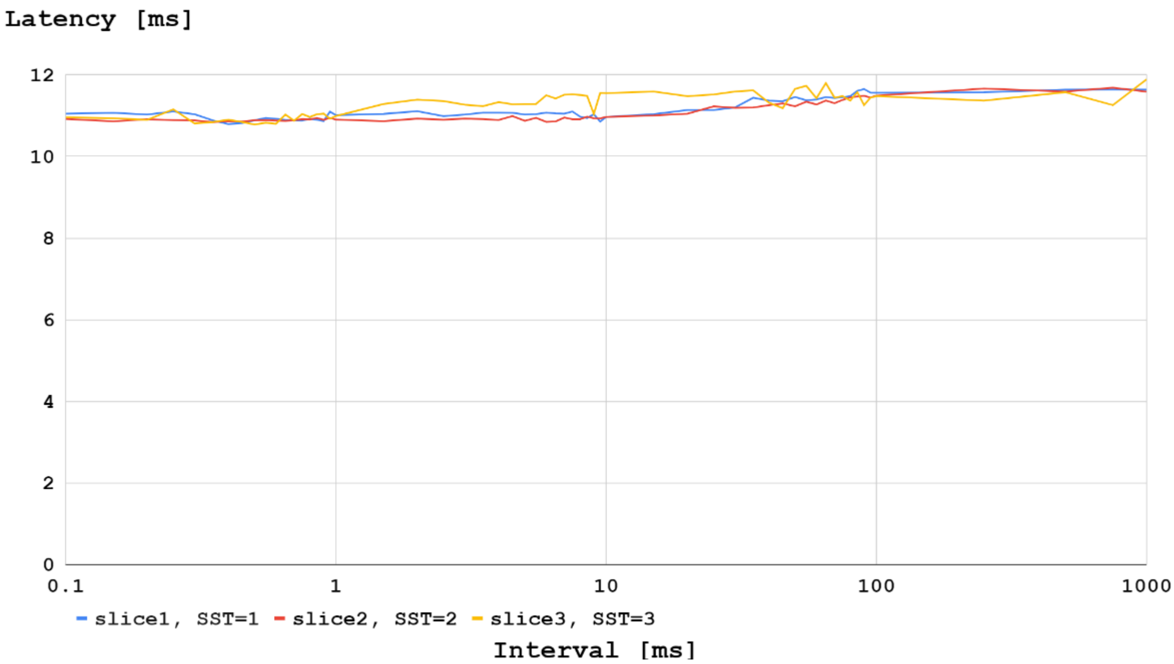


Figure 15. End-to-end user latency for different slices in the in-band peering scenario.

In Figure 16, we display the throughput in Mbit/s for the out-of-band peering scenario. Throughput is measured for all 30 user sessions using the *iperf3* [78] tool for an interval of 60 s. Through these measurements, we showcase the connectivity between the radio access network and the user plane functions. We can see that the throughput for the UPF1 hosted on the local cluster is consistently higher than the other two user plane functions that reside on the remote cluster.

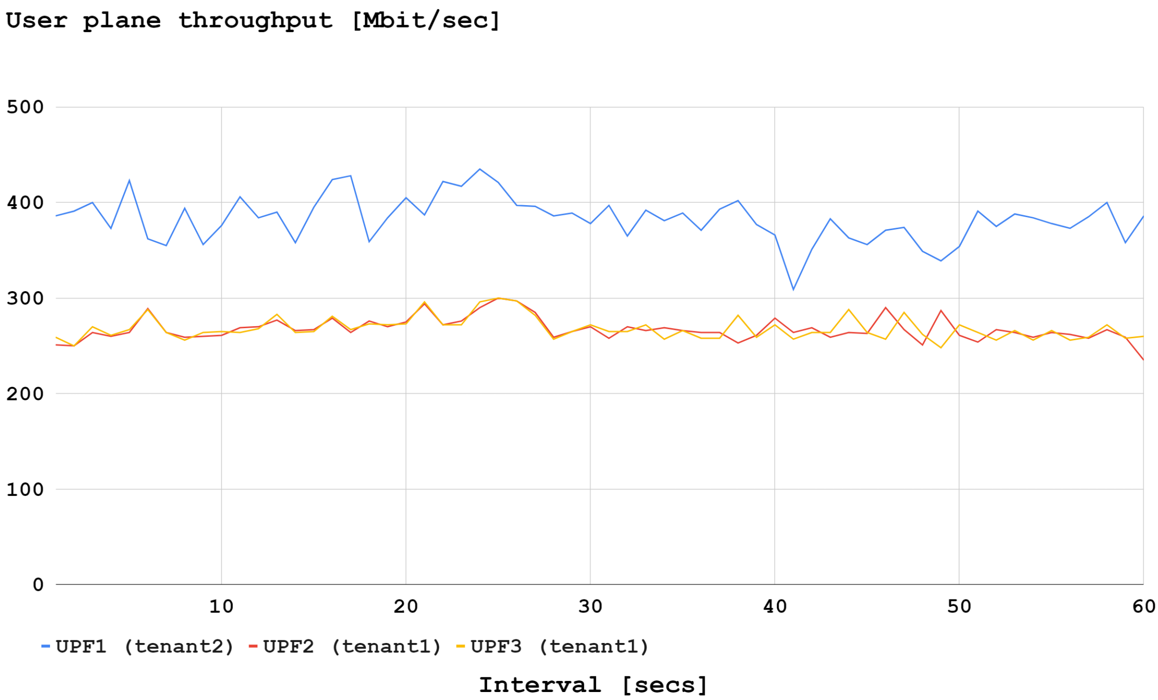


Figure 16. User Plane throughput for the out-of-band peering scenario.

8. Conclusions and Future Work

This paper presented a novel approach to the multi-site cloud native 5G capabilities in terms of vertical and horizontal scalability for Kubernetes clusters and cross-connectivity in different data centers by employing the Liqo operator to share resources. In this manner, we highlighted the on-demand network capacity, which is limited only by the number of utilized resources and clusters in the current testbed, as well as the programability of the network and mobile core configuration by leveraging the benefits of APIs. To complement the full stack of the MANO framework, a service mesh solution was integrated to ensure observability and monitoring for each service. Moreover, this proposed design is enriched with the CI/CD capabilities of a cloud-native solution that eases the deployment and integration of components compared to the MANO framework. Due to the increase in traffic and different vertical needs in terms of QoS, the proposed approach could bring not only cost savings to MNOs but also improve network performance along with a higher computational power that can be triggered upon request as well as high availability and fault tolerance of the infrastructure ensured by the cloud provider.

In our setup, we analyze two peering scenarios (out-of-band and in-band) for offloading of both user and control planes. The aim of this paper was also to provide insight into inter-service monitoring; therefore, the above measurements were validated through a service-mesh monitoring solution dedicated to cloud-native applications. In addition, we analyze the response and request throughput of the AMF and SMF across data centers to demonstrate the system's performance. We observed that maximum registered values for the AMF request throughput offloaded user plane and control plane measured at the outbound source are two times higher than the AMF response throughput measured at the destination for the offloaded control plane, whereas the values registered for the SMF request throughput for both user and control planes are twice or three times higher for the SMF request throughput registered on the outbound source interface. Since SMF is responsible for the AMF selection, higher values for the SMF throughput are achieved for the offloaded control plane in the remote clusters.

In addition, we determined the latency for the three UPF tunnels and the end-to-end user latency for three instantiated network slices. We observed that for in-band peering where a VPN tunnel is established, the user plane latency and the achieved throughput for two offloaded user plane functions have similar values compared to the latency registered for the UPF deployed in the local cluster, which is by 100 Mbit/s lower than the registered latency for the other two user plane functions. In terms of the measured end-to-end latency for the user connectivity to the Internet, the latency for all three configured network slices is slightly lower (by 1–2 ms) in the case of the in-band peering.

As a proposed future work, our goal is to replicate the deployment across different public cloud providers, addressing the proposed test scenarios in a different setup using a physical RAN solution. Furthermore, we can also extend the end-to-end network slicing to new use cases for 6G to serve vertical demands, such as streaming, IoT, and vehicular communication, to validate the level of service guarantee accommodating different vertical demands.

Author Contributions: Conceptualization, O.-M.D.-G. and V.C.; methodology, O.-M.D.-G. and V.C.; software, O.-M.D.-G.; validation, V.C. and R.K.; formal analysis, O.-M.D.-G.; investigation, O.-M.D.-G.; resources, O.-M.D.-G.; data curation, O.-M.D.-G. and V.C.; writing—original draft preparation, O.-M.D.-G.; writing—review and editing, O.-M.D.-G.; visualization, O.-M.D.-G.; supervision, V.C. and R.K.; project administration, V.C.; funding acquisition, V.C. and R.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Delft University of Technology (TUDelft), The Netherlands.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AMF	Access and Mobility Management Function
APN	Access Point Name
BSS	Business support system
CAPI	Cluster API
CI/CD	Continuous Integration and Continuous Delivery
CIS	Container Infrastructure Service
CISM	Container Infrastructure Service Management
CISI	Container Infrastructure Service Instance
CNF	Containerized Network Functions
DC	Data Center
DNN	Data Network Name
ETSI MANO	Network Functions Virtualization Management and Orchestration
eMBB	enhanced Mobile Broadband
HPA	Horizontal Pod Autoscaler
MEC	Multi-access Edge Computing
NSM	Network Service Mesh
MNO	Mobile Network Operators
NFV	Network Functions Virtualization
NFV MANO	Network Functions Virtualization Management and Orchestration
NFO	NFV Orchestrator
OSS	Operation Support System
PDU	Protocol Data Units
RAN	Radio Access Network
SBA	Service-Based Architecture
SCTP	Stream Control Transmission Protocol
SLA	Service-level Agreement
SSD	Service Differentiator
SMF	Session Management Function
SST	Slice/Service Types
UPF	User Plane Function
URLLC	Ultra-Reliable Low-Latency Communications
VIM	Virtualized Infrastructure Manager
VNF	Virtual Network Functions
VNFM	VNF Manager
VPA	Vertical Pod Autoscaler

References

1. Tam, P.; Ros, S.; Song, I.; Kim, S. QoS-Driven Slicing Management for Vehicular Communications. *Electronics* **2024**, *13*, 314. [CrossRef]
2. ETSI GS. 5G; System Architecture for the 5G System (5GS) (3GPP TS 23.501 Version 16.6.0 Release 16). Available online: https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf (accessed on 10 March 2024).
3. ONAP. Open Network Automation Platform. Available online: https://www.onap.org (accessed on 9 April 2024).
4. ETSI. Open Source MANO. Available online: https://osm.etsi.org (accessed on 15 March 2024).
5. Yilma, G.M.; Yousaf, F.; Sciancalepore, V.; Costa-Pérez, X. Benchmarking Open-Source NFV MANO Systems: OSM and ONAP. *Comput. Commun.* **2020**, *161*, 86–98. [CrossRef]
6. Alawe, I.; Hadjadj-Aoul, Y.; Ksentini, A.; Bertin, P.; Viho, C.; Darche, D. Smart Scaling of the 5G Core Network: An RNN-Based Approach. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6. [CrossRef]
7. Cunha, J.; Ferreira, P.; Castro, E.; Oliveira, P.; Nicolau, M.; Núñez, I.; Sousa, X.; Serodio, C. Enhancing Network Slicing Security: Machine Learning, Software-Defined Networking, and Network Functions Virtualization-Driven Strategies. *Future Internet* **2024**, *16*, 226. [CrossRef]
8. Tipantuña, C.; Hesselbach, X. Adaptive Energy Management in 5G Network Slicing: Requirements, Architecture, and Strategies. *Energies* **2020**, *13*, 3984. [CrossRef]
9. Moreno-Vozmediano, R.; Montero, R.S.; Huedo, E.; Llorente, I.M. Intelligent Resource Orchestration for 5G Edge Infrastructures. *Future Internet* **2024**, *16*, 103. [CrossRef]

10. Dolente, F.; Garroppo, R.G.; Pagano, M. A Vulnerability Assessment of Open-Source Implementations of Fifth-Generation Core Network Functions. *Future Internet* **2024**, *16*, 1. [CrossRef]
11. Kim, J.; Xie, M. A Study of Slice-Aware Service Assurance for Network Function Virtualization. In Proceedings of the 2019 IEEE Conference on Network Softwarization (NetSoft), Paris, France, 24–28 June 2019; pp. 489–497. [CrossRef]
12. Rodriguez, V.Q.; Guillemin, F.; Boubendir, A. 5G E2E Network Slicing Management with ONAP. In Proceedings of the 2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 24–27 February 2020; pp. 87–94. [CrossRef]
13. GSMA. E2E Network Slicing Architecture, Version 1.0. Available online: https://www.gsma.com/newsroom/wp-content/uploads/NG.127-v1.0-2.pdf (accessed on 30 September 2023).
14. Kahvazadeh, S.; Khalili, H.; Nikbakht Silab, R.; Bakhshi, B.; Mangues-Bafalluy, J. Vertical-oriented 5G platform-as-a-service: User-generated content case study. In Proceedings of the 2022 IEEE Future Networks World Forum (FNWF), Montreal, QC, Canada, 10–14 October 2022.
15. Liqo. Enable Dynamic and Seamless Kubernetes Multi-Cluster Topologies. Available online: https://liqo.io/ (accessed on 12 March 2024).
16. Virtual Kubelet. Available online: https://virtual-kubelet.io/docs/ (accessed on 10 March 2024).
17. Open5GS. Open-Source Project of 5GC and EPC (Release 16). Available online: https://open5gs.org (accessed on 8 March 2024).
18. UERANSIM. Available online: https://github.com/aligungr/UERANSIM (accessed on 20 March 2024).
19. Istio. The Istio Service Mesh. Available online: https://istio.io (accessed on 18 March 2024).
20. SONATA. SONATA NFV Platform. Available online: https://www.sonata-nfv.eu (accessed on 18 March 2024).
21. OpenStack. OpenStack Tacker. Available online: https://wiki.openstack.org/wiki/Tacker (accessed on 18 March 2024).
22. Cloudify. Available online: https://docs.cloudify.co/ (accessed on 18 March 2024).
23. Yilma, G.M.; Yousaf, F.Z.; Sciancalepore, V.; Costa-Pérez, X. On the Challenges and KPIs for Benchmarking Open-Source NFV MANO Systems: OSM vs. ONAP. *arXiv* **2019**, arXiv:1904.10697.
24. Arampatzis, D.; Apostolakis, K.; Margetis, G.; Stephanidis, C.; Atxutegi, E.; Amor, M.; Pietro, N.; Henriques, J.; Cordeiro, L.; Carapinha, J.; et al. Unification architecture of cross-site 5G testbed resources for PPDR verticals. In Proceedings of the 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), Athens, Greece, 7–10 September 2021; pp. 13–19. [CrossRef]
25. Foukas, X.; Marina, M.; Kontovasilis, K. Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture. In Proceedings of the MobiCom'17: The 23rd Annual International Conference on Mobile Computing and Networking, Snowbird, UT, USA, 16–20 October 2017; pp. 127–140. [CrossRef]
26. Shorov, A. 5G Testbed Development for Network Slicing Evaluation. In Proceedings of the 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Saint Petersburg and Moscow, Russia, 28–31 January 2019; pp. 39–44. [CrossRef]
27. OPNFV. Available online: https://www.opnfv.org/ (accessed on 11 March 2024).
28. OpenStack. Available online: https://www.openstack.org (accessed on 11 March 2024).
29. VMware. Available online: https://www.vmware.com/ (accessed on 27 July 2024).
30. Garcia-Aviles, G.; Gramaglia, M.; Serrano, P.; Banchs, A. POSENS: A Practical Open Source Solution for End-to-End Network Slicing. *IEEE Wirel. Commun.* **2018**, *25*, 30–37. [CrossRef]
31. Garcia-Aviles, G.; Gramaglia, M.; Serrano, P.; Gringoli, F.; Fuente-Pascual, S.; Labrador Pavon, I. Experimenting with open source tools to deploy a multi-service and multi-slice mobile network. *Comput. Commun.* **2020**, *150*, 1–12. [CrossRef]
32. Huang, C.Y.; Ho, C.Y.; Nikaein, N.; Cheng, R.G. Design and Prototype of A Virtualized 5G Infrastructure Supporting Network Slicing. In Proceedings of the 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP), Shanghai, China, 19–21 November 2018; pp. 1–5. [CrossRef]
33. Chang, W.-C.; Lin, F. Coordinated Management of 5G Core Slices by MANO and OSS/BSS. *J. Comput. Commun.* **2021**, *9*, 52–72. [CrossRef]
34. Wang, Q.; Alcaraz-Calero, J.; Ricart-Sanchez, R.; Weiss, M.; Gavras, A.; Nikaein, N.; Vasilakos, X.; Bernini, G.; Pietro, G.; Roddy, M.; et al. Enable Advanced QoS-Aware Network Slicing in 5G Networks for Slice-Based Media Use Cases. *IEEE Trans. Broadcast.* **2019**, *65*, 444–453. [CrossRef]
35. Esmaeily, A.; Kralevska, K.; Gligoroski, D. A Cloud-based SDN/NFV Testbed for End-to-End Network Slicing in 4G/5G. In Proceedings of the 2020 6th IEEE Conference on Network Softwarization (NetSoft), Ghent, Belgium, 29 June–3 July 2020; pp. 29–35. [CrossRef]
36. Sarrigiannis, I.; Kartsakli, E.; Ramantas, K.; Antonopoulos, A.; Verikoukis, C. Application and Network VNF migration in a MEC-enabled 5G Architecture. In Proceedings of the 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Barcelona, Spain, 17–19 September 2018; pp. 1–6. [CrossRef]
37. ONF. Aether. Available online: https://docs.aetherproject.org/aether-2.0 (accessed on 12 December 2023).
38. 5G-Berlin. Available online: https://5g-berlin.de/5g-testbed (accessed on 29 July 2024).
39. 5G-GENESIS. Available online: https://www.thegene5is5g.com/ (accessed on 29 July 2024).
40. Free5GMANO. Available online: https://github.com/free5gmano/free5gmano (accessed on 29 July 2024).
41. Prometheus. Monitoring System & Time Series Database. Available online: https://prometheus.io/ (accessed on 29 July 2024).

42. OpenTelemetry. Available online: https://opentelemetry.io/ (accessed on 29 July 2024).
43. Arouk, O.; Nikaein, N. 5G Cloud-Native: Network Management & Automation. In Proceedings of the NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–2.
44. Barrachina-Muñoz, S.; Baranda, J.; Payaró, M.; Mangues-Bafalluy, J. Intent-Based Orchestration for Application Relocation in a 5G Cloud-native Platform. In Proceedings of the 2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Phoenix, AZ, USA, 14–16 November 2022; pp. 94–95. [CrossRef]
45. Kaur, K.; Guillemin, F.; Rodriguez, V.Q.; Sailhan, F. Latency and network aware placement for cloud-native 5G/6G services. In Proceedings of the 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC), Phoenix, AZ, USA, 14–16 November 2022; pp. 114–119. [CrossRef]
46. Khichane, A.; Fajjari, I.; Aitsaadi, N.; Gueroui, M. Cloud Native 5G: An Efficient Orchestration of Cloud Native 5G System. In Proceedings of the NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, Las Vegas, NV, USA, 8–11 January 2022; pp. 1–9. [CrossRef]
47. Arouk, O.; Nikaein, N. Kube5G: A Cloud-Native 5G Service Platform. In Proceedings of the GLOBECOM 2020-2020 IEEE Global Communications Conference, Budapest, Hungary, 25–29 April 2020; pp. 1–6. [CrossRef]
48. Barrachina-Muñoz, S.; Nikbakht, R.; Baranda, J.; Payaró, M.; Mangues-Bafalluy, J.; Kokkinos, P.; Soumplis, P.; Kretsis, A.; Varvarigos, E. Deploying cloud-native experimental platforms for zero-touch management 5G and beyond networks. *IET Netw.* **2023**, *12*, 305–315. [CrossRef]
49. Grafana. Grafana: The Observability Platform. Available online: https://grafana.com/ (accessed on 29 July 2024).
50. Barrachina-Muñoz, S.; Payaró, M.; Mangues-Bafalluy, J. Cloud-native 5G experimental platform with over-the-air transmissions and end-to-end monitoring. *arXiv* **2022**, arXiv:2207.11936.
51. AMARI Callbox Ultimate. Available online: https://www.amarisoft.com/app/uploads/2022/01/AMARI-Callbox-Ultimate.pdf (accessed on 30 September 2023).
52. RedHad. How We Designed a 5G Core Platform That Scales Well. Available online: https://www.redhat.com/architect/autoscale-5g-core (accessed on 12 March 2024).
53. Ungureanu, O.M.; Vlădeanu, C.; Kooij, R. Collaborative Cloud-Edge: A Declarative API orchestration model for the NextGen 5G Core. In Proceedings of the 2021 IEEE International Conference on Service-Oriented System Engineering (SOSE), Oxford, UK, 23–26 August 2021; pp. 124–133. [CrossRef]
54. Kubernetes SIGs. ClusterAPI. Available online: https://cluster-api.sigs.k8s.io (accessed on 12 March 2024).
55. Rancher. K3s. Available online: https://k3s.io (accessed on 12 March 2024).
56. Mfula, H.; Ylä-Jääski, A.; Nurminen, J. Seamless Kubernetes Cluster Management in Multi-Cloud and Edge 5G Applications. In Proceedings of the International Conference on High Performance Computing & Simulation (HPCS 2020), Virtual, 22–27 March 2021.
57. Osmani, L.; Kauppinen, T.; Komu, M.; Tarkoma, S. Multi-Cloud Connectivity for Kubernetes in 5G Networks. *IEEE Commun. Mag.* **2021**, *59*, 42–47. [CrossRef]
58. Ungureanu, O.M.; Vlădeanu, C. Leveraging the cloud-native approach for the design of 5G NextGen Core Functions. In Proceedings of the 2022 14th International Conference on Communications (COMM), Bucharest, Romania, 16–18 June 2022; pp. 1–7. [CrossRef]
59. Linkerd. A Different Kind of Service Mesh. Available online: https://linkerd.io/ (accessed on 22 March 2024).
60. ETSI. Network Functions Virtualisation (NFV) Release 3; Architecture; Report on the Enhancements of the NFV Architecture towards Cloud-Native and PaaS. Available online: https://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/029/03.03.01_60/gr_NFV-IFA029v030301p.pdf (accessed on 12 March 2024).
61. AWS. Mapping ETSI MANO to Kubernetes. Available online: https://docs.aws.amazon.com/pdfs/whitepapers/latest/ETSI-NFVO-compliant-orchestration-in-kubernetes/ETSI-NFVO-compliant-orchestration-in-kubernetes.pdf (accessed on 29 July 2024).
62. ONF. SD-Core. Available online: https://opennetworking.org/sd-core/ (accessed on 29 July 2024).
63. Hossein Ashtari, Spiceworks. Horizontal vs. Vertical Cloud Scaling: Key Differences and Similarities. Available online: https://www.spiceworks.com/tech/cloud/articles/horizontal-vs-vertical-cloud-scaling/ (accessed on 12 March 2024).
64. Kubernetes. Horizontal Pod Autoscaling. Available online: https://docs.sd-core.opennetworking.org (accessed on 12 March 2024).
65. Kubernetes. Vertical Pod Autoscaler. Available online: https://github.com/kubernetes/autoscaler/blob/master/vertical-pod-autoscaler/README.md (accessed on 12 March 2024).
66. RedHat. RedHat OpenShift. Available online: https://docs.openshift.com/ (accessed on 12 March 2024).
67. Crossplane. Available online: https://www.crossplane.io (accessed on 12 March 2024).
68. Iorio, M.; Risso, F.; Palesandro, A.; Camiciotti, L.; Manzalini, A. Computing Without Borders: The Way Towards Liquid Computing. *IEEE Trans. Cloud Comput.* **2023**, *11*, 2820–2838. [CrossRef]
69. Kubernetes. Kubernetes Components. Available online: https://kubernetes.io/docs/concepts/overview/components/ (accessed on 15 March 2024).
70. Ionos. Ionos Cloud. Available online: https://docs.ionos.com/cloud/compute/networks/overview (accessed on 15 March 2024).

71. Techplayon. 5G RAN and 5GC Network Slice Signaling. Available online: https://www.techplayon.com/5g-ran-and-5gc-network-slice-signaling (accessed on 15 March 2024).
72. Wu, Y.; Dai, H.N.; Wang, H.; Xiong, Z.; Guo, S. A Survey of Intelligent Network Slicing Management for Industrial IoT: Integrated Approaches for Smart Transportation, Smart Energy, and Smart Factory. *IEEE Commun. Surv. Tutor.* **2022**, *24*, 1175–1211. [CrossRef]
73. Helm. Available online: https://helm.sh/ (accessed on 15 March 2024).
74. Envoy. Envoy Proxy. Available online: https://www.envoyproxy.io (accessed on 30 April 2024).
75. Flannel. Available online: https://github.com/flannel-io (accessed on 15 March 2024).
76. WireGuard. Available online: https://www.wireguard.com/ (accessed on 29 April 2024).
77. Derek Phanekham, R.J. What a Trip. Measuring Network Latency in the Cloud. Available online: https://cloud.google.com/blog/products/networking/using-netperf-and-ping-to-measure-network-latency (accessed on 15 March 2024).
78. ESnet. Iperf3. Available online: https://github.com/esnet/iperf (accessed on 15 March 2024).