UNIVERSITY OF BERN

BACHELOR THESIS

Management of SDN/NFV based Mobile Networks

Author:
Balz ASCHWANDEN

Mentors:
Dr. Eryk SCHILLER
Dr. Thiago GENEZ
Supervisor:
Prof. Dr. Torsten BRAUN

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science in Computer Science

in the

Communication and Distributed Systems Group (CDS)
Institute of Computer Science

August 13, 2019

Declaration of Authorship

I, Balz ASCHWANDEN, declare that this thesis titled, "Management of SDN/NFV based Mobile Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ________________________________

Date: ________________________________

UNIVERSITY OF BERN

Abstract

Faculty of Science
Institute of Computer Science

Bachelor of Science in Computer Science

Management of SDN/NFV based Mobile Networks

by Balz ASCHWANDEN

The ever-increasing global Internet traffic challenges network providers. In order for them to meet these demands, they have to embrace Network Function Virtualization (NFV). While Radio Access Network (RAN) implementations are being improved rapidly, many operators still have to work with legacy technologies (e.g. Universal Terrestrial Radio Access Network (UTRAN)/GSM EDGE Radio Access Network (GERAN)), where a large burden still rest with the Evolved Packet Core (EPC). Therefore, network providers have to over-provision their data centers to account for peak demand but cannot downsize the infrastructure when demand is low, because powering machines up or down takes too long. This leads to wasted resources. In this thesis, we use container technology, a special form of virtualization, to implement NFV and to develop an elastic version of EPC. The EPC is provided by the OpenAirInterface (OAI) 5G Project. An elastic EPC scales out or in, depending on demand, and saves resources in times of low traffic. We design a containerized EPC and provide evaluation considerations for the tools we use for management, monitoring, virtualization, and traffic generation. Our work shows that two choices by OAI pose an obstacle for implementing a fully elastic EPC with current container orchestration providers. The first is the reliance of OAI on IP addresses instead of DNS. The second is Stream Control Transmission Protocol (SCTP), which is not fully supported in all major container orchestration implementations. We also demonstrate the limits of the OAI System Emulation (oaisim), a Radio Access Network (RAN) simulation utility provided by OAI. To address these issues, we have developed our own scaling engine which can use the advantages of established container orchestration implementations where possible, but addresses the shortcomings where necessary. Results are promising, but further reserach needs to be undertaken in order to acheave a fully elastic EPC.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Global network traffic is increasing on a daily basis. In large parts, this is due to the gaining popularity of the Internet of Things (IoT). A forecast by Cisco predicts a 46 times increase in global Internet traffic from 2017 to 2022. By 2022, the annual rate of mobile data traffic will be more than 100 times larger than in 2012.[1] This rapid increase in mobile network traffic and IoT devices are the main challenges that network providers are facing today.[2] This challenge is further increased by the fact that most network infrastructure is dominated by proprietary, tailored hardware appliances. These appliances are costly to acquire, difficult to integrate into an existing infrastructure and challenging to scale. Due to all these difficulties, network providers find it hard to adapt to a changing environment. If they want to keep up with the increasing demands, they have to look at new technologies to optimize their infrastructure. A Whitepaper by the European Telecommunications Standards Institute (ETSI) suggests Network Function Virtualization (NFV) as a solution to cope with these issues.[3, 4]

NFV brings several advantages. By leveraging virtualization technologies, applications can be run on standardized hardware instead of custom hardware appliances. This simplifies the entire hardware life-cycle as well as organizing the physical layout of a data center. Because virtualized applications can run in cluster mode, where one piece of hardware can compensate for the failure of another, NFV can also be resistant to hardware failures. Since every aspect of a network can be depicted in software, testing and rollback scenarios become easy to handle as well. With custom hardware, the hardware has to be acquired and installed in the data center and testing is limited to a dedicated testing setup, which consists of expensive hardware and is often not representative of the production environment. With NFV, the state of a network is entirely represented as machine readable text. This text can be used by several computer programs to automatically configure an environment. This in turn simplifies switching between different setups and configurations.

NFV does not only simplify testing and hardware lifecycles, it assists network providers in one more important area: Scaling. Since hardware appliances are time-consuming to change, they cannot be moved into or out of a data center on demand. This leads to data centers often being over-provisioned to buffer peak demand.[5] Virtual Machines (VMs) and containers can be started and stopped much faster than traditional hardware appliances. This allows for just-in-time provisioning of network resources. By eliminating the need for an overprovisioned data center, NFV can help reducing the number of running services, saving resources and ultimately reducing expenses.

Several projects aim at leveraging NFV to assist network providers in optimizing 4G/LTE and implementing 5G. These projects range from providing functions that can be deployed on an existing infrastructure (e.g. GNF, formally known as GLANF[6]) to a complete implementation of LTE networks. Open EPC[7] and OpenAirInterface (OAI)[8] are two recent efforts of the latter. To the best of our knowledge, Open EPC has only licensed code for commercial deployment available. OAI which has been developed by EURECOM, on the other hand, is an open source Software Defined Radio (SDR) implementation of LTE including both RAN and EPC. Previous studies have also shown that OAI can be deployed on a cloud infrastructure.[9]

Uptake of container technology grows within the IT industry and more possibilities are suddenly available to network providers. These possibilities grow even further, if one does keep in mind that edge devices such as wireless routers and switches have become smarter and more powerful over time as well.[10] Moving network functionality away from central data centers and closer to the user would not only increase performance – and thus user experience – it would also reduce traffic and processing-time in the data center. Container technology is essential for this undertaking, as VMs – even optimized ones – often have to run on custom hardware and take too long to be provisioned in order to satisfy the rapidly changing topology at the network edge. Containers, on the other hand, can be created and started within seconds and can run on commodity hardware.[10]

Today's mobile networking infrastructure in Switzerland is largely based on Long Term Evolution (LTE), which is one type of the fourth generation (4G) implementation of mobile phone technology, often referred to as 4G/LTE or 4G LTE. This is the most commonly used version of 4G worldwide.[11] LTE is based on three main components: The User Equipment (UE), the Evolved UMTS Terrestrial Radio Access Network (E-ULTRAN) and the Evolved Packet Core (EPC).[12] The move to a 5G network is the next technological challenge network providers are facing. In preparation to this change, several infrastructure components are being updated but as [13] point out: while Radio Access Network (RAN) implementations are being improved rapidly, many operators still have to work with legacy hardware, with a large burden still rest with the EPC. EPC was not designed with elasticity in mind. This means that a traditional EPC cannot scale to traffic demand and is vulnerable to hardware failure.[13]

The future holds a number of challenges for network providers. However, we now have an overview of several ways to meet these challenges – both from a technological as well as an architectural point of view. In this thesis, we will address the problem of overprovisioning by focusing on the EPC, one of the main components of LTE. By its nature, EPC has to be overprovisioned because all traffic has to be routed through. As such, there is a strong need to redesign the EPC.[14] It is the one element of LTE that can profit the most from a NFV and scaling approach.

For this thesis, we start out with the code provided by OAI, isolate and containerize the individual components and scale them according to demand. We will also use OAI System Emulation (oaisim), a simulation tool provided by OAI to do simulated testing against our virtualized EPC.[15]

1.2 Contributions

The goal of this thesis is to provide, evaluate, and implement a containerized EPC. Our implementation of the OAI EPC in Docker allows for several EPC core components to be running in parallel. This allows for load balancing between the different instances of these core components, allowing for optimal resource allocation with regards to situational requirements. A fully elastic EPC is not possible at the moment due to the design decisions made by OAI. However, we provide a valid prove of concept and propose a list of changes which could be implemented to the OAI EPC to make fully elastic EPC possible. Through our work we also expose the limitations of popular container orchestration solutions like Kubernetes, docker-swarm and docker-compose when confronted with an application that operates on the lower levels of the networking stack. OAI uses protocols which are not supported by Kubernetes. It also relies heavily on IP-address-based instead of DNS-based routing, which causes a problem with all of the build-in scaling solutions for Kubernetes as well as docker-swarm and docker-compose. To overcome these obstacles, we leverage several of the existing components of docker-compose to implement our own orchestration solution. This solution includes monitoring of network traffic as well as fully configurable auto-scaling features.

In our solution, we implemented an autoscaling engine as well as a monitoring solution. The monitoring solution uses cAdvisor[16] and Prometheus[17] to gather information about the load on individual EPC containers. To react to this load, we built a custom autoscaling engine called balancer.py. It collects information about EPC performance from Prometheus and uses this data to determine whether running instances of an EPC component should be stopped or new instances should be added. Changing the number of running instances is done by leveraging the management tools of docker-compose inside balancer.py.

We tested the functionality of our EPC using the OpenAirInterface System Emulation (oaisim), a virtualization tool created by OAI which provides a virtualized eNB and UE.[18] We also used iperf[19] to test how fast our engine could react to changing network traffic demands. We found that our solution outperforms traditional approaches using VMs by a large margin in terms of start up time and therefore also in terms of flexibility.

1.3 Thesis Structure

The rest of this thesis is organized as follows: Chapter 2 provides an overview over of the key concepts that form the foundation of our thesis, while Chapter 3 looks at the state of research in the area of NFV with a focus on auto-scaling as well as the tools commonly available. Chapter 4 illustrates our choices of architecture and implementation of the EPC and our auto-scaling engine. Chapter 5 introduces our testing and results, Chapter 6 our conclusions and ideas for future research in the areas of NFV and virtualization.

Chapter 2

State of the Art

2.1 LTE Network

LTE is one type of 4G. However, it is the most popular implementation and often referred to as 4G/LTE. Other 4G networks are Evolved High Speed Packet Access (HSPA+) and Worldwide Interoperability for Microwave Access (WiMax).[11] The Third Generation Partnership Project (3GPP) initiated LTE with the goal of providing a high data rate, low latency and packet optimized radio access technology supporting flexible bandwidth deployments. The network architecture was designed with the goal to support packet-switched traffic with seamless mobility and great quality of service. LTE is based on three main components, as illustrated in Figure 2.1: User Equippment (UE), Evolved-UMTS Terrestrial Radio Access Network (E-UTRAN) and Evolved Packet Core (EPC).[12]

The E-UTRAN handles the radio communication between UE and EPC and has the evolved NodeB (eNB) as its only component. Each eNB acts as a base station, controlling the UEs in a given area that is divided into one or more cells. The EPC consists of a Home Subscriber Server (HSS), a Mobility Management Entity (MME), a Serving Gateway (SGW) and a Packet Data Network Gateway (PGW).[20] The last two are often bundled together as a Serving Packet Gateway (SPGW). This is also the case for the OAI implementation of EPC. The interactions between the different EPC elements and E-UTRAN are illustrated in Figure 2.2.

When an UE wants to connect to an EPC, it sends an attach request message to the MME via its eNB. The MME will then authenticate the UE with the help of the HSS. Once the UE is authenticated, secure communication is established and a data tunnel is set up. This is called the default bearer. OAI uses a mix of different General Packet Radio Service (GPRS) Tunneling Protocols (GTPs) for its bearers. GTPV2-C is used for S11, the interface between the MME and SPGW. GTPV1-U is used for the S1

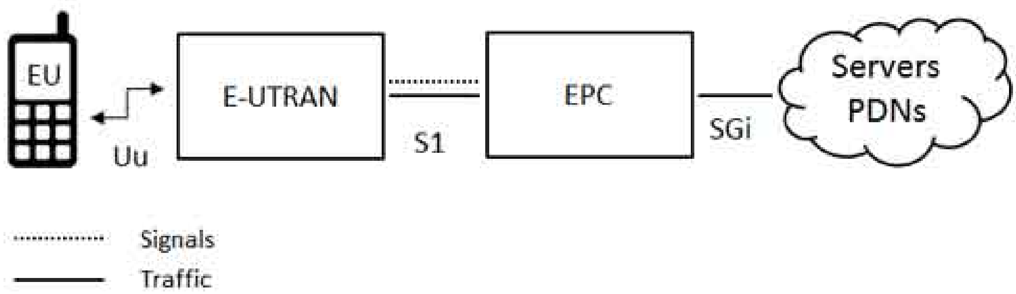

FIGURE 2.1: High-Level overview of LTE network.[12]

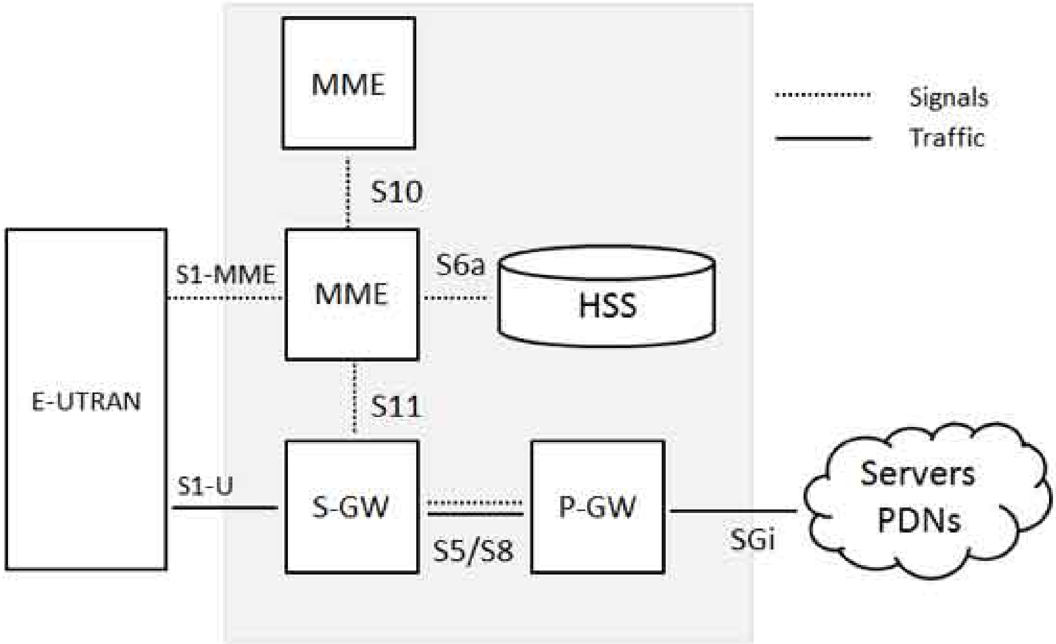

FIGURE 2.2: EPC elements and their interaction.[12]

user plane external interface (S1-U), which is defined between eNodeB and SPGW. See [21] for an example configuration.

For this process to function an IP address is assigned from the PGW to the UE and the tunnel endpoint identifier values for the bearer are exchanged between eNB, MME, SGW, and PGW. At the end of this process, a tunnel for the UE from eNB to PGW is established and the UE can receive data from a Packet Data Network (PDN).[14] A very good illustration of the attachment process is provided by [22]. They also show how traffic between the different components can be intercepted by listening on the appropriate interfaces.

2.2 NFV and SDN

Network Function Virtualization (NFV) is a network architecture concept that uses virtualization to abstract entire classes of network node functions into building blocks that may connect or chain together to create communication services. Unlike traditional Network Functions, they do not necessarily depend on particular hardware but can run on cloud computing infrastructure. Examples of NFV are load balancers, firewalls, and intrusion detection devices.[23] Advantages of NFV include the reduced cost and overall dependency toward a hardware vendor, as NFV can run on commodity hardware. The biggest advantage, however, is the increased flexibility and reduced time to deploy new network services. This flexibility also allows for quick scaling of services to address changing demands.[24]

NFV enables dynamic modification of Network Services (NSs), enabling a service provider to quickly react to user demand. Maintaining infrastructure for peak requirements is one of the main cost factors for cloud infrastructures.[25] This factor can be greatly reduced by using dynamically deployed NFV. Traditionally, Quality of Service (QoS) was guaranteed by oversizing the capacity of Network Functions (NF). However, this leads to a system that is optimized for peak performance and

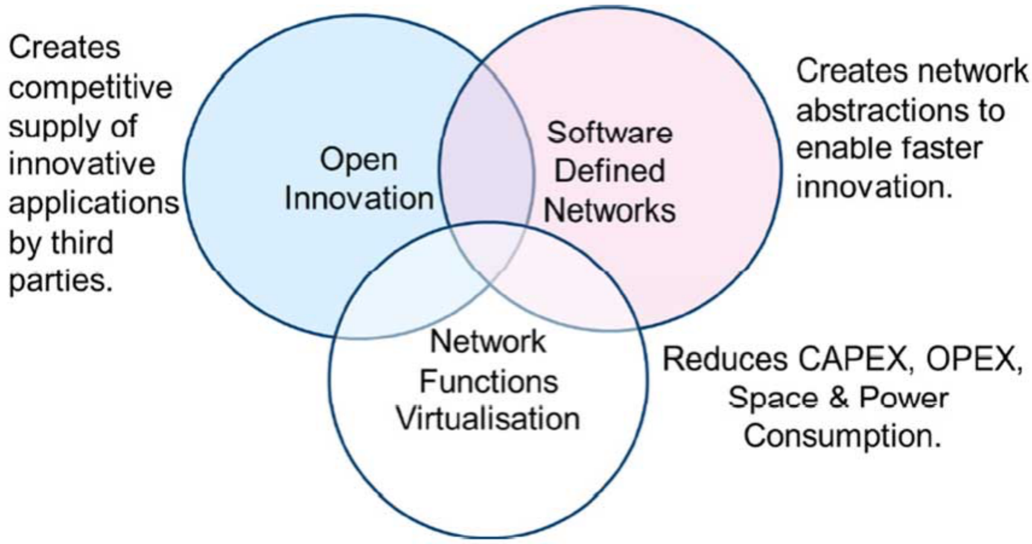

FIGURE 2.3: Complementary characteristics of NFV and SDN.[3]

overprovisioned for every other situation. NFV enables a service provider to deliver optimal performance to the user while at the same time optimizing resource allocation on the backend.[5]

NFV is often used in combination with Software Defined Networking (SDN) with which it shares many key concepts. Both NFV and SDN are software-based approaches to networking with the goal of being scalable, agile and innovative.[26] When SDN first emerged, its main focus was on separating data plane from control plane. The data plane is concerned with moving packets through a network as fast as possible. The control plane is concerned with the logic of where the packets should be routed to. This, in turn, enables administrators to configure and reconfigure an entire net-work from a central point of control. Furthermore, it reduces vendor lock-in by separating controllers from data plan devices. And last but not least, it reduces costs because controllers and applications can be run on standardized server hardware. This concept was soon adopted to various products and technologies.[27] The emergence of NFV can be seen as being part of the same idea, but both technologies are independent and can be implemented separately. However, they tend to complement each other, as Figure 2.3 illustrates.[28]

Before NFV, applications were running on bare metal machines. As servers grew in capacity, bare metal applications were no longer able to exploit the abundance of resources.[29] Scaling was also an issue, as every new service had to run on proprietary hardware, which had to be integrated in an existing facility. This led to a rapid increase in complexity.[3] Virtual Machines (VMs) soon replaced bare metal. VMs offer many benefits: By running the OS virtualized they are independent of the initial hardware and can be deployed easily. This also reduces complexity in the data center because appliances can be replaced with standardized hardware.

The operator still has to deploy and manage an entire OS, however. This is a good deal of work, if the operator wants to scale into the thousands of machines. Here is where container technology comes into play. The technology behind containers – Linux containers – was initially introduced in 2008.[30] But it has been popularized by Docker which made containers much more portable and easier to use.[31] To use Docker, one only needs to install it on a Linux server and tell it which image to run. An image is a Docker term for the service the container should run. Docker containers can be built with a straight-forward yaml syntax,[32] making VMs much more complicated to deploy in comparison.[31]

Yaml stands for *YAML Ain't Markup Language*. It is a data serialization language which can be used by all programming languages.[32] Yaml has a strong focus on readability and has grown in popularity over the last few years. It is often used as a format for configuration files. This is also the case for Docker.

The following is an example of the yaml syntax. It shows the beginning of the build instruction for the MySQL database Docker container we use as backend for our HSS. The complete source of this file can be found on GitHub.[33] We added comments starting with % to explain the meaning of the different instructions. The full reference to these instructions can be found on the Docker website.[34]

```
% Docker Containers can be built based on other, existing
% containers by specifying a so called base image.
% The FROM instruction tells Docker to use the mysql
% container version 5.6 and as a base image.
% It can be red as "Use the container image FROM mysql with
% version 5.6
FROM mysql:5.6

% The MAINTAINER instruction server no other purpose than
% to give contact information for users of a
% given container.
MAINTAINER Yan Grunenberger <yan@grunenberger.net>

% Lines starting with # are comments and are being
% ignored when building the container image.
#### CUSTOMIZE YOUR FIRST SIM DETAILS
% ARG is used to declare variables for later use.
ARG IMSI='901550000000000'
ARG MSISDN='6789'
ARG KI=0x912e7221941577df083e1591d35f4c42
ARG OPC=0x4487d12562bd21df3b076852f4d74eec
ARG APN='internet'

#### CUSTOMIZE YOUR HSS DETAILS
ARG REALM='openair4G.eur'
ARG MME='mme.openair4G.eur'

######################## Docker build instructions

% Set environment variables.
% Do not confuse with ARG which defines variables but
% does not make them available in the environment.
ENV MYSQL_USER=root
ENV MYSQL_ROOT_PASSWORD=linux

% RUN specifies a command to be executed
% inside a container. However, this one will be ignored
% because of the leading # character
#RUN apt-get update && apt-get -qy install curl
#RUN curl \
# https://gitlab.eurecom.fr/oai/openair-cn/raw/develop/src/oai_hss/db/
    oai_db.sql \
# -o /docker-entrypoint-initdb.d/oai_db.sql

% ADD is used to copy a file from the build machine
% into the container
ADD oai_db.sql /docker-entrypoint-initdb.d/oai_db.sql
```

As we can see, these build instructions are easy to comprehend and write. But not only the usability is better, containers are also advantageous from a resource management point of view. This stems mostly from the location of the virtualization layer and the way that operating system resources are used.[29, 35] To find out more, we have to take a closer look at how the two concepts work.

VMs rely on a hypervisor usually installed directly on a bare-metal system. Once the software is installed, VMs can be deployed. Each VM will receive its unique operating system and applications. Each VM is running fully isolated and is not aware of other VMs running along-side. With containers, the case is different. Containers do not rely on a hypervisor. A host operating system is installed on a bare-metal system or in a VM and the container layer is installed in this VM. Once this is set up, container instances can be deployed. Isolation is different than in the case of a VM – each containerized application shares the same underlying operating system.[35] This also results in a vastly improved startup time compared to VMs.[2, 36]

Studies analyzing resource utilization in containers and VMs mostly show a favorable outcome for containers. This is also the case in a recent study on power consumption in virtualized environments. Docker containers did not have lower energy consumption than VMs, the consumption also increased more slowly when the workload was increasing. Due to the quick startup time, containers can also be shut down and started on demand, leading to significantly reduced energy consumption compared to a VM.[37]

Due to the limited isolation, containers can share a great deal of resources. They are regarded as more resource-efficient. They are also created and started much faster than a VM. All this leads to containers being embraced by cloud providers. This is nicely illustrated in Figure 2.4.

With all this being said, containers have also downsides when compared with VMs. Due to the shared resources – mainly the same kernel space –, the underlying OS would also crash all hosted containers. Also, as isolation is less strong compared to VMs, this could open up attack vectors. This is likely to change soon thanks to the gVisor project by Google, which is a container sandbox runtime focused on security, efficiency, and ease of use.[38, 39]

2.3 NFV and EPC

Glasgow Network Functions (GNF) is a promising project combining the advantages of NFV and SDN using Docker and OpenFlow.[6] It supports every UNIX version and does not depend on a specific programming environment. A GNF Manager exposes a REST API for centralized orchestration – separating coordination logic (Manager) and operation logic (Agent).[4] GNF could be used to solve three major concerns in today's networking infrastructure: IoT DDoS mitigation, distributed on-demand measuring and troubleshooting of network connectivity and Roaming Network Functions for the 5G network.[2]

Arteaga et al. apply the gained elasticity capacity to the Evolved Packet Core (EPC), more precisely, to the Mobile Management Entity (MME).[5] Performance is measured in terms of Mean Response Time (MRT) and scaling out/in is achieved by adding additional instances of MME Service Logic (SL) behind a MME front-end that acts as a load balancer.

Jain et al. have identified the LTE Evolved Packet Core (EPC) as the main component in need of a redesign.[14] For this redesign, several new architectures have been proposed, which incorporate Software Defined Networking (SDN) and Network

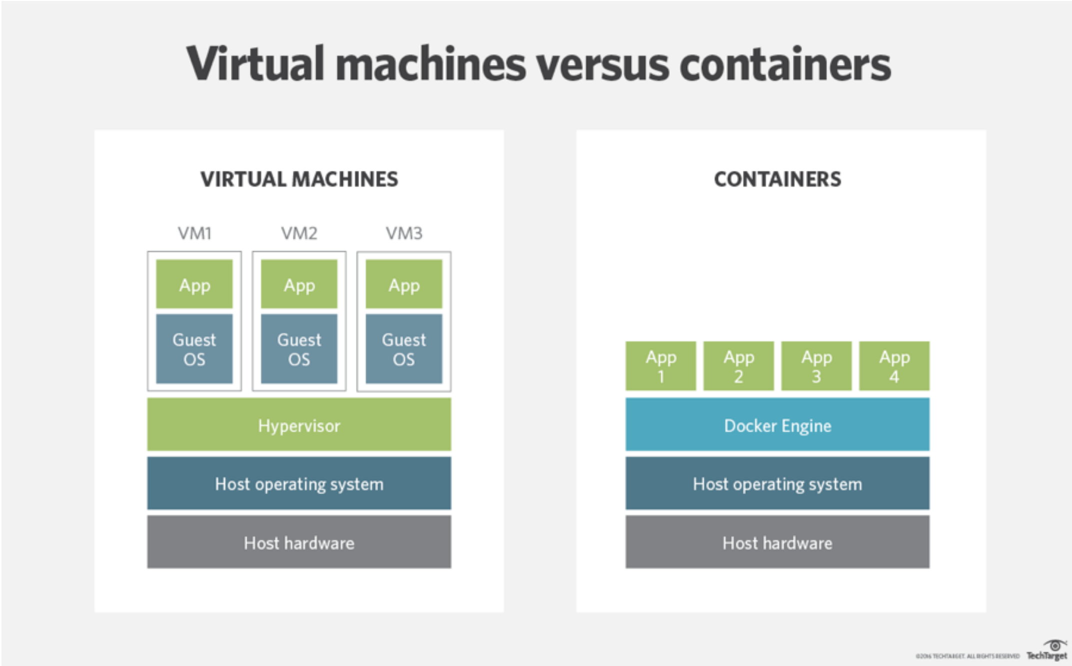

FIGURE 2.4: Isolation and Resource Sharing in VM and Container.[29]

Function Virtualization (NFV). The authors continue by reviewing several related papers and criticizing them for not following through on their research by providing a complete EPC implementation. This is where the authors go one step further and provide two reference implementations of EPC that they published as an open-source project for testing and further research. The authors compare two different implementations of LTE EPC: One using SDN and another using NFV. Both provide the basic elements of an EPC consisting of HSS, MME, SGW and PGW. They then go on testing these two implementations. Testing is done with two types of traffic: control traffic (mainly UE attach-requests) and data traffic (mainly TCP). The authors find the NFV implementation of EPC performing better in a network with heavy signaling traffic and vice versa; the SDN implementation has the upper hand when it comes to data traffic.

2.4 OpenAirInterface (OAI)

The OpenAirInterface (OAI) Software Alliance (OSA) is a non-profit consortium fostering a community of industrial as well as research contributors for open source software and hardware development for the core network (EPC), access network and user equipment (EUTRAN) of 3GPP cellular networks.[15] The EPC provided by OAI is one of only few open source EPC implementations. It has been studied before and was successfully used in a fully cloudified mobile network infrastructure.[9]

OAI also provides the OpenAirInterface System Emulation (oaisim). This tools allows for simulation and emulation of an OpenAirLTE network, including eNB and UE virtualization.[18] It allows a researcher or developer to test functionality of the OAI EPC without the use of expensive hardware.

Chapter 3

Related Works

3.1 Containerization and Scaling

There are, broadly speaking, two types of scaling. Scaling up/down (or vertical scaling) that implies adding resources to an existing machine, e.g., growing or shrinking a VM by manipulating resources like CPU or RAM that are allocated to that machine. The advantage of this approach is its simplicity regarding the architectural implementation. If a problem demands it, a more powerful machine is used. However, this approach comes with the added complexity of reducing the size of a VM – which is not easily accomplished. (See e.g. [5, 13, 25]). Scaling out/in (horizontal scaling), on the other hand, has a simplified life cycle. A VM or – more commonly – a container is created and added to the pool of workers charged with a task. Once the container is no longer needed, it is simply destroyed. The released resources can be used for another task. Of course, this adds architectural complexity. A static frontend has to be provided for clients who want to use this service. This is most commonly accomplished with a load balancing service. An example for this is JUJU, an open source modelling tool for operating software in the cloud, developed by Canonical.[40]

3.2 Scaling Algorithms for EPC

Scaling in the context of IT services referrers to the process of changing the amount of resources allocated to a service. Several scaling strategies have already been proposed in the literature. The different categories for scaling are (1) reactive, when a certain performance threshold is not reached, (2) predictive scaling based on historic data before the system overloads and (3) machine-learning based.[41] The first two strategies can lead to oscillations, when performance is just around the defined threshold and because historic data can differ from actual events, strategy 2 is not always optimized (e.g. can over-provision the NF). Lastly, machine learning is based on trial and error that can negatively impact QoS.[41] Arteaga et al. propose an adaptive scaling mechanism that is utilized by an agent to carry out improvement strategies of the scaling policy.[5]

Carella et al. have noted that requirement changes against a network can be described in two different categories: Predictable and unpredictable events.[42] They first describe what information can be deduced by knowing the time of the day and the day of the year (e.g. different peaks around the beginning of a working day than at midnight.) The resource demands can be planned ahead of time and can be deduced from network utilization and other factors. By applying an optimization strategy consisting of these two factors, Carella et al. show that it is possible to save up to 60% of resources compared to a traditional approach that involves static allocation of resources.[42]

While VMs have a startup time of several minutes, containers do the same in a few seconds.[2, 36] This fundamentally changes the way resources are provisioned, as services can now be started on demand without the end-user noticing it. This makes much pre-container research uninteresting for our work, as there is a huge difference between real-time and ahead-of-time resource allocation.[41, 43] Yet this research is still relevant when it comes to the underlying data center operations of providing seemingly endless numbers of VMs or Docker hosts to users. However, scaling follows different strategies, when used with containers than it would with VMs.

3.3 ETSI Compatible Implementation of EPC

Many researchers have proposed an additional ETSI MANO compliant element to introduce auto-scaling into the system. Several of them aim at making their proposed scaling implementation ETSI NFV MANO compatible and ETSI itself moved into this direction by its propositions in ETSI GANA. Dutta et al. propose additional functions, namely QoE Assessor (QA), Resource Usage Monitor (RUM) as well as Elasticity Decision Maker (EDM) and integrate them within the ETSI Service Manager (SM) and Service Orchestrator (SO).[25] Carella et al. propose an Auto-scaling Engine (AE) and integrate it into ETSI NFV as an additional functional element.[42] Their implementation has been published as an open source project with the name Open Baton.[44] Their AE consists of three main components: The detector (detecting KPIs and reports if they are no longer acceptable), the decision maker (notified by the detector, decides whether scaling action needs to be undertaken or not (either because not necessary or not possible)) and the executor (executing scaling action, initializing cool-down counter). We will take inspiration from the latter approach and use it to implement our own auto-scaling logic.

3.4 Existing NFV Architecture Frameworks

Containers are superior to VMs in terms of resource utilization and startup time.[35, 36] As they run fewer processes than VMs, they are also easier to manage. However, even the simplest systems will likely increase in complexity when it increases in size. ETSI realized this from the start and the ETSI MANO includes definitions for ways to connect and load balance the different NFVs. Figure 3.1 shows the ETSI MANO reference. We will now look at some of the important elements of the ETSI framework and how they apply to our study.

ETSI NFV MANO leveraged cloud orchestration and management to improve NFV. The MANO architecture has to be integrated into an existing system, providing an open Representational State Transfer (REST) Application Programming Interface (API) to make user requests for VNFs easy to automate. The three ETSI NFV MANO Function Blocks are illustrated in 3.1. They consist of:

1. Virtualized Infrastructure Manager (VIM), which controls and manages the underlying infrastructure components such as storage and network resources

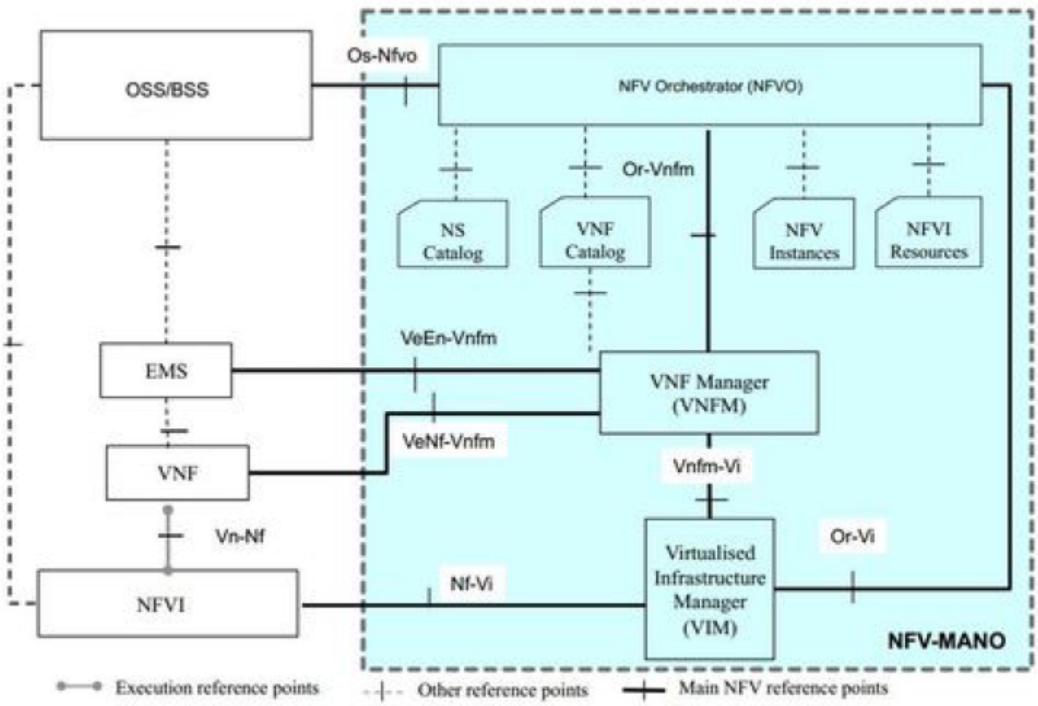

FIGURE 3.1: ETSI MANO Function Blocks.[45]

2. NFV Orchestrator (NFVO), which bridges MANO with the Operations Support Systems (OSS) and Business Support Systems (BSS) elements. It authorizes infrastructure resource requests and is in charge of on-boarding new Network Services (NS).

3. VNF Manager, which is in charge of VNF instance life cycles.

ETSI NFV MANO provides clear guidelines for implementing functional components that can react to outside demands – it is designed for **automation**. However, because its concern is the implementation of a flexible architecture, it does not go into detail about the optimization aspects. This is, where ETSI GANA comes into play. ETSI GANA is designed for **autonomy**. Automation is about workflow reduction, whereas autonomy, or rather autonomic management emphasizes learning, reasoning, and adaptation e.g., the ability to scale a service in or out depending on current demand. GANA introduces the autonomic Decision-Making-Element (DE), which is responsible for adaptive control of a system and its resources. DE can control Managed Entities (MEs) such as NFVs to improve the overall efficiency of a system. The role of DEs is further generalized with the concept of the Knowledge Plane (KP). The role of a KP is to have a high-level model of the network's tasks. It should collaborate with the DEs at the lower layers.

3.5 Container Orchestration and ETSI Standards

When ETSI MANO was first released, containers were not as widely used as today. Nakajima shows, that many of the roles proposed by ETSI MANO can be fulfilled by containers.[46] Figure 3.3 illustrates how containers are used for NFV and how OpenStack is taking over the part of NFVO and other ETSI MANO elements.

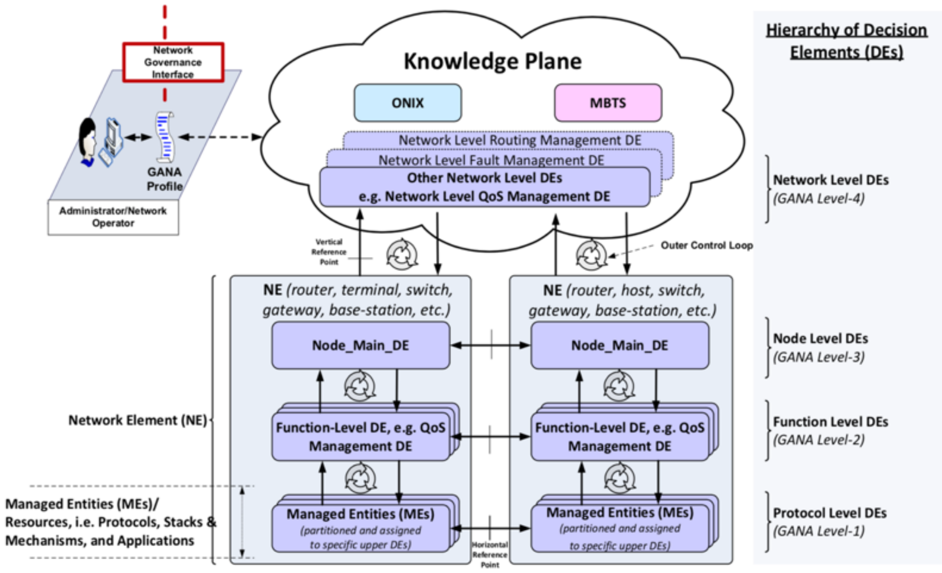

FIGURE 3.2: Visualization of the different Planes and Elements taken from ETSI GANA.[45]

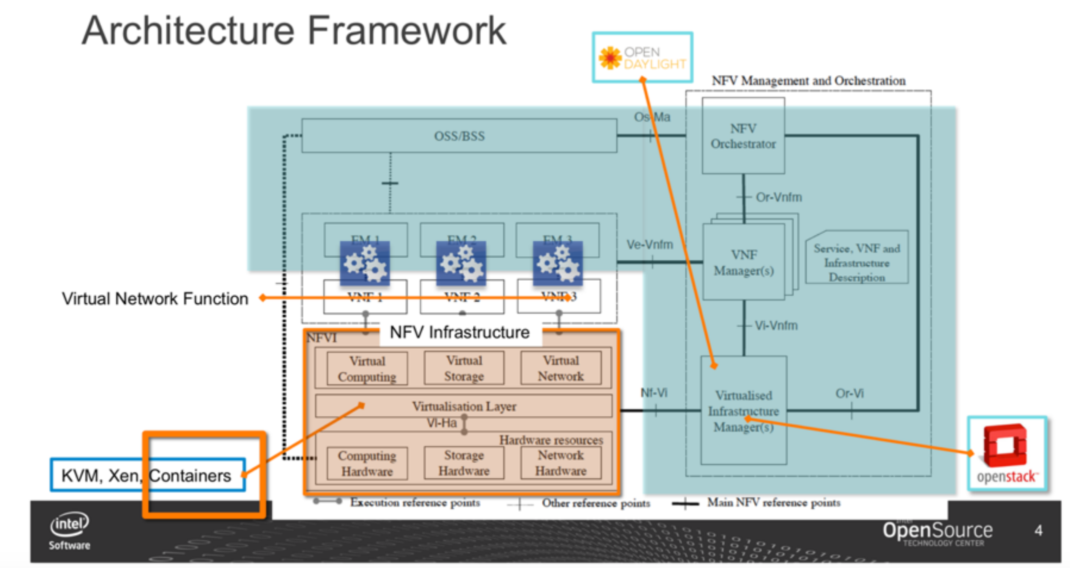

FIGURE 3.3: Containerized NFV running on OpenStack.[46]

We argue that any container orchestration tool can replace OpenStack. As Kubernetes (originally developed by Google) is the most popular orchestration tool, it makes sense to use it in this example.[31]

Kubernetes is a container orchestration tool designed to run as a cluster with a central Master and a number of nodes, which are VMs or bare-metal machines. To deploy applications, you then create so-called deployments (with deployment configurations). Using this configuration information, the Kubernetes master schedules the applications into individual nodes in the cluster. It also monitors these instances and will replace instances if they go down. This feature is called a self-healing mechanism.[47]

With every deployment, Kubernetes creates a pod to host the application instance. A pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker), and some shared resources for those containers. Once the pod is up and running, we have to expose it to the Internet. We do this explicitly because although every pod has a unique IP address, pods can stop working (either due to scaling, malfunction or a failed up-date and the old version of a pod has to be replaced with a new one). When the master spins up a new pod, it will have a new IP address. We need a way to explicitly assign this. The way of doing it in Kubernetes is with services. A service in Kubernetes is an abstraction which defines a logical set of pods and a policy by which to access them.[48]

To the best of our knowledge, Kubernetes is currently the only orchestration tool offering IP-based load balancing out of the box. With its design of nodes and service endpoints, Kubernetes offers a simple way for administrators to scale a service out and in. Unfortunately, by the time of this writing, Kubernetes does not support the SCTP protocol yet. This means that there is no way to expose ports that will route this traffic to the outside.[49] This is a problem because freeDiameter (an open source tool used by OAI)[50] and OAI are relying on it. We were able to compile OAI EPC without SCTP, but that was of no use. Although the No_SCTP option is already set in the preference file provided by OAI, OAI EPC still depends on SCTP for the S1 interface between MME and eNB.

Fortunately, docker-swarm and docker-compose (both developed by Docker Inc.) are both valid alternatives to Kubernetes. They share many of the core concepts but lack the IP address-based automatic load balancing Kubernetes provides. We will get back to this in Chapter 4.

3.6 Monitoring

3.6.1 Key Performance Indicators

To access the performance of a network as well as the Quality of Service (QaS) perceived by the user, Key Performance Indicators (KPIs) are measured and evaluated. Several KPIs have been proposed for LTE in general and for the EPC in particular. ETSI defines KPIs for the EPC within the 3GPP Project and makes a distinction between the following three types of KPIs: Accessibility, Mobility and Utilization.[51] This is the full list of defined KPIs as defined by ETSI:

1. Accessibility KPIs
    - EPS Attach Success Rate
    - Dedicated EPS Bearer Creation Success Rate
    - Dedicated Bearer Set-up Time by MME (Mean)
    - Service Request Success Rate
2. Mobility KPIs
    - Inter-RAT Outgoing Handover Success Rate (EPS→ GSM)
    - Inter-RAT Outgoing Handover Success Rate (EPS→ UMTS)
    - Inter-RAT Outgoing Handover Success Rate (EPS→ CDMA2000)
    - Inter-RAT Incoming Handover Success Rate (GSM→ EPS)
    - Inter-RAT Incoming Handover Success Rate (UMTS→ EPS)
    - Inter-RAT Incoming Handover Success Rate (CDMA2000→ EPS)
    - Tracking Area Update Success Rate
3. Utilization KPI
    - Mean Active Dedicated EPS Bearer Utilization

In addition to these, researchers often use throughput,[52, 53] latency, or one-way delay (OWD), as indicators for network performance.[52, 54] Latency is defined by 3GPP in two ways: control-plane latency and user-plane latency. Control-plane latency is the time required by the call-setup procedure and user-plane latency is the one-way transmission time of an IP data packet from the UE to the RAN edge-node or vice versa.[54, 55] Studies on EPC performance usually focus on the second of these two definitions and analyze user-plane latency (e.g. in [54] and [56]). Other research also used EPC Mean Response Time (MRT) to indicate the QoS.[5] KPIs can also include Received Power, Handover Time and Round-trip time.[57]

3.6.2 Measuring Key Performance Indicators

Unfortunately, neither Kubernetes nor Docker (neither docker-compose nor docker-swarm) offer a native way of measuring network related KPIs to the level of detail we require. Also, popular monitoring tools such as Jaeger[58] have to be integrated into the source code – this goes beyond the scope of this thesis. This forces researchers to fall back on tools outside of the framework (e.g. in Medel et al., where iperf is used to assess the performance of a Kubernetes setup.)[59] However, some of the information received from the Docker daemon can still be used to know the state of the network. Docker offers the *stats* command, which lets the user query several metrics, for instance information about CPU and memory usage as well as net I/O. Throughput and latency in production networks can be measured directly on the UE, for example by using an Android App.[52]

Another popular measure is the use of iperf[19], which is used for synthetic traffic generation and system profiling for LTE networks (see e.g. in [60] and [14, 60] or for networks in general [59, 61, 62]). For us, the work done by Jain et al. is particularly interesting.[14] The authors compare performance for signaling and data-plane traffic in SDN-based versus NFV-based EPCs. By using throughput as well as responsiveness (establishment of UE attach/detachment) the authors combine KPIs in the sense defined by ETSI with more traditional network performance indicators.[14]

After gathering all this information, it seems clear that using iperf assessing the performance of a network is inevitable. Hence, this is a tool we will use as well. For signaling traffic, Jain et al. used a custom RAN simulator.[14] We hope to achieve the same or similar effect by using the oaisim tool provided by OAI.[8] There is research on a dedicated OAI traffic generator by Hafsaoui et al., but to the best of our knowledge, it has never been publicly available.[56]

Chapter 4

Architecture and Implementation

4.1 Architecture

To design our architecture, we have to consider several factors. For implementing VNF we have to take the MANO reference architecture by ETSI into account,[3, 45] and, since we also want for our implementation to be able to be self-optimizing (autonomic), we have to consider the ETSI proposal for Generic Autonomic Networking Architecture (GANA).[63]. We provide a fully working load-balancing engine which is able to control docker-compose to start/stop Docker containers based on demand on the EPC.

A detailed list of all the used technologies is provided in Appendix E. Figure 4.1 also shows an overview over the used components. They are the following:

- **Configuration**: Controls the behavior of the load balancer. Information about the number of container instances as well as thresholds are configured here.
- **Load Balancer**: Controls docker-compose to start/stop Docker containers. Queries information about performance from Prometheus. Takes into account performance over a range of time as well as information in the configuration file to determine the optimal number of running Docker containers. The goal is to have to optimal number of Docker containers for OpenAirInterface EPC running. This load balamcer has been written and implemented by us for this thesis.
- **docker-compose**: Orchestration of different Docker containers. Ensures containers are in the correct virtual network. Starts and stops containers.
- **Docker container**: Core technology to containerize services. Controlled by docker-compose. Several Docker containers are running at the same time.
- **OpenAirInterface EPC**: EPC provided by the OpenAirInterface organization. The different functionalities of OpenAirInterface EPC are running in different Docker containers. OAI EPC has been modified by us to be run in individual Docker containers with dynamic commands that are run at startup.
- **cAdvisor**: Collects performance metrices from Docker containers and makes them available to Prometheus.
- **Prometheus**: Collects information from cAdvisor. Makes it available as time series.
- **Graphana**: Can visualize the information collected by Prometheus.

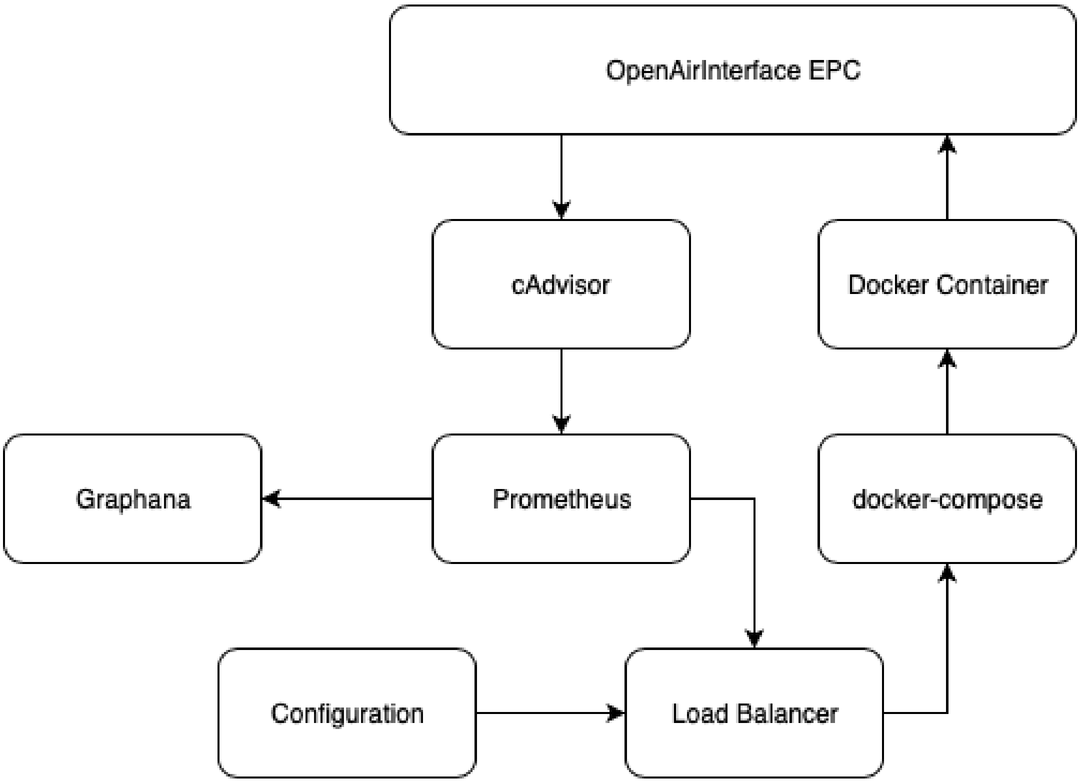

FIGURE 4.1: Component overview

4.2 Monitoring Implementation

There are several ways to monitor distributed applications such as cloudified apps or the EPC in question. The most common ones are the ELK Stack and a combination of Prometheus and Grafana. ELK is an acronym for three components: Elastic Search, LogStash and Kibana.[64] Elastic Search and its indexing components are quite resource intensive and it is not recommended to run the whole ELK stack on a single laptop. Luckily, Prometheus and Grafana are simpler to set up and are less resource intense.

Prometheus is an open source monitoring solution originally developed by soundcloud.[17] It is part of the Cloud Native Computing Foundation (CNCF) and as such well equipped to handle the various demands brought forward by containerized applications.[65] The main component we use for our work is the ability to collect time series exposed by endpoints. This information can then be displayed as a dashboard (using Grafana). They can also be queried using the extensive Prometheus API. See 4.2 for an example.

To get our monitoring data, we use cAdvisor originally developed by Google and now available as open source software on GitHub.[16] It hooks into the processes used by Docker and exports the information in a form readable by Prometheus. This gives us time series information about the most important Docker performances including network I/O and CPU usage. Other authors have used similar approaches, e.g., Dutta et al., used built-in tools of OpenStack for monitoring.[25] Figure 4.3 shows the data flow in our architecture. cAdvisor is running in a Docker container which collects metrics from all running containers. This data is then collected by Prometheus which promotes it to Grafana and exposes it over an API.

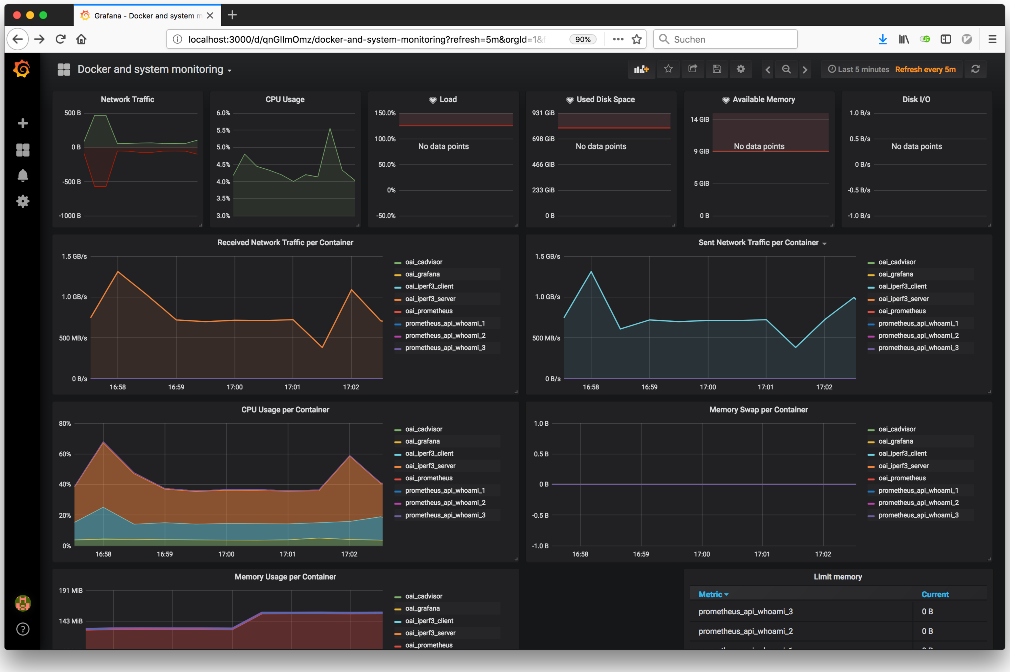

FIGURE 4.2: Example of the Grafana visualization of Prometheus time series.

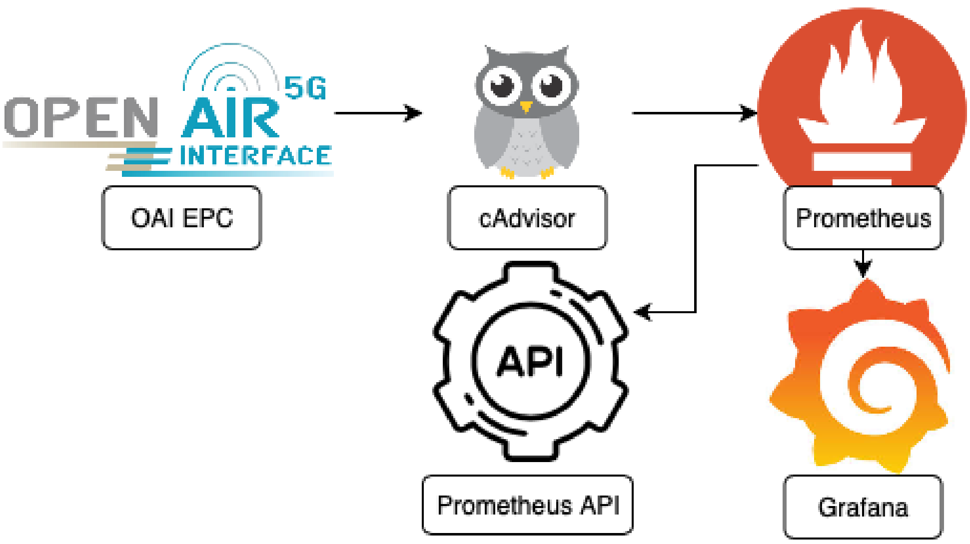

FIGURE 4.3: Data flow from left to right:
OAI EPC: Running in different Docker containers.
cAdvisor: Collecting data from OAI EPC Docker containers and exposing it to be collected by Prometheus.
Prometheus: Collecting data from cAdvisor. Forwarding it to Grafana and exposing it over the Prometheus API (symbol image).

4.3 Containerization and Scaling

As explained in section 3.2, scaling strategies can be categorized into three groups: reactive, predictive and machine-learning based. The latter two require existing data to model their approaches. We do not posses an existing data model which could be used for a predictive or a machine-learning based strategy. Therefore, we have to use the first strategy: reactive scaling. When choosing reactive scaling, we have to keep in mind the danger of oscillation, which in this context refers to an ongoing and disproportionate adaptation of allocated resources around certain edge cases.

When scaling a service, we will consider the following parameters for each service: traffic received and sent, memory, and CPU usage. These values are measured in a given interval, e.g. 10 minutes. Also, minimum and maximum sizes can be specified to control the number of instances a service should have. The received information will be divided by the number of running instances and compared against two thresholds. If the result exceeds the upper threshold, scaling out will occur. If the result is below the lower threshold, scaling in will occur.

Two mechanisms are in place to avoid oscillation. First, performance will be measured in a defined time interval. Using the average performance will give us a more accurate picture of the overall load on a system. It will also prevent overreacting to sudden spike or drop in traffic. Measuring performance is done leveraging Prometheus' functionality. Prometheus can output performance of a KPI at a given time or it can aggregate it over a given interval. Our balancer utility is written in such a way that the interval is configurable. Second, a so called cool down timer will be set every time scaling for a service occurs. No scaling actions will be performed during the count down of the cool down timer. This reduces the impact of possible oscillation effects. In our current implementation, the cool down timer is the amount of time our balancer will wait after every scaling action. This wait time can be configured to further optimize the scaling algorithm.

Arteaga et al. patched the MME to divide it into its different functions and used part of it as a load balancer.[5] Doing the same with OAI is out of scope for this thesis. Also, such an approach has the disadvantage of having to patch every part of the EPC one would like to scale. By using a generic load-balancing server – such as nginx in our case – we hope to find a more generic approach.

We know from previous work that containerization of OAI is feasible.[66] However, in order to adapt OAI to our work we have to overcome several challenges. First, we have to implement the individual OAI EPC components as Docker containers. The Docker container provided by the OAI organization has not been updated for over two years. Also, it is running all the services in one instance and can, therefore, not be scaled based in it's individual components.[67] Fortunately, we are able to profit from an open source implementation.[68] However, these containers were not designed with scalability in mind. Several variables such as host name and certificate are determined and/or generated when building the container. In order to take the advantage of the fast container startup time, building has to be completed ahead of time. Individual instances of the underlying build can then be spawned. Building on demand is time-consuming and inefficient. Therefore, we have to patch the open source implementation to allow for several elements to be determined at runtime, making for a more flexible installation.[69]

4.4 Load Balancing

We use docker-compose to document and implement our architecture. To simplify, docker-compose is like docker-swarm but without the ability to distribute containers on multiple machines. Since implementing EPC on a cluster is not needed for a proof of concept, we do not add this additional layer of complexity. As it stands, docker-compose offers a sufficiently good orchestration and command line utility to implement our EPC and to scale it out/in according to traffic demands. However, scaling our architecture over a cluster would be a good follow-up to current research.

Load balancing is a fundamental requirement for containerized deployments, where scaling is achieved by running several instances of a service next to each other. A load balancer has to make sure that load is evenly distributed across all running instances. As expected, Docker and Kubernetes offer several tools to achieve this essential feature. Unfortunately, we are limited in the choices we can use because OAI EPC and eNB require both the SCTP protocol and hardcoded IP addresses to function properly. This in turn means that we can use neither Kubernetes services nor the Docker HAProxy, an open source load balancing and proxying for TCP and HTTP-based applications.[70] Instead we have to fall back to using a more elaborate setup with a dedicated load balancing server.

Docker-compose does offer load balancing based on service names, e.g. a *service* named Home Subscriber Server (HSS) in the docker-compose with two *instances* will have both instances running behind a single service name. The default load balancing strategy is Round Robin, see Docker Hub for a more detailed example.[71] This feature would eliminate the need to implement our own load balancing. Unfortunately, we failed to make OAI EPC work with Docker name spaces. Only IP addresses can be specified in the relevant configuration. However, we can still leverage Docker load balancing by putting it behind a proxy load balancer. For this, we are using the official nginx Docker container.[72] Nginx has been used for other cloud related research[73] and offers streaming support for a variety of inputs.[74] Nginx streaming setup is straightforward; working with OAI is causing some problems though. Nginx streaming is port-based and as not all the ports used by OAI EPC are documented, this needs some testing. In the end, the setup is simple; the service name can be used when defining a backend service for nginx, which will pick up on new instances when reloaded. This solves the issue of service discovery. By reloading the server instead of restarting it, we make sure that active connections will not be interrupted.

The following nginx.conf is an example configuration file and shows how Docker name space can be used to redirect traffic to different instances of the same NFV. We added comments marked with %.

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}

% The following is the only element we have to configure
% nginx will listen to the port defined in 'listen'
% and forward the traffic to a service named 'db'
```

```
% Docker namespace can be used here.
stream {
  server {
      listen        3306;
      proxy_pass db:3306;
  }
}
```

Using this method, we were able to use the standard Docker container without modification. When starting the container, we will link the appropriate configuration file into the container. Every nginx container is assigned to a fixed IP address. OAI NFV can use this address and will then be redirected to an available container.

4.5 Autoscaling Logic

The goal of our autoscaling logic is to enable elasticity, which is the capability to dynamically scale the allocation of cloud resources to the current demand. Rapid elasticity was originally defined by NIST.[75] We can distinguish four distinct versions of elasticity: no elasticity, horizontal elasticity (allocated resources), vertical elasticity (allocated instances), and overall elasticity (both horizontal and vertical). There is also a proposed classification by Galante et al. for elasticity depending on the optimization goal and some detection algorithms which could be integrated in the AE.[42, 76] It is important to note that action only needs to be taken when a KPI deteriorates and its performance is no longer accepted.

cAdvisor is collecting KPIs from SPGW, MME, HSS and SQL, which form the EPC. The information is forwarded to Prometheus from which it is then exported to Grafana for visualization. The autoscaling logic is the only element not implemented as a Docker container. It is instead implemented as a program running on the Docker host. See Appendix A. The autoscaling engine queries information from Prometheus and interacts with docker-compose via the open API. See Figure 4.7 for a run down of the data flow.

UE and eNB are virtualized using the oaisim tool provided by OAI. We use this to collect information about the state of the EPC and whether it needs to be scaled out or in. In line with the KPIs presented already and in accordance with the measurements we get from cAdvisor we use the following KPIs to decide whether we shall scale a service or not: Network traffic, CPU usage, memory usage. These factors along with the number of already existing service instances will determine whether or not a new instance will be added or an existing one removed. As the autoscaling unit is running outside Docker on the host machine, it can seamlessly interact with the docker-compose command line utility to trigger a scaling action.

The code example in Listing 4.5 shows our interaction with the Prometheus API: The method named get_traffic_received takes the name of a docker image and a time interval as input. It does construct the query for Prometeus and sends it to the send_request method. This method is then calls the Prometheus API and asks information for the docker image specified in query_string. This will be used as a loop to query Prometheus regularly and to react to changing traffic load. The implementation can be enhanced with more KPIs and a more complex scaling algorithm in future work.

```
def send_request(query_string):
        """Send a request to the Prometheus API and return the response or
    None."""
        try:
            response = requests.get(
```

```
                url=prometheus_url,
                params={
                    "query": query_string,
                },
            )
        response_parsed = json.loads(response.content)
        try:
            result = response_parsed['data']['result'][0]['value'][1]
        except IndexError:
            # Prometheus returned empty response for this query
            result = None
        return result

def get_traffic_received(image, interval):
    """Query Prometheus API for network traffic information.
    Return received and sent traffic size for a given interval or None.
    """
    query_string =
  "sum(rate(container_network_receive_bytes_total{{image=\"{}\"}}[{}]))".
    format(image, interval)
    result = send_request(query_string)
    return result
```

4.6 Architecture Design

Figure 4.4 shows the relationship between ETSI MANO and our implementation, which uses docker for many of the underlying functions and adds our own autoscaling engine to control the docker containers running OAI EPC. This graphical representation is inspired by Nakajima et al.[46] Docker containers and docker-compose are used for the underlying infrastructure and MANO logic. The full docker-compose file which includes all containers and their source repositories can be found in Appendix B. OAI is providing the network functions and, as proposed by Carella et al., an Autoscaling Engine (AE) has been added, using the open API of docker-compose to trigger scaling actions.[42] This AE consists of cAdvisor, Prometheus, Grafana and our custom Logic Unit written in Python. Appendix E shows a more detailed overview over the tools used and their role in the architecture.

Several Docker containers are necessary to realize this functionality. We have different containers for the following tools:

- Cadvisor (part of AE)
- Prometheus (part of AE)
- Grafana (part of AE)
- MySQL (backend for HSS)
- phpMyAdmin (User interface for MySQL)
- HSS
- MME
- SPGW

We will now go through this architecture step by step. The basic ETSI MANO Functional Blocks are NFVO, VNFM and VIM.[45] The roles of NFVO and VIM will

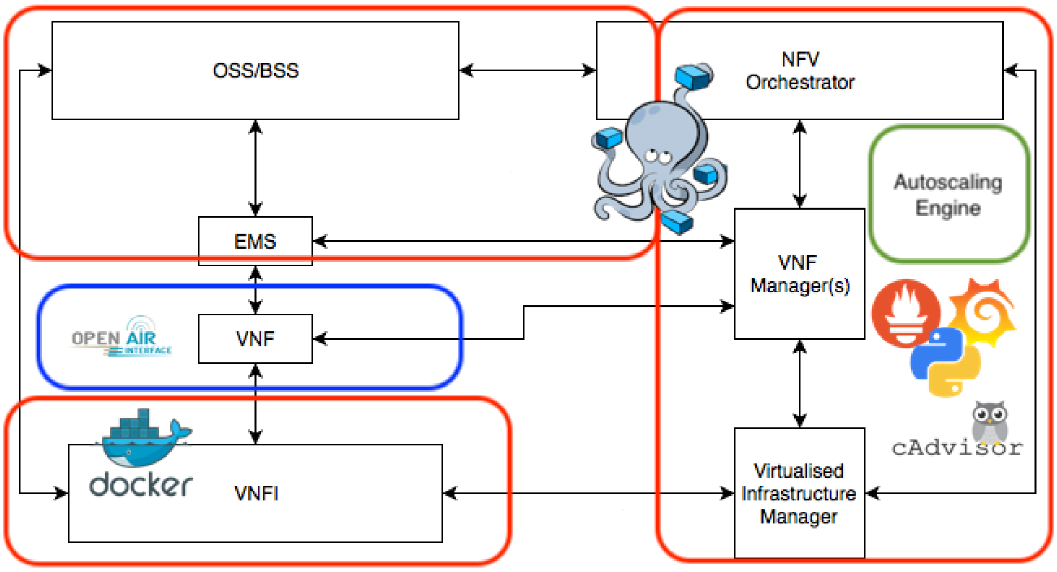

FIGURE 4.4: ETSI MANO including AE.
Red: Parts provided by Docker with docker-compose as orchestrator.
Blue: Parts provided by OAI.
Green: Autoscaling Engine (AE) with its components.

be under the responsibility of docker-compose. Docker-compose is responsible for controlling and managing the infrastructure as well as the network. However, as explained in section 4.4, we will leverage docker-compose networking to implement our own load balancing.

The VNFM is in charge of the lifecycle management of VNF instances. Here we have a mix of docker-compose and our own small Autoscaling Engine (AE). The AE will decide on how many instances of a service should run at a given time and when scaling should occur. This information will then be passed down to docker-compose, which will start/stop the actual containers. This brings us to the NFV Infrastructure (NFVI), where we rely on Docker container engines running on Ubuntu 16.04. Docker will provide the necessary abstraction to run different containerized Virtual Network Functions (VNF). These functions are provided by OAI and are split into individual containers. The load balancing will be done by nginx, as described in section 4.4. Finally, OSS/BSS as well as the Element Management System (EMS) are taken on by docker-compose again.

4.7 Docker-Compose

Figure 4.5 shows the NFV architecture in more detail with the different elements of EPC containerized. The goal of the Figure is to show which parts of our work are running inside docker as containerized applications and which are running on the host machine. In this Figure, we also see a more detailed overview of the different elements of the AE. Of these elements, the AE was developed by us. cAdvisor, Prometheus and Grafana were implemented by us to collect data from the EPC and the EPC itself was modified from the work done by [33] to allow for a more modular setup where parameters can be passed into a docker container without rebuilding the whole image. Details with regards to the docker containers can be found in Appendix B. The original configuration of oaisim was also changed, to work with our

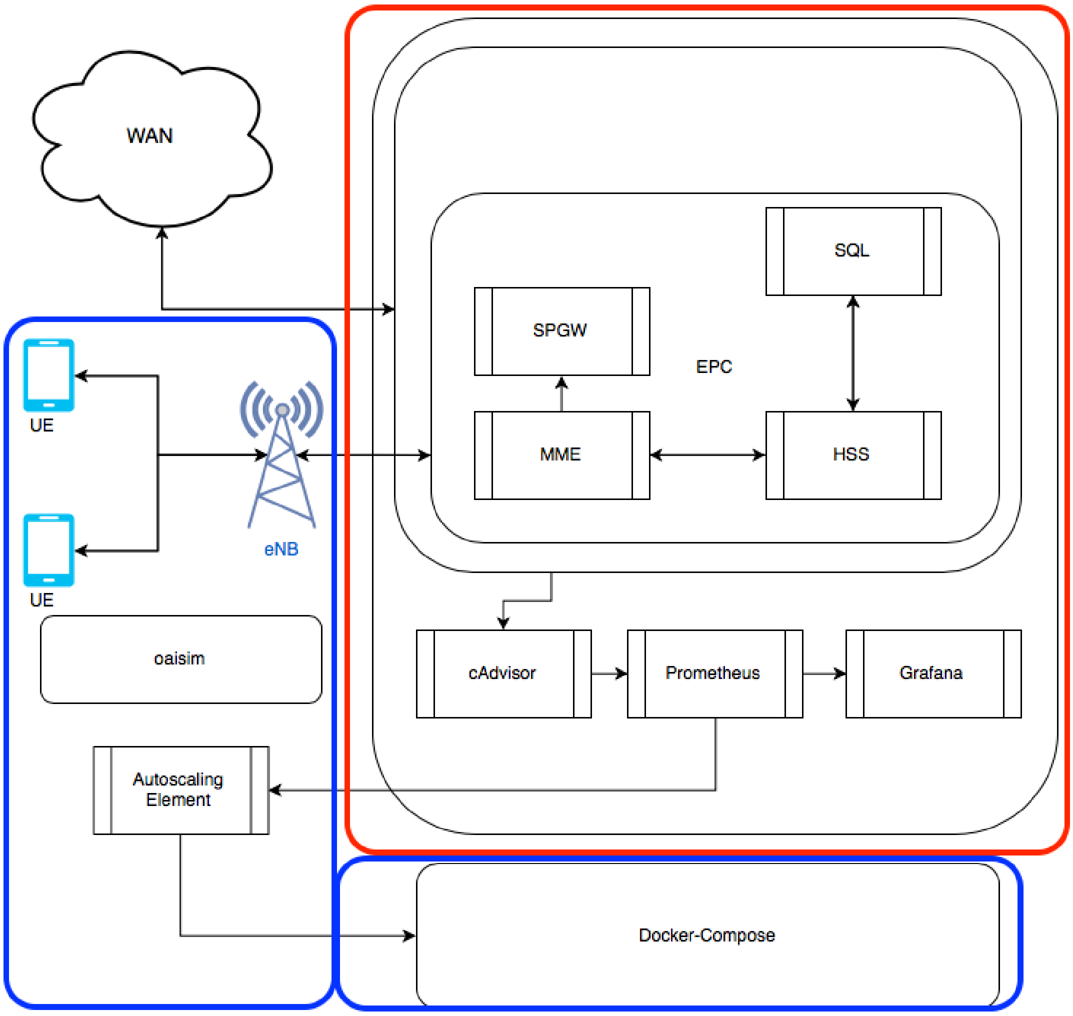

FIGURE 4.5: Detailed View of NF Architecture.
Red: Services running inside Docker containers.
Blue: Services running natively on the host machine.

EPC. Details of the configuration of the eNB simulated by oaisim can be found in Appendix C.

The entire setup was done using docker-compose. This allows for easy replication of the setup to replicate our work and to do further research. This is also an advantage over a possible implementation in Kubernetes or Docker-Swarm. Since docker-compose only relies on Docker being installed, setup is easy and the EPC including monitoring can be started by typing docker-compose up on a Linux machine.

The elements not included in the Docker setup are user equipment (UE) and radio access network (RAN), eNB in our case. Both are provided by oaisim, but could be replaced with the appropriate hardware. In its basic form, oaisim does simulate an eNB with attached UE. Successful attachment to EPC through oaisim results in a new network interface being created on the computer running oaisim. It is usually named oip1 and can be used to route traffic through the EPC. Figure 4.6 shows the interplay between oaisim and the different EPC components managed by docker-compose.

The EPC itself consists of SPGW, MME, HSS and an SQL database all running

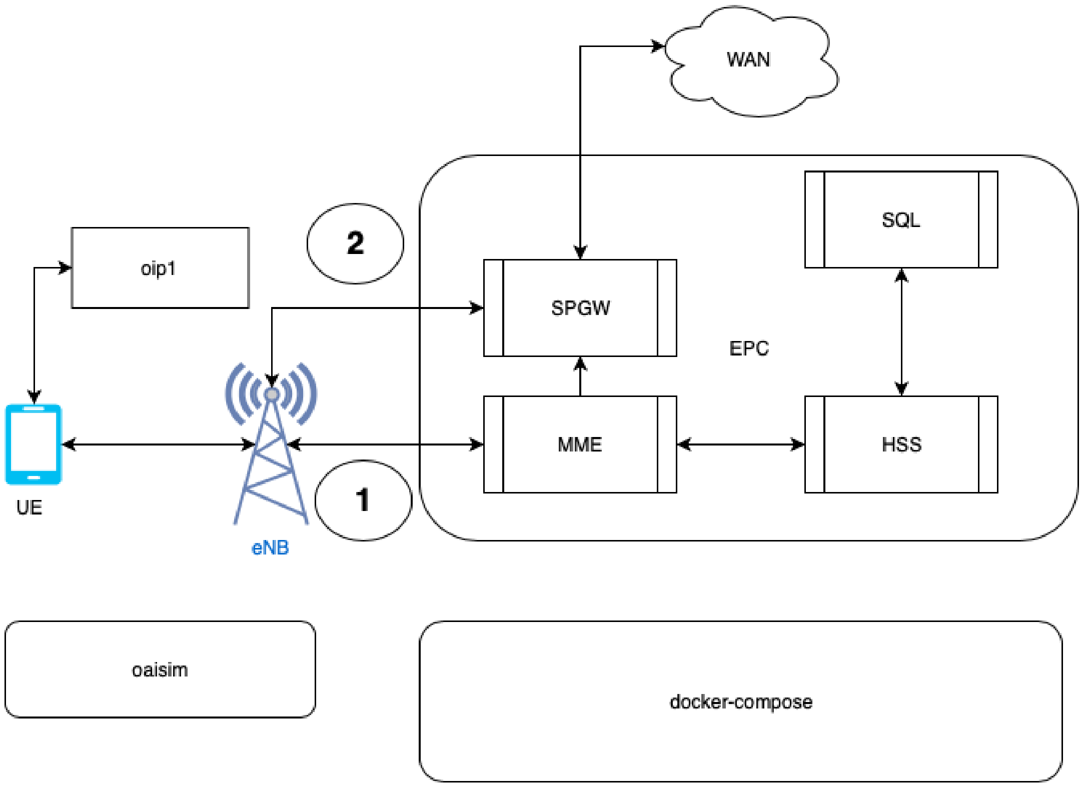

FIGURE 4.6: Interplay between oaisim and the different EPC
1: Attachment of UE to EPC.
2: After attachment, traffic is handled by SPGW.

inside Docker containers. Performance information about every container is being collected by cAdvisor and forwarded to Prometheus. The only element not running as a Docker container is our Autoscaling Engine (AE), which uses the APIs exposed by Prometheus and by docker-compose. The API exposed by Prometheus is used to query performance information. The API exposed by docker-compose is used to react to the information received from the Prometheus API to scale individual services. The full docker-compose file and detailed information about the different Docker containers is provided in Appendix B.

4.8 Data Generation

Data traffic is generated using iperf[19], which is a popular tool used by many researchers as established earlier. We are using the oaisim simulator to simulate both eNB and Unified Software Radio Peripheral (USRP) and create a virtual interface. This can then be used to route traffic from the testing machine running oaisim and the dockerized EPC to a public iperf server. By using the *--bandwith* option on the client we can determine how much traffic should be generated (e.g. *iperf3 --bandwidth 100M* will equal a traffic of 100Mbit/second, *--bandwidth 0* will send unlimited traffic.). Additional streams can be specified with the *-P* option and the number of desired streams. For our testing, we will initialize bursts of traffic over a given period of time and observe the reaction of our EPC.

Unfortunately, the unlimited traffic option is not feasible right now with oaisim,

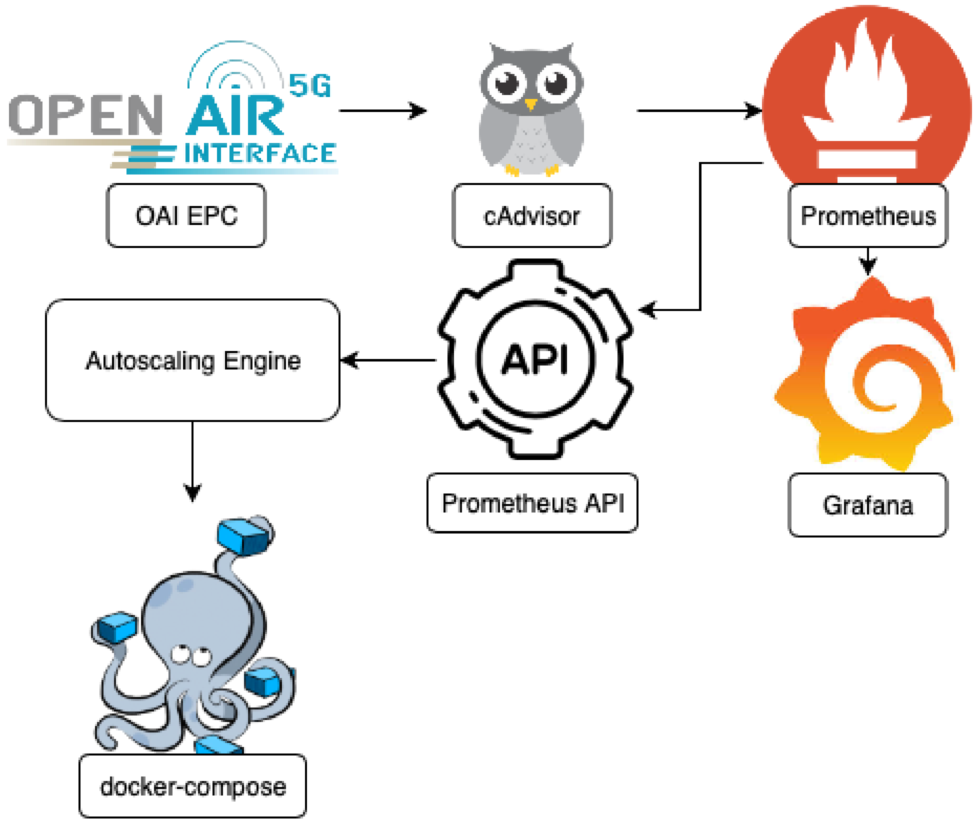

FIGURE 4.7: Data flow from left to right:
OAI EPC: Running in different Docker containers.
cAdvisor: Collecting data from OAI EPC Docker containers and exposing it to be collected by Prometheus.
Prometheus: Collecting data from cAdvisor. Forwarding it to Grafana and exposing it over the Prometheus API (symbol image).
Autoscaling Engine: Collecting data from Prometheus API.
Calling docker-compose to scale infrastructure.

since packet size is limited to 1000 kilobytes. This limit is set as a parameter. However, increasing it over the threshold of 1000 KB leads to arbitrary crashes of oaisim. These settings can be found in several places the perf_oai.bash on the official OAI GitLab Repository is a good example.[77]

Chapter 5

Testing and Results

5.1 Testing

The goal of the following series of tests is twofold. First, we will show we have build a fully functional EPC inside individual docker containers. Second, we will test the reaction time of our system by showing how our setup can react to changing network traffic.

Testing was done on Ubuntu 16.04 with Kerner 4.13 (4.13.0-45-generic). To virtualize RAN and UE, we are using OpenAirInterface System Emulation (oaisim). Oaisim is built according to the instructions found in the OAI Wiki.[18] EPC is built by opening a terminal window, changing into the appropriate directory and running docker-compose. Our AE controls EPC startup and shutdown. For traffic simulation, we use iperf3.[19]

Due to the complex setup with 10+ Docker containers, testing is done in several steps. First, we show how our containerized EPC can be used with oaisim. We start the EPC and use oaisim for UE attachment. Second, we repeat this process with our own load balancing solution to show that it provides the same functionality as docker-compose. We also show how traffic between UE and EPC can be inspected to gather further information about the UE attachment.

As explained in Chapter 4.8, packet size and traffic are limited with the current version of oaisim provided by OAI. However, by using iperf3 to inject traffic directly into our system we can work around this limitation and provide a scenario in which we can test how fast our system can react to changing requirements. This is done in Section 5.1.3.

5.1.1 Containerized EPC

For a basic testing setup, we build and start the Docker containers manually to start EPC. Once the EPC is running, oaisim is started in a different terminal window by running the run_enb_ue_virt_s1 executable. We configured oaisim to attach a single UE to EPC at startup. When this connection is established, the EPC network is accessed through the oip1 interface. Traffic going through this interface will be routed through the EPC. Presented in 5.1 are the commands to reproduce our experiment assuming both git repositories for OAI oaisim and EPC have been cloned to the test machine. After these steps have been performed, traffic can be routed into the EPC network using the oip1 interface. EPC logs demonstrating setup and attachment have been omitted due size. (Over 2000 lines of logs.)

```
# Build EPC
cd ~/docker-openairinterface-epc/
docker-compose build --no-cache

# Build oaisim
```

```
cd  ~/openairinterface5g/
source oaienv
cd cmake_targets/
./build_oai -c --UE --oaisim

# Run EPC
cd ~/docker-openairinterface-epc/
docker-compose up

# In a different terminal window
cd ~/openairinterface5g/cmake_targets/tools
sudo -E ./run_enb_ue_virt_s1 \
--config-file \
    ~/docker-openairinterface-epc/oaisim/\
    enb.band7.generic.oaisim.local_mme.conf

# Basic test in a different terminal window
ping google.com -I oip1
```

LISTING 5.1: Terminal commands to start and test a dockerized EPC.

5.1.2 Scaling EPC with balancer.py

The setup described in Section 5.1.1 provides a fully functional EPC with each of its components running in different Docker containers. However, the setup does not yet offer the desired scaling functionality. As explained in Chapter 3, we are currently unable to leverage the automatic scaling functions provided by the different container orchestration providers. Therefore, we wrote a custom utility called balance.py which uses performance metrics gathered from cAdvisor and Prometheus to decide whether EPC is running with the optimal amount of instances or it should be scaled in or out. See Appendix A for a more detailed look at balance.py. A log file demonstrating setup and signaling traffic between the EPC components can be found in Appendix F.

Tracking User Attachment

Using the technique outlined in [22], we can intercept network traffic to detect the initial context setup between EPC and UE. This will allow us to detect the unique identifiers of the different components.

To simplify this process, we wrote a tool based on work by Schiller.[78] The example in Listing 5.2 shows the process of identifying the TEID of ENB and MME by sniffing network traffic. Details with regards to the traffic sniffing utility can be found in Appendix D. The traffic sniffing utility works by listening on the interface that is created by oaisim. It will listen to the attachment request coming from the UE virtualized by oaisim.

```

# Run EPC using balance.py
cd ~/docker-openairinterface-epc/
python balance.py

# In a different terminal window
# This will start our traffic sniffing utility
cd ~/docker-openairinterface-epc/traffic
sudo python main.py

# In a different terminal window
```

```
cd ~/openairinterface5g/cmake_targets/tools
sudo -E ./run_enb_ue_virt_s1 \
--config-file \
    ~/docker-openairinterface-epc/oaisim/\
    enb.band7.generic.oaisim.local_mme.conf

# traffic/main.py does sniff the initial
# attachment traffic and will print out the TEIDs.
# The following are the results when using our
# standard setup:
ENB TEID is 00000001
MME TEID is 86ab21c3
```

LISTING 5.2: Terminal commands to start and test a dockerized EPC with load balancer.

5.1.3 Scaling using balancer.py and iperf3

By using the load generating capabilities of iperf3, we can test our system under load. The question we are trying to answer here is the following: How fast can new instances of SPGW be deployed, when the load balancing algorithm detects an increase or decrease in network traffic?

When testing the reaction time of our balancer.py, we have to take into account the following fact: There will be a delay between the time, when the system recognizes a change and the time, when a new instance of a service is deployed. To observe this difference, we log ever scaling action that is executed by balancer.py. At the same time, we measure the number of running docker containers through an independent utility running outside of balancer.py. This container instance counter script is only needed to assure accuracy of our experiment and is not part of the overall architecture. The traffic generated by iperf3 fluctuates between 10 and 300MBit per second. The detailed traffic numbers are omitted from the test results. The objective of this test was to identify the reaction time of our setup and not the maximum throughput. We are also limiting the maximum number of deployable instances of SPGW to 27. The reason for this is a simple hardware constraint. More than 27 instances, and our host would become unresponsive. This is not something we consider a problem because in production, more powerful hardware would be used - ideally a cluster of some kind.

The autoscaling engine balancer.py as well as its exact settings can be found in Appendix A. Both the traffic generation utility, including the exact parameters for the traffic generated, and the docker container instance counting script can be found in Appendix G.

Increasing and decreasing traffic

We explained our autoscaling engine balancer.py in a Section 4.5. It uses KPIs to decide whether an instance of a service should be added or removed. However, much like VMs, containers need some time to start up. Hence, we expect to see a difference between the number of planned instances (the number of containers determined optimal by balancer.py) and the number of deployed instances (the number of containers running on a host). Figure 5.1 shows the difference between the number of deployed instances and the number of instances determined as optimal for the working of the system by balancer.py. We can clearly see the delay the system has in adopting to the new demands. However, we also notice that these delays are very small, generally speaking. Figure 5.2 shows a detailed look at the reaction

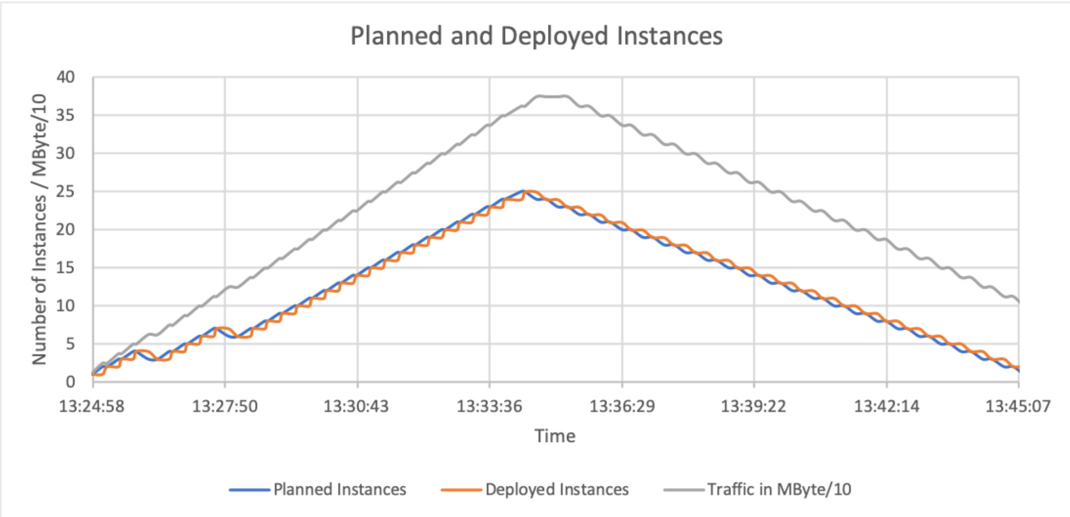

FIGURE 5.1: Planned and deployed Docker container instances

time for this scenario. Here we observe an average reaction time in between three and five seconds for starting a new container and an average time between eleven and thirteen seconds for shutting down an existing container. Both reaction times are much faster than booting an existing VM.

5.2 Startup Time and Elasticity

Before a docker container can be used, it has to be built.[79] Building the components of OAI EPC includes compilation of the different elements. Compilation time for the entire EPC takes well over an hour. However, each docker container only has to be built once. After building, several copies of the same container can be used without the need of rebuilding. Building a docker container can be accomplished well before the time of its actual use. It can be compared to compiling a binary file which can then be distributed to several systems. Therefore, we do not take build time into account when giving performance information about our setup. Because they are not part of OAI EPC, we will also ignore performance information with regards to cAdvisor, Prometheus and Graphana. They are running separately to EPC and, apart from collecting performance information, do not influence the running EPC.

When it comes to startup time, we have to differentiate between two variables: The startup time of the docker container and the time the program running inside the container is ready to be used. The second is determined by the start up time of the OAI EPC component running inside an individual container.. The first is extremely fast and can be measured using the series of commands shown in Listing 5.3.

```
# Provide an initial measurement and start containers
date --utc && docker-compose up
Sun Jul  7 07:33:58 UTC 2019
# Measure startup time of all containers
docker ps -q | xargs docker inspect \
            --format='{{.Name}}:{{.State.StartedAt}}'
spgw_1:  2019-07-07T07:34:11.205097303Z
mme_1:   2019-07-07T07:34:09.257947645Z
hss_1:   2019-07-07T07:34:07.933439931Z
db_1:    2019-07-07T07:34:01.894049977Z
phpmyadmin_1:    2019-07-07T07:34:06.206968141Z
```

LISTING 5.3: Measuring startup time for docker containers.

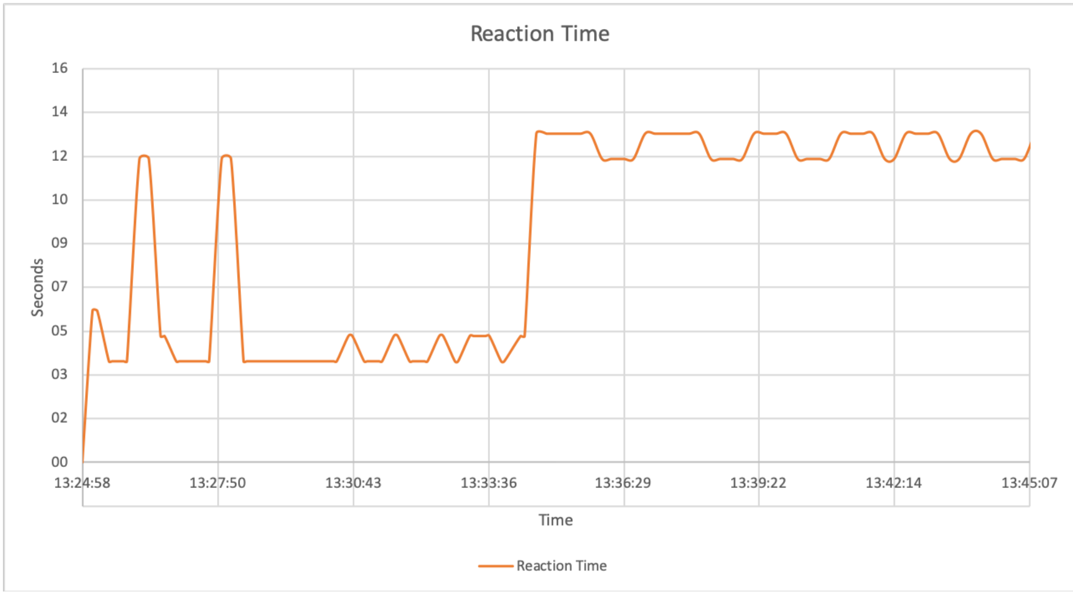

FIGURE 5.2: Reaction time until the number of planned instances is reached

As we can see from these results, startup time for all containers is within well under 30 seconds. Average startup time for Linux VMs has been estimated at over 90 seconds.[80] As expected, our containerized solution is outperforming VMs in terms of startup time by a significant amount. This advantage still persists when we take into account the following: Once a docker container is started, it does not mean that the program running inside the container is ready to be used. Our measurements show that it takes on average 40 seconds from the time the docker-compose command was executed to the time EPC starts accepting connections. Starting an individual instance of an EPC service like SPGW takes much less time. This number stayed consistent when running docker-compose on its own as shown in Listing 5.1 as well as inside our load balancing utility as shown in Listing **??**. Therefore, we are still outperforming a setup using a VM. Before the 90 seconds it takes for the VM to boot completely, we have already established a fully functional EPC.

Furthermore, we can work with the short startup time of Docker containers for scaling the EPC. We can manipulate allocated resources by adding or removing instances of a certain part of the EPC. This makes elasticity for EPC within containers a much more feasible approach then accepting the large startup overhead added by a VM. Our load balancing utility performs as expected, with startup and shutdown times of SPGW containers well within the 30 second mark.

5.3 Performance of balancer.py

As shown in our experiments in Section 5.1.3, reaction time for our balancer.py is lower than the startup time for an entire EPC, beating VM startup time by a large margin. These results stay consistent even with a large number of containers and were independent of the overall traffic load on the system. Shut down time for individual containers was slightly higher than start up time, but stayed consistent as well. These results show the great potential container technologies such as Docker bring to infrastructure projects.

The overall performance of the scaling algorithm could certainly be improved. However, as previously mentioned in Section 3.2, scaling algorithms are a complex undertaking. They are not the main focus of this thesis.

5.4 Issues Encountered

We faced several issues during our work with the OAI EPC. Some of them were related to design decisions and technologies used by OAI, others to our setup. The first major roadblock was the use of SCTP by OAI EPC. We first encountered the issue when working on setting up Kubernetes services for the individual EPC components. Although some of the EPC components, such as HSS, have settings to disable the use of SCTP, they will still attempt to set up these connections and crash, if they do not succeed. The relevant settings can be found on the OAI gitlab repository.[8] OAI EPC uses freeDiameter[50] to control SCTP traffic which is needed for the S1 interface between MME and eNB. Both freeDiameter and OAI EPC can be compiled without SCTP and the setting can be disabled in the OAI configuration.[50] Nevertheless, this still leads to a crashing HSS. OAI does not respect this setting and still required SCTP to be enabled. By the time of this writing, Kubernetes does not support SCTP yet.[49] More details about the relationship between SCTP and Kubernetes can be found in section 4.4. Patching OAI to work without SCTP is out of scope of this thesis. We, therefore, had to opt for another container orchestration tool and chose docker-compose. It is not only popular with the open-source community, but it can also be integrated easily with docker-swarm.

Contrary to Kubernetes, docker-compose does not block SCTP traffic between different containers. However, we faced other issues: When using multiple instances of a service, docker-compose will load-balance them based on the service name.[71] Unfortunately, OAI EPC mostly uses IP addresses and not DNS names. MME for instance requires an IP address for the location of SPGW, making it impossible to use the Docker DNS resolution.[15] We showed our workaround in Chapter 4.8. However, this proved only successful in balancing the database back end for HSS as well as the HSS itself. SPGW could not be properly balanced because it requires binding to a specific IP address - 192.168.142.30 in our case – and not to 0.0.0.0 or localhost. Attempts at patching the appropriate code in EPC did not resolve the issue. This prevents the initiation of multiple SPGW container instances as they cannot bind to the same IP address/port simultaneously.

Nginx, the load balancing server of our choice also prevented us from scaling MME. This was due to an issue with SCTP and nginx. The issue was very likely related to a problem with OAI EPC and Linux Kernel 4.13. However, we decided to not pursue the issue any further. Because of the design choices made by OAI, we ended up with an only partial elastic EPC. Nevertheless, we now have build instruction for docker-compose to build a fully functional EPC with some elastic parts that can be scaled out/in.

This brings us to the next issue we encountered. The different tools provided by OAI can be used in several combinations. The most common scenario is using OAI EPC and eNB with physical USRP and UE.[81] However, RAN and UE can also be virtualized using oaisim.[82] This is very convenient as it eliminates the need for an extensive and costly physical setup. Basic testing using ping or curl works well. Unfortunately, oaisim crashed in our testing as soon as the iperf3 client established a connection with a public iperf3 server. This scenario repeated with TCP and with UDP connections as well as different bandwidth options. We did our best generating load using different tools as alternatives to iperf3, including scapy, a build tool for custom pakets,[83] and curl but cannot provide any significant results due to oaisim being unreliable.

Chapter 6

Conclusion and Outlook

6.1 Conclusion

In this thesis, we provide a containerized implementation of OAI EPC that is ETSI MANO compliant. We complete this implementation with state of the art performance measuring on the container level and provide a scaling logic by leveraging the API provided by docker-compose. We can query standardized performance metrics and react to them in an automated fashion. Every element of our implementation is built using Docker and docker-compose making it easy to reproduce our setup on an appropriate system. It is also worth nothing that we haven't come across a publication were EPC performance is accessed using cAdvisor and Prometheus. Hence, to the best of our knowledge, we are the first to implement such a monitoring stack for EPC.

A single instance of OAI EPC is containerized in several Docker container and can be deployed. We encountered several issues trying to make the EPC fully elastic. What we mean by that is updating the existing EPC in a way that every one of its individual components can be scaled independently. We realised soon that such an undertaking was out of scope for a bachelor thesis. The reason for this are in part the limitations of existing container orchestration solutions, which do not fully support certain protocols and/or IP based scaling of services, and in part the design decisions made by the developers of OAI EPC. We developed a custom scaling solution to work around some of these issues. We also settled on scaling an individual element of the EPC: the SPGW.

6.2 Future Work

The issues we encountered by building our implementation show that a lot of work still needs to be done by all players involved to seamlessly integrate NFV with container orchestration applications. NFV required for EPC is often structured in a way that prohibits using the more powerful features of container networking. In particular the reliance on IP addresses instead of DNS names poses an obstacle.

NFV still has the potential to offering enormous benefits to networking providers. However, we are convinced that many of the issues we faced in our work could be avoided by first focusing on the edge of the network or on individual network functions not directly related to EPC. These can be integrated into the container networking logic.

Our current EPC implementation is not fully elastic. Making it possible to run several MME and SPGW instances in parallel would be an interesting follow up. The same can be said for patching oaisim. Another area to focus on could be the container orchestration tool of choice. We are currently using docker-compose and a single machine for our test setup. This includes oaisim, EPC and AE. Scaling EPC to a cluster using docker-swarm or another orchestration tool would provide a valuable follow-up to our thesis.

We only use a limited set of KPIs and a straight forward scaling logic. We suggest exploring additional KPIs such as connection time between MME and UE as well as more powerful scaling algorithms in future work. Our balancing tool is very flexible and a more complex scaling logic could easily be introduced.

Appendix A

Autoscaling Engine

A.1 Implementation - balancer.py

```python
#!/usr/bin/python3

import requests
import subprocess
import json
import time
import sys
import os
import configparser
import logging
import datetime

# For more information about prometheus metrices see:
# https://github.com/dashpole/cadvisor/blob/1
    dcd0cee2b05590a8b5515f5c41a80905d2fc1c2/metrics/prometheus.go

_ROOT = os.path.abspath(os.path.dirname(__file__))
CFG = configparser.ConfigParser()
CFG.read(os.path.join(_ROOT, "balancer.cfg"))
verbosity = 0

logger = logging.getLogger("balancer")
ch = logging.StreamHandler()
logger.addHandler(ch)
logger.setLevel(logging.NOTSET)
if verbosity == 0:
    logger.setLevel(logging.ERROR)
if verbosity == 1:
    logger.setLevel(logging.WARNING)
if verbosity == 2:
    logger.setLevel(logging.INFO)
if verbosity == 3:
    logger.setLevel(logging.DEBUG)


def result_to_int(result_str):
    """Parse result received by prometheus query and return rounded value
    as int or 0."""
    if result_str:
        result_int = int(result_str.split(".")[0])
        return result_int
    return 0


def send_request(query_string, prometheus_url):
```

```python
    """Send a request to the Prometheus API and return the response or None."""
    try:
        response = requests.get(url=prometheus_url, params={"query": query_string})
        logger.debug(
            "Response HTTP Status Code: {status_code}".format(
                status_code=response.status_code
            )
        )
        logger.debug(
            "Response HTTP Response Body: {content}".format(content=response.content)
        )
    except requests.exceptions.RequestException:
        logger.warning("HTTP Request failed")
        raise

    # https://stackoverflow.com/questions/40059654/python-convert-a-bytes-array-into-json-format
    bytes_value = response.content
    bytes_to_str = bytes_value.decode("utf8").replace("'", '"')
    response_parsed = json.loads(bytes_to_str)
    try:
        result = response_parsed["data"]["result"][0]["value"][1]
    except IndexError:
        # Prometheus returned empty response for this query
        result = None

    return result


def get_traffic_received(image, interval, prometheus_url):
    """Query Prometheus API for network traffic information.
    Return received and sent traffic size for a given interval or None.

    eg: sum(rate(container_network_receive_bytes_total{image="networkstatic/iperf3"}[10m]))
    """
    query_string = 'sum(rate(container_network_receive_bytes_total{{image="{}"}}[{}]))'.format(
        image, interval
    )
    result = send_request(query_string, prometheus_url)
    result = result_to_int(result)
    logger.info("Traffic received for image {}: {}.".format(image, result))
    return result


def get_traffic_sent(image, interval, prometheus_url):
    """Query Prometheus API for network traffic information.
    Return received and sent traffic size for a given interval or None.

    Eg.: sum(rate(container_network_transmit_bytes_total{image="networkstatic/iperf3"}[10m]))
    """
    query_string = 'sum(rate(container_network_transmit_bytes_total{{image="{}"}}[{}]))'.format(
        image, interval
    )
    result = send_request(query_string, prometheus_url)
    result = result_to_int(result)
```

```
    logger.info("Traffic sent for image {}: {}.".format(image, result))
    return result


def get_cpu_usage(image, interval, prometheus_url):
    """Query Prometheus API for cpu usage information.
    Return usage for a given interval or None.

    Eg.: sum(rate(process_cpu_seconds_total{image="networkstatic/iperf3"}[10m]))
    """
    query_string = 'sum(rate(process_cpu_seconds_total{{image="{}"}}[{}]))'.format(
        image, interval
    )
    result = send_request(query_string, prometheus_url)
    result = result_to_int(result)
    logger.info("CPU usage for image {}: {}.".format(image, result))
    return result


def get_memory_usage(image, interval, prometheus_url):
    """Query Prometheus API for memory usage information.
    Return usage for a given interval or None.

    Eg.: sum(rate(container_memory_usage_bytes{image="networkstatic/iperf3"}[10m]))
    """
    query_string = 'sum(rate(container_memory_usage_bytes{{image="{}"}}[{}]))'.format(
        image, interval
    )
    result = send_request(query_string, prometheus_url)
    result = result_to_int(result)
    logger.info("Memory usage for image {}: {}.".format(image, result))
    return result


def get_scale_to(
    section, traffic_received, traffic_sent, memory_used, cpu_used, size_current
):
    """Parse traffic, cpu and memory usage depending on the number of instances
    (size) and return a touple if scaling should occure and if so to what size.
    """

    scaling_should_occure = False
    size_difference = 0

    traffic_received = traffic_received / size_current
    # print("Traffic per instance: {}".format(traffic_received))
    traffic_sent = traffic_sent / size_current
    memory_used = memory_used / size_current
    cpu_used = cpu_used / size_current

    # Scale out
    if traffic_received > int(CFG[section]["traffic_received_limit_upper"]):
        size_difference += 1
#     if traffic_sent > int(CFG[section]["traffic_sent_limit_upper"]):
#         size_difference += 1
```

```python
#     if memory_used > int(CFG[section]["memory_used_limit_upper"]):
#         size_difference += 1
#     if cpu_used > int(CFG[section]["cpu_used_limit_upper"]):
#         size_difference += 1

    # Scale in
    if traffic_received < int(CFG[section]["traffic_received_limit_lower"]):
        size_difference -= 1
#     if traffic_sent < int(CFG[section]["traffic_sent_limit_lower"]):
#         size_difference -= 1
#     if memory_used < int(CFG[section]["memory_used_limit_lower"]):
#         size_difference -= 1
#     if cpu_used < int(CFG[section]["cpu_used_limit_lower"]):
#         size_difference -= 1

    size_ideal = size_current + size_difference
#     logger.warning(
#         "size_ideal: {}\nsize_current: {}\nsize_difference: {}".format(size_ideal, size_current, size_difference)
#     )
    if size_ideal > int(CFG[section]["size_max"]):
        size_ideal = int(CFG[section]["size_max"])
    if size_ideal < int(CFG[section]["size_min"]):
        size_ideal = int(CFG[section]["size_min"])
    if size_ideal != size_current:
        scaling_should_occure = True
    if int(CFG[section]["scale_service"]) == 0:
        scaling_should_occure = False

    logger.info("Ideal size for {} is {}.".format(section, size_ideal))
    return scaling_should_occure, size_ideal


def scale_service(service, size_ideal):
    """Call docker-compose to scale the service in
    question to the appropriate size.
    Return docker-compose exit code.
    """
    cmd = [
        "docker-compose",
        "up",
        "--detach",
        "--scale",
        "{}={}".format(service, size_ideal),
    ]
    logger.debug(cmd)
    #print("Scaling to {} instances now: {:%H:%M:%S}".format(size_ideal, datetime.datetime.now()))
    print("{} {:%H:%M:%S}".format(size_ideal, datetime.datetime.now()))
    proc = subprocess.Popen(
        cmd, bufsize=-1, stdout=subprocess.PIPE, stderr=subprocess.PIPE
    )
    out, err = proc.communicate()
    rc = proc.returncode
    if rc != 0:
        logger.warning(
            "Error while scaling service {} to size {}.".format(service, size_ideal)
        )
        stop_epc()
        sys.exit(err.decode("utf8"))
    return rc
```

```python


def start_epc():
    """Run docker-compose in detach mode to start
    the dockerized EPC.
    Must be started with the root of our EPC repo
    as working directory in order for the docker-
    compose file to be found.
    """
    cmd = ["docker-compose", "up", "--detach"]
    proc = subprocess.Popen(
        cmd, bufsize=-1, stdout=subprocess.PIPE, stderr=subprocess.PIPE
    )
    out, err = proc.communicate()
    rc = proc.returncode
    if rc != 0:
        logger.warning(
            "Error while starting docker containers. Stopping running instances"
        )
        stop_epc()
        sys.exit(err.decode("utf8"))
    logger.warning("Container(s) started")
    logger.warning(err.decode("utf8"))
    return rc


def stop_epc():
    """Run docker-compose to stop funning containers.
    Same restrictions as for start_epc apply."""
    proc = subprocess.Popen(
        ["docker-compose", "stop"],
        bufsize=-1,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
    )
    out, err = proc.communicate()
    rc = proc.returncode
    if rc != 0:
        logger.warning("Error while shutting down instances.")
        sys.exit(err.decode("utf8"))
    logger.warning("Container(s) stopped")
    logger.warning(err.decode("utf8"))
    return rc


def reload_loadbalancer(container_name):
    """Reload nginx inside docker container. Return exit code of subprocess."""
    cmd = ["docker", "container", "exec", container_name, "nginx", "-s", "reload"]
    proc = subprocess.Popen(
        cmd, bufsize=-1, stdout=subprocess.PIPE, stderr=subprocess.PIPE
    )
    out, err = proc.communicate()
    rc = proc.returncode
    if rc != 0:
        logger.warning("Error reloading nginx.")
    return rc


def balance(section):
    """Run our balancer logic.
```

```

    Some information with regards to the output:
    * Traffic information is in bytes
    """
    service = CFG[section]["service"]
    logger.info("Balance service {}".format(service))
    traffic_received = get_traffic_received(
        CFG[section]["docker_image"],
        CFG[section]["interval"],
        CFG[section]["prometheus_url"],
    )
    traffic_sent = get_traffic_sent(
        CFG[section]["docker_image"],
        CFG[section]["interval"],
        CFG[section]["prometheus_url"],
    )
    memory_used = get_memory_usage(
        CFG[section]["docker_image"],
        CFG[section]["interval"],
        CFG[section]["prometheus_url"],
    )
    cpu_used = get_cpu_usage(
        CFG[section]["docker_image"],
        CFG[section]["interval"],
        CFG[section]["prometheus_url"],
    )
    scaling_should_occure, size_ideal = get_scale_to(
        section,
        traffic_received,
        traffic_sent,
        memory_used,
        cpu_used,
        int(CFG[section]["size_current"]),
    )

    # Update cooldown timer
    timer = int(CFG[section]["cool_down_timer"])
    if timer > 0:
        timer -= 1
        CFG[section]["cool_down_timer"] = str(timer)

    if scaling_should_occure:
        if timer > 0:
            logger.debug("Timer is {}. Not scaling.".format(timer))
            return str(int(CFG[section]["size_current"]))

        logger.info(
            "Ideal size is {}, current is {}. Scaling service {}.".format(
                size_ideal, int(CFG[section]["size_current"]), service
            )
        )
        rc = scale_service(service, size_ideal)
        if rc != 0:
            logger.warning(
                "Error: Could not scale service {} to size {}.".format(
                    service, size_ideal
                )
            )
            return str(int(CFG[section]["size_current"]))

        CFG[section]["cool_down_timer"] = CFG[section]["cool_down_timer_max"]
        CFG[section]["size_current"] = str(size_ideal)
```

```
        logger.info(
            "Service {} scaled to {}.".format(
                service, int(CFG[section]["size_current"])
            )
        )
        reload_loadbalancer(CFG[section]["load_balancer"])
    return str(int(CFG[section]["size_current"]))


def start_traffic_generation(mode_option):
    """Start iperf3 traffic generation with a given mode as argument.
    Possible options for mode are:
    * full: Maximum throughput for a given time
    * raise: Raise traffic in a determined interval
    * fall: Start from high traffic and go down
    * static: Raise, stay at a level and don't change
    * fluctuate: Fluctuate around a given value to test balancing around a
     threashold
    """
    cmd = [
        "docker-compose",
        "run",
        "--detach",
        "--rm",
        "iperf3_client",
        "/code/traffictest.sh",
        "{}".format(mode_option),
    ]
    logger.debug(cmd)
    proc = subprocess.Popen(
        cmd, bufsize=-1, stdout=subprocess.PIPE, stderr=subprocess.PIPE
    )
    out, err = proc.communicate()
    rc = proc.returncode
    return rc

if __name__ == "__main__":
    """Start dockerized EPC and continue to
    query the prometheus API
    """
    try:
        start_epc()
        time.sleep(int(CFG[CFG.sections()[0]]["wait_time"]))
        start_traffic_generation("raise_and_fall")
        while True:
            time.sleep(int(CFG[CFG.sections()[0]]["wait_time"]))
            for section in CFG.sections():
                CFG[section]["size_current"] = balance(section)
    except KeyboardInterrupt:
        logger.warning("\nShutting down...")
        stop_epc()
    except:
        stop_epc()
        raise
```

A.2 Configuration File - balancer.cfg

```
[DEFAULT]
prometheus_url: http://localhost:9090/api/v1/query
interval: 5s
size_max: 30
size_min: 1
```

```
size_current: 1
wait_time: 5

cool_down_timer = 0
cool_down_timer_max = 3
# 5 MB
traffic_received_limit_upper = 4000000
# 2.5 MB
traffic_received_limit_lower = 2000000
traffic_sent_limit_upper = 5000000
traffic_sent_limit_lower = 2500000
memory_used_limit_upper = 55040
memory_used_limit_lower = 27520
cpu_used_limit_upper = 500
cpu_used_limit_lower = 0

# Set to 1 to enable scaling for this service
scale_service = 0

# Workaround  to introduce traffic to the system and start new instances
# of  spgw
[spgw]
service = spgw2
docker_image = networkstatic/iperf3
load_balancer = n/a
scale_service = 1

# [hss]
# service: hss
# docker_image: docker-openairinterface-epc_backend_hss
# load_balancer: docker-openairinterface-epc_hss_1
# scale_service = 1
#
# [mme]
# service: mme
# docker_image: docker-openairinterface-epc_backend_mme
# load_balancer: n/a
#
# [spgw]
# service: spgw
# docker_image: docker-openairinterface-epc_backend_spgw
# load_balancer: n/a
#
```

Appendix B

Docker EPC Containers

The code for the epc containers is based on the work done by Grunenberger.[68]

B.1 Docker-compose.yml

```
version: '2'

services:
    ######################
    # Monitring Services #
    ######################
    cadvisor:
      image: google/cadvisor:latest
      container_name: oai_cadvisor
      ports:
          - 8080:8080
      volumes:
          - /:/rootfs:ro
          - /var/run:/var/run:rw
          - /sys:/sys:ro
          - /var/lib/docker/:/var/lib/docker:ro
          - /dev/disk/:/dev/disk:ro
      networks:
          - monitoring
    prometheus:
      image: prom/prometheus
      container_name: oai_prometheus
      ports:
          - 9090:9090
      volumes:
          - ./prometheus.yml:/etc/prometheus/prometheus.yml
      networks:
          - monitoring
    grafana:
      image: grafana/grafana
      container_name: oai_grafana
      ports:
          - 3000:3000
      networks:
          - monitoring

    #####################
    #  Traffic   Generator #
    #####################
    iperf3_server:
      image: networkstatic/iperf3
      container_name: oai_iperf3_server
      ports:
```

```
        - 5201:5201
    command: ["-s"]
    networks:
      default:
  iperf3_client:
    depends_on:
        - iperf3_server
    build: iperf3_client
    container_name: oai_iperf3_client
    command: /code/traffictest.sh
    volumes:
     - /tmp/iperf_status:/tmp/iperf_status
     - ./iperf3_client/traffictest.sh:/code/traffictest.sh
    networks:
      default:

  ###############
  # EPC Network #
  ###############

  ######
  # DB #
  ######
  db:
    build: db
    environment:
        - MYSQL_ROOT_PASSWORD=linux
    networks:
      default:
    volumes:
     - ./db/oai_db.sql:/docker-entrypoint-initdb.d/oai_db.sql

  balance_db:
      image: nginx
      networks:
          default:
      ports:
          - 3306:3306
      volumes:
          - ./balance/balance_db.conf:/etc/nginx/nginx.conf
      depends_on:
          - "db"

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    links:
     - db:db
    ports:
        - 8088:80
    environment:
        - MYSQL_ROOT_PASSWORD=linux
        - MYSQL_USER=root
        - MYSQL_PASSWORD=linux
  #######
  # HSS #
  #######
  backend_hss:
    build: hss
    volumes:
     - ./hss/hss.conf:/usr/local/etc/oai/hss.conf:ro
    depends_on:
        - "balance_db"

```

```
  hss:
    image: nginx
    ports:
        - 3868
    volumes:
        - ./balance/balance_hss.conf:/etc/nginx/nginx.conf
    depends_on:
        - "backend_hss"
    networks:
      default:
      epc:
        ipv4_address: 192.168.142.10
    domainname: openair4G.eur
    hostname: hss

  #######
  # MME #
  #######
  mme:
    build: mme
    volumes:
     - ./mme/mme.conf:/usr/local/etc/oai/mme.conf:ro
    depends_on:
        - "hss"
    networks:
      default:
      epc:
        ipv4_address: 192.168.142.20
    domainname: openair4G.eur
    hostname: mme

#   balance_mme:
#       image: nginx
#       # ports:
#           # - 2123:2123
#       volumes:
#           - ./balance/balance_mme.conf:/etc/nginx/nginx.conf
#       depends_on:
#           - "mme"
  ########
  # SPGW #
  ########
  spgw:
    build: spgw
    privileged: true
    volumes:
     - /lib/modules:/lib/modules
     - ./spgw/spgw.conf:/usr/local/etc/oai/spgw.conf:ro
    depends_on:
        - "mme"
    networks:
      default:
      epc:
        ipv4_address: 192.168.142.30
    domainname: openair4G.eur
    hostname: spgw
    ports:
        - 2152
        - 2123

  spgw2:
    build: spgw
    privileged: true
```

```
      volumes:
       - /lib/modules:/lib/modules
       - ./spgw/spgw2.conf:/usr/local/etc/oai/spgw.conf:ro
      depends_on:
          - "mme"
      networks:
        default:
        epc:
      domainname: openair4G.eur
      hostname: spgw2
      ports:
          - 2152
          - 2123
#     spgw:
#       image: nginx
#       volumes:
#             - ./balance/balance_spgw.conf:/etc/nginx/nginx.conf
#       depends_on:
#             - "backend_spgw"

#############
# Networks #
############
networks:
  monitoring:
    driver: bridge
  epc:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 192.168.142.0/24
          gateway: 192.168.142.1
```

B.2 Docker file for DB

```
FROM mysql:5.6
MAINTAINER Yan Grunenberger <yan@grunenberger.net>

#### CUSTOMIZE YOUR FIRST SIM DETAILS
ARG IMSI='901550000000000'
ARG MSISDN='6789'
ARG KI=0x912e7221941577df083e1591d35f4c42
ARG OPC=0x4487d12562bd21df3b076852f4d74eec
ARG APN='internet'

#### CUSTOMIZE YOUR MME DETAILS
ARG REALM='openair4G.eur'
ARG MME='mme.openair4G.eur'

############################################ Docker build instructions

ENV MYSQL_ROOT_PASSWORD=linux

#RUN apt-get update && apt-get -qy install curl
#RUN curl https://gitlab.eurecom.fr/oai/openair-cn/raw/develop/src/oai_hss
    /db/oai_db.sql -o  /docker-entrypoint-initdb.d/oai_db.sql
# ADD oai_db.sql  /docker-entrypoint-initdb.d/oai_db.sql


# Customize the SQL based on the arguments passed on build time

```

```
# MME settings
# RUN sed -i s/'mme.openair4G.eur'/$MME/g /docker-entrypoint-initdb.d/oai_db.sql
# RUN sed -i s/'openair4G.eur'/$REALM/g /docker-entrypoint-initdb.d/oai_db.sql

# SIM card record
# RUN sed -i s/'oai.ipv4'/$APN/g /docker-entrypoint-initdb.d/oai_db.sql

# RUN sed -i s/208920100001100/$IMSI/g /docker-entrypoint-initdb.d/oai_db.sql
# RUN sed -i s/33638020000/$MSISDN/g /docker-entrypoint-initdb.d/oai_db.sql
# RUN sed -i s/0x8baf473f2f8fd09487cccbd7097c6862/$KI/g /docker-entrypoint-initdb.d/oai_db.sql
# RUN sed -i s/0xe734f8734007d6c5ce7a0508809e7e9c/$OPC/g  /docker-entrypoint-initdb.d/oai_db.sql
```

B.3 DB SQL Dump

```
-- phpMyAdmin SQL Dump
-- version 4.8.0.1
-- https://www.phpmyadmin.net/
--
-- Host: db
-- Generation Time: Jun 23, 2018 at 01:09 PM
-- Server version: 5.6.40
-- PHP Version: 7.2.4

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `oai_db`
--
CREATE DATABASE oai_db;
USE oai_db;

-- --------------------------------------------------------

--
-- Table structure for table `apn`
--

CREATE TABLE `apn` (
  `id` int(11) NOT NULL,
  `apn-name` varchar(60) NOT NULL,
  `pdn-type` enum('IPv4','IPv6','IPv4v6','IPv4_or_IPv6') NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------


--
-- Table structure for table `mmeidentity`
```

```
--

CREATE TABLE `mmeidentity` (
  `idmmeidentity` int(11) NOT NULL,
  `mmehost` varchar(255) DEFAULT NULL,
  `mmerealm` varchar(200) DEFAULT NULL,
  `UE-Reachability` tinyint(1) NOT NULL COMMENT 'Indicates whether the MME
     supports UE Reachability Notifcation'
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `mmeidentity`
--

INSERT INTO `mmeidentity` (`idmmeidentity`, `mmehost`, `mmerealm`, `UE-
     Reachability`) VALUES
(1, 'mme.openair4G.eur', 'openair4G.eur', 0),
(46, 'mme.openair4G.eur.openair4G.eur', 'openair4G.eur', 0);

-- --------------------------------------------------------

--
-- Table structure for table `pdn`
--

CREATE TABLE `pdn` (
  `id` int(11) NOT NULL,
  `apn` varchar(60) NOT NULL,
  `pdn_type` enum('IPv4','IPv6','IPv4v6','IPv4_or_IPv6') NOT NULL DEFAULT
     'IPv4',
  `pdn_ipv4` varchar(15) DEFAULT '0.0.0.0',
  `pdn_ipv6` varchar(45) CHARACTER SET latin1 COLLATE latin1_general_ci
     DEFAULT '0:0:0:0:0:0:0:0',
  `aggregate_ambr_ul` int(10) UNSIGNED DEFAULT '50000000',
  `aggregate_ambr_dl` int(10) UNSIGNED DEFAULT '100000000',
  `pgw_id` int(11) NOT NULL,
  `users_imsi` varchar(15) NOT NULL,
  `qci` tinyint(3) UNSIGNED NOT NULL DEFAULT '9',
  `priority_level` tinyint(3) UNSIGNED NOT NULL DEFAULT '15',
  `pre_emp_cap` enum('ENABLED','DISABLED') DEFAULT 'DISABLED',
  `pre_emp_vul` enum('ENABLED','DISABLED') DEFAULT 'DISABLED',
  `LIPA-Permissions` enum('LIPA-prohibited','LIPA-only','LIPA-conditional
     ') NOT NULL DEFAULT 'LIPA-only'
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `pdn`
--

INSERT INTO `pdn` (`id`, `apn`, `pdn_type`, `pdn_ipv4`, `pdn_ipv6`, `
     aggregate_ambr_ul`, `aggregate_ambr_dl`, `pgw_id`, `users_imsi`, `qci`,
      `priority_level`, `pre_emp_cap`, `pre_emp_vul`, `LIPA-Permissions`)
     VALUES
(1, 'internet', 'IPv4', '0.0.0.0', '0:0:0:0:0:0:0:0', 50000000, 100000000,
     1, '901550000000000', 9, 15, 'DISABLED', 'ENABLED', 'LIPA-only'),
(60, 'internet', 'IPv4', '0.0.0.0', '0:0:0:0:0:0:0:0', 50000000,
     100000000, 1, '208930100001111', 9, 15, 'DISABLED', 'ENABLED', 'LIPA-
     only');

-- --------------------------------------------------------

--
-- Table structure for table `pgw`
```

```sql
--

CREATE TABLE `pgw` (
  `id` int(11) NOT NULL,
  `ipv4` varchar(15) NOT NULL,
  `ipv6` varchar(39) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `pgw`
--

INSERT INTO `pgw` (`id`, `ipv4`, `ipv6`) VALUES
(1, '127.0.0.1', '0:0:0:0:0:0:0:1');

-- --------------------------------------------------------

--
-- Table structure for table `terminal-info`
--

CREATE TABLE `terminal-info` (
  `imei` varchar(15) NOT NULL,
  `sv` varchar(2) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Table structure for table `users`
--

CREATE TABLE `users` (
  `imsi` varchar(15) NOT NULL COMMENT 'IMSI is the main reference key.',
  `msisdn` varchar(46) DEFAULT NULL COMMENT 'The basic MSISDN of the UE (Presence of MSISDN is optional).',
  `imei` varchar(15) DEFAULT NULL COMMENT 'International Mobile Equipment Identity',
  `imei_sv` varchar(2) DEFAULT NULL COMMENT 'International Mobile Equipment Identity Software Version Number',
  `ms_ps_status` enum('PURGED','NOT_PURGED') DEFAULT 'PURGED' COMMENT 'Indicates that ESM and EMM status are purged from MME',
  `rau_tau_timer` int(10) UNSIGNED DEFAULT '120',
  `ue_ambr_ul` bigint(20) UNSIGNED DEFAULT '50000000' COMMENT 'The Maximum Aggregated uplink MBRs to be shared across all Non-GBR bearers according to the subscription of the user.',
  `ue_ambr_dl` bigint(20) UNSIGNED DEFAULT '100000000' COMMENT 'The Maximum Aggregated downlink MBRs to be shared across all Non-GBR bearers according to the subscription of the user.',
  `access_restriction` int(10) UNSIGNED DEFAULT '60' COMMENT 'Indicates the access restriction subscription information. 3GPP TS.29272 #7.3.31',
  `mme_cap` int(10) UNSIGNED ZEROFILL DEFAULT NULL COMMENT 'Indicates the capabilities of the MME with respect to core functionality e.g. regional access restrictions.',
  `mmeidentity_idmmeidentity` int(11) NOT NULL DEFAULT '0',
  `key` varbinary(16) NOT NULL DEFAULT '0' COMMENT 'UE security key',
  `RFSP-Index` smallint(5) UNSIGNED NOT NULL DEFAULT '1' COMMENT 'An index to specific RRM configuration in the E-UTRAN. Possible values from 1 to 256',
  `urrp_mme` tinyint(1) NOT NULL DEFAULT '0' COMMENT 'UE Reachability Request Parameter indicating that UE activity notification from MME has been requested by the HSS.',
```

```
  `sqn` bigint(20) UNSIGNED ZEROFILL NOT NULL,
  `rand` varbinary(16) NOT NULL,
  `OPc` varbinary(16) DEFAULT NULL COMMENT 'Can be computed by HSS'
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `users`
--

INSERT INTO `users` (`imsi`, `msisdn`, `imei`, `imei_sv`, `ms_ps_status`,
    `rau_tau_timer`, `ue_ambr_ul`, `ue_ambr_dl`, `access_restriction`, `
    mme_cap`, `mmeidentity_idmmeidentity`, `key`, `RFSP-Index`, `urrp_mme`,
     `sqn`, `rand`, `OPc`) VALUES
('901550000000000', '6789', '35609204079200', NULL, 'PURGED', 120,
    50000000, 100000000, 47, 0000000000, 1, 0
    x912e7221941577df083e1591d35f4c42, 1, 0, 00000000000000000351, 0x00, 0
    x8caac204d4ff07140c23ea7f2e191e12),
('208930100001111', '6789', '356113022094149', NULL, 'NOT_PURGED', 120,
    50000000, 100000000, 47, 0000000000, 46, 0
    x8baf473f2f8fd09487cccbd7097c6862, 1, 0, 00000000000000001727, 0
    x90b69a4032fc642fc17bfd3462aed44d, 0xe734f8734007d6c5ce7a0508809e7e9c);

--
-- Indexes for dumped tables
--

--
-- Indexes for table `apn`
--
ALTER TABLE `apn`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `apn-name` (`apn-name`);

--
-- Indexes for table `mmeidentity`
--
ALTER TABLE `mmeidentity`
  ADD PRIMARY KEY (`idmmeidentity`);

--
-- Indexes for table `pdn`
--
ALTER TABLE `pdn`
  ADD PRIMARY KEY (`id`,`pgw_id`,`users_imsi`),
  ADD KEY `fk_pdn_pgw1_idx` (`pgw_id`),
  ADD KEY `fk_pdn_users1_idx` (`users_imsi`);

--
-- Indexes for table `pgw`
--
ALTER TABLE `pgw`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `ipv4` (`ipv4`),
  ADD UNIQUE KEY `ipv6` (`ipv6`);

--
-- Indexes for table `terminal-info`
--
ALTER TABLE `terminal-info`
  ADD UNIQUE KEY `imei` (`imei`);

--
-- Indexes for table `users`
```

```
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`imsi`,`mmeidentity_idmmeidentity`),
  ADD KEY `fk_users_mmeidentity_idx1` (`mmeidentity_idmmeidentity`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `apn`
--
ALTER TABLE `apn`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `mmeidentity`
--
ALTER TABLE `mmeidentity`
  MODIFY `idmmeidentity` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT
    =47;

--
-- AUTO_INCREMENT for table `pdn`
--
ALTER TABLE `pdn`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=61;

--
-- AUTO_INCREMENT for table `pgw`
--
ALTER TABLE `pgw`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

B.4 Docker file for HSS

```
FROM ubuntu:16.04
MAINTAINER Yan Grunenberger <yan@grunenberger.net>

### dependencies and downloads

ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update
RUN apt-get -yq dist-upgrade

# General utilities
RUN apt-get -qy install git wget apt-utils autoconf  \
 automake  \
 bison     \
 build-essential \
 cmake \
 cmake-curses-gui  \
 doxygen \
 doxygen-gui\
 debhelper \
 flex  \
 gdb \
```

```
pkg-config \
subversion \
libconfig8-dev \
libgcrypt-dev \
libidn2-0-dev \
libpq-dev \
libidn11-dev \
libmysqlclient-dev \
libpthread-stubs0-dev \
libsctp1 \
libsctp-dev \
libxml2-dev \
libssl-dev \
libtool \
libgmp-dev \
libtasn1-6-dev \
libp11-kit-dev \
libtspi-dev \
libtspi1 \
libidn2-0-dev \
libidn11-dev \
openssl \
mercurial \
python-dev \
ssl-cert \
swig


WORKDIR /root
RUN wget https://ftp.gnu.org/gnu/nettle/nettle-2.5.tar.gz
RUN tar -xzf nettle-2.5.tar.gz

WORKDIR /root
RUN wget http://mirrors.dotsrc.org/gcrypt/gnutls/v3.1/gnutls-3.1.23.tar.xz
RUN tar -xJf gnutls-3.1.23.tar.xz

WORKDIR /root
RUN git clone https://gitlab.eurecom.fr/oai/freediameter.git -b eurecom
    -1.2.0

# other mirror : ftp://ftp.lysator.liu.se/pub/security/lsh/nettle-2.5.tar.
    gz
WORKDIR /root
RUN cd nettle-2.5/ && ./configure --disable-openssl --enable-shared --
    prefix=/usr && make -j`nproc` && make check && make install

WORKDIR /root
RUN cd gnutls-3.1.23/ && ./configure --prefix=/usr && make -j`nproc` &&
    make install

# Run freediameter (hard dependencies on gnutls)
WORKDIR /root/freediameter
RUN mkdir build && cd build && cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr ../
    && make -j`nproc` && make install

# cloning directory
WORKDIR /root
RUN mkdir .ssh
RUN ssh-keyscan github.com >> .ssh/known_hosts
COPY id_rsa .ssh/id_rsa
COPY id_rsa.pub .ssh/id_rsa.pub
RUN git clone git@github.com:aschwanb/openair-cn.git

```

```
###################### START OF CUSTOMIZATION

#### CUSTOMIZE YOUR DATABASE PARAMETERS
ARG MYSQLHOSTNAME=db.openair4G.eur
ARG MYSQLUSER=root
ARG MYSQLPASSWORD=linux
ARG MYSQLDATABASE=oai_db

#### CUSTOMIZE YOUR BUILD PARAMETER
ARG OAIBRANCH=develop

#### CUSTOMIZE YOUR OPERATOR KEY
ARG OPKEY=63bfa50ee6523365ff14c1f45f88737d

#### CUSTOMIZE YOUR HSS HOSTNAME (used in certificates)
ARG HSS_CN_NAME=hss.openair4G.eur

###################### END OF CUSTOMIZATION

# cloning directory
WORKDIR /root/openair-cn
RUN git checkout $OAIBRANCH

#  install_asn1c_from_source
#WORKDIR /root
#RUN apt-get -qy install autoconf automake bison build-essential flex gcc libtool
#RUN git clone https://gitlab.eurecom.fr/oai/asn1c.git
#RUN cd asn1c && ./configure && make && make install

# compiling OAI HSS executable oai_hss
WORKDIR /root/openair-cn/build/hss
RUN mkdir build
WORKDIR /root/openair-cn/build/hss/build
RUN cmake -DOPENAIRCN_DIR=/root/openair-cn ../
RUN make -j `nproc`

RUN mkdir -p /usr/local/etc/oai/freeDiameter
# RUN cp /root/openair-cn/etc/hss.conf /usr/local/etc/oai/
RUN cp /root/openair-cn/etc/hss_fd.conf /usr/local/etc/oai/freeDiameter/
RUN cp /root/openair-cn/etc/acl.conf /usr/local/etc/oai/freeDiameter/

ENV MYSQLUSER=root
ENV MYSQLPASSWORD=linux
ENV MYSQLDATABASE=oai_db
ENV MYSQLHOSTNAME=db.openair4G.eur


ENV OPKEY=63bfa50ee6523365ff14c1f45f88737d

ENV HSS_CN_NAME=hss.openair4G.eur
#ready to work
WORKDIR /root
COPY start.sh /root/start.sh
RUN chmod +x /root/start.sh
ENTRYPOINT "/root/start.sh"
```

B.5 HSS configuration

```
################################################################################
```

```
# Licensed to the OpenAirInterface (OAI) Software Alliance under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The OpenAirInterface Software Alliance licenses this file to You under
# the Apache License, Version 2.0  (the "License"); you may not use this file
# except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------------------------------
# For more information about the OpenAirInterface (OAI) Software Alliance:
#      contact@openairinterface.org
################################################################################
HSS :
{
## MySQL mandatory options
MYSQL_server = "balance_db";     # HSS S6a bind address
MYSQL_user   = "root";  # Database server login
MYSQL_pass   = "linux";  # Database server password
MYSQL_db     = "oai_db";        # Your database name

## HSS options
# OPERATOR_key = "63bfa50ee6523365ff14c1f45f88737d"; # OP key matching your database
OPERATOR_key = "1006020f0a478bf6b699f15c062e42b3"; # OP key matching your database
#OPERATOR_key = "11111111111111111111111111111111"; # OP key matching your database

RANDOM = "true";                                    # True random or only pseudo random (for subscriber vector generation)

## Freediameter options
FD_conf = "/usr/local/etc/oai/freeDiameter/hss_fd.conf";
};
```

B.6 HSS start script

```
#!/bin/bash
# MySQL database configuration
sed -i "s/@MYSQL_user@/$MYSQLUSER/g" /usr/local/etc/oai/hss.conf
sed -i "s/@MYSQL_pass@/$MYSQLPASSWORD/g" /usr/local/etc/oai/hss.conf
sed -i "s/127.0.0.1/$MYSQLHOSTNAME/g" /usr/local/etc/oai/hss.conf
sed -i "s/oai_db/$MYSQLDATABASE/g" /usr/local/etc/oai/hss.conf
sed -i "s/db.openair4G.eur/$MYSQLHOSTNAME/g" /usr/local/etc/oai/hss.conf

# Operator key (OP)
sed -i "s/1006020f0a478bf6b699f15c062e42b3/$OPKEY/g" /usr/local/etc/oai/hss.conf

# HSS Configuration
sed -i "s/hss.openair4G.eur/$HSS_CN_NAME/g" /usr/local/etc/oai/hss.conf
```

```
sed -i "s/hss.openair4G.eur/$HSS_CN_NAME/g" /usr/local/etc/oai/
    freeDiameter/hss_fd.conf
sed -i "s/hss.openair4G.eur/$HSS_CN_NAME/g" /usr/local/etc/oai/
    freeDiameter/acl.conf

# Generation of certificate for diameter
cd /root
if [[ ! -d /root/demoCA ]];then
  mkdir demoCA && touch demoCA/index.txt && echo 01 > demoCA/serial
  openssl req  -new -batch -x509 -days 3650 -nodes -newkey rsa:1024 -out
    hss.cacert.pem -keyout hss.cakey.pem -subj /CN=$HSS_CN_NAME/C=FR/ST=
    PACA/L=Aix/O=Eurecom/OU=CM
  openssl genrsa -out hss.key.pem 1024
  openssl req -new -batch -out hss.csr.pem -key hss.key.pem -subj /CN=
    $HSS_CN_NAME/C=FR/ST=PACA/L=Aix/O=Eurecom/OU=CM
  openssl ca -cert hss.cacert.pem -keyfile hss.cakey.pem -in hss.csr.pem -
    out hss.cert.pem -outdir . -batch
  mv /root/hss.cakey.pem /usr/local/etc/oai/freeDiameter/
  mv /root/hss.cert.pem /usr/local/etc/oai/freeDiameter/
  mv /root/hss.cacert.pem /usr/local/etc/oai/freeDiameter/
  mv /root/hss.key.pem /usr/local/etc/oai/freeDiameter/
fi

# Start hss
sleep 15 && /root/openair-cn/build/hss/build/oai_hss
```

B.7 Docker file for MME

```
FROM ubuntu:16.04
MAINTAINER Yan Grunenberger <yan@grunenberger.net>
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update
RUN apt-get -yq dist-upgrade

# General utilities
RUN apt-get -qy install git wget apt-utils

# cloning directory
WORKDIR /root
RUN mkdir .ssh
RUN ssh-keyscan github.com >> .ssh/known_hosts
COPY id_rsa .ssh/id_rsa
COPY id_rsa.pub .ssh/id_rsa.pub
RUN git clone git@github.com:aschwanb/openair-cn.git

WORKDIR /root/openair-cn
RUN git checkout develop

WORKDIR /root

# Fixing default mysql root password to "linux". This is the default
    assumed by OAI building scripts
RUN echo 'mysql-server mysql-server/root_password password linux' |
    debconf-set-selections
RUN echo 'mysql-server mysql-server/root_password_again password linux' |
    debconf-set-selections
RUN echo 'phpmyadmin phpmyadmin/dbconfig-install boolean true' | debconf-
    set-selections
RUN echo 'phpmyadmin phpmyadmin/app-password-confirm password linux ' |
    debconf-set-selections
RUN echo 'phpmyadmin phpmyadmin/mysql/admin-pass password linux' | debconf
    -set-selections
```

```
RUN echo 'phpmyadmin phpmyadmin/mysql/app-pass password linux' | debconf-set-selections
RUN echo 'phpmyadmin phpmyadmin/reconfigure-webserver multiselect apache2' | debconf-set-selections

# (Build script - MME dependencies...)
# (from build_helper) Remove incompatible softwares
RUN apt-get -qy --purge remove libgnutls-dev \
 'libgnutlsxx2?' \
 nettle-dev \
 nettle-bin

# (from build_helper) Compilers, Generators
RUN apt-get -qy install autoconf \
 automake \
 bison \
 build-essential \
 cmake \
 cmake-curses-gui \
 doxygen \
 doxygen-gui\
 flex \
 gdb \
 pkg-config

# (from build_helper) git/svn
RUN apt-get -qy install git \
 subversion

# (from build_helper) librairies
RUN apt-get -qy install libconfig8-dev \
 libgcrypt11-dev \
 libidn2-0-dev \
 libidn11-dev \
 libmysqlclient-dev \
 libpthread-stubs0-dev \
 libsctp1 \
 libsctp-dev \
 libssl-dev \
 libtool \
 mysql-client \
 mysql-server \
 openssl

# (from build_helper) compile nettle from source
RUN apt-get -qy install \
 autoconf \
 automake \
 build-essential \
 libgmp-dev
WORKDIR /root
# other mirror : ftp://ftp.lysator.liu.se/pub/security/lsh/nettle-2.5.tar.gz
RUN wget https://ftp.gnu.org/gnu/nettle/nettle-2.5.tar.gz
RUN tar -xzf nettle-2.5.tar.gz
RUN cd nettle-2.5/ && ./configure --disable-openssl --enable-shared --prefix=/usr && make -j`nproc` && make check && make install

# (from build_helper) install_gnutls_from_source $1
WORKDIR /root
RUN apt-get -qy install \
 autoconf \
 automake \
```

```
 build-essential
#       libtasn1-6-dbg \  libp11-kit0-dbg \
RUN apt-get -qy install libtasn1-6-dev \
 libp11-kit-dev \
 libtspi-dev \
 libtspi1 \
 libidn2-0-dev \
 libidn11-dev
RUN wget http://mirrors.dotsrc.org/gcrypt/gnutls/v3.1/gnutls-3.1.23.tar.xz
RUN tar -xJf gnutls-3.1.23.tar.xz
RUN cd gnutls-3.1.23/ &&  ./configure --prefix=/usr && make -j`nproc` &&
    make install

# (from build_helper)
RUN apt-get -qy install autoconf  \
 automake  \
 bison     \
 build-essential \
 cmake \
 cmake-curses-gui  \
 debhelper \
 flex  \
 g++ \
 gcc \
 gdb \
 libgcrypt-dev \
 libidn11-dev \
 libmysqlclient-dev \
 libpq-dev \
 libsctp1 \
 libsctp-dev \
 libxml2-dev \
 mercurial \
 python-dev \
 ssl-cert \
 swig

# Run freediameter (hard dependencies on gnutls)
WORKDIR /root
RUN git clone https://gitlab.eurecom.fr/oai/freediameter.git -b eurecom
    -1.2.0
WORKDIR /root/freediameter
RUN mkdir build && cd build && cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr ../
    && make -j`nproc` && make install

# PHPmyadmin for the MME database management
WORKDIR /root
RUN apt-get -qy install phpmyadmin \
 python-pexpect \
 php \
 libapache2-mod-php

RUN apt-get -qy install check \
 phpmyadmin \
 python-dev \
 python-pexpect \
 unzip

#  install_asn1c_from_source
WORKDIR /root
RUN apt-get -qy install autoconf automake bison build-essential flex gcc
    libtool
RUN git clone https://gitlab.eurecom.fr/oai/asn1c.git
```

```
RUN cd asn1c && ./configure && make && make install

#   install_libgtpnl_from_source

WORKDIR /root
RUN apt-get -qy install     autoconf   \
    automake   \
    build-essential \
    libmnl-dev

RUN  git clone git://git.osmocom.org/libgtpnl
RUN cd libgtpnl && autoreconf -fi && ./configure && make -j`nproc` && make install && ldconfig

RUN apt-get -qy install ethtool \
    iproute \
    vlan \
    tshark

# compiling OAI mme executable
WORKDIR /root/openair-cn/build/mme
RUN cp CMakeLists.template CMakeLists.txt
RUN echo 'include(${CMAKE_CURRENT_SOURCE_DIR}/../CMakeLists.txt)' >> CMakeLists.txt
RUN mkdir build
WORKDIR /root/openair-cn/build/mme/build
RUN cmake -DOPENAIRCN_DIR=/root/openair-cn ../
RUN make -j`nproc`

# Configuration files
RUN mkdir -p /usr/local/etc/oai/freeDiameter
# RUN cp /root/openair-cn/etc/mme.conf /usr/local/etc/oai/
RUN cp /root/openair-cn/etc/mme_fd.conf /usr/local/etc/oai/freeDiameter/

ENV HSS_CN_NAME=hss.openair4G.eur
ENV MME_CN_NAME=mme.openair4G.eur
ENV MME_IPV4_ADDRESS_FOR_S1_MME="192.168.142.20"
ENV HSS_IPV4_ADDRESS="192.168.142.10"

#ready to work
WORKDIR /root
COPY start.sh /root/start.sh
RUN chmod +x /root/start.sh
ENTRYPOINT "/root/start.sh"
```

B.8 MME configuration

```
################################################################################
# Licensed to the OpenAirInterface (OAI) Software Alliance under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The OpenAirInterface Software Alliance licenses this file to You under
# the Apache License, Version 2.0  (the "License"); you may not use this file
# except in compliance with the License.
# You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
```

```
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------------------------
# For more information about the OpenAirInterface (OAI) Software Alliance:
#       contact@openairinterface.org
################################################################################

MME :
{
    REALM                                     = "openair4G.eur";        # YOUR REALM HERE
    PID_DIRECTORY                             = "/var/run";
    # Define the limits of the system in terms of served eNB and served UE.
    # When the limits will be reached, overload procedure will take place.
    MAXENB                                    = 2;                      # power of 2
    MAXUE                                     = 16;                     # power of 2
    RELATIVE_CAPACITY                         = 10;

    EMERGENCY_ATTACH_SUPPORTED                     = "no";
    UNAUTHENTICATED_IMSI_SUPPORTED                 = "no";

    # EPS network feature support
    EPS_NETWORK_FEATURE_SUPPORT_IMS_VOICE_OVER_PS_SESSION_IN_S1       = "no";    # DO NOT CHANGE
    EPS_NETWORK_FEATURE_SUPPORT_EMERGENCY_BEARER_SERVICES_IN_S1_MODE = "no";    # DO NOT CHANGE
    EPS_NETWORK_FEATURE_SUPPORT_LOCATION_SERVICES_VIA_EPC            = "no";    # DO NOT CHANGE
    EPS_NETWORK_FEATURE_SUPPORT_EXTENDED_SERVICE_REQUEST             = "no";    # DO NOT CHANGE

    # Display statistics about whole system (expressed in seconds)
    MME_STATISTIC_TIMER                       = 10;

    IP_CAPABILITY = "IPV4V6";                                           # UNUSED, TODO


    INTERTASK_INTERFACE :
    {
        # max queue size per task
        ITTI_QUEUE_SIZE            = 2000000;
    };

    S6A :
    {
        S6A_CONF                   = "/usr/local/etc/oai/freeDiameter/mme_fd.conf"; # YOUR MME freeDiameter config file path
        HSS_HOSTNAME               = "hss";                             # THE HSS HOSTNAME
    };

    # ------- SCTP definitions
    SCTP :
    {
        # Number of streams to use in input/output
```

```
        SCTP_INSTREAMS  = 8;
        SCTP_OUTSTREAMS = 8;
    };

    # ------------- S1AP definitions
    S1AP :
    {
        # outcome drop timer value (seconds)
        S1AP_OUTCOME_TIMER = 10;
    };

    # ------------- MME served GUMMEIs
    # MME code DEFAULT  size = 8 bits
    # MME GROUP ID size = 16 bits
    GUMMEI_LIST = (
         {MCC="208" ; MNC="93"; MME_GID="4" ; MME_CODE="1"; }          # YOUR GUMMEI CONFIG HERE
    );

    # ------------- MME served TAIs
    # TA (mcc.mnc:tracking area code) DEFAULT = 208.34:1
    # max values = 999.999:65535
    # maximum of 16 TAIs, comma separated
    # !!! Actually use only one PLMN
    TAI_LIST = (
         {MCC="208" ; MNC="93";  TAC = "1"; }                       # YOUR TAI CONFIG HERE
    );


    NAS :
    {
        # 3GPP TS 33.401 section 7.2.4.3 Procedures for NAS algorithm selection
        # decreasing preference goes from left to right
        ORDERED_SUPPORTED_INTEGRITY_ALGORITHM_LIST = [ "EIA2" , "EIA1" , "EIA0" ];
        ORDERED_SUPPORTED_CIPHERING_ALGORITHM_LIST = [ "EEA0" , "EEA1" , "EEA2" ];

        # EMM TIMERS
        # T3402 start:
        # At attach failure and the attempt counter is equal to 5.
        # At tracking area updating failure and the attempt counter is equal to 5.
        # T3402 stop:
        # ATTACH REQUEST sent, TRACKING AREA REQUEST sent.
        # On expiry:
        # Initiation of the attach procedure, if still required or TAU procedure
        # attached for emergency bearer services.
        T3402                                 =  1                           # in minutes (default is 12 minutes)

        # T3412 start:
        # In EMM-REGISTERED, when EMM-CONNECTED mode is left.
        # T3412 stop:
        # When entering state EMM-DEREGISTERED or when entering EMM-CONNECTED mode.
        # On expiry:
        # Initiation of the periodic TAU procedure if the UE is not attached for
```

```
    # emergency bearer services. Implicit detach from network if the UE is
    # attached for emergency bearer services.
    T3412                                  = 54     # in minutes (default is 54 minutes, network dependent)
    # T3422 start: DETACH REQUEST sent
    # T3422 stop: DETACH ACCEPT received
    # ON THE 1st, 2nd, 3rd, 4th EXPIRY: Retransmission of DETACH REQUEST
    T3422                                  = 6      # in seconds (default is 6s)

    # T3450 start:
    # ATTACH ACCEPT sent, TRACKING AREA UPDATE ACCEPT sent with GUTI, TRACKING AREA UPDATE ACCEPT sent with TMSI,
    # GUTI REALLOCATION COMMAND sent
    # T3450 stop:
    # ATTACH COMPLETE received, TRACKING AREA UPDATE COMPLETE received, GUTI REALLOCATION COMPLETE received
    # ON THE 1st, 2nd, 3rd, 4th EXPIRY: Retransmission of the same message type
    T3450                                  = 6      # in seconds (default is 6s)

    # T3460 start: AUTHENTICATION REQUEST sent, SECURITY MODE COMMAND sent
    # T3460 stop:
    # AUTHENTICATION RESPONSE received, AUTHENTICATION FAILURE received,
    # SECURITY MODE COMPLETE received, SECURITY MODE REJECT received
    # ON THE 1st, 2nd, 3rd, 4th EXPIRY: Retransmission of the same message type
    T3460                                  = 6      # in seconds (default is 6s)

    # T3470 start: IDENTITY REQUEST sent
    # T3470 stop: IDENTITY RESPONSE received
    # ON THE 1st, 2nd, 3rd, 4th EXPIRY: Retransmission of IDENTITY REQUEST
    T3470                                  = 6      # in seconds (default is 6s)

    # ESM TIMERS
    T3485                                  = 8      # UNUSED in seconds (default is 8s)
    T3486                                  = 8      # UNUSED in seconds (default is 8s)
    T3489                                  = 4      # UNUSED in seconds (default is 4s)
    T3495                                  = 8      # UNUSED in seconds (default is 8s)
};

NETWORK_INTERFACES :
{
    # MME binded interface for S1-C or S1-MME  communication (S1AP), can be ethernet interface, virtual ethernet interface, we don't advise wireless interfaces
    MME_INTERFACE_NAME_FOR_S1_MME          = "eth0";      # YOUR NETWORK CONFIG HERE
    MME_IPV4_ADDRESS_FOR_S1_MME            = "0.0.0.0/24"; # YOUR NETWORK CONFIG HERE

```

```
        # MME binded interface for S11 communication (GTPV2-C)
        MME_INTERFACE_NAME_FOR_S11_MME        = "eth0";        # YOUR NETWORK CONFIG HERE
        MME_IPV4_ADDRESS_FOR_S11_MME          = "0.0.0.0/24";  # YOUR NETWORK CONFIG HERE
        MME_PORT_FOR_S11_MME                  = 2123;          # YOUR NETWORK CONFIG HERE
    };

    LOGGING :
    {
        # OUTPUT choice in { "CONSOLE", "SYSLOG", `path to file`", "`IPv4@`:`TCP port num`"}
        # `path to file` must start with '.' or '/'
        # if TCP stream choice, then you can easily dump the traffic on the remote or local host: nc -l `TCP port num` > received.txt
        OUTPUT              = "CONSOLE";
        #OUTPUT              = "SYSLOG";
        #OUTPUT              = "/tmp/mme.log";
        #OUTPUT              = "127.0.0.1:5656";

        # THREAD_SAFE choice in { "yes", "no" } means use of thread safe intermediate buffer then a single thread pick each message log one
        # by one to flush it to the chosen output
        THREAD_SAFE         = "yes";

        # COLOR choice in { "yes", "no" } means use of ANSI styling codes or no
        COLOR               = "yes";

        # Log level choice in { "EMERGENCY", "ALERT", "CRITICAL", "ERROR", "WARNING", "NOTICE", "INFO", "DEBUG", "TRACE"}
        SCTP_LOG_LEVEL      = "TRACE";
        S11_LOG_LEVEL       = "TRACE";
        GTPV2C_LOG_LEVEL    = "TRACE";
        UDP_LOG_LEVEL       = "TRACE";
        S1AP_LOG_LEVEL      = "TRACE";
        NAS_LOG_LEVEL       = "TRACE";
        MME_APP_LOG_LEVEL   = "TRACE";
        S6A_LOG_LEVEL       = "TRACE";
        UTIL_LOG_LEVEL      = "TRACE";
        MSC_LOG_LEVEL       = "ERROR";
        ITTI_LOG_LEVEL      = "ERROR";
        MME_SCENARIO_PLAYER_LOG_LEVEL = "TRACE";

        # ASN1 VERBOSITY: none, info, annoying
        # for S1AP protocol
        ASN1_VERBOSITY      = "none";
    };
    TESTING :
    {
        # file should be copied here from source tree by following command: run_mme --install-mme-files ...
        SCENARIO_FILE = "/usr/local/share/oai/test/mme/no_regression.xml";
    };
};

S-GW :
{
    # S-GW binded interface for S11 communication (GTPV2-C), if none selected the ITTI message interface is used
    SGW_IPV4_ADDRESS_FOR_S11                  = "192.168.142.30/8";        # YOUR NETWORK CONFIG HERE
```

```

};
```

B.9 MME start script

```
#!/bin/bash
# MME Configuration
sed -i s/"mme.openair4G.eur"/$MME_CN_NAME/g /usr/local/etc/oai/mme.conf
sed -i s/"yang.openair4G.eur"/$MME_CN_NAME/g /usr/local/etc/oai/
    freeDiameter/mme_fd.conf
sed -i s/"mme.openair4G.eur"/$MME_CN_NAME/g /usr/local/etc/oai/
    freeDiameter/mme_fd.conf
sed -i s/"hss.openair4G.eur"/$HSS_CN_NAME/g /usr/local/etc/oai/
    freeDiameter/mme_fd.conf

# set IP addr
sed -i s/"192.168.11.17"/"$MME_IPV4_ADDRESS_FOR_S1_MME"/g /usr/local/etc/
    oai/mme.conf
sed -i s/"127.0.0.1"/"$HSS_IPV4_ADDRESS"/g /usr/local/etc/oai/freeDiameter
    /mme_fd.conf


# Generation of certificate for diameter
cd /root
if [[ ! -d /root/demoCA ]];then
  mkdir demoCA && touch demoCA/index.txt && echo 01 > demoCA/serial
  openssl req  -new -batch -x509 -days 3650 -nodes -newkey rsa:1024 -out
    mme.cacert.pem -keyout mme.cakey.pem -subj /CN=$MME_CN_NAME/C=FR/ST=
    PACA/L=Aix/O=Eurecom/OU=CM
  openssl genrsa -out mme.key.pem 1024
  openssl req -new -batch -out mme.csr.pem -key mme.key.pem -subj /CN=
    $MME_CN_NAME/C=FR/ST=PACA/L=Aix/O=Eurecom/OU=CM
  openssl ca -cert mme.cacert.pem -keyfile mme.cakey.pem -in mme.csr.pem -
    out mme.cert.pem -outdir . -batch
  mv /root/mme.cakey.pem /usr/local/etc/oai/freeDiameter/
  mv /root/mme.cert.pem /usr/local/etc/oai/freeDiameter/
  mv /root/mme.cacert.pem /usr/local/etc/oai/freeDiameter/
  mv /root/mme.key.pem /usr/local/etc/oai/freeDiameter/
fi
# Start mme
sleep 17 && /root/openair-cn/build/mme/build/mme
```

B.10 Docker file for SPGW

```
FROM ubuntu:16.04
MAINTAINER Yan Grunenberger <yan@grunenberger.net>
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update
RUN apt-get -yq dist-upgrade

# General utilities
RUN apt-get -qy install kmod git wget apt-utils

# Adding OAI certificates
RUN mkdir -p /usr/local/share/ca-certificates/eurecom
RUN echo -n | openssl s_client -showcerts -connect gitlab.eurecom.fr:443
    2>/dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > /
    usr/local/share/ca-certificates/eurecom/eurecom.crt
RUN update-ca-certificates

# cloning directory
```

```
WORKDIR /root
RUN mkdir .ssh
RUN ssh-keyscan github.com >> .ssh/known_hosts
COPY id_rsa .ssh/id_rsa
COPY id_rsa.pub .ssh/id_rsa.pub
RUN git clone git@github.com:aschwanb/openair-cn.git

WORKDIR /root/openair-cn
RUN git checkout develop

WORKDIR /root

# Fixing default mysql root password to "linux". This is the default assumed by OAI building scripts
RUN echo 'mysql-server mysql-server/root_password password linux' | debconf-set-selections
RUN echo 'mysql-server mysql-server/root_password_again password linux' | debconf-set-selections
RUN echo 'phpmyadmin phpmyadmin/dbconfig-install boolean true' | debconf-set-selections
RUN echo 'phpmyadmin phpmyadmin/app-password-confirm password linux ' | debconf-set-selections
RUN echo 'phpmyadmin phpmyadmin/mysql/admin-pass password linux' | debconf-set-selections
RUN echo 'phpmyadmin phpmyadmin/mysql/app-pass password linux' | debconf-set-selections
RUN echo 'phpmyadmin phpmyadmin/reconfigure-webserver multiselect apache2' | debconf-set-selections

# (Build script - MME dependencies...)
# (from build_helper) Remove incompatible softwares
RUN apt-get -qy --purge remove libgnutls-dev \
 'libgnutlsxx2?' \
 nettle-dev \
 nettle-bin

# (from build_helper) Compilers, Generators
RUN apt-get -qy install autoconf \
 automake \
 bison \
 build-essential \
 cmake \
 cmake-curses-gui \
 doxygen \
 doxygen-gui\
 flex \
 gdb \
 pkg-config

# (from build_helper) git/svn
RUN apt-get -qy install git \
 subversion

# (from build_helper) librairies
RUN apt-get -qy install libconfig8-dev \
 libgcrypt11-dev \
 libidn2-0-dev \
 libidn11-dev \
 libmysqlclient-dev \
 libpthread-stubs0-dev \
 libsctp1 \
 libsctp-dev \
 libssl-dev \
```

```
 libtool \
 mysql-client \
 mysql-server \
 openssl

# (from build_helper) compile nettle from source
RUN apt-get -qy install  \
 autoconf  \
 automake  \
 build-essential \
 libgmp-dev
WORKDIR /root
# other mirror : ftp://ftp.lysator.liu.se/pub/security/lsh/nettle-2.5.tar.gz
RUN wget https://ftp.gnu.org/gnu/nettle/nettle-2.5.tar.gz
RUN tar -xzf nettle-2.5.tar.gz
RUN cd nettle-2.5/ &&  ./configure --disable-openssl --enable-shared --prefix=/usr  &&  make -j`nproc` &&  make check  &&  make install

# (from build_helper) install_gnutls_from_source $1
WORKDIR /root
RUN apt-get -qy install \
 autoconf  \
 automake  \
 build-essential
#       libtasn1-6-dbg \  libp11-kit0-dbg \
RUN apt-get -qy install libtasn1-6-dev \
 libp11-kit-dev \
 libtspi-dev \
 libtspi1 \
 libidn2-0-dev \
 libidn11-dev
RUN wget http://mirrors.dotsrc.org/gcrypt/gnutls/v3.1/gnutls-3.1.23.tar.xz
RUN tar -xJf gnutls-3.1.23.tar.xz
RUN cd gnutls-3.1.23/ &&  ./configure --prefix=/usr && make -j`nproc` && make install

# (from build_helper)
RUN apt-get -qy install autoconf  \
 automake  \
 bison     \
 build-essential \
 cmake \
 cmake-curses-gui  \
 debhelper \
 flex  \
 g++ \
 gcc \
 gdb \
 libgcrypt-dev \
 libidn11-dev \
 libmysqlclient-dev \
 libpq-dev \
 libsctp1 \
 libsctp-dev \
 libxml2-dev \
 mercurial \
 python-dev \
 ssl-cert \
 swig

# Run freediameter (hard dependencies on gnutls)
WORKDIR /root
```

```
RUN git clone https://gitlab.eurecom.fr/oai/freediameter.git -b eurecom-1.2.0
WORKDIR /root/freediameter
RUN mkdir build && cd build && cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr ../ && make -j`nproc` && make install

# PHPmyadmin for the MME database management
WORKDIR /root
RUN apt-get -qy install phpmyadmin \
 python-pexpect \
 php \
 libapache2-mod-php

RUN apt-get -qy install check \
 phpmyadmin \
 python-dev \
 python-pexpect \
 unzip

#  install_asn1c_from_source
WORKDIR /root
RUN apt-get -qy install autoconf automake bison build-essential flex gcc libtool
RUN git clone https://gitlab.eurecom.fr/oai/asn1c.git
RUN cd asn1c && ./configure && make && make install

#   install_libgtpnl_from_source

WORKDIR /root
RUN apt-get -qy install    autoconf  \
    automake  \
    build-essential \
    libmnl-dev

RUN  git clone git://git.osmocom.org/libgtpnl
RUN cd libgtpnl && autoreconf -fi && ./configure && make -j`nproc` && make install && ldconfig



RUN apt-get -qy install autoconf  \
    automake  \
    bison     \
    build-essential \
    cmake \
    cmake-curses-gui  \
    doxygen \
    doxygen-gui\
    flex  \
    gccxml \
    gdb  \
    git \
    pkg-config \
    subversion

RUN apt-get -qy install guile-2.0-dev \
    libconfig8-dev \
    libgcrypt11-dev \
    libgmp-dev \
    libhogweed? \
    libgtk-3-dev \
    libidn2-0-dev \
    libidn11-dev \
```

```
    libpthread-stubs0-dev \
    libtool \
    libxml2 \
    libxml2-dev \
    mscgen \
    openssl \
    python

RUN apt-get -qy install  ethtool \
    iperf \
    iproute \
    vlan \
    tshark

RUN apt-get -qy install python-dev \
    python-pexpect \
    unzip

# compiling OAI mme executable
WORKDIR /root/openair-cn/build/

WORKDIR /root/openair-cn/build/spgw
RUN cp /root/openair-cn/build/spgw/CMakeLists.template ./CMakeLists.txt
RUN echo 'include(${CMAKE_CURRENT_SOURCE_DIR}/../CMakeLists.txt)' >> ./
    CMakeLists.txt
RUN mkdir build
WORKDIR /root/openair-cn/build/spgw/build
RUN cmake -DOPENAIRCN_DIR=/root/openair-cn ../
RUN make -j `nproc`


RUN mkdir -p /usr/local/etc/oai/freeDiameter
RUN cp /root/openair-cn/etc/spgw.conf /usr/local/etc/oai/
RUN apt-get -qy install iptables

ENV SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP="192.168.142.30"
ENV PGW_INTERFACE_NAME_FOR_SGI="eth1"

#ready to work
WORKDIR /root
COPY start.sh /root/start.sh
RUN chmod +x /root/start.sh
ENTRYPOINT "/root/start.sh"
```

B.11 SPGW configuration

```
################################################################################
# Licensed to the OpenAirInterface (OAI) Software Alliance under one or
    more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The OpenAirInterface Software Alliance licenses this file to You under
# the Apache License, Version 2.0  (the "License"); you may not use this
    file
# except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
```

```
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------------------------
# For more information about the OpenAirInterface (OAI) Software Alliance:
#      contact@openairinterface.org
################################################################################
S-GW :
{
    NETWORK_INTERFACES :
    {
        # S-GW binded interface for S11 communication (GTPV2-C), if none selected the ITTI message interface is used
        SGW_INTERFACE_NAME_FOR_S11              = "eth0";            # STRING, interface name, YOUR NETWORK CONFIG HERE
        SGW_IPV4_ADDRESS_FOR_S11                = "192.168.142.30/24";   # STRING, CIDR, YOUR NETWORK CONFIG HERE

        # S-GW binded interface for S1-U communication (GTPV1-U) can be ethernet interface, virtual ethernet interface, we don't advise wireless interfaces
        SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP    = "eth0";            # STRING, interface name, YOUR NETWORK CONFIG HERE, USE "lo" if S-GW run on eNB host
        SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP      = "192.168.142.30/24";   # STRING, CIDR, YOUR NETWORK CONFIG HERE
        SGW_IPV4_PORT_FOR_S1U_S12_S4_UP         = 2152;              # INTEGER, port number, PREFER NOT CHANGE UNLESS YOU KNOW WHAT YOU ARE DOING

        # S-GW binded interface for S5 or S8 communication, not implemented, so leave it to none
        SGW_INTERFACE_NAME_FOR_S5_S8_UP         = "none";            # STRING, interface name, DO NOT CHANGE (NOT IMPLEMENTED YET)
        SGW_IPV4_ADDRESS_FOR_S5_S8_UP           = "0.0.0.0/24";      # STRING, CIDR, DO NOT CHANGE (NOT IMPLEMENTED YET)
    };

    INTERTASK_INTERFACE :
    {
        # max queue size per task
        ITTI_QUEUE_SIZE            = 2000000;                      # INTEGER
    };

    LOGGING :
    {
        # OUTPUT choice in { "CONSOLE", "SYSLOG", `path to file`", "`IPv4@`:`TCP port num`"}
        # `path to file` must start with '.' or '/'
        # if TCP stream choice, then you can easily dump the traffic on the remote or local host: nc -l `TCP port num` > received.txt
        OUTPUT            = "CONSOLE";                              # see 3 lines above
        #OUTPUT            = "SYSLOG";                              # see 4 lines above
        #OUTPUT            = "/tmp/spgw.log";                       # see 5 lines above
        #OUTPUT            = "127.0.0.1:5656";                      # see 6 lines above

```

```
        # THREAD_SAFE choice in { "yes", "no" } means use of thread safe intermediate buffer then a single thread pick each message log one
        # by one to flush it to the chosen output
        THREAD_SAFE        = "no";

        # COLOR choice in { "yes", "no" } means use of ANSI styling codes or no
        COLOR              = "yes";

        # Log level choice in { "EMERGENCY", "ALERT", "CRITICAL", "ERROR", "WARNING", "NOTICE", "INFO", "DEBUG", "TRACE"}
        UDP_LOG_LEVEL      = "TRACE";
        GTPV1U_LOG_LEVEL   = "TRACE";
        GTPV2C_LOG_LEVEL   = "TRACE";
        SPGW_APP_LOG_LEVEL = "TRACE";
        S11_LOG_LEVEL      = "TRACE";
    };
};

P-GW =
{
    NETWORK_INTERFACES :
    {
        # P-GW binded interface for S5 or S8 communication, not implemented, so leave it to none
        PGW_INTERFACE_NAME_FOR_S5_S8          = "none";    # STRING, interface name, DO NOT CHANGE (NOT IMPLEMENTED YET)

        # P-GW binded interface for SGI (egress/ingress internet traffic)
        PGW_INTERFACE_NAME_FOR_SGI            = "eth0";    # STRING, YOUR NETWORK CONFIG HERE
        PGW_MASQUERADE_SGI                    = "yes";     # STRING, {"yes", "no"}. YOUR NETWORK CONFIG HERE, will do NAT for you if you put "yes".
        UE_TCP_MSS_CLAMPING                   = "no";      # STRING, {"yes", "no"}.
    };

    # Pool of UE assigned IP addresses
    # Do not make IP pools overlap
    # first IPv4 address X.Y.Z.1 is reserved for GTP network device on SPGW
    # Normally no more than 16 pools allowed, but since recent GTP kernel module use, only one pool allowed (TODO).
    IP_ADDRESS_POOL :
    {
        IPV4_LIST = (
                      "172.16.0.0/12"    # STRING, CIDR, YOUR NETWORK CONFIG HERE.
                    );
    };

    # DNS address communicated to UEs
    DEFAULT_DNS_IPV4_ADDRESS     = "8.8.8.8";    # YOUR NETWORK CONFIG HERE
    DEFAULT_DNS_SEC_IPV4_ADDRESS = "8.8.4.4";    # YOUR NETWORK CONFIG HERE

    # Non standard feature, normally should be set to "no", but you may need to set to yes for UE that do not explicitly request a PDN address through NAS signalling
    FORCE_PUSH_PROTOCOL_CONFIGURATION_OPTIONS = "no";    # STRING, {"yes", "no"}.
```

```
    UE_MTU                                         = 1500
            # INTEGER
};
```

B.12 SPGW start script

```
#!/bin/bash
sed -i s/"192.168.11.17"/"$SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP"/g /usr/local/etc/oai/spgw.conf
sed -i s/"eth3"/"$PGW_INTERFACE_NAME_FOR_SGI"/g /usr/local/etc/oai/spgw.conf

# Start spgw
sleep 15 && /root/openair-cn/build/spgw/build/spgw
```

Appendix C

Additional Configuration

C.1 eNB configuration

```
Active_eNBs = ( "eNB_Eurecom_LTEBox");
# Asn1_verbosity, choice in: none, info, annoying
Asn1_verbosity = "annoying";

eNBs =
(
 {
    ////////// Identification parameters:
    eNB_ID    =  0xe00;

    cell_type =  "CELL_MACRO_ENB";

    eNB_name  =  "eNB_Eurecom_LTEBox";

    // Tracking area code, 0x0000 and 0xfffe are reserved values
    tracking_area_code  =  "1";

    mobile_country_code =  "208";

    mobile_network_code =  "93";

       ////////// Physical parameters:

    component_carriers = (
      {
            node_function                                     = "eNodeB_3GPP";
            node_timing                                       = "synch_to_ext_device";
            node_synch_ref                                    = 0;

          frame_type                     = "FDD";
            tdd_config                     = 3;
            tdd_config_s                     = 0;
      prefix_type                      = "NORMAL";
        eutra_band                     = 7;
            downlink_frequency               = 2680000000L;
            uplink_frequency_offset          = -120000000;
        Nid_cell                     = 0;
            N_RB_DL                          = 25;
            Nid_cell_mbsfn                   = 0;
      nb_antenna_ports             = 2;
            nb_antennas_tx                   = 2;
            nb_antennas_rx                   = 2;
      tx_gain                                = 25;
      rx_gain                                = 20;
```

```
            prach_root                          = 0;
            prach_config_index                  = 0;
            prach_high_speed                    = "DISABLE";
            prach_zero_correlation              = 1;
            prach_freq_offset                   = 2;
      pucch_delta_shift                   = 1;
            pucch_nRB_CQI                       = 1;
            pucch_nCS_AN                        = 0;
            pucch_n1_AN                         = 32;
            pdsch_referenceSignalPower            = 0;
            pdsch_p_b                             = 0;
            pusch_n_SB                            = 1;
            pusch_enable64QAM                     = "DISABLE";
      pusch_hoppingMode                         = "interSubFrame";
      pusch_hoppingOffset                       = 0;
          pusch_groupHoppingEnabled             = "ENABLE";
          pusch_groupAssignment                 = 0;
          pusch_sequenceHoppingEnabled          = "DISABLE";
          pusch_nDMRS1                              = 0;
          phich_duration                            = "NORMAL";
          phich_resource                            = "ONESIXTH";
          srs_enable                                = "DISABLE";
          /*   srs_BandwidthConfig                  =;
          srs_SubframeConfig                        =;
          srs_ackNackST                             =;
          srs_MaxUpPts                              =;*/

          pusch_p0_Nominal                          = -108;
          pusch_alpha                               = "AL1";
          pucch_p0_Nominal                          = -108;
          msg3_delta_Preamble                       = 6;
          pucch_deltaF_Format1                      = "deltaF2";
          pucch_deltaF_Format1b                     = "deltaF3";
          pucch_deltaF_Format2                      = "deltaF0";
          pucch_deltaF_Format2a                     = "deltaF0";
            pucch_deltaF_Format2b               = "deltaF0";

            rach_numberOfRA_Preambles                 = 64;
            rach_preamblesGroupAConfig                = "DISABLE";
/*
            rach_sizeOfRA_PreamblesGroupA             = ;
            rach_messageSizeGroupA                    = ;
            rach_messagePowerOffsetGroupB             = ;
*/
            rach_powerRampingStep                     = 2;
            rach_preambleInitialReceivedTargetPower   = -100;
            rach_preambleTransMax                     = 10;
            rach_raResponseWindowSize                 = 10;
            rach_macContentionResolutionTimer         = 48;
            rach_maxHARQ_Msg3Tx                       = 4;

            pcch_default_PagingCycle                  = 128;
            pcch_nB                                   = "oneT";
            bcch_modificationPeriodCoeff        = 2;
            ue_TimersAndConstants_t300            = 1000;
            ue_TimersAndConstants_t301            = 1000;
            ue_TimersAndConstants_t310            = 1000;
            ue_TimersAndConstants_t311            = 10000;
            ue_TimersAndConstants_n310            = 20;
      ue_TimersAndConstants_n311              = 1;

      ue_TransmissionMode               = 2;
        }
```

```
    );


    srb1_parameters :
    {
        # timer_poll_retransmit = (ms) [5, 10, 15, 20,... 250, 300, 350, ... 500]
        timer_poll_retransmit    = 80;

        # timer_reordering = (ms) [0,5, ... 100, 110, 120, ... ,200]
        timer_reordering         = 35;

        # timer_reordering = (ms) [0,5, ... 250, 300, 350, ... ,500]
        timer_status_prohibit    = 0;

        # poll_pdu = [4, 8, 16, 32 , 64, 128, 256, infinity(>10000)]
        poll_pdu                 =  4;

        # poll_byte = (kB) [25,50,75,100,125,250,375,500,750,1000,1250,1500,2000,3000,infinity(>10000)]
        poll_byte                =  99999;

        # max_retx_threshold = [1, 2, 3, 4 , 6, 8, 16, 32]
        max_retx_threshold       =  4;
    }

    # -------- SCTP definitions
    SCTP :
    {
        # Number of streams to use in input/output
        SCTP_INSTREAMS  = 2;
        SCTP_OUTSTREAMS = 2;
    };

    ////////// MME parameters:
    mme_ip_address      = ( { ipv4       = "192.168.142.20";
                              ipv6       = "192:168:30::17";
                              active     = "yes";
                              preference = "ipv4";
                            }
                          );

    NETWORK_INTERFACES :
    {
        ENB_INTERFACE_NAME_FOR_S1_MME            = "br-3cb0e0ddb8a9";
        ENB_IPV4_ADDRESS_FOR_S1_MME              = "192.168.142.1/24";

        ENB_INTERFACE_NAME_FOR_S1U               = "br-3cb0e0ddb8a9";
        ENB_IPV4_ADDRESS_FOR_S1U                 = "192.168.142.1/24";
        ENB_PORT_FOR_S1U                         = 2152; # Spec 2152
    };

    log_config :
    {
    global_log_level                      ="trace";
    global_log_verbosity                  ="medium";
    hw_log_level                          ="info";
    hw_log_verbosity                      ="medium";
    phy_log_level                         ="trace";
    phy_log_verbosity                     ="medium";
    mac_log_level                         ="trace";
    mac_log_verbosity                     ="medium";
```

```
    rlc_log_level                              ="trace";
    rlc_log_verbosity                          ="medium";
    pdcp_log_level                             ="trace";
    pdcp_log_verbosity                         ="medium";
    rrc_log_level                              ="trace";
    rrc_log_verbosity                          ="medium";
    gtpu_log_level                             ="debug";
    gtpu_log_verbosity                         ="medium";
    udp_log_level                              ="debug";
    udp_log_verbosity                          ="medium";
    osa_log_level                              ="debug";
    osa_log_verbosity                          ="low";

   };

  }
);
```

C.2 Prometheus configuration

```
global:
  # How frequently to scrape targets by default.
  scrape_interval: 15s

  # The labels to add to any time series or alerts when communicating with
  # external systems (federation, remote storage, Alertmanager).
  external_labels:
    monitor: 'epc-monitor'

# A list of scrape configurations.
scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'cadvisor'
    scrape_interval: 5s
    static_configs:
      - targets: ['cadvisor:8080']
```

C.3 Nginx configuration for DB Load Balancing

```

user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}

# https://nginx.org/en/docs/stream/ngx_stream_proxy_module.html
stream {
  server {
      listen       3306;
      proxy_pass db:3306;
  }
}
```

C.4 Nginx configuration for HSS Load Balancing

```

user   nginx;
worker_processes   1;

error_log   /var/log/nginx/error.log warn;
pid         /var/run/nginx.pid;


events {
    worker_connections   1024;
}

# https://nginx.org/en/docs/stream/ngx_stream_proxy_module.html
stream {
  server {
      listen        3868;
      proxy_pass  backend_hss:3868;
  }
}
```

C.5 Nginx configuration for MME Load Balancing

```

user   nginx;
worker_processes   1;

error_log   /var/log/nginx/error.log warn;
pid         /var/run/nginx.pid;


events {
    worker_connections   1024;
}

# https://nginx.org/en/docs/stream/ngx_stream_proxy_module.html
stream {
  server {
      listen        2123;
      proxy_pass mme:2123;
  }
}
```

C.6 Nginx configuration for SPGW Load Balancing

```

user   nginx;
worker_processes   1;

error_log   /var/log/nginx/error.log warn;
pid         /var/run/nginx.pid;


events {
    worker_connections   1024;
}

# https://nginx.org/en/docs/stream/ngx_stream_proxy_module.html
stream {
```

```
  server {
      listen        2123;
      proxy_pass spgw:2123;
  }
  server {
      listen        2152;
      proxy_pass spgw:2152;
  }
}
```

Appendix D

Traffic Sniffing Utility

D.1 Implementation - main.py

```
#!/bin/env python

import logging
import tracker_new


if __name__ == "__main__":
  global logger
  logger = logging.getLogger('mylogger')
  ch = logging.StreamHandler()
  ch.setLevel(logging.DEBUG)
  logger.addHandler(ch)
  logger.setLevel(logging.INFO)

  iface = 'br-3cb0e0ddb8a9'
  (enb_teid, mme_teid) = tracker_new.trace_pkt(iface, 0)
  logger.info("ENB TEID is %s" % enb_teid)
  logger.info("MME TEID is %s" % mme_teid)
```

D.2 Underlying utility class - tracker_new.py

This class is largely based on the work done by Dr. Eryk Schiller.

```
'''
Packet sniffer in python using the pcapy python library

Project website
http://oss.coresecurity.com/projects/pcapy.html
'''

import socket
import logging
from scapy.all import *
from struct import *
import datetime
import pcapy
import sys
from libmich.asn1.processor import *
from types import *


logger = logging.getLogger(__name__)
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
logger.addHandler(ch)
logger.setLevel(logging.DEBUG)
```

```

# generate_modules({'S1AP':'S1AP_36413-c10'})
load_module('S1AP')

db = {}
inner = {}

ASN1.ASN1Obj.CODEC = PER
PER.VARIANT = 'A'
pdu = GLOBAL.TYPE['S1AP-PDU']

def decode_string(buf):
    error = 0
    try:
        error = 0
        pdu.decode(buf);
        val = pdu()
    except:
        error = 1;

    return error

def get_val_from_tuple(text, tup):
    index = 0

    if not isinstance(tup, tuple):
        return 1, ""

    try:
        index = tup.index(text);

    except:
        return 1, ""

    if index + 1 >= len(tup):
        return 1, ""

    else:
        return 0, index+1


def get_enb_gtp_teid():
    error = 0
    index = 0

    pdu_val = pdu()

    error, index = get_val_from_tuple('initiatingMessage', pdu_val)

    if error:
        return

    inm = pdu_val[index]

    if not isinstance (inm, dict):
        return

    if not 'value' in inm:
        return

    error, index = get_val_from_tuple('InitialContextSetupRequest', inm['
    value'])

```

```
    if error:
        return

    icsr = inm['value'][index]

    if not isinstance(icsr, dict):
        return

    if not 'protocolIEs' in icsr:
        return

    for item in icsr['protocolIEs']:
        if not isinstance(item, dict):
            continue
        else:
            if 'value' in item:
                err1, ind1 = get_val_from_tuple('E-RABToBeSetupListCtxtSUReq', item['value'])
                if err1:
                    continue
                else:
                    x = item['value'][ind1]
                    for y in x:
                        if isinstance(y, dict):
                            if 'value' in y:
                                z = y['value']
                                err2, ind2 = get_val_from_tuple('E-RABToBeSetupItemCtxtSUReq', z)

                                if err2:
                                    continue
                                else:
                                    if isinstance(z[ind2], dict):
                                        if 'e-RAB-ID' in z[ind2] and 'gTP-TEID' in z[ind2]:
                                            if isinstance(z[ind2]['e-RAB-ID'], int) and isinstance(z[ind2]['gTP-TEID'], str):
                                                if not z[ind2]['e-RAB-ID'] in db:
                logger.debug('####################')
                                                    logger.debug("Adding e-RAB-ID to db")
                logger.debug('####################')
                                                    db[z[ind2]['e-RAB-ID']] = {}

                                                db[z[ind2]['e-RAB-ID']]['ENB-TEID'] = z[ind2]['gTP-TEID'].encode('hex')
            logger.debug('****')
                                                logger.debug(db)
            logger.debug('****')

def get_mme_gtp_teid():
    error = 0
    index = 0

    pdu_val = pdu()
    error, index = get_val_from_tuple('successfulOutcome', pdu_val)

    if error:
        return

    inm = pdu_val[index]
```

```

    if not isinstance (inm, dict):
        return

    if not 'value' in inm:
        return

    error, index = get_val_from_tuple('InitialContextSetupResponse', inm['value'])

    if error:
        return

    icsr = inm['value'][index]

    if not isinstance(icsr, dict):
        return

    if not 'protocolIEs' in icsr:
        return

    for item in icsr['protocolIEs']:
        if not isinstance(item, dict):
            continue
        else:
            if 'value' in item:
                err1, ind1 = get_val_from_tuple('E-RABSetupListCtxtSURes', item['value'])
                if err1:
                    continue
                else:
                    x = item['value'][ind1]
                    for y in x:
                        if isinstance(y, dict):
                            if 'value' in y:
                                z = y['value']
                                err2, ind2 = get_val_from_tuple('E-RABSetupItemCtxtSURes', z)

                                if err2:
                                    continue
                                else:
                                    if isinstance(z[ind2], dict):
                                        if 'e-RAB-ID' in z[ind2] and 'gTP-TEID' in z[ind2]:
                                            if isinstance(z[ind2]['e-RAB-ID'], int) and isinstance(z[ind2]['gTP-TEID'], str):
                                                if not z[ind2]['e-RAB-ID'] in db:
                                                    db[z[ind2]['e-RAB-ID']] = {}

                                                db[z[ind2]['e-RAB-ID']]['MME-TEID'] = z[ind2]['gTP-TEID'].encode('hex')
            logger.debug('####################')
                                                logger.debug("Adding e-RAB-ID to db")
                                                logger.debug('####################')
            logger.debug('****')
                                                logger.debug(db)
            logger.debug('****')

```

```
#function to parse a packet
def parse_packet(packet) :

    #parse ethernet header
    eth_length = 14

    eth_header = packet[:eth_length]
    eth = unpack('!6s6sH' , eth_header)
    eth_protocol = socket.ntohs(eth[2])

    #Parse IP packets, IP Protocol number = 8
    if eth_protocol == 8 :
        logger.debug( "Packet with ETH Protocol 8")
        #Parse IP header
        #take first 20 characters for the ip header
        ip_header = packet[eth_length:20+eth_length]

        #now unpack them :)
        iph = unpack('!BBHHHBBH4s4s' , ip_header)

        version_ihl = iph[0]
        version = version_ihl >> 4
        ihl = version_ihl & 0xF

        iph_length = ihl * 4

        ttl = iph[5]
        protocol = iph[6]
        s_addr = socket.inet_ntoa(iph[8]);
        d_addr = socket.inet_ntoa(iph[9]);

        # logger.debug( 'Version : ' + str(version) + ' IP Header Length : ' + str(ihl) + ' TTL : ' + str(ttl) + ' Protocol : ' + str(protocol) + ' Source Address : ' + str(s_addr) + ' Destination Address : ' + str(d_addr))

        #UDP packets
        if protocol == 17 :
            logger.debug( "Packet with protocol 17")
            u = iph_length + eth_length
            udph_length = 8
            udp_header = packet[u:u+8]

            #now unpack them :)
            udph = unpack('!HHHH' , udp_header)

            source_port = udph[0]
            dest_port = udph[1]
            length = udph[2]
            checksum = udph[3]

            # logger.debug( 'Source Port : ' + str(source_port) + ' Dest Port : ' + str(dest_port) + ' Length : ' + str(length) + ' Checksum : ' + str(checksum))

            h_size = eth_length + iph_length + udph_length
            data_size = len(packet) - h_size

            #get data from the packet
            # 2152 GPRS

            if data_size > 8 and source_port == 2152 and dest_port == 2152:
```

```
            gprs_header = packet[h_size:h_size+8]
            gprsh = unpack('!BBH4s', gprs_header)

            gprs_flags = gprsh[0]
            gprs_type = gprsh[1]
            gprs_size = gprsh[2]
            gprs_teid = gprsh[3]

            # T-PDU = 0xff
            if gprs_type == 0xff:
                # logger.debug( "GPRS size: " + str(gprs_size))

                data_size = len(packet) - h_size - 8

                data = packet[h_size + 8:]

                if len(data) > 20:
                    inner_ip_header = data[0:20]

                    inner_iph = unpack('!BBHHHBBH4s4s' , inner_ip_header)

                    inner_version_ihl = inner_iph[0]
                    inner_version = inner_version_ihl >> 4
                    inner_ihl = inner_version_ihl & 0xF

                    inner_iph_length = inner_ihl * 4

                    inner_ttl = inner_iph[5]
                    inner_protocol = inner_iph[6]
                    inner_s_addr = socket.inet_ntoa(inner_iph[8]);
                    inner_d_addr = socket.inet_ntoa(inner_iph[9]);

                    # logger.debug( 'Inner source IP addr: ' + str(inner_s_addr) + ' Innder dest IP addr: ' + str (inner_d_addr) + 'Source IP addr: ' + str(s_addr) + ' Dest IP addr: ' + str (d_addr) + ' TEID: ' + gprs_teid.encode('hex'))

                    index = gprs_teid.encode('hex')

                    if not index in inner:
                        inner[index] = {}

                    inner[index]['inner_s_addr'] = inner_s_addr
                    inner[index]['inner_d_addr'] = inner_d_addr
                    inner[index]['s_addr'] = s_addr
                    inner[index]['d_addr'] = d_addr

  logger.debug( '****')
  logger.debug( db)
                    for key, value in inner.iteritems():
                        logger.debug( "%s: %s" % (key, value))
  logger.debug( '****')
  try:
    enb_teid = db[5]["ENB-TEID"]
    mme_teid = db[5]["MME-TEID"]
    return enb_teid, mme_teid
  except KeyError:
    logger.debug( "KeyError while parsing db content")
    pass
```

```

            # logger.debug( 'Data : ' + data)

        elif protocol == 132 :
            logger.debug( "Packet with protocol 132")
            # logger.debug( 'received sctp packet')
            u = iph_length + eth_length
            sctp_length = 12

            sctp_header = packet[u:u+12]
            sctph = unpack('!HH4s4s', sctp_header)

            source_port = sctph[0]
            dest_port = sctph[1]
            verification_tag = sctph[2].encode('hex')
            checksum = sctph[3].encode('hex')

            # logger.debug( 'Source port: ' + str(source_port) + ' Destinarion port: ' + str(dest_port) + ' Verification Tag: ' + '0x' + verification_tag + ' Checksum: ' + '0x' + checksum)

            u += sctp_length

            while len(packet) - u >= 4:
                chunk_header = packet[u:u+4]

                chknh = unpack('!BBH', chunk_header)

                chunk_type = chknh[0]
                chunk_flags = chknh[1]
                chunk_length = chknh[2]

                chunk_pad = 0

                if chunk_length % 4:
                    chunk_pad = 4 - chunk_length % 4

                #logger.debug( 'Chunk type: ' + str(hex(chunk_type)) + ' Chunk flags: ' + str(hex(chunk_flags)) + ' Chunk size: ' + str(chunk_length))

                # DATA = 0, data hader should be inside, chunk should fit a packet
                if chunk_type == 0 and u + 12 <= len(packet) and u + chunk_length <= len(packet):
                    chunk_data = packet[u+4:u+4+12]
                    chdth = unpack('!IHHI', chunk_data)

                    chunk_data_transmission_sequence_number = chdth[0]
                    chunk_data_stream_identifier = chdth[1]
                    chunk_data_stream_sequence_number = chdth[2]
                    chunk_data_payload_protocol_identifier = chdth[3]

                    logger.debug( 'Transmission Sequence Number' + str(chunk_data_transmission_sequence_number) + ' Stream Identifier: ' + str(chunk_data_stream_identifier) + 'Stream Sequence Number: ' + str(chunk_data_stream_sequence_number) + ' Payload Protocol Identifier: ' + str(chunk_data_payload_protocol_identifier))

                    if chunk_data_payload_protocol_identifier == 18:
                        buf=packet[u+4+12:u+chunk_length]
                        error = decode_string(buf)
                        if not error:
```

```
                            get_mme_gtp_teid()
                            get_enb_gtp_teid()

                u += chunk_length + chunk_pad

        #some other IP packet like IGMP
    return None, None

def trace_pkt(iface, vlevel=0):
    logger.setLevel(logging.ERROR)
    if vlevel == 1:
      logger.setLevel(logging.WARNING)
    if vlevel == 2:
      logger.setLevel(logging.INFO)
    if vlevel >= 3:
      logger.setLevel(logging.DEBUG)

    # Check access level
    if os.geteuid() != 0:
        exit("You need to have root privileges to run this script.\n"
             "Please try again, this time using 'sudo'.\n"
             "Exiting.")
    '''
    open device
    # Arguments here are:
    #   device
    #   snaplen (maximum number of bytes to capture _per_packet_)
    #   promiscious mode (1 for true)
    #   timeout (in milliseconds)
    '''
    cap = pcapy.open_live(iface, 65536 , 1 , 1000)
    # sniff(iface=iface, prn=pkt_callback)
    logger.debug( "Start packet capture now.")
    while(1) :
        try:
      (header, packet) = cap.next()
            # logger.debug( "Header: %s\nPacket: %s" % (type(header), type
    (packet)))
            # logger.debug( "Header: %s\nPacket: %s" % (str(header), str(
    packet)))
            (enb_teid, mme_teid) = parse_packet(packet)
      if enb_teid and mme_teid:
        return enb_teid, mme_teid
  except KeyboardInterrupt:
            logger.debug( "\nKeyboard Interrupt.\nShutting down.")
            sys.exit()
        except error as e:
      logger.debug(e)
            pass

if __name__ == "__main__":
  # Docker bridge docker-openairinterface-epc_epc
  iface = 'br-3cb0e0ddb8a9'
  trace_pkt(iface, 3)
```

Appendix E

List of tools in architecture

The following presents a detailed explanation of the different tools and their role in our implementation.

Name: OpenAirInterface
Description: The OpenAirInterface Software Alliance (OSA) is a non-profit consortium fostering a community of industrial as well as research contributors for open source software and hardware development for the core network (EPC), access network and user equipment (EUTRAN) of 3GPP cellular networks.
Usage: The OpenAirInterface implementation of EPC forms the basis of our architecture. We also use the OpenAirInterface Simulator for eNodeB and UE.
Source: https://www.openairinterface.org

Name: Docker
Description: Docker is probably the most popular container solution available today.
Usage: Docker containers are used to separate the different EPC elements from each other.
Source: https://www.docker.com

Name: Docker-Compose
Description: Compose is a tool for defining and running multi-container Docker applications.
Usage: Docker-Compose is used to manage the relation between the different containers. We also leverage the docker-compose command line to scale in/out by running several instances of a given docker container.
Source: https://docs.docker.com/compose/

Name: Python
Description: Python is a general-purpose programming language.
Usage: We use Python to write most of our balancing and sniffing functions.
Source: https://www.python.org

Name: cAdvisor (Container Advisor)
Description: cAdvisor provides container users an understanding of the resource usage and performance characteristics of their running containers.
Usage: cAdvisor is used to export performance metrices from individual containers and make them available to monitoring tools such as prometheus.
Source: https://github.com/google/cadvisor

Name: Prometheus
Description: Prometheus is a monitoring system and a time series database.
Usage: Prometheus does scrape the information exposed by cAdvisor. It is also used to querie performance metrices for later use by balance.py
Source: https://prometheus.io

Name: Grafana
Description: Grafana is an open source metric analytics and visualization suite.
Usage: Grafana is used to provide a graphical front end for the data collected in Prometheus. It is not directly used to determine whether a service should be scaled or not.
Source: https://grafana.com

Appendix F

Balancer.py logs

```
/home/setup/.local/lib/python2.7/site-packages/requests/__init__.py:83: RequestsDependencyWarning: Old version of cryptography ([1, 2, 3]) may cause slowdown.
  warnings.warn(warning, RequestsDependencyWarning)
Container(s) started
Starting oai_cadvisor ...
Starting oai_iperf3_server ...
Starting docker-openairinterface-epc_db_1 ...
Starting oai_prometheus                  ...
Starting oai_grafana                     ...
[5A[2K
Starting oai_cadvisor                    ... [32mdone[0m
[5B[3A[2K
Starting docker-openairinterface-epc_db_1 ... [32mdone[0m
[3BStarting docker-openairinterface-epc_balance_db_1 ...
Starting docker-openairinterface-epc_phpmyadmin_1 ...
[3A[2K
Starting oai_grafana                              ... [32mdone[0m
[3B[4A[2K
Starting oai_prometheus                           ... [32mdone[0m
[4B[6A[2K
Starting oai_iperf3_server                        ... [32mdone[0m
[6BStarting oai_iperf3_client                         ...
[3A[2K
Starting docker-openairinterface-epc_balance_db_1 ... [32mdone[0m
[3BStarting docker-openairinterface-epc_backend_hss_1 ...
[3A[2K
Starting docker-openairinterface-epc_phpmyadmin_1  ... [32mdone[0m
[3B[2A[2K
Starting oai_iperf3_client                         ... [32mdone[0m
[2B[1A[2K
Starting docker-openairinterface-epc_backend_hss_1 ... [32mdone[0m
[1BStarting docker-openairinterface-epc_hss_1          ...
[1A[2K
Starting docker-openairinterface-epc_hss_1         ... [32mdone[0m
[1BStarting docker-openairinterface-epc_mme_1          ...
[1A[2K
Starting docker-openairinterface-epc_mme_1         ... [32mdone[0m
[1BStarting docker-openairinterface-epc_spgw2_1        ...
Starting docker-openairinterface-epc_spgw_1        ...
[2A[2K
Starting docker-openairinterface-epc_spgw2_1       ... [32mdone[0m
[2B[1A[2K
Starting docker-openairinterface-epc_spgw_1        ... [32mdone[0m
[1B
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 30.
Traffic sent for image docker-openairinterface-epc_backend_hss: 16.
Memory usage for image docker-openairinterface-epc_backend_hss: 33162.
```

```
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 32.
Traffic sent for image docker-openairinterface-epc_backend_hss: 18.
Memory usage for image docker-openairinterface-epc_backend_hss: 37645.
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 33.
Traffic sent for image docker-openairinterface-epc_backend_hss: 19.
Memory usage for image docker-openairinterface-epc_backend_hss: 37900.
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 33.
Traffic sent for image docker-openairinterface-epc_backend_hss: 19.
Memory usage for image docker-openairinterface-epc_backend_hss: 37885.
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
```

```
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 33.
Traffic sent for image docker-openairinterface-epc_backend_hss: 19.
Memory usage for image docker-openairinterface-epc_backend_hss: 37876.
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 39.
Traffic sent for image docker-openairinterface-epc_backend_hss: 24.
Memory usage for image docker-openairinterface-epc_backend_hss: 37998.
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 39.
Traffic sent for image docker-openairinterface-epc_backend_hss: 24.
Memory usage for image docker-openairinterface-epc_backend_hss: 38000.
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 39.
Traffic sent for image docker-openairinterface-epc_backend_hss: 24.
Memory usage for image docker-openairinterface-epc_backend_hss: 37996.
```

```
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 39.
Traffic sent for image docker-openairinterface-epc_backend_hss: 24.
Memory usage for image docker-openairinterface-epc_backend_hss: 37988.
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 40.
Traffic sent for image docker-openairinterface-epc_backend_hss: 24.
Memory usage for image docker-openairinterface-epc_backend_hss: 37978.
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.
Balance service hss
Traffic received for image docker-openairinterface-epc_backend_hss: 40.
Traffic sent for image docker-openairinterface-epc_backend_hss: 24.
Memory usage for image docker-openairinterface-epc_backend_hss: 37970.
CPU usage for image docker-openairinterface-epc_backend_hss: 0.
Ideal size for hss is 1.
Balance service mme
Traffic received for image docker-openairinterface-epc_backend_mme: 0.
Traffic sent for image docker-openairinterface-epc_backend_mme: 0.
Memory usage for image docker-openairinterface-epc_backend_mme: 0.
CPU usage for image docker-openairinterface-epc_backend_mme: 0.
Ideal size for mme is 1.
Balance service spgw
```

```
Traffic received for image docker-openairinterface-epc_backend_spgw: 0.
Traffic sent for image docker-openairinterface-epc_backend_spgw: 0.
Memory usage for image docker-openairinterface-epc_backend_spgw: 0.
CPU usage for image docker-openairinterface-epc_backend_spgw: 0.
Ideal size for spgw is 1.

Shutting down...
Container(s) stopped
Stopping docker-openairinterface-epc_spgw2_1        ...
Stopping docker-openairinterface-epc_spgw_1         ...
Stopping docker-openairinterface-epc_mme_1          ...
Stopping docker-openairinterface-epc_hss_1          ...
Stopping docker-openairinterface-epc_backend_hss_1  ...
Stopping docker-openairinterface-epc_phpmyadmin_1   ...
Stopping docker-openairinterface-epc_balance_db_1   ...
Stopping docker-openairinterface-epc_db_1           ...
Stopping oai_iperf3_client                          ...
Stopping oai_cadvisor                               ...
Stopping oai_prometheus                             ...
Stopping oai_grafana                                ...
Stopping oai_iperf3_server                          ...
[2A[2K
Stopping oai_grafana                                ... [32mdone[0m
[2B[4A[2K
Stopping oai_cadvisor                               ... [32mdone[0m
[4B[8A[2K
Stopping docker-openairinterface-epc_phpmyadmin_1   ... [32mdone[0m
[8B[3A[2K
Stopping oai_prometheus                             ... [32mdone[0m
[3B[5A[2K
Stopping oai_iperf3_client                          ... [32mdone[0m
[5B[13A[2K
Stopping docker-openairinterface-epc_spgw2_1        ... [32mdone[0m
[13B[12A[2K
Stopping docker-openairinterface-epc_spgw_1         ... [32mdone[0m
[12B[1A[2K
Stopping oai_iperf3_server                          ... [32mdone[0m
[1B[11A[2K
Stopping docker-openairinterface-epc_mme_1          ... [32mdone[0m
[11B[10A[2K
Stopping docker-openairinterface-epc_hss_1          ... [32mdone[0m
[10B[9A[2K
Stopping docker-openairinterface-epc_backend_hss_1  ... [32mdone[0m
[9B[7A[2K
Stopping docker-openairinterface-epc_balance_db_1   ... [32mdone[0m
[7B[6A[2K
Stopping docker-openairinterface-epc_db_1           ... [32mdone[0m
[6B
```

Appendix G

Balancer.py logs

G.1 Traffic Test Bash Script

```
#!/bin/bash
# Simple utility to hand different options to iperf3
# Transmission time is 10 secs (iperf3 default)
# Bandwith 0 is unlimitted TCP traffic
#  Protocoll   is TCP (iperf3 default)

mode="$1"
help_string="""\
No mode provided. Possible options for mode are:
* full: Maximum throughput for a given time
* raise: Raise traffic in a determined interval
* fall: Start from high traffic and go down
* static: Raise, stay at a level and don't change
* fluctuate: Fluctuate around a given value to test balancing around a
    threashold
* raise_and_fall
"""

full=(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
raise=(100Mbit 200Mbit 300Mbit 400Mbit 500Mbit 600Mbit 700Mbit 800Mbit 900
    Mbit 1000Mbit 1100Mbit 1200Mbit 1300Mbit 1400Mbit 1500Mbit 1600Mbit
    1700Mbit 1800Mbit 1900Mbit 2000Mbit 2100Mbit 2200Mbit 2300Mbit 2400Mbit
     2500Mbit 2600Mbit 2700Mbit 2800Mbit 2900Mbit 3000Mbit)
fall=(300Mbit 2900Mbit 2800Mbit 2700Mbit 2600Mbit 2500Mbit 2400Mbit 2300
    Mbit 2200Mbit 2100Mbit 2000Mbit 1900Mbit 1800Mbit 1700Mbit 1600Mbit
    1500Mbit 1400Mbit 1300Mbit 1200Mbit 1100Mbit 1000Mbit 900Mbit 800Mbit
    700Mbit 600Mbit 500Mbit 400Mbit 300Mbit 200Mbit 100Mbit)
static=(1000Mbit 1500Mbit 2000Mbit 2500Mbit 3000Mbit 3000Mbit 3000Mbit
    3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit
     3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000
    Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit 3000Mbit)
fluctuate=(1000Mbit 1500Mbit 2000Mbit 2500Mbit 3000Mbit 2500Mbit 3000Mbit
    2500Mbit 3000Mbit 2500Mbit 3000Mbit 2500Mbit 3000Mbit 2500Mbit 3000Mbit
     2500Mbit 3000Mbit 2500Mbit 3000Mbit 2500Mbit 3000Mbit 2500Mbit 3000
    Mbit 2500Mbit 3000Mbit 2500Mbit 3000Mbit 2500Mbit 3000Mbit 2500Mbit)
raise_and_fall=(100Mbit 200Mbit 300Mbit 400Mbit 500Mbit 600Mbit 700Mbit
    800Mbit 900Mbit 1000Mbit 1100Mbit 1200Mbit 1300Mbit 1400Mbit 1500Mbit
    1600Mbit 1700Mbit 1800Mbit 1900Mbit 2000Mbit 2100Mbit 2200Mbit 2300Mbit
     2400Mbit 2500Mbit 2600Mbit 2700Mbit 2800Mbit 2900Mbit 3000Mbit 300Mbit
     2900Mbit 2800Mbit 2700Mbit 2600Mbit 2500Mbit 2400Mbit 2300Mbit 2200
    Mbit 2100Mbit 2000Mbit 1900Mbit 1800Mbit 1700Mbit 1600Mbit 1500Mbit
    1400Mbit 1300Mbit 1200Mbit 1100Mbit 1000Mbit 900Mbit 800Mbit 700Mbit
    600Mbit 500Mbit 400Mbit 300Mbit 200Mbit 100Mbit)

full(){
```

```
    for i in "${full[@]}";do
        cmd="/usr/bin/iperf3 -c iperf3_server -b $i"
        echo "###### $cmd ######"; $cmd
    done
}

raise(){
    for i in "${raise[@]}";do
        cmd="/usr/bin/iperf3 -c iperf3_server -b $i"
        echo "###### $cmd ######"; $cmd
    done
}

fall(){
    for i in "${fall[@]}";do
        cmd="/usr/bin/iperf3 -c iperf3_server -b $i"
        echo "###### $cmd ######"; $cmd
    done
}

static(){
    for i in "${static[@]}";do
        cmd="/usr/bin/iperf3 -c iperf3_server -b $i"
        echo "###### $cmd ######"; $cmd
    done
}

fluctuate(){
    for i in "${fluctuate[@]}";do
        cmd="/usr/bin/iperf3 -c iperf3_server -b $i"
        echo "###### $cmd ######"; $cmd
    done
}

raise_and_fall(){
    for i in "${raise_and_fall[@]}";do
        cmd="/usr/bin/iperf3 -c iperf3_server -b $i"
        echo "###### $cmd ######"; $cmd
    done
}

if [[ "$mode" == "full" ]];then
    full
elif [[ "$mode" == "raise" ]];then
    raise
elif [[ "$mode" == "fall" ]];then
    fall
elif [[ "$mode" == "static" ]];then
    static
elif [[ "$mode" == "fluctuate" ]];then
    fluctuate
elif [[ "$mode" == "raise_and_fall" ]];then
    raise_and_fall
else
    echo "$help_string"
fi
```

G.2 Docker Instance Counter

```
#!/bin/bash

instances=0
```

```
while true;do
  inst="$(docker ps | grep 'spgw2' | wc -l | tr -d '\n')"
  if [[ "$instances" != "$inst" ]];then
    instances="$inst"
    echo -n "$instances "
    date +%H:%M:%S
  fi
done
```

Bibliography

[1] Cisco, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast, 2017–2022*, Feb. 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-forecast-qa.pdf.

[2] R. Cziva and D. P. Pezaros, "Container Network Functions: Bringing NFV to the Network Edge", *IEEE Communications Magazine*, vol. 55, no. 6, pp. 24–31, 2017, ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1601039.

[3] ETSI, *Network Functions Virtualisation (White Paper)*. 2012. [Online]. Available: https://portal.etsi.org/nfv/nfv_white_paper.pdf.

[4] R. Cziva, S. Jouet, K. J. S. White, and D. P. Pezaros, "Container-based network function virtualization for software-defined networks", *2015 IEEE Symposium on Computers and Communication (ISCC)*, pp. 415–420, Jul. 2015. DOI: 10.1109/ISCC.2015.7405550.

[5] C. H. T. Arteaga, F. Rissoi, and O. M. C. Rendon, "An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an NFV-based EPC", *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–7, Nov. 2017. DOI: 10.23919/CNSM.2017.8255982.

[6] *Glasgow Network Functions (GNF)*. [Online]. Available: https://netlab.dcs.gla.ac.uk/projects/glasgow-network-functions.

[7] *Open EPC*. [Online]. Available: https://www.openepc.com.

[8] *OpenAirInterface MME Configuration*. [Online]. Available: https://gitlab.eurecom.fr/oai/openair-cn/blob/develop/etc/mme.conf.

[9] B. Sousa, L. Cordeiro, P. Simões, A. Edmonds, S. Ruiz, G. A. Carella, M. Corici, N. Nikaein, A. S. Gomes, E. Schiller, T. Braun, and T. M. Bohnert, "Toward a Fully Cloudified Mobile Network Infrastructure", *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 547–563, 2016, ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2598354.

[10] R. Cziva, S. Jouët, D. Stapleton, F. P. Tso, and D. P. Pezaros, "SDN-Based Virtual Machine Management for Cloud Data Centers", *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 212–225, 2016, ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2528220.

[11] U. Varshney, "4g Wireless Networks", *IT Professional*, vol. 14, no. 5, pp. 34–39, 2012, ISSN: 1520-9202. DOI: 10.1109/MITP.2012.71.

[12] *LTE Tutorial*. [Online]. Available: https://www.tutorialspoint.com/lte/index.htm.

[13] T. Taleb, M. Corici, C. Parada, A. Jamakovic, S. Ruffino, G. Karagiannis, and T. Magedanz, "EASE: EPC as a service to ease mobile core network deployment over cloud", *IEEE Network*, vol. 29, no. 2, pp. 78–88, 2015, ISSN: 0890-8044. DOI: 10.1109/MNET.2015.7064907.

[14] A. Jain, N. S. Sadagopan, S. K. Lohani, and M. Vutukuru, "A comparison of SDN and NFV for re-designing the LTE Packet Core", *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 74–80, Nov. 2016. DOI: 10.1109/NFV-SDN.2016.7919479.

[15] *OpenAirInterface – 5g software alliance for democratising wireless innovation*, en-US. [Online]. Available: https://www.openairinterface.org/ (visited on 08/02/2019).

[16] Google, *cAdvisor*. [Online]. Available: https://github.com/google/cadvisor.

[17] *Prometheus*. [Online]. Available: https://prometheus.io.

[18] *OpenAirLTEEmulation · Wiki · oai / openairinterface5g*, en. [Online]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/OpenAirLTEEmulation (visited on 07/02/2019).

[19] *Iperf*. [Online]. Available: https://iperf.fr.

[20] F. Lobillo, Z. Becvar, M. A. Puente, P. Mach, F. L. Presti, F. Gambetti, M. Goldhamer, J. Vidal, A. K. Widiawan, and E. Calvanesse, "An architecture for mobile computation offloading on cloud-enabled LTE small cells", *2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 1–6, Apr. 2014. DOI: 10.1109/WCNCW.2014.6934851.

[21] lionelgo, *Contribute to OPENAIRINTERFACE/openair-cn development by creating an account on GitHub*, original-date: 2017-09-08T14:46:56Z, May 2019. [Online]. Available: https://github.com/OPENAIRINTERFACE/openair-cn (visited on 05/21/2019).

[22] E. Schiller, "MEC Caching Prototype", ICT - Information and Communication Technologies, Tech. Rep., Jan. 2017.

[23] Wikipedia, *Network function virtualization*. [Online]. Available: https://en.wikipedia.org/wiki/Network_function_virtualization.

[24] sdx, "What is NFV – Network Functions Virtualization – Definition?",

[25] S. Dutta, T. Taleb, and A. Ksentini, "QoE-aware elasticity support in cloud-native 5g systems", *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2016. DOI: 10.1109/ICC.2016.7511377.

[26] sdx, *Which is Better – SDN or NFV?* [Online]. Available: https://www.sdxcentral.com/networking/nfv/definitions/which-is-better-sdn-or-nfv/.

[27] J. Burke, *What is SDN? The answer now includes automation and virtualization*. [Online]. Available: https://searchnetworking.techtarget.com/tip/What-is-SDN-The-answer-now-includes-automation-and-virtualization.

[28] P. Pate, "NFV and SDN: What's the Difference?", 2013. [Online]. Available: https://www.sdxcentral.com/articles/contributed/nfv-and-sdn-whats-the-difference/2013/03/.

[29] P. Brey, "Containers vs. Virtual Machines (VMs): What's the Difference?", 2018. [Online]. Available: https://blog.netapp.com/blogs/containers-vs-vms/.

[30] *LXC (Linux Containers)*. [Online]. Available: https://en.wikipedia.org/wiki/LXC.

[31] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes", *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014, ISSN: 2325-6095. DOI: 10.1109/MCC.2014.51.

[32] *The Official YAML Web Site*. [Online]. Available: https://yaml.org/ (visited on 06/24/2019).

[33] Y. Grunenberger, *Dockerfile for EPC Database*. [Online]. Available: https://github.com/ravens/docker-openairinterface-epc/blob/master/db/Dockerfile.

[34] *Dockerfile reference*. [Online]. Available: https://docs.docker.com/engine/reference/builder/#usage.

[35] A. M. Joy, "Performance comparison between Linux containers and virtual machines", *2015 International Conference on Advances in Computer Engineering and Applications*, pp. 342–346, Mar. 2015. DOI: 10.1109/ICACEA.2015.7164727.

[36] T. Wei, M. Malhotra, B. Gao, T. Bednar, D. Jacoby, and Y. Coady, "No such thing as a "free launch"? Systematic benchmarking of containers", *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pp. 1–6, Aug. 2017. DOI: 10.1109/PACRIM.2017.8121922.

[37] S. S. Tadesse, C. F. Chiasserini, and F. Malandrino, "Characterizing the power cost of virtualization environments", *Transactions on Emerging Telecommunications Technologies*, vol. 0, no. 0, e3462, 2018, ISSN: 2161-3915. DOI: 10.1002/ett.3462.

[38] *gVisor*, en. [Online]. Available: https://gvisor.dev/ (visited on 07/29/2019).

[39] *gVisor - Container Runtime Sandbox*. [Online]. Available: https://github.com/google/gvisor.

[40] C. Ltd., *Introduction to Juju Charms*. [Online]. Available: https://docs.jujucharms.com/2.0/en/charms.

[41] P. Tang, F. Li, W. Zhou, W. Hu, and L. Yang, "Efficient Auto-Scaling Approach in the Telco Cloud Using Self-Learning Algorithm", *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Dec. 2015. DOI: 10.1109/GLOCOM.2015.7417181.

[42] G. A. Carella, M. Pauls, L. Grebe, and T. Magedanz, "An extensible Autoscaling Engine (AE) for Software-based Network Functions", *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 219–225, Nov. 2016. DOI: 10.1109/NFV-SDN.2016.7919501.

[43] X. Wang, C. Wu, F. Le, A. Liu, Z. Li, and F. Lau, "Online VNF Scaling in Datacenters", *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pp. 140–147, Jun. 2016. DOI: 10.1109/CLOUD.2016.0028.

[44] *Open Baton*. [Online]. Available: http://openbaton.github.io/index.html.

[45] ETSI, *MANO - Network Functions Virtualizatio*. [Online]. Available: http://network-functions-virtualization.com/mano.html.

[46] J. Nakajima, "Building High-Performance NFV Solutions Using Containers", 2015. [Online]. Available: https://events.static.linuxfound.org/sites/events/files/slides/Jun_Nakajima_NFV_Container_final.pdf.

[47] *Using Minikube to Create a Cluster*, en. [Online]. Available: https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/ (visited on 04/28/2019).

[48] *Viewing Pods and Nodes*, en. [Online]. Available: https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/ (visited on 04/28/2019).

[49] *Kubernetes service can't support SCTP protocol*. [Online]. Available: https://github.com/kubernetes/kubernetes/issues/44485.

[50] *freeDiameter - Requirements*. [Online]. Available: http://www.freediameter.net/trac/wiki/Requirements.

[51] ETSI, "TS 132 455; Telecommunication management; Key Performance Indicators (KPI) for the Evolved Packet Core (EPC); Definitions (3gpp TS 32.455 version 11.0.0 Release 11)", Tech. Rep., 2012.

[52] M. Lauridsen, I. Rodriguez, L. M. Mikkelsen, L. C. Gimenez, and P. Mogensen, "Verification of 3g and 4g received power measurements in a crowdsourcing Android app", *2016 IEEE Wireless Communications and Networking Conference*, pp. 1–6, Apr. 2016. DOI: 10.1109/WCNC.2016.7564930.

[53] H. Mfula and J. K. Nurminen, "Adaptive Root Cause Analysis for Self-Healing in 5g Networks", *2017 International Conference on High Performance Computing & Simulation (HPCS)*, pp. 136–143, Jul. 2017. DOI: 10.1109/HPCS.2017.31.

[54] M. Laner, P. Svoboda, P. Romirer-Maierhofer, N. Nikaein, F. Ricciato, and M. Rupp, "A comparison between one-way delays in operating HSPA and LTE networks", *2012 10th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, pp. 286–292, May 2012.

[55] 3GPP, "TS 25.913, Requirements for Evolved UTRA and Evolved UTRAN", Tech. Rep.

[56] A. Hafsaoui, N. Nikaein, and L. Wang, "OpenAirInterface Traffic Generator (OTG): A Realistic Traffic Generation Tool for Emerging Application Scenarios", *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 492–494, Aug. 2012, ISSN: 1526-7539. DOI: 10.1109/MASCOTS.2012.62.

[57] M. Lauridsen, L. C. Gimenez, I. Rodriguez, T. B. Sorensen, and P. Mogensen, "From LTE to 5g for Connected Mobility", *IEEE Communications Magazine*, vol. 55, no. 3, pp. 156–162, 2017, ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1600778CM.

[58] *Jaeger*. [Online]. Available: https://www.jaegertracing.io.

[59] V. Medel, O. Rana, J. Bañares, and U. Arronategui, "Modelling Performance & Resource Management in Kubernetes", Dec. 2016, pp. 257–262.

[60] K. Han, S. Li, S. Tang, H. Huang, S. Zhao, G. Fu, and Z. Zhu, "Application-Driven End-to-End Slicing: When Wireless Network Virtualization Orchestrates with NFV-based Mobile Edge Computing", *IEEE Access*, pp. 1–1, 2018. DOI: 10.1109/ACCESS.2018.2834623.

[61] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kuklinski, and T. Rasheed, "Programming Abstractions for Software-Defined Wireless Networks", *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 146–162, 2015, ISSN: 1932-4537. DOI: 10.1109/TNSM.2015.2417772.

[62] L. Cui, F. P. Tso, and W. Jia, "Enforcing network policy in heterogeneous network function box environment", *Computer Networks*, vol. 138, pp. 108–118, Jun. 2018, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.03.029.

[63] T. B. Meriem and R. Chaparadza, "GANA - Generic Autonomic Networking Architecture (White Paper)", 2016, ISSN: 979-10-92620-10-8. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp16_gana_Ed1_20161011.pdf.

[64] *An Introduction to the ELK Stack for Logs and Metrics*. [Online]. Available: https://www.elastic.co/webinars/introduction-elk-stack.

[65] *Cloud Native Computing Foundation*. [Online]. Available: https://www.cncf.io.

[66] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, and T. Braun, "Network Slices toward 5g Communications: Slicing the LTE Network", *IEEE Communications Magazine*, vol. 55, no. 8, pp. 146–154, 2017, ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1600936.

[67] R. Gupta, *Oaiindocker*. [Online]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/OAIinDocker.

[68] Y. Grunenberger, *Docker-openairinterface-epc*. [Online]. Available: https://github.com/ravens/docker-openairinterface-epc.

[69] B. Aschwanden, *Pull request: Add startup script for docker container*. [Online]. Available: https://github.com/ravens/docker-openairinterface-epc/pull/2.

[70] *Dockercloud/haproxy - Docker Hub*. [Online]. Available: https://hub.docker.com/r/dockercloud/haproxy/ (visited on 06/27/2019).

[71] B. Christner, *How to scale Docker Containers with Docker-Compose*. [Online]. Available: https://www.brianchristner.io/how-to-scale-a-docker-container-with-docker-compose/.

[72] *Nginx Docker Container*. [Online]. Available: https://hub.docker.com/_/nginx/.

[73] W. Yuan, H. Sun, X. Wang, and X. Liu, "Towards Efficient Deployment of Cloud Applications through Dynamic Reverse Proxy Optimization", *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pp. 651–658, Nov. 2013. DOI: 10.1109/HPCC.and.EUC.2013.97.

[74] *Module ngx_stream_core_module*. [Online]. Available: http://nginx.org/en/docs/stream/ngx_stream_core_module.html.

[75] P. Mell and T. Grance, "The NIST Definition of Cloud Computing", *Recommendations of the National Institute of Standards and Technology*, vol. Special Publication, no. 800-145, 2011.

[76] G. Galante and L. C.E. d. Bona, "A Survey on Cloud Computing Elasticity", *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, pp. 263–270, Nov. 2012. DOI: 10.1109/UCC.2012.30.

[77] C. Roux, *Cmake_targets/tools/perf_oai.bash · master · oai / openairinterface5g*, en. [Online]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g/blob/master/cmake_targets/tools/perf_oai.bash (visited on 05/31/2019).

[78] E. Schiller, *FIRE LTE testbeds for open experimentation. Contribute to ejschiller/FLEX development by creating an account on GitHub*, original-date: 2017-01-27T15:07:00Z, Feb. 2017. [Online]. Available: https://github.com/ejschiller/FLEX (visited on 07/07/2019).

[79] *Docker build*, en, Jul. 2019. [Online]. Available: https://docs.docker.com/engine/reference/commandline/build/ (visited on 07/07/2019).

[80] M. Mao and M. Humphrey, "A Performance Study on the VM Startup Time in the Cloud", en, in *2012 IEEE Fifth International Conference on Cloud Computing*, Honolulu, HI, USA: IEEE, Jun. 2012, pp. 423–430, ISBN: 978-1-4673-2892-0 978-0-7695-4755-8. DOI: 10.1109/CLOUD.2012.103. [Online]. Available: http://ieeexplore.ieee.org/document/6253534/ (visited on 07/07/2019).

[81] *How to Connect OAI eNB (USRP B210) with COTS UE*. [Online]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/HowToConnectCOTSUEwithOAIeN

[82] *OpenAirInterface HSS Configuration*. [Online]. Available: https://gitlab.eurecom.fr/oai/openair-cn/blob/develop/etc/hss_fd.conf.

[83] P. Biondi, *Scapy*, en. [Online]. Available: https://secdev.github.io/ (visited on 07/02/2019).