NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY POLITEHNICA BUCUREŞTI

Doctoral School of Electronics, Telecommunications and Information Technology

Decizie nr. 206 **din** 21-09-2024

PH.D. THESIS SUMMARY

Ing. Oana-Mihaela Ungureanu (Dumitru-Guzu)

METODE DE VIRTUALIZARE ŞI GESTIONARE A CONTAINERELOR ÎN MODELELE ARHITECTURALE BAZATE PE MICROSERVICII

PATTERNS OF VIRTUALIZATION AND CONTAINER MANAGEMENT IN MICROSERVICES-BASED ARCHITECTURE MODELS

THESIS COMMITTEE

Prof. Dr. Ing. Ion MARGHESCU U.N.S.T Politehnica of Bucharest	President
Prof. Dr. Ing. Călin VLĂDEANU U.N.S.T Politehnica of Bucharest	PhD Supervisor
Prof. Dr. Ing. Romulus-Mircea TEREBEŞ Technical University of Cluj-Napoca	Referee
Conf. Dr. Ing. Cristian-Iulian RÎNCU Military Technical Academy Ferdinand I Bucharest	Referee
Conf. Dr. Ing. Şerban Georgică OBREJA U.N.S.T Politehnica of Bucharest	Referee

BUCHAREST

Table of contents

List of tables

List of figures

Chapter 1

Introduction

1.1 Presentation of the field of the doctoral thesis

The present paper "Patterns of virtualization and container management in microservices-based architecture models" addresses a current topic - the integration of native cloud technologies in the current generation of 5G mobile core as well as in future generations to improve network scalability, reducing operational costs as well as bringing programmability to the network level.

1.2 Scope of the doctoral thesis

Following the ever-growing market expansion for various industries, the 5G Next Gen Core design is on the verge of evolving into a service-based architecture. Therefore, this paper presents a novel approach to deploy next-generation 5G network functions as cloud-native applications that are easily scaled on demand to align with the constraints of different sectors.

1.3 Content of the doctoral thesis

In Chapter 1 the context of cloud computing and native cloud technologies such as containers, orchestrators, etc. are presented compared to the notions of virtualization. Moreover, to meet the requirements of the current generation of 5G mobile core in terms of infrastructure that must be scaled to aggregate and process massive amounts of data coming from different industries, cloud-native technologies offer multiple benefits in terms of cost, cross-site scalability and network-level abstraction capabilities.

Chapter 2 gives an introduction to the new 5G architecture and describes the concepts of NFV and SDN as well as the MANO architecture. A taxonomy of 4G and 5G mobile core virtualization open-source projects that use virtualization is presented. In addition, an analysis of the main container orchestration solutions and an introduction to the 5G

architecture are made, highlighting the main differences between the non-standalone 5G NSA architecture and the standalone (SA) type autonomous architecture. Next, the architecture of the Kubernetes orchestrator that is the subject of these works is presented, as well as the provisioning capabilities in multi-cloud and multi-cluster environments.

Chapter 3 addresses the implementation of the 5G SA functionality in a virtualized environment running in a public cloud with programmable interfaces based on services and orchestrated by Kubernetes. Furthermore, to validate the communication between the microservices, a service-mesh solution was used. The proposed configuration aims to validate the 5G SA functionality by integrating two of the open-source simulators: Open5GS and UERANSIM used as a RAN emulator. In addition, the behavior of multiple concurrent PDU session establishment requests was analyzed.

In Chapter 4 the security risks on the main AMF and SMF interfaces by injecting traffic with a tool called 5Greplay are presented. Next, two attack scenarios are addressed for which the response throughput of the AMF function is measured.

Chapter 5 presents a declarative model based on APIs that enables multi-cloud deployment while delegating control logic to the network edge. The model based on the ClusterAPI implementation allows the use of lighter Kubernetes distributions, therefore the two K3s and Kind integrations with ClusterAPI are analyzed. Various container orchestration models designed for edge computing were also evaluated. Through a service mesh solution, called Linkerd, the latency generated by the two tools is compared in the response throughput of the Open5GS functions.

Chapter 6 addresses the cloud-native 5G multi-site capabilities by requesting network capacity on demand, as well as network programmability and mobile core configuration by leveraging the benefits of APIs. Further, two cross-site communication scenarios are presented, hereafter referred to as in-band peering and out-of-band peering, which are analyzed for the offloading of the user and control plane in the peered cluster. A comparison is also made between the MANO framework and the new proposed containerized function management model based on Kubernetes orchestration. Moreover, three network slicing scenarios are validated and the end-to-end traffic measured for simultaneous sessions of users accessing the three slices.

Chapter 7 looks at a hybrid Kubernetes-based scheduling framework model with shared states that delegates most of the tasks to distributed schedulers and has a scheduling correction function that mainly handles unscheduled and non-priority tasks. Also, the main types of schedulers existing in the literature based on Kubernetes implementation are presented, and the capabilities of the proposed new scheduler are analyzed compared to three Kubernetes scheduling frameworks.

Chapter 8 summarizes the experimental results obtained in the research together with the original contributions and traces the future development directions in order to deepen this field of cloud computing and possible applications in new mobile architectures.

Chapter 2

Technologies and systems used in the architecture of the new generation of 5G mobile core

This chapter aims to introduce the main technologies used in the new generation 5G mobile core architecture as well as the differences between the non-autonomous architecture (NSA) and the standalone architecture (SA).

2.1 Standalone Architecture

The non-standalone architecture (NSA) allows compatibility between the old 4G core and the new 5G mobile core, since the standalone architecture (SA) aims to accommodate new services coming from different sectors and industries (self-driving vehicles, precision robotics , IoT, etc.) and provide ultra-reliable low-latency communications (URLCC). Moreover, this chapter explains the functionality of the main network functions: **AMF** (Access and Mobility Management Function) ensures authentication, authorization and mobility between the user and the network, the **SMF** function (Session Management Function) ensures authentication, authorization and mobility between user and network. Function **AF** (Application Function) - provides information about resources, and function **UPF** (User Plane Function) has the role of routing and transfer of packets, maintains the transfer session **PDU** (Protocol Data Unit) between the user and the network.

The network function **AUSF** (Authentication Server Function) - provides the capability of the AMF function to authenticate the user. The function **UDM** (Unified Data Management) - is responsible for storing user subscription data. **UDR** (Unified Data Repository) - mainly stores subscription data and customer profiles, **NSSF** (Network Slice Selection Function) maintains a list of carrier-defined network instance slicing. **NEF** function (Network Exposure Function) - exposes services and resources via internal or external API to the 5G network. The **NRF** (Network Repository Function) maintains a

list of available network functions and their associated profiles. **PCF** (Control Function) - is responsible for policy enforcement and quality of service (QoS) fulfillment based on subscription.

Another component introduced in the SA architecture is the MEC (Multi-access Edge Computing) component, which is located close the base stations and aims to minimize the response time. The main idea in MEC architecture is to provide low latency, higher bandwidth as well as massive processing power at the edge of mobile networks close to terminals.

2.2 Introducing the concepts of NFV and SDN

The ETSI NFV group is in the process of standardizing the orchestration interface with NFV management and orchestration functions, also called MANO (Management and Orchestration).

The ETSI NFV Management and Orchestration (MANO) framework consists of three functional blocks: Virtualized Infrastructure Manager (VIM), NFV Orchestrator (NFVO) and VNF Manager (VNFM). The vCloud NFV platform includes an integrated VIM, which exposes northbound interfaces to VNFM and NFVO.

2.3 Brief taxonomy of existing solutions using SDN and NFV

The literature is mainly divided between different solutions to orchestrate NFVs in a cloud architecture presented in the papers [1], [2] and virtualization of functions developed for 4G and 5G mobile core. Table 2.1 summarizes the existing OSS projects mainly dedicated to the virtualization of the mobile packet core according to the 4G specifications and presents few initiatives for the development of NFs for the 5G SBA architecture.

Among the projects that support both 4G and 5G VNFs as well as CNF are OpeNess [11] and Open5GS [7] which we will consider next in our configuration. The Open Network Edge Services (OpenNESS) software is a tool to simulate the MEC architecture in order to provide CNFs. This project was developed in collaboration with Intel and runs entirely on a microservices architecture that provides APIs to the open-source community.

Open5GS [7] is an open-source project developed in C language that implements network functions according to the 3GPP Release 16 standard (ie AMF, SMF, PCF, UDM, AUSF, NRF, NSSF, UDR) and UPF) [12]. It also offers a graphical WebUI interface developed in Node.JS and React. Some of these functions have their equivalent in 4G Evolved Packet Core (EPC): AMF, SMF, PCF, UDM and AUSF. The Basic Access

Tabel 2.1 Open-source projects for the virtualization of the 4G and 5G mobile core

OSS Project	Langiage	Licence	4G	5G	CNF	Contributors
OpenAirInterface [3]	C	Apache v2.0	yes	yes	wip*	OpenAir Software Alliance EURO-COM
NextEPC [4]	C	GNU AG-PLv3	yes	wip*	wip *	NextEPC
corenet [5]	Python	GPL-2.0 License	yes	no	no	Corenet
openLTE [6]	C++	GNU AG-PLv3	yes	no	no	openLTE
open5GS [7]	C	GNU AG-PLv3	yes	yes	yes	Open5GS
OMEC [8]	C++	Apache v2.0	yes	yes	no	ONF, Intel, Deutsche Telekom, Sprint, AT&T
free5GC [9]	Go, C	Apache v2.0	no	yes	wip*	Free5C
srsLTE [10]	C++	GNU AG-PLv3	yes	no	no	srsLTE
OpenNESS [11]	Go	Apache-2.0	yes	yes	yes	Intel

* încă în lucru.

and Mobility Management Function (AMF) is responsible for user access and network authorization, interacts with other NFs (ie SMF, AUSF) to manage UE mobility and corresponds to the Mobility Management Entity (MME) in 4G.

2.4 The cloud native perspective

Nowadays, Kubernetes has become a popular container orchestrator to help mobile operators as well as customers manage 5G services in a containerized framework, regardless of their infrastructure.

A successful migration path from a legacy monolithic architecture to a cloud-native one addresses not only the hardware decoupling of VNFs, but also a modular design through APIs and a new distributed architecture based on automation and self-management. Since the aforementioned MANO framework does not address the management of CNFs, a legitimate candidate for container orchestration is Kubernetes [13] to help MNOs modernize their expensive infrastructure and turn it into a modular platform

designed for multiple integrations with Operational Support Systems (OSS) for better performance and resilience.

A service (or micro-service) is composed of one or more pods and policies. A job can run multiple operations to create one or more bridges. At the level of the control plane there is an API server responsible for updating the state of the pods, a manager controller that monitors the state of the cluster, a scheduler that dictates which nodes (virtual machines) the pods are assigned to, an agent that runs on each node and a proxy responsible for policies of traffic.

The main advantages for running next-generation mobile containerized kernel in multiple Kubernetes clusters deployed in different cloud platforms are **complete isolation** and **scalability**. For example, a URLLC application can run in "slice 1", the eMBB application in "slice 2", and the mMTC application in "slice 3". In order for the software to follow development (ie, DevOps) compliance rules, it is possible to separate the development and staging environments by running these slices in different isolated production clusters.

The concepts of **multi-cloud** and **multi-region** are also explained. Multi-region hosting covers availability and failover issues, as well as latency or geographic location in case of data protection and GDPR compliance, while multi-cloud hosting ensures disaster recovery capabilities and avoids lock-in with a particular cloud provider .

Chapter 3

A cloud-native approach to design network functions in future generations of 5G mobile cores

In this chapter, a cloud-native architecture is proposed along with implementing the Open5GS solution in containers running in a public cloud by using a package manager dedicated to the Kubernetes orchestrator, the results being published in the work [14].

Migrating to cloud-native network functions (CNF) provides a solution for orchestrating VNFs because microservices can run in containers and accommodate requirements such as autoscaling, resiliency, and network monitoring.

3.1 Description of the proposed solution

The proposed configuration uses the containerized version of the open-source tool Open5GS [7], version 15 which introduces the concept of "standalone" for the 5G core consisting of a fully virtualized core network to leverage the implementation and management of network functions virtualized (NFV) along with network segregation and segmentation capabilities. Mobile terminals and base stations in 5G are simulated in the proposed configuration using UERANSIM which is an open-source simulator for UE and 5G gNB deployment. UERANSIM is compatible with Open5GS and has two main components: the gNodeB (gNB) which connects to the AMF and handles data traffic in the Internet through the simulated radio link, and the users (UEs) represent mobile phone subscribers that generate Internet traffic. Both Open5GS and UERANSIM simulators are hosted in a public cloud.

For both UE and gNB configurations, the number of network segments called network slices are defined. The purpose of using Network Slices in 5G is to provide isolation and QoS for different end-to-end network requirements sharing the same physical resources. For example, two network segments can share the same SMF.

Furthermore, the functionality is validated on three main interfaces: radio interface - between UE and RAN, control interface - between RAN and AMF and user interface - between RAN and UPF. Due to the limitation imposed by the UERANSIM simulator to only allow the configuration of a maximum of 15 sessions simultaneously, the UE script was run three times to test the connection of 45 users, respectively the establishment of 45 PDU sessions within an hour.

A "service-mesh" solution is also implemented that shows us inter-service communication in real-time in terms of traffic distribution, request duration and throughput for all network services of the 5G mobile core. To visualize the service mesh graph with the traffic distribution and percentages, the Istio tool [15] was used, on top of which a plugin called "kiali" [16] was added.

The experimental analysis shows that the throughput of the request when querying the SMF is higher than the other service requests. The AMF is responsible for Network Slice selection and determining the most appropriate SMF function to handle the connection request by querying the NRF (see Figure 3.2).

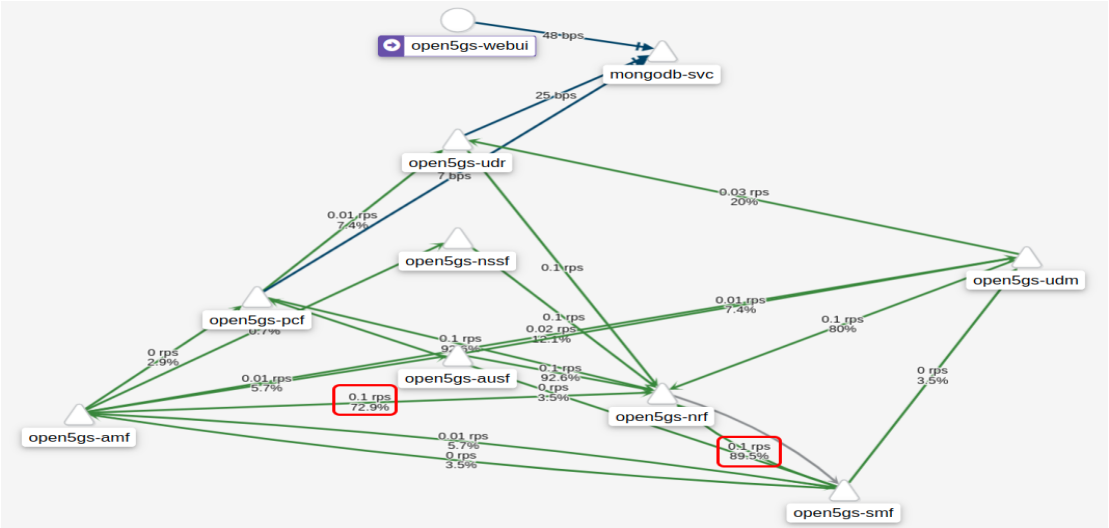
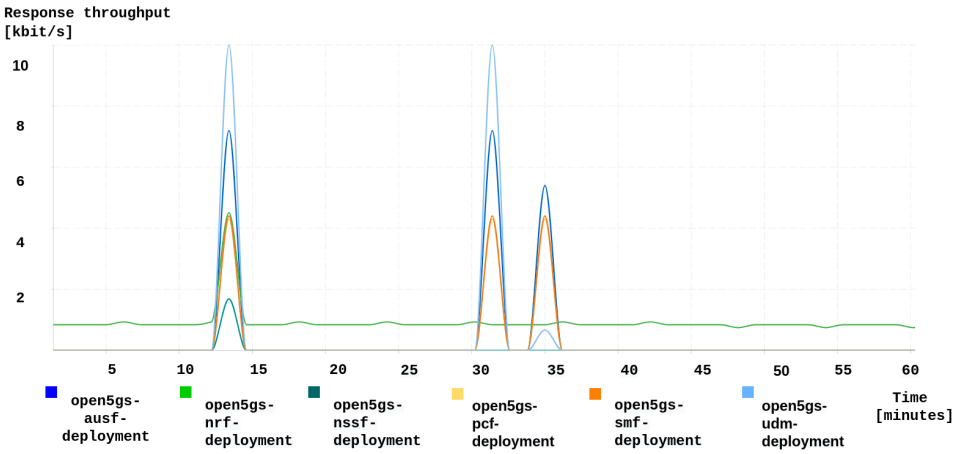
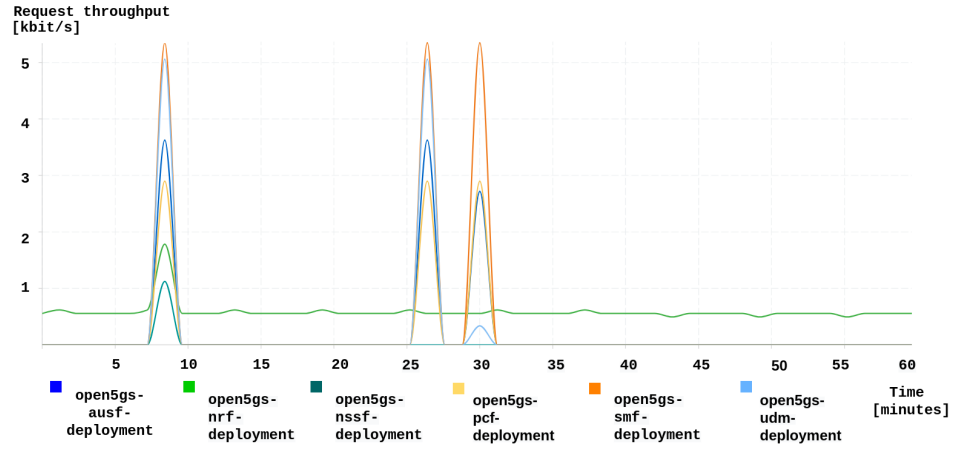


Fig. 3.1 The "service mesh" graph with the distribution and percentages of traffic



(a) AMF response throughput measured at source



(b) AMF request throughput measured at destination

Fig. 3.2 AMF response and request throughput on the outbound interface

Chapter 4

Analysis of potential vulnerabilities in the implementation of the new generation of mobile core 5G

This chapter presents two attack scenarios on the main control plane and user interfaces. The results were published in the work [17].

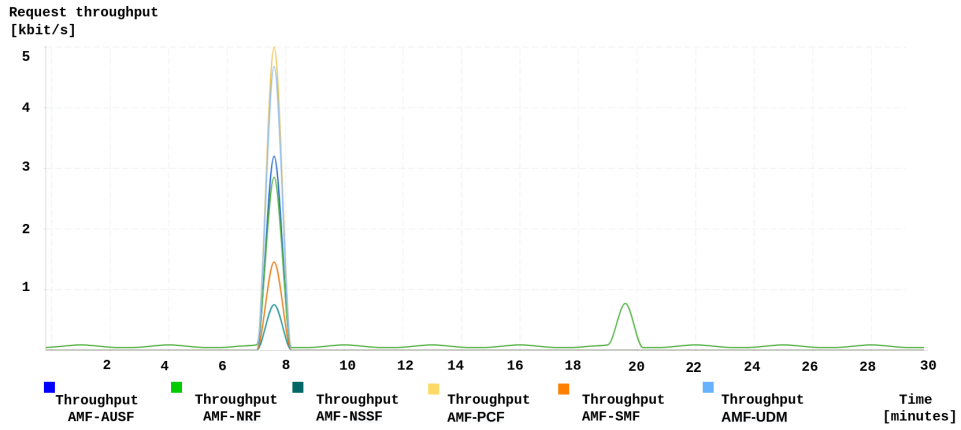
The interfaces connecting the 5G RAN with AMF and UPF (User Plane Function) respectively are targets for attackers because they carry sensitive user plane data between the access network and the core network.

4.1 The objective of the proposed solution

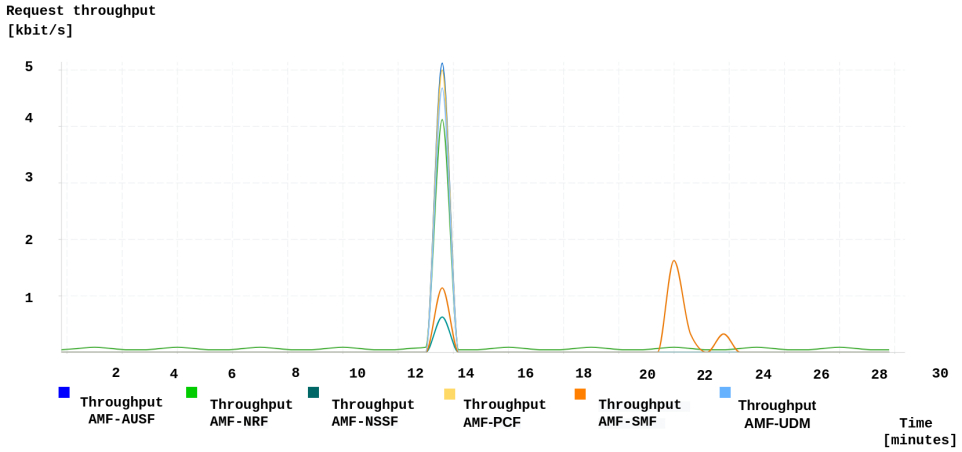
The open-source tool 5Greplay was used to simulate the two attack scenarios, which is a solution that can be configured to replay traffic and forward modified network packets to the network interface card (NIC) on predefined ports. The traffic generated by the tool conforms to the standardization protocols in 3GPP Release 16 and is compatible with existing open-source 5G frameworks such as Open5GS.

In the first scenario after the 15 PDU sessions have been successfully established, i.e. the 15 user connections, the target protocol is specified, i.e. SCTP and the address of the host where AMF is running. It was found that the TUN interface (the user plane between the UE and the UPF) is not affected, the UE's connection to the Internet remains available. However, a warning "Unhandled SCTP connection received" is displayed and when starting a new PDU session, the user plan is no longer established. Therefore, if the AMF does not implement a protection mechanism against this type of attack, the network will not drop the packet. Additionally, this vulnerability could be exploited by a malicious actor who can impersonate the user.

In the second scenario, an attack on the N3 interface between gNB and UPF is simulated. For this simulation the 5Greplay script was run where the target host which is



(a) AMF response throughput following the 5g Replay attack on the N1/N2 interfaces



(b) AMF response throughput following the 5g Replay attack on the N3 interfaces

Fig. 4.1 AMF response throughput in the both attack scenarios with 5Greplay

UERANSIM and the target port for UDP was changed. PCAP traffic from the gNB is injected into the 5G core, which interrupts the PDU session. Next, the gNB script is run to restore data plan connectivity.

Monitoring within the Kubernetes cluster was also achieved by implementing a service mesh solution from Istio that serves as a proxy for each of the core 5G functions.

It is observed that when the 5Greplay attack is injected, the user's plane is interrupted, so we see the AMF to SMF response which is almost half of the initial values (see Figure 4.1 -b)). To request a new session, the UE and gNB use the NGAP protocol to carry NAS messages. The AMF function receives these requests and is responsible for handling mobility management while forwarding network session requirements to the SMF. In this manner, the AMF determines which SMF is best suited to handle the connection request by querying the NRF.

Chapter 5

Cloud-Edge Communication: A Declarative API-Based Orchestration Model for the New Generation 5G Core

This chapter presents the benefits of deploying services and network functions in different clusters hosted in multiple cloud providers to ensure full isolation between tenants/tenants. Different container orchestration models designed for edge computing were also evaluated.

One of the main objectives was the comparison of two deployment models from the perspective of latency for 5G functions run in containers and memory resource consumption for the control plane. The results were published in the paper [18].

5.1 Description of cloud-edge context

The concept of network segmentation or slicing has recently introduced a new component in the 5G architecture called Multi-Access Edge Computing (MEC) and can be implemented at the edge network closer to the end users.

5.2 Container orchestrators for edge computing

In this section, three orchestration models are presented: **kind** [19], **K3s** [20] and **KubeEdge** [21]. The open-source **Kind** tool was developed by the Kubernetes Special Interest Groups (SIGs) and supports multi-node and multi-cluster deployments, plus it can be easily integrated with other Kubernetes APIs , for example ClusterAPI (CAPI) [22].

Kubedge is another open-source project developed under CNCF (Cloud Native Computing Foundation) built on Kubernetes with the aim of orchestrating IoT Appli-

cations. The KubeEdge architecture has two main components: cloud and edge with corresponding "hub" and "edge" modules.

Another tool dedicated to running Kubernetes clusters at the network edge is **K3s** which comes with a lightweight version of Kubernetes. This project was developed by Rancher Labs and is mainly suitable for IoT use cases and cutting edge technologies. The deployment is split into K3s servers and multiple agents that can run at the network edge and, unlike KubeEdge, do not require cloud-side communication.

5.3 The ClusterAPI model

The CAPI framework consists of three concepts: the management cluster that stores the information from all cloud providers, the bootstrap providers used to install the nodes dedicated to the Kubernetes control plane, and the "workload" or "tenant" clusters created as cluster object resources associated with [23] cloud providers.

The core of the mobile packet network can run in different cloud tenants but in a different framework called a virtual private cloud (VPC) corresponding to a "slice" of the network. The primary role of a VPC is to provide network segmentation and security throughout the policy configuration.

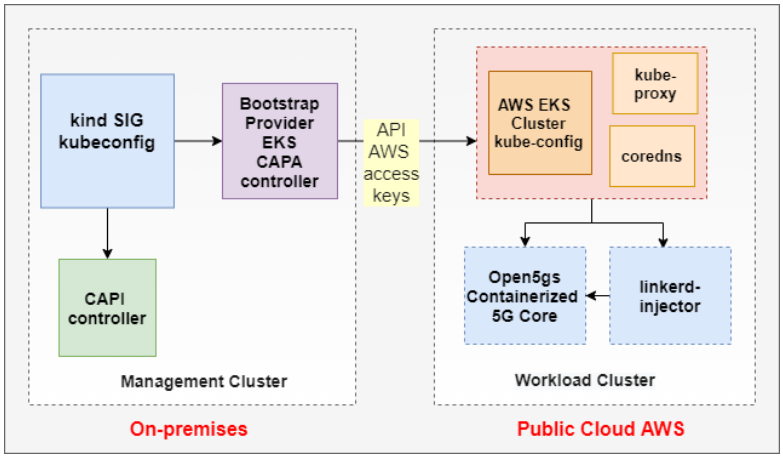
Each cloud and on-premises environment has its own dedicated ClusterAPI provider that provides cluster provisioning, the CAPI provider for AWS (CAPA), while CAPI for vSphere (CAPV) is dedicated to the VMware provider. The CAPI provider for AWS (CAPA) covers both Elastic Compute (EC2) and Elastic Kubernetes Service (EKS) deployments, i.e. CAPA EKS. A user can define different types of resources in a declarative manner using resource templates, Custom Resource Definition (CRD) that differ slightly depending on the cloud provider's infrastructure.

5.4 Experimental Setup - Deploying 5G NextGen Core in Kubernetes Clusters

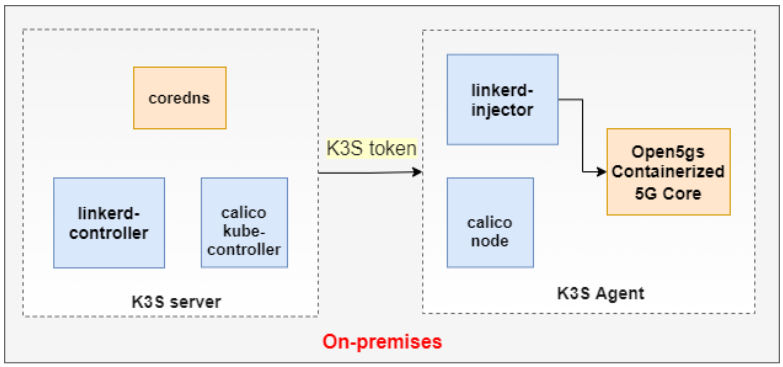
In the first test scenario there are two Kubernetes clusters with different configurations: the *SIG kubeconfig* generated with Kind corresponding to the management cluster and a second configuration of *kubeconfig* for the EKS cluster where the workload cluster is running.

In the second test setup (see Figure 5.1 (b)) runs in a local VMware (on-premises) which consists of a K3s cluster composed of a K3s server that acts as a master node and an agent K3s as a worker node.

Both scenarios share a component called linkerd-injector. This is part of Linkerd [24], a popular service mesh tool deployed on Kubernetes cluster that enables and monitors communication between services and routes traffic and API calls between



(a) Open5GS deployed in the EKS cluster using kind and CAPI

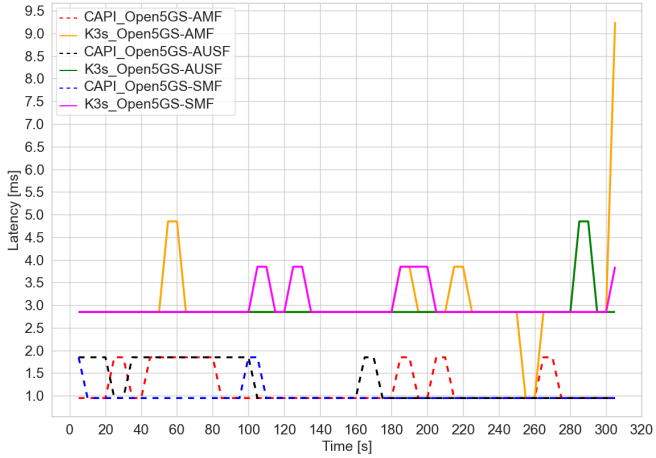


(b) Open5GS deployed in the K3S cluster

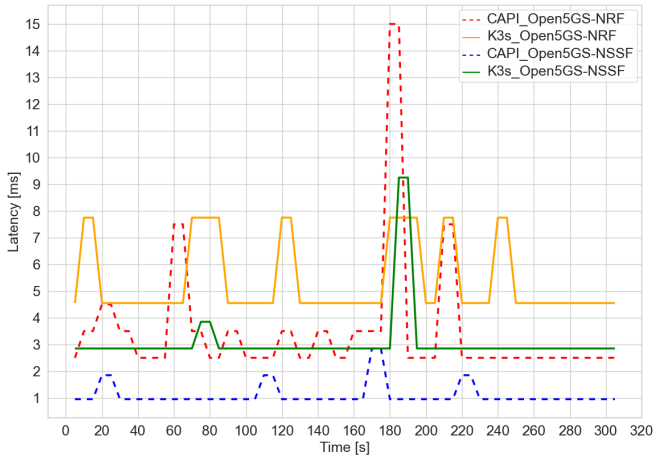
Fig. 5.1 The setup used for 5G Core SBA deployed with Kubernetes orchestrator at the edge of the network (Figure 5.4 from the thesis)

services/endpoints. The "service mesh" layer implemented on top of the Kubernetes infrastructure role is to provide the abstraction of the application logic and mainly provides service monitoring and observability.

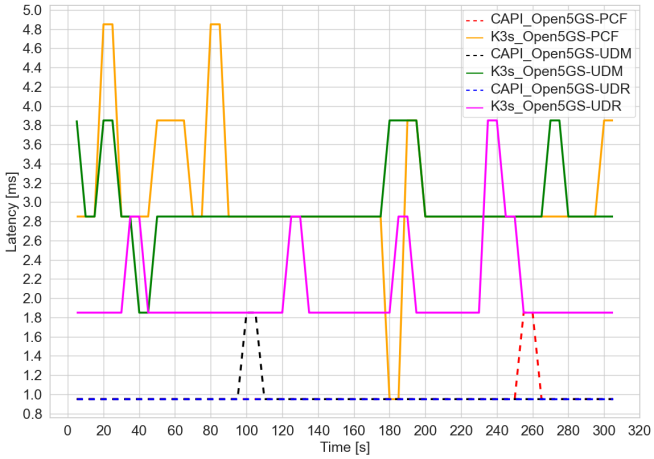
The main objective of this research was to evaluate the performance obtained from communication between Open5GS containerized application processes using application programming interfaces (APIs) for both proposed implementation solutions, CAPI and K3s. The obtained results show better response time values for the CAPI configuration compared to K3s (Figure 5.2) which validates our hypothesis regarding the benefit of running Open5GS in a public cloud because when we used the resources on demand we not only increased the scalability, but also led to faster communication between services.



(a) Authentication, mobility and session management functions - AMF/AUSF



(b) Network Resource Management Functions - NRF/NSSF



(c) Policy Control and Data Management Functions - PCF/UDM/UDR

Fig. 5.2 Latency comparison of Open5GS implementation using CAPI vs K3s (Figure 5.5 from the thesis)

Chapter 6

An innovative approach to scaling 5G Core network slicing deployed in a cloud-native framework across multiple sites

In this chapter, a new approach based on the "liquid computing" paradigm is presented, which has the advantage of adapting to the changing environment. In this manner, the 5G mobile core can be managed as a single cluster entity running in a public cloud following cloud-native models for declarative configuration based on Kubernetes APIs and on-demand resource allocation. Moreover, two scenarios of offloading the Open5GS user functions and the control plane are analyzed.

Three end-to-end network use cases are also validated, showcasing full 5G core automation and leveraging the capabilities of Kubernetes multi-cluster deployments and inter-service monitoring through the applied service network solution.

6.1 Comparison between ETSI MANO framework and Kubernetes orchestrator

Figure 6.1 presents the proposed solution for the containerization of the 5G mobile core compared to the ETSI MANO framework where the network functions are virtualized.

The ETSI MANO framework differs from the Kubernetes Model because the VNFM (VNF Manager) has a detailed view of the associated deployed VNFs and a northward exposure to the NFVO. In Kubernetes, this information is not exported to the upper layers, because Kubernetes provides a better way to control it by defining the intent through object definitions (such as labels, tags, selectors, etc.). In Kubernetes, the concept of Pods exists to define where application containerization resides.

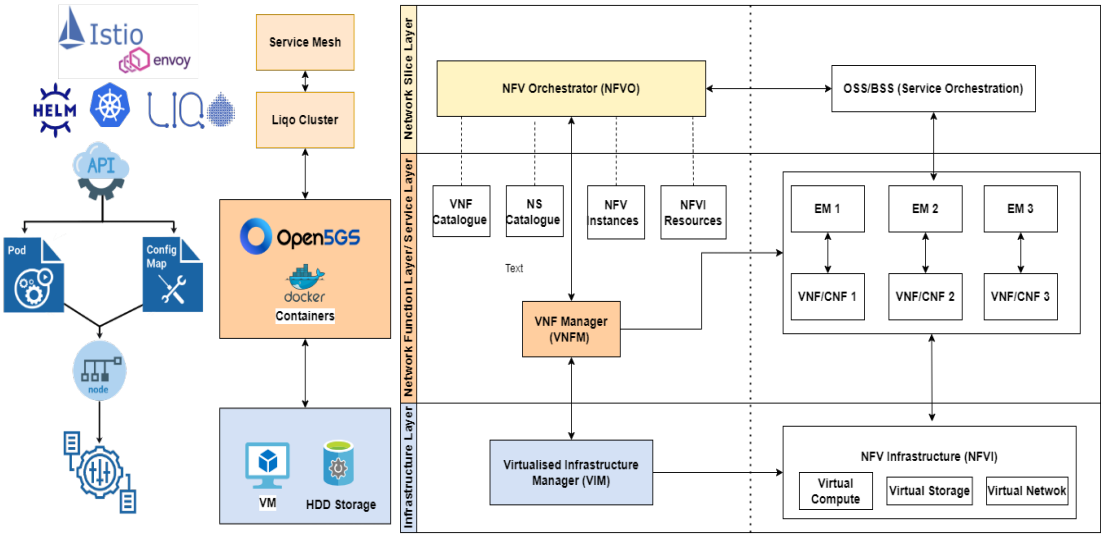


Fig. 6.1 Proposed framework for network slicing in cloud-native 5G mobile core

6.2 Horizontal scaling vs. vertical in the cloud

In Kubernetes, horizontal scaling means expanding the number of pods in response to increased workload. The HorizontalPodAutoscaler (HPA) feature automatically updates a workload resource (such as a Deployment or StatefulSet) in order to automatically scale the workload to match [25] capacity demands.

In terms of vertical scaling, the Kubernetes scheduler points more resources (for example, memory or CPU) to already running pods as part of the workload. The key role of the Vertical Pod Autoscaler (VPA) [26] is to automatically set container requests based on pod usage through an optimal scheduling mechanism to allocate the compute resources required to run each pod on demand.

6.3 Scaling in multi-cluster and multi-cloud Kubernetes deployments

The open-source tool Liqo [27] uses the virtual node abstraction as an extension of the Virtual Kubelet [28] project. In Kubernetes, the kubelet is the master node agent, responsible for registering the node with the control plane and handling bridge scheduling. Virtual Kubelet replaces a traditional kubelet for a physical node through standard Kubernetes APIs for both local and remote [29] clusters.

It also automatically pushes the negotiated configuration (Services, ConfigMaps, Secrets, Storage) into the capacity needed to properly execute offloaded/offloaded workloads, through a mechanism called resource reflection. All available resources that exist in a given namespace and are selected for offload are automatically propagated to the corresponding replicated namespaces created in the selected cluster.

6.4 Description of the test setup and the two peering scenarios

Liqo extends Kubernetes namespaces across cluster boundaries by creating replicated namespaces in the selected subset of remote clusters whenever a namespace is selected for offloading. The namespaces host the actual pods downloaded to the corresponding cluster as well as the additional resources propagated by the resource mirroring process.

In the first "out-of-band peering" scenario, we initiate peering between two K3S clusters that are isolated from each other because they are hosted in different tenants in different hyperscaler data centers from the public cloud. The Liqo Network Manager is the control plane of the Liqo network.

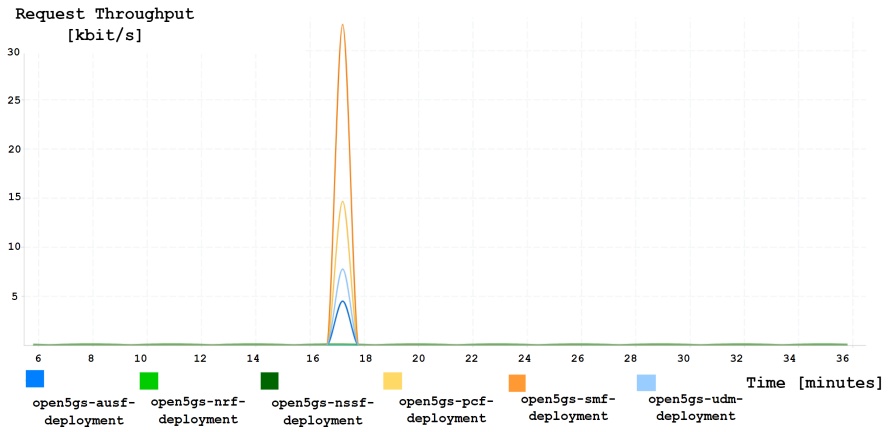


Fig. 6.2 AMF Request throughput measured at the source for the control plane downloaded to the remote cluster (Figure 6.13 in Thesis)

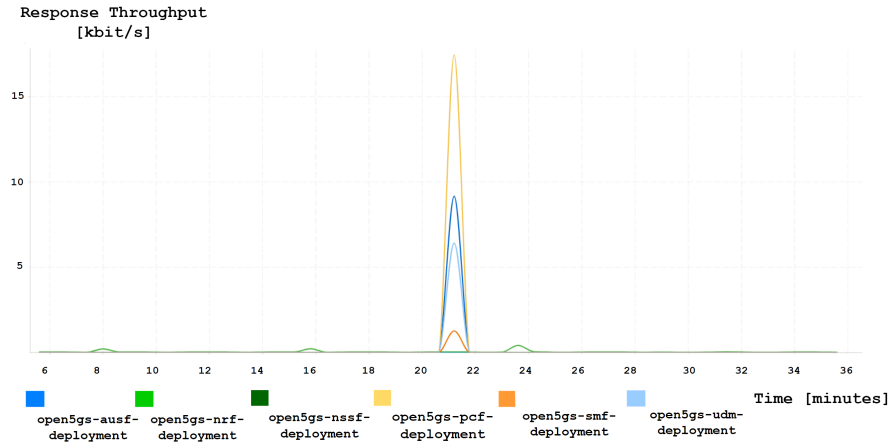


Fig. 6.3 AMF response throughput measured at the destination for the control plane downloaded to the remote cluster (Figure 6.14 in the thesis)

It is observed that we measure a higher throughput for the AMF request throughput in the offloaded user plane compared to the Open5GS control plane shown in Figure

6.2, while the AMF response throughput measured at the destination for the control plane offloaded to the remote cluster is approximately half of the AMF query thoughput measured at the source for the control plane offloaded to the remote cluster (Figure 6.3).

The reason is mainly due to the session establishment PDU traffic generated along with the creation of interfaces in the user plane that is done between tenants for *out-of-band* peering.

On the other hand, the **request throughput** for the SMF service on the egress destination for the offloaded Open5GS control plane was observed to be three times higher than for the offloaded user plane. On the other hand, for the **response throughput** at the source in the case of a user plan downloaded to the remote cluster, we record half of the value of the **request throughput** measured at the destination for the same scenario (see Figure 6.4) and (Figure 6.5) .

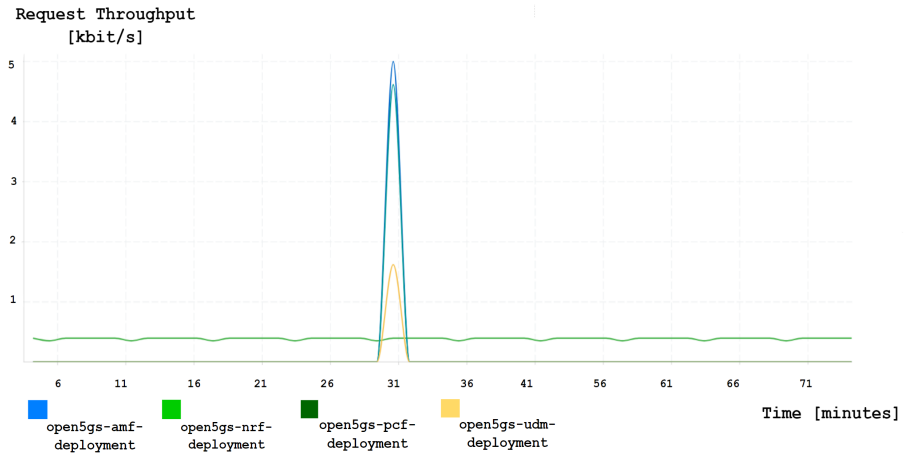


Fig. 6.4 SMF response throughput measured at the destination for the user plane downloaded to the remote cluster (Figure 6.15 from the thesis)

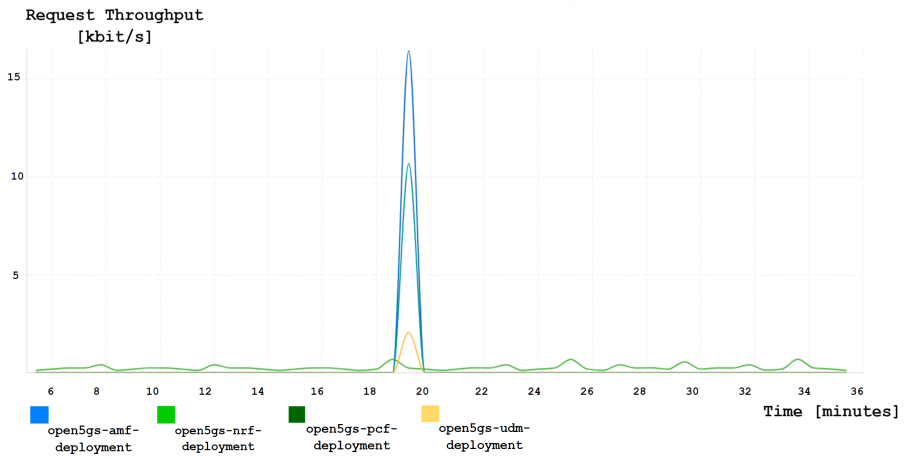


Fig. 6.5 SMF request throughput measured at the destination for the control plane downloaded to the remote cluster (Figure 6.16 in the thesis)

In the second "inband peering" scenario, we analyze the Open5GS control plane services downloaded to the remote cluster. The remote cluster resides in a different tenant and region.

The particularity of in-band peering for the two clusters is that all traffic in the control plane (including authentication services) goes through the VPN tunnel. In this scenario, the Kubernetes API service is not exposed outside the cluster. In-band peering involves several steps to authenticate and establish the VPN tunnel using the WireGuard [30] client.

In the case of the in-band peering, the latency values for the three UPF functions are similar because the user plane traffic between the two clusters communicates through the VPN tunnel (see Figure 6.6).

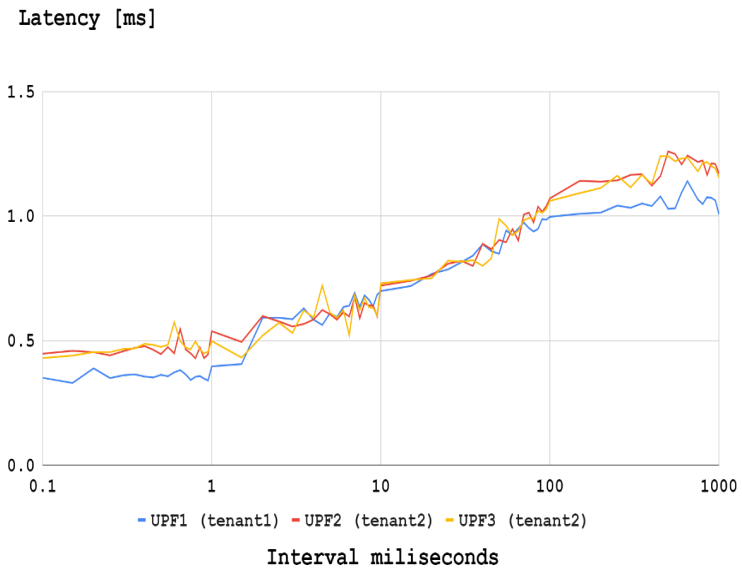


Fig. 6.6 Latency in the user plane for the in-band peering scenario (Figure 6.19 in the thesis)

Chapter 7

Optimizing Kubernetes Clusters Using Hybrid Shared State Scheduling

This chapter presents a new Kubernetes-based scheduler model that uses a hybrid scheduling method with a shared state correction feature. Also, the comparison between this type of planner and the main types of planners existing in the specialized literature are presented.

7.1 Related works and a brief taxonomy of existing planning mechanisms

The Kubernetes orchestrator is a widely used **centralized scheduling mechanism** and implements a shared persistent storage mechanism exposed through APIs. Building management APIs around containers is considered more developer-oriented and shifts the primary concern of distributing data from the machine layer to the application layer. Table 7.1 provides a classification of the various scheduler models existing in the current literature.

7.2 Issues Identified in the Kubernetes Scheduler

The main issues that can cause the need to move already running pods to other nodes for various reasons have been identified: Nodes under or overused, tags added or removed from nodes, or pod/node affinity requirements are no longer met. Nodes may also display errors and their bridges have been moved to other nodes or when new nodes are added to clusters.

Following the proposed test scenario, it was found that in a long-running Kubernetes cluster, after a node fails, the workload is not evenly balanced between nodes.

Tabel 7.1 Taxonomy of the different planning frameworks existing in the current literature

Scheduler	Centralized	Two-level	Shared-state	Hibrid	Distributed
Borg	✓				
Omega			✓		
Apache Hadoop YARN	✓				
Apollo			✓		
Kubernetes	✓				
Hadoop-on-Demand		✓			
Apache Mesos		✓			
Docker Swarm	✓				
Firmament-Poseidon	✓				
Nomad			✓		
Sparrow					✓
Tarcil			✓	✓	

7.3 Description of the proposed scheduling method: hybrid-shared state scheduler

The proposed method uses a similar shared-state hybrid architecture, where the interference of collocated tasks is handled by a scheduling correction function. This scheduling correction function compares the waiting time burden reported by the resource nodes with its estimate, calculated based on the number of concurrent transactions performed by each of the schedulers over time.

The architecture presented in Figure 7.1 consists of several *Scheduling Agents* (SA) configured as slaves, a master that has an overview of the cluster state, *Master-State Agent* (MSA) and a *Scheduling Correction* (SC) function that estimates the waiting time for remaining unscheduled and unprioritized jobs. *Resource Node* (RN) stores information regarding resource usage (eg memory usage, CPU, throughput, and execution time) and shares this information with the SA. RNs manage the lifetime of each task, while SAs run the life cycle of a job. Each SA processes this information together with job priorities and *Node Feasibility* (NF), thus, after making a decision to place a job, it sends the updated status to the MSA which further sends it to the SC and all SAs.

7.4 Correction Function (SC) Logic

We consider the total execution time of a job queued in the Correction Function (SC) queue $S = [S_1, S_2, \ldots, S_n]$. We define the total execution time of a task (S_i) as the sum of the estimated waiting time of the task (T_{w_i}) plus the estimated execution time of the task (R_i), where $i \in \{1, ..., n\}$. The interference time for a received job is $I = [I_1, I_2, \ldots, I_n]$.

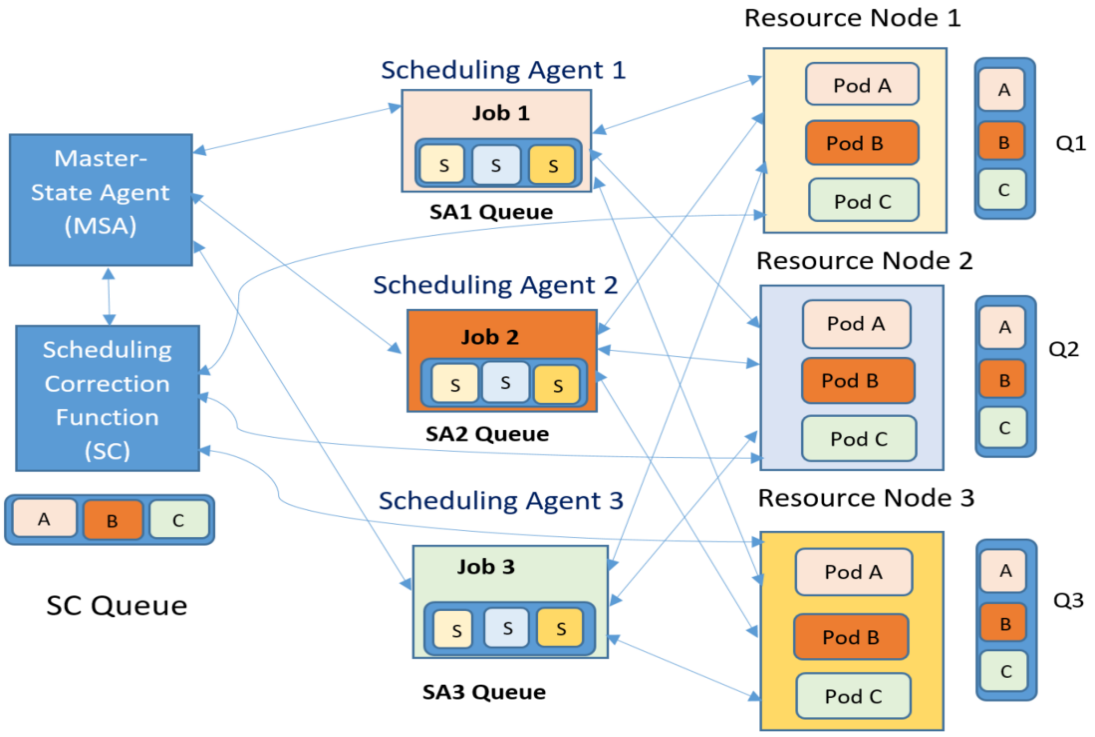


Fig. 7.1 Proposed shared-state hybrid scheduling architecture

To calculate the total execution time for an unscheduled task, S'_i, the SC function assigns a weight to the time derived from the interference latency (see equation 7.1)

$$w_i = 1 - \frac{T_a}{T_{wi}} \tag{7.1}$$

Where T_a represents the average estimated task runtime (T_a) normalized to the average total execution time for an unscheduled job ($\overline{S'_n} = \frac{Ta}{T_{exec}}$).

Afterwards, the prioritized SC queue contains the new values for the total execution time of an unscheduled job $S' = [S'_1, S'_2, \ldots, S'_n]$ to which the minimum collocation interference time is added (see equation 7.2). Prioritization is then done based on the task with the highest total execution time so that it can be executed on the chosen RN.

$$S'_i = S_i + w_i \times I_i \tag{7.2}$$

In this manner we determine the collocation cases: if P_C =1, the resource allocation is optimal and the result is sent to the SA, if $P_c < \overline{S'_n}$ we have a suboptimal schedule, otherwise the SC function determines the entire queue rescheduling (Equation 7.3):

$$P_C = \begin{cases} 1 - F_n, & \text{if } F_n \geq \overline{S'_n}. \\ F_n & F_n < \overline{S'_n}. \end{cases} \tag{7.3}$$

Tabel 7.2 Comparison of features between Kubernetes integrated schedulers

Caracteristică	Planificator Default	Descheduler	Poseidon-Firmament	Planificator propus
`Node Affinity/Anti-Affinity`	yes	yes	yes	yes
`Inter-pod Affinity /Anti-Affinity`	yes	partially	yes	yes
`Taints/Tolerations`	yes	no	yes	yes
`Basic Scheduling/Optimal Scheduling`	no	yes	partially	yes
`Colocation interference avoidance`	no	no	partially	yes
`High-availability`	yes	no	no	yes
`Priority Preemption`	yes	no	partially	yes
`Inherent Rescheduling`	no	no	yes	yes

7.5 Analysis of the hybrid-shared state scheduler method compared to other existing scheduling solutions

Table 7.2 shows a comparison of common features as well as differences between the three scheduling candidates integrated with the Kubernetes scheduling framework (**Descheduler, Firmament-Poseidon** and the proposed solution, the scheduler **Hybrid-state**).

The *Node Affinity/Anti-Affinity* rule defines how nodes are selected by the scheduler using custom labels on nodes and selectors on bridges, while the *Inter-pod Affinity /Anti-Affinity* rule represents a constrain against bridge labels rather than node labels.

Taints and Tolerations are the opposite of node affinity, as they allow a node to reject a set of bridges. *Colocation interference avoidance* occurs between bridges competing for the same node resources.

Basic Scheduling/Optimal Scheduling - To make scheduling decisions according to resource requirements, the scheduler uses predicates and priorities.

High availability the scenario of high service availability where multiple pod replicas can be scheduled on the same node without considering service availability shows that the default Kubernetes scheduler does not have a mechanism adapted to cluster dynamics.

Priority Preemption - If a bridge cannot be scheduled, the scheduler tries to remove lower-priority bridges to reschedule available bridges.

Inherent Rescheduling – When a node is terminated, pods that were previously running on that node will be rescheduled on available nodes.

Chapter 8

Conclusions

8.1 Obtained results

In Chapter 1, the terminology and technologies of the 5G spectrum are introduced, as well as the presentation of the cloud ecosystem with related technologies, and Chapter 2 presents the new 5G architecture and the description of the concepts of NFV and SDN, as well as the MANO architecture.

In Chapter 3 the 5G SA functionality was tested in a public cloud running Open5GS adapted to a Kubernetes orchestrated environment. In order to validate the communication between the microservices, a service mesh solution from Istio was used that monitors the throughput in case of session stabilization of 45 of registered users. Further, in Chapter 4 two attack scenarios on the main 5G SA interfaces were addressed to analyze the behavior in the control and user planes.

Chapter 5 presented a declarative model based on APIs that allows multi-cloud implementation, the model based on the ClusterAPI implementation. Furthermore, the two K3s and Kind integrations with ClusterAPI are analyzed. Through a new service mesh solution called Linkerd, the latency generated by the two tools is compared in the response throughput of the Open5GS functions.

In Chapter 6 a new scaling model of network slices based on Liqo that respects the "liquid computing" paradigm was addressed. In addition, this tool meets the scaling criteria in multi-cluster Kubernetes deployments. Also, three network segmentation scenarios were validated and end-to-end traffic measured for concurrent user sessions.

Chapter 7 aimed to propose a new scheduler model based on Kubernetes that uses a hybrid scheduling method with a shared state correction function, being compared with the main types of schedulers based on the Kubernetes implementation.

8.2 Original contributions

This section presents the main original contributions that have achieved the objective of this paper together with the published articles listed in Section 8.3.

(1) Proposing a scheduler model with shared states and a correction function for the efficient use of resources in a Kuberenetes cluster. Analysis of functionality compared to other types of planners.

(2) Orchestrating the 5G mobile core running in a multi-cloud infrastructure and bringing resource allocation logic to the edge of the network through a declarative cloud-to-edge communication model.

(3) Implementing a 5G SBA architecture running in microservices in a public cloud as well as validating the user connection by using an emulator for the access radio network.

(4) Analysis of the main vulnerabilities that can be exploited in a 5G architecture running in the cloud and validation of some attack scenarios on the main control plane and user interfaces.

(5) Multi-cloud and multi-region scaling of the new generation of 5G core through a new model called "liquid computing". Analysis of communication scenarios between different Kubernetes clusters and end-to-end connection testing of users accessing multiple slices concurrently.

8.3 List of original publications

1. O.-M. Ungureanu, C. Vlădeanu, and R. Kooij, “Kubernetes cluster optimization using hybrid shared-state scheduling framework”, pp. 1–12,
   doi 10.1145/3341325.3341992,
   Proceedings of the 3rd International Conference on Future Networks and Distributed Systems, **ICFNDS, Paris, France, Best Paper Award, July 2019**

2. O.-M. Ungureanu, C. Vlădeanu, and R. Kooij, “Collaborative cloud - edge: A declarative API orchestration model for the NextGen 5G core,” pp. 124– 133,
   doi 10.1109/SOSE52839.2021.00019,
   IEEE International Conference on Service-Oriented System Engineering, **SOSE, Oxford, United Kingdom, August 2021**

3. O.-M. Ungureanu and C. Vlădeanu, “Leveraging the cloud-native approach for the design of 5G NextGen core functions,” IEEE Press, p. 1–7,
   doi 10.1109/COMM54429.2022.9817268,
   14th International Conference on Communications, **COMM, Bucharest, Romania, June 2022**

4. O.-M. Dumitru-Guzu and C. Vlădeanu, “Analysis of potential threats in NextGen 5G core,”, pp. 1–4,
   doi 10.1109/ISETC56213.2022.10010163,
   International Symposium on Electronics and Telecommunications, **ISETC, Timișoara, Romania, November 2022**

5. O.-M. Dumitru-Guzu and C. Vlădeanu and R. Kooij, “A novel framework for cross-cluster scaling in cloud-native 5G NextGenCore”, pp. 1-22
   doi.org/10.3390/fi16090325,
   MDPI Journal, Future Internet, **MDPI, Switzerland, selected as cover photo for Vol. 16, Iss. 9, September 2024**

8.4 Perspectives for further developments

A further development may be to replicate the implementation of multi-cloud scenarios for different public cloud providers addressing the proposed test scenarios in a different configuration using physical devices for the radio access solution. Another direction could be the integration of the MANO framework or other open-source virtualization solutions such as Open5GS in an environment orchestrated by Kubernetes.

As a further perspective, the scheduler proposed in Chapter 7 could be integrated with the tested and validated solution in Chapter 6 to improve how services are deployed on multiple Kubernetes clusters.

Moreover, we could also address and test new network slicing scenarios such as streaming, IoT and vehicle communication, or address new use cases for 6G to test network performance and validate different QoS requirements.

References

[1] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead," *Computer Networks*, vol. 182, p. 107516, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620311786

[2] S. Imadali and A. Bousselmi, "Cloud Native 5G Virtual Network Functions: Design Principles and Use Cases," 11 2018, pp. 91–96.

[3] OpenAir Software Alliance EUROCOMcorene t. OpenAirInterface, Accessed: Mar. 2021. [Online]. Available: https://openairinterface.org

[4] NextEPC. NextEPC, Accessed: Mar. 2021. [Online]. Available: https://nextepc.org/

[5] Cellular Network Infrastructure. Corenet, Accessed: Mar. 2021. [Online]. Available: https://github.com/mitshell/corenet

[6] openLTE. openLTE, Accessed: Mar. 2021. [Online]. Available: http://openlte.sourceforge.net

[7] Open5GS. Open source project of 5GC and EPC (Open5GS) (Release-16), Accessed: Mar. 2021. [Online]. Available: https://open5gs.org

[8] Omec. "Lightweight Kubernetes", Accessed: Feb. 2024. [Online]. Available: https://opennetworking.org/omec/

[9] free5GC. free5GC, Accessed: Mar. 2021. [Online]. Available: https://www.free5gc.org

[10] srsLTE. srsLTE, Accessed: Mar. 2021. [Online]. Available: https://www.srslte.com

[11] Intel. Open Network Edge Services Software (OpenNESS), Accessed: Mar. 2021. [Online]. Available: https://www.openness.org

[12] Open5Gs. "Open source project of 5GC and EPC (Release 16)", Accessed: Feb. 2022. [Online]. Available: https://open5gs.org,Accessed:Feb.2022

[13] Kubernetes. Kubernetes K8S, Accessed: Mar. 2024. [Online]. Available: https://kubernetes.io

[14] O.-M. Ungureanu and C. Vlădeanu, "Leveraging the cloud-native approach for the design of 5g nextgen core functions," in *2022 14th International Conference on Communications (COMM)*. IEEE Press, 2022, p. 1–7. [Online]. Available: https://doi.org/10.1109/COMM54429.2022.9817268

[15] Rancher. "The Istio service mesh", Accessed: Feb. 2024. [Online]. Available: https://istio.io

[16] Kiali. "Kiali", Accessed: Feb. 2024. [Online]. Available: https://kiali.io

[17] O.-M. Dumitru-Guzu and C. Vlădeanu, “Analysis of potential threats in nextgen 5g core,” in *2022 International Symposium on Electronics and Telecommunications (ISETC)*, 2022, pp. 1–4.

[18] O.-M. Ungureanu, C. Vladeanu, and R. Kooij, “Collaborative cloud - edge: A declarative api orchestration model for the nextgen 5g core,” 08 2021, pp. 124–133.

[19] Kubernetes SIGs. Kind, Accessed: Mar. 2024. [Online]. Available: https://kind.sigs.k8s.io

[20] k3s. "Lightweight Kubernetes", Accessed: Feb. 2024. [Online]. Available: https://k3s.io

[21] CNCF. KubeEdge, Accessed: Mar. 2024. [Online]. Available: https://kubeedge.io/en

[22] Kubernetes SIGs. ClusterAPI, Accessed: Mar. 2024. [Online]. Available: https://cluster-api.sigs.k8s.io

[23] G. Chandra, “Multi-Cloud and Multi-Cluster Declarative Kubernetes Cluster Creation and Management with Cluster API (CAPI — v1alpha3),” ITNEXT, Jul 24 2020. [Online]. [Online]. Available: https://itnext.io/

[24] Linkerd. Linkerd, Accessed: Mar. 2023. [Online]. Available: https://linkerd.io

[25] Kubernetes. "Horizontal Pod Autoscaling", accessed: Sep. 2023. [Online]. Available: https://docs.sd-core.opennetworking.org

[26] kubernetes. Vertical Pod Autoscaler, accessed: Sep. 2023. [Online]. Available: https://github.com/kubernetes/autoscaler/blob/master/vertical-pod-autoscaler/README.md

[27] Liqo. "Enable Dynamic and seamless Kubernetes Multi-cluster Topologies", Accessed: Sep. 2023. [Online]. Available: https://liqo.io/

[28] Virtual Kubelet. "Virtual Kubelet", Accessed: Sep. 2023. [Online]. Available: https://virtual-kubelet.io/docs/

[29] Kubernetes. "Kubernetes components", accessed: Sep. 2023. [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/

[30] WireGuard. "WireGuard", Accessed: Apr. 2023. [Online]. Available: https://www.wireguard.com/

[31] ETSI. "Open Source MANO (OSM)", accessed: Sep. 2023. [Online]. Available: https://etsi.org/technologies/open-source-mano

[32] ETSI GS. "Network Function Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework, Dec., 2015. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-EVE/001_099/005/01.01.01_60/gs_nfv-eve005v010101p.pdf

[33] ETSI. "Network Functions Virtualisation (NFV); Management and Orchestration, Dec., 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-MAN/001_099/001/01.02.01_60/gr_NFV-MAN001v010201p.pdf

[34] O.-M. Ungureanu, C. Vladeanu, and R. Kooij, “Kubernetes cluster optimization using hybrid shared-state scheduling framework,” 07 2019, pp. 1–12.