ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCHOOL OF ENGINEERING AND ARCHITECTURE

MASTER'S DEGREE
IN
TELECOMMUNICATIONS ENGINEERING

OPEN RAN SOLUTIONS FOR MOBILE NETWORK AUTOMATION SCENARIOS

Master Thesis
in
Laboratory of Networking M

Supervisor
Prof. WALTER CERRONI

Candidate
BEGÜM GÖKTENAY GÜZEL

Co-supervisor
Ing. FRANCESCO FORESTA

SESSION III
ACADEMIC YEAR 2021/2022

Contents

Abstract

The softwarization and virtualization in mobile networks have been gaining importance as they provide high flexibility, scalability, agility, and cost savings. Hence, the world shifts in this direction at a fast pace. For the purpose of transforming the industry into open, intelligent, virtualized, and interoperable mobile networks, Open Radio Access Network (O-RAN) framework is a concept based on openness and intelligence, with a goal to provide standards for white-box hardware and open source software elements. Gaining momentum in the telecommunications industry in recent years, O-RAN aims to define Open RAN architecture integrated with a modular base station software stack. This thesis aims to investigate the architecture and solutions proposed by O-RAN Alliance. The work focuses on Open Networking Foundation Software Defined RAN (ONF SD-RAN) platform as well as the procedure and challenges of deploying an open network operating system, micro-ONOS, using virtualization platforms such as Docker and container orchestration system like Kubernetes.

Chapter 1

Introduction

Communication technologies have been shaping the world and society for the last 40 years. In order to provide for the increasing demand, as well as for new services and applications emerging continuously, approximately every 10 years, a new generation of cellular standards is being defined [1]. With every generation there have been modifications and transformations regarding the structure and deployment strategies. The transformation in the operator deployment strategies has already sparked significant changes in the cellular core towards a disaggregated, user plane and control plane separated, services-oriented architecture. These allow for dynamic creation and life cycle management of use case optimized network slices. Nonetheless, to a large extent, the radio access network (RAN) has remained untouched until recently. However, this changes with emerging technologies as operators migrate to a more open, disaggregated network. RAN disaggregation is about moving away from mobile networks based on costly monolithic tech and splitting out functions, especially those that can be implemented in software as virtualized functions. CU, DU and RU disaggregation can be seen in Figure 1.1.

In the last few years, the focus on softwarization and virtualization has increased as they are highly considered in mobile networks. Running a functionality in software rather than in hardware assures a high degree of flexibility and reconfigurability. Virtualization, in addition, boosts the split of the software and hardware, which means software runs in COTS (commercial-off-the-shelf) servers by using a virtual machine (VM) instead of hardware. From a network point of view, the main realization of the softwarization concept is Software Defined Networking (SDN), whereas, it is Network Function Virtual-

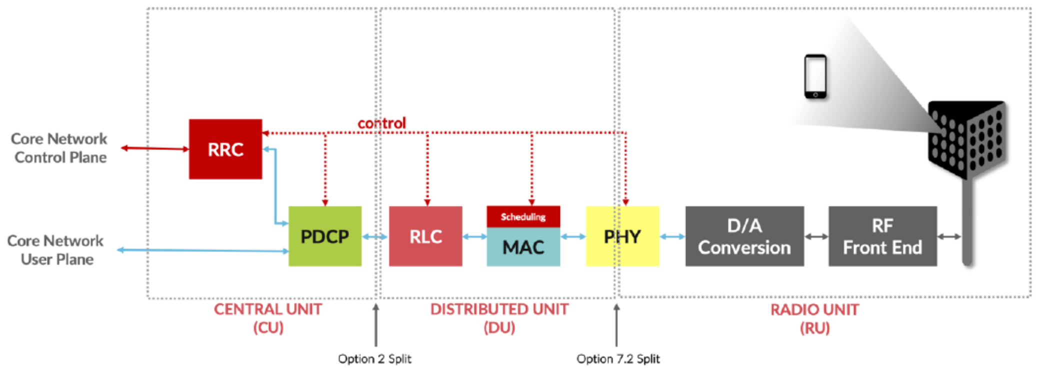

Figure 1.1: Disaggregated CU, DU and RU [1]

ization (NFV) for virtualization. Together, they provide the flexibility needed in 4G and 5G networks [2].

Virtualized radio access networks (vRAN) are an approach to run base band functions as software by virtualizing the functions of a traditional RAN and offering them on flexible and scalable cloud platforms. This reduces CAPEX and OPEX greatly and helps the networks provide the required performance and use cases of the 5G era [3]. Thus, vRAN, Software Defined Radio (SDR), and SDN have become increasingly important in the world of networking providing flexibility, reconfigurability, and programmability in order to support heterogeneous 5G use cases [2]. vRAN has been seen as the next step in the cellular network as it separates software and hardware. However, it might be challenging to keep the promise of flexibility and cost efficiency as more resources are required, and acceleration technologies are expensive in some cases [4]. This can be overcome by O-RAN's approach of providing the aimed performance without sacrificing flexibility and cost efficiency.

Driven by the emerging need of transparent, open and programmable communications O-RAN architecture offers openness and intelligence to RANs by migrating it into a virtualized environment. This enables economies of scale by using O-RAN's open source components, interfaces, and orchestrators, which are all hosted on the standardized O-Cloud hosting platform. Vendors and providers that want to benefit from O-RAN architecture are free to use open-source VNFs such as O-RAN Central Unit (O-CU), O-RAN Distributed Unit (O-DU), Near-Real Time RAN Intelligent Controller (Near-RT RIC) as long as

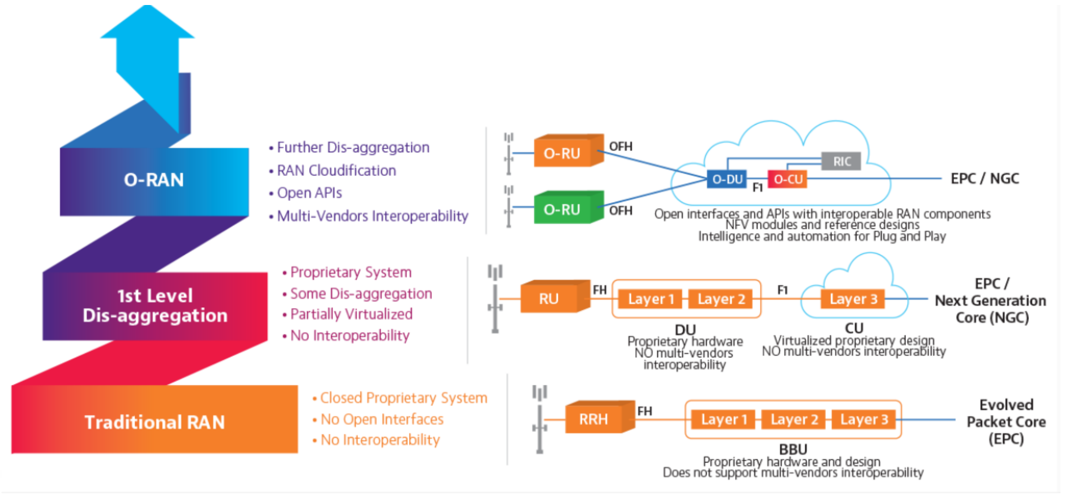
O-RAN
• Further Dis-aggregation
• RAN Cloudification
• Open APIs
• Multi-Vendors Interoperability
O-RU
OFH
O-RU
OFH
O-DU
F1
O-CU
RIC
EPC / NGC
Open interfaces and APIs with interoperable RAN components
NFV modules and reference designs
Intelligence and automation for Plug and Play
1st Level Dis-aggregation
• Proprietary System
• Some Dis-aggregation
• Partially Virtualized
• No Interoperability
RU
FH
Layer 1
Layer 2
F1
Layer 3
EPC / Next Generation Core (NGC)
DU
Proprietary hardware
NO multi-vendors interoperability
CU
Virtualized proprietary design
NO multi-vendors interoperability
Traditional RAN
• Closed Proprietary System
• No Open Interfaces
• No Interoperability
RRH
FH
Layer 1
Layer 2
Layer 3
Evolved Packet Core (EPC)
BBU
Proprietary hardware and design
Does not support multi-vendors interoperability

Figure 1.2: High Level Design of O-RAN [7]

they are compliant with the O-RAN hardware specifications. This erases the boundaries between different hardware and software vendors and providers, allowing RANs to extend easily and effortlessly to new services. Together with virtualization, O-RAN functions can be disaggregated from their hardware by network service providers, which enables dynamic instantiating of services and reducing their capital and operating costs [5].

In other words, as a game changer in mobile communications, the aim of O-RAN is to create a multi supplier RAN solution that allows for the separation or disaggregation between hardware and software. It is about disaggregated RAN functionality built by open interface specifications between building blocks, and can be implemented in hardware and software based on open interfaces and standards developed by the O-RAN community [6]. Figure 1.2 shows a high level design of O-RAN, starting from traditional RAN through first level disaggregation.

The work presented in this thesis aims to investigate the O-RAN scenarios for network automation, proposed solutions, and deployment procedure. Starting from the current architecture of O-RAN, with the help of open source documentations, specifications and use cases, available solutions provided by O-RAN community will be examined. Then, practical aspects of the deployment will be studied, along with the issues encountered while implementing the required systems, tools and components of the framework. With the motivation of exploring O-RAN, its maturity and success in implementing the proposed platforms and promised innovations, the main objective is to focus on evaluating the current status of the possible solutions and deployment procedure.

In Chapter 2 an overview and architecture of O-RAN will be presented. A few O-RAN key components, such as frameworks, functions and interfaces will be explained. In Chapter 3 the solutions and projects that were proposed by O-RAN will be investigated in detail. The focus will be on Open Network Foundation Software Define Networking RAN (ONF SD-RAN) platform, explaining its structure and deployment procedure. Chapter 4 will provide the detailed steps of the deployment of elements necessary for chosen platform i.e., micro-ONOS. This chapter will also include Docker, Kubernetes, Helm and micro-services of micro-ONOS, along with the problems faced during the deployment procedure. Finally, in Chapter 5, deployment procedure and issues that are encountered will be discussed.

Chapter 2

Open RAN

2.1 Overview

A new alliance named O-RAN strives to re-shape Radio Access Networks to be more intelligent, open, virtualized and fully interoperable. A few years ago ORAN which stands for Open Radio Access Network was born. It is a non-proprietary version of RAN and aims to increase interoperability between vendors, flexibility, and standardization of RAN elements. Founded by AT&T, China Mobile, Deutsche Telekom, NTT DOCOMO, and Orange in Germany, O-RAN aims to transform the RAN industry into an open and more intelligent mobile network, O-RAN is gaining momentum across the mobile industry [8].

O-RAN aims to lead the industry to open, interoperable interfaces, Big Data and AI enable RAN intelligence and maximize the use of COTS and minimize proprietary hardware [9]. The proprietary remote radio head (RRH) and baseband units (BBUs) are now disaggregated to radio units (O-RUs), distributed units (O-DUs), and central units (O-CUs) such that many of them can be virtualized or containerized, and the interfaces between these components are open and interoperable. ORAN's help to service providers can be summarized in three categories: reducing CAPEX and OPEX, increasing network efficiency and performance, and excellent agility.

To achieve the stated goals, O-RAN Alliance is actively working on three main streams which are Specification effort, O-RAN Software Community (OSC) and Test & Integration. O-RAN Specifications aim to lead the industry towards Open and Intelligent RAN by extending RAN standards, these specifications are available for public use. O-RAN Software Community supports the software development for O-RAN solutions available to everyone. Partnering with O-RAN Alliance and Linux Foundation, OSC aims to align with O-RAN architecture and specifications created and defined by O-RAN Alliance, in order to deliver a software solution for open and intelligent 5G RAN. Lastly Testing & Integration supports member companies in their O-RAN implementations, and defines the overall approach of O-RAN Alliance to testing and integration. This is realized by taking into account test and integration specifications, approaches to meet requirements. There are Open Testing & ıntegration Centers (OTIC) providing a collaborative, open and vendor-independent environments to support, demonstrate, test and verify the implementations and solutions based on O-RAN specifications.

Open RAN is the concept for a more open and flexible radio access network (RAN) architecture than the one which is used today [10]. It is a term used for industry-wide standards for RAN interfaces that brings interoperability between vendors' equipment and increase network flexibility at a lower cost. It offers interoperability standard for RAN element including hardware and software. Before going into details, it must be known that ORAN can also be used to refer to Open RAN movement but O-RAN with the hyphen refers to the O-RAN Alliance [11]. O-RAN Alliance has been founded in February 2018 by AT&T, China Mobile, Deutsche Telekom, NTT DOCOMO and Orange. It has been established as a German entity in August 2018. Since then, O-RAN ALLIANCE has become a world-wide community of mobile network operators, vendors, and research academic institutions operating in the RAN industry [8]. O-RAN Alliance aims to transform the RAN industry into something more open, interoperable, virtualized, and intelligent. Another objective of O-RAN is maximizing the usage of COTS hardware and minimizing proprietary ones [9]. O-RAN architecture and interface specifications shall be consisted with 3GPP as much as possible.

Next generation 5G and 6G networks are transforming inflexible and monolithic systems into flexible, agile and disaggregated architectures to support service heterogeneity, interoperability, and fast deployments. These transformations are enabled by O-RAN framework. The O-RAN based 5G and future 6G networks' goal is to integrate artificial intelligence into the deployment, operation, and maintenance of the network. The O-RAN architecture aims to offer a new dawn of services and applications for cellular networks, such as virtual network slices and dynamic spectrum sharing. O-RAN takes advantage of Cloud RAN principles and leverages the software-defined implementations of wireless communications and networking functions. Instead of vendor-specific legacy interfaces that are controlled by major industry entities, it defines open interfaces and an open architecture for innovation at all layers. It assumes that cellular network management will be more and more data driven and establishes the generic modules and interfaces for data collection, distribution, and processing accordingly [12].

O-RAN is based on ongoing research on open and intelligent networks. Empowered by the principles of openness and intelligence, O-RAN is the foundation for transformation into virtualized RAN on open hardware and cloud with AI-powered radio control embedded. RAN disaggregation splits base stations into functional units, hence providing the disaggregation model proposed by 3GPP for NR Next Generation Node Bases (gNBs). The gNBs are split into Central Unit (O-CU), Distributed Unit (O-DU), and a Radio Unit (O-RU) in O-RAN specifications. The CU is then split into Control Plane (CP) and User Plane (UP). This logical split allows the deployment of different functionalities at different locations in the network, or hardware platforms. The split options proposed by 3GPP for O-RU and O-DU were evaluated by O-RAN Alliance and it was found that 7.2x split yields a balance between simplicity of RU and data rates, and latency between O-RU and O-DU. O-RU only performs Fast Fourier Transform (FFT) and cyclic prefix addition and removal operations which makes it inexpensive and easily deployed. O-DU manages the remaining functionalities of physical layer, the Medium Access Control (MAC), and the Radio Link Control (RLC) layers. O-CU units i.e., CP and UP, implement the hgigher layers of 3GPP stack which are Radio Resource Control (RRC) layer managing the life cycle of connection, and the Service Data Adaptation Protocol Layer (SDAP) handling the Quality of Service (QoS) of the traffic flows, and also the Packet Data Convegence Protocol (PDCP) layer which controls reordering, packet duplication, and encryption for the air interface [13]. The evolution of traditional black-box base station architecture towards a virtualized gNBs with a functional split can be seen in Figure 2.1.

The O-RAN specifications have been divided into technical Working Groups (WG), all under the supervision of the Technical Steering Committee. Each of the WGs covers a part of the O-RAN Architecture and is open to all Members and Contributors. O-RAN Focus Groups (FG) work with the topics involving technical WGss, and they are relevant for the whole organization. The center architecture of O-RAN with corresponding working groups and focus groups can be seen in Figure 2.2

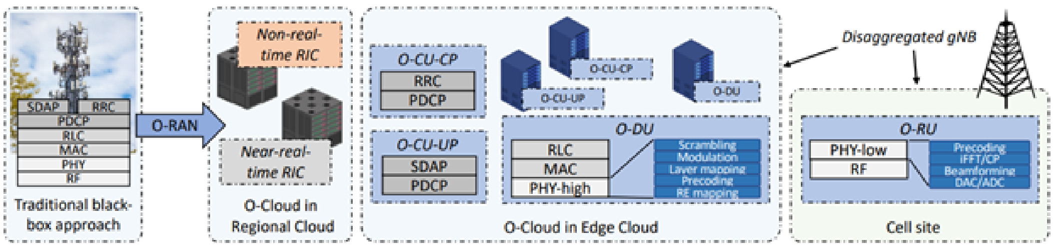

Figure 2.1: Evolution of Disaggregation in O-RAN [13]

2.2 O-RAN Architecture

The logical architecture of O-RAN is given as in Figure 2.3

O-RAN architecture consists of the following Service Management and Orchestration (SMO) framework containing Non-RT RIC network function, A1, O1, O2 and Open Fronhaul M-plane interfaces, O-RAN network functions (Near-RT RIC, O-CU-CP, O-CU-UP, O-DU) and O-Cloud. It also has three control loops involving different O-RAN functions. These are Non-RT, Near-RT and RT control loops. The components of O-RAN such as RIC functions, frameworks, interfaces and control loops are listed and explained below.

- SMO: this framework is responsible for RAN domain management in O-RAN architecture. To provide RAN support, SMO maintains FCAPS (Fault, Configuration, Accounting, Performance, Security) interfaced to O-RAN network functions, and Non-RT RIC for RAN optimization. In addition, SMO provides O-Cloud management, and supports the orchestration of platform, application elements, and workflow management. These services are performed via four key interfaces of O-RAN which are A1, O1, Open Fronthaul M-plane and O2. [9] SMO and Non-RT RIC services can be seen in Figure 2.4.

- A1: It is the interface between Non-RT RIC in SMO and Near-RT RIC functions and it supports three types of services: policy management service (for the fulfilment of RAN intent), enrichment information (for RAN optimization) and ML model management. A1 interface policies are not critical to traffic, have temporary validity. It is non-persistent which means it does not survive a restart of the Near-RT RIC.

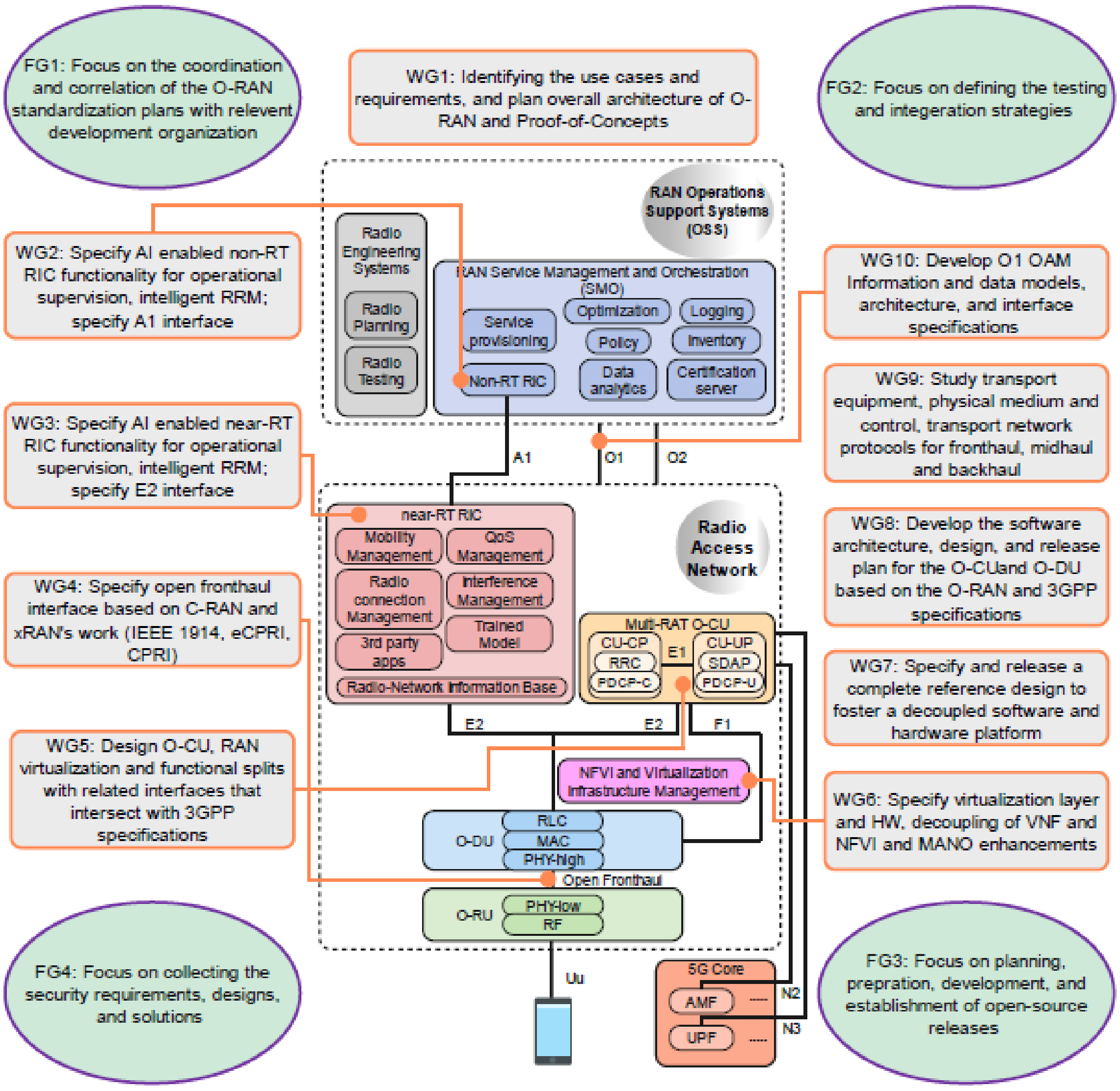
FG1: Focus on the coordination and correlation of the O-RAN standardization plans with relevent development organization
WG1: Identifying the use cases and requirements, and plan overall architecture of O-RAN and Proof-of-Concepts
FG2: Focus on defining the testing and integeration strategies
RAN Operations Support Systems (OSS)
Radio Engineering Systems
Radio Planning
Radio Testing
RAN Service Management and Orchestration (SMO)
Service provisioning
Optimization
Logging
Policy
Inventory
Non-RT RIC
Data analytics
Certification server
WG2: Specify AI enabled non-RT RIC functionality for operational supervision, intelligent RRM; specify A1 interface
WG10: Develop O1 OAM Information and data models, architecture, and interface specifications
WG3: Specify AI enabled near-RT RIC functionality for operational supervision, intelligent RRM; specify E2 interface
WG9: Study transport equipment, physical medium and control, transport network protocols for fronthaul, midhaul and backhaul
A1
O1
O2
near-RT RIC
Mobility Management
QoS Management
Radio connection Management
Interference Management
3rd party apps
Trained Model
Radio-Network Information Base
Radio Access Network
WG8: Develop the software architecture, design, and release plan for the O-CUand O-DU based on the O-RAN and 3GPP specifications
WG4: Specify open fronthaul interface based on C-RAN and xRAN's work (IEEE 1914, eCPRI, CPRI)
Multi-RAT O-CU
CU-CP
RRC
PDCP-C
E1
CU-UP
SDAP
PDCP-U
WG7: Specify and release a complete reference design to foster a decoupled software and hardware platform
E2
E2
F1
WG5: Design O-CU, RAN virtualization and functional splits with related interfaces that intersect with 3GPP specifications
NFVI and Virtualization Infrastructure Management
WG6: Specify virtualization layer and HW, decoupling of VNF and NFVI and MANO enhancements
O-DU
RLC
MAC
PHY-high
Open Fronthaul
O-RU
PHY-low
RF
Uu
5G Core
AMF
UPF
N2
N3
FG4: Focus on collecting the security requirements, designs, and solutions
FG3: Focus on planning, prepration, development, and establishment of open-source releases

Figure 2.2: O-RAN Architecture, Working and Focus Groups Related [12]

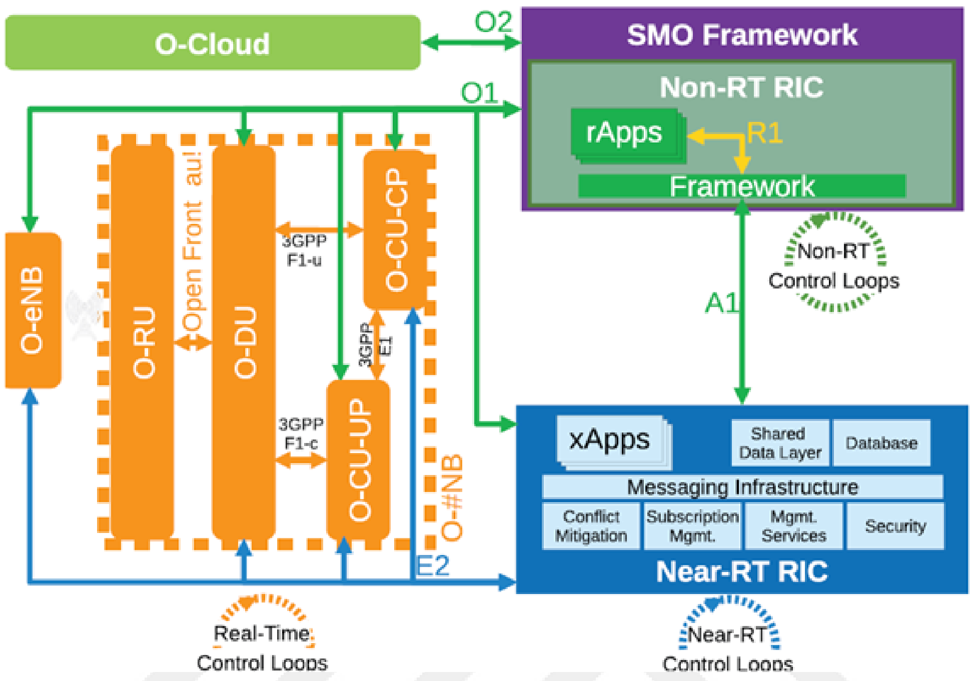

Figure 2.3: O-RAN Architecture [4]

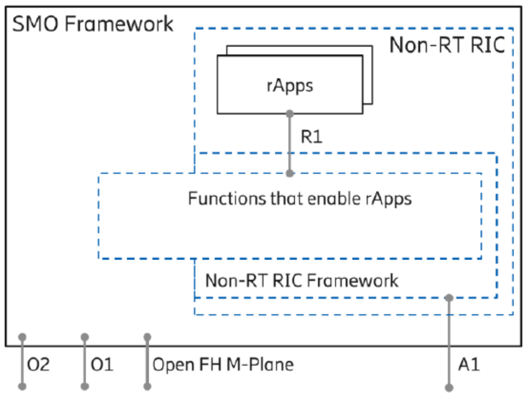

Figure 2.4: SMO and Non-RT RIC Services [9]

- O1: It is the interface between O-RAN managed element and the management entity.

- Open-Fronthaul Interface: It is between O-DU and O-RU functions and consists of Control User Synchronization (CUS) plane and Management (M) plane.

- Open-Fronthaul M-plane(in hybrid mode): In the hybrid mode it is between O-RU and SMO for FCAPS functionality.

- O2: Interface between SMO and O-Cloud.

- E2: It is a logical interface bridging Near-RT RIC and E2 node. An E2 node can be connected to only one Near-RT RIC while a Near-RT RIC might be connected to more than one E2 nodes. The protocols are based on the control plane. Near-RT RIC services and Non-RT RIC services are two functions of E2.

There are many other interfaces that are not detailed in because they are maintained by 3GPP even though seen as part of the O-RAN architecture as well: E1, F1-c, F1-u, NG-c, NG-u, X2-c, X2-u, Xn-c, Xn-u, Uu.

- Non-RT RIC: It communicates with Near-RT RIC elements in the RAN via the A1 interface. Using A1 interface, Non-RT RIC implementation helps policies for individual UEs or UE groups, monitors and supplies feedback on policy state from Near-RT RICs, gives the information required by near-RT TICs, and facilitates ML model training, distribution and interpretation with Near-RT RICs [14]. Non-RT RIC will be examined in detail in section 2.4.

- Near-RT RIC: It is a logical function that handles near real-time control and E2 node function and resource optimization. It hosts one or multiple xApps that use E2 interface to collect near real-time information and provide value-added services. In case Near-RT RIC fails, E2 node is able to provide services. However, there might be an outage of value-added services which can be done by Near-RT RIC only.

- O-CU-CP: It terminates the NG-c, X2-c, Xn-c, F1-c and E1 interfaces.

- O-CU-UP: It terminates the NG-u, X2-u, S1-u, Xn-u, F1-u and E1 interfaces.
- O-DU: It terminates the E2 and the F1 interfaces.
- O-Cloud: It is a cloud computing platform of a collection of physical infrastructure nodes that satisfy O-RAN requirements in order to host relevant O-RAN functions, software components, and appropriate management and orchestration functions exporting: O-RAN 02 interface, O-RAN Accelerator Abstraction Layer (AAL) API, O-Cloud Notification interface.

2.3 Control Loops

Control loops are designed for normal operation and optimization of the network by feeding it with real-time intelligence and analytics. Thereby they realize the vision of autonomous and self-optimizing networks. Control loops exist at various levels and run simultaneously, in parallel to each other and. Depending on the use case, these loops may or may not have any interaction [15].

O-RAN architecture supports three control loops involving different O-RAN functions, these loops can be seen in Figure 2.5.

- Non-RT control loop: The aim of this control loop is to execute O-RAN orchestration decisions which are policy configuration and machine learning (ML) training. It is large timescale operation in seconds or minutes i.e,, more than 1s. Non-RT control loop helps Non-RT RIC provide guidance, enrichment information and management of ML and AI models for Near-RT RIC, also has an important impact on SMO operations.
- Near-RT control loop: The goal is to perform operations such as policy enforcement and radio resource management. It is sub-second time-scale operation. The timing of the Near-RT control loop is between 10ms and 1s. [4]
- RT control loop: Real-time operation performing legacy radio operations such as hybrid automatic repeat request (HARQ), beamforming,

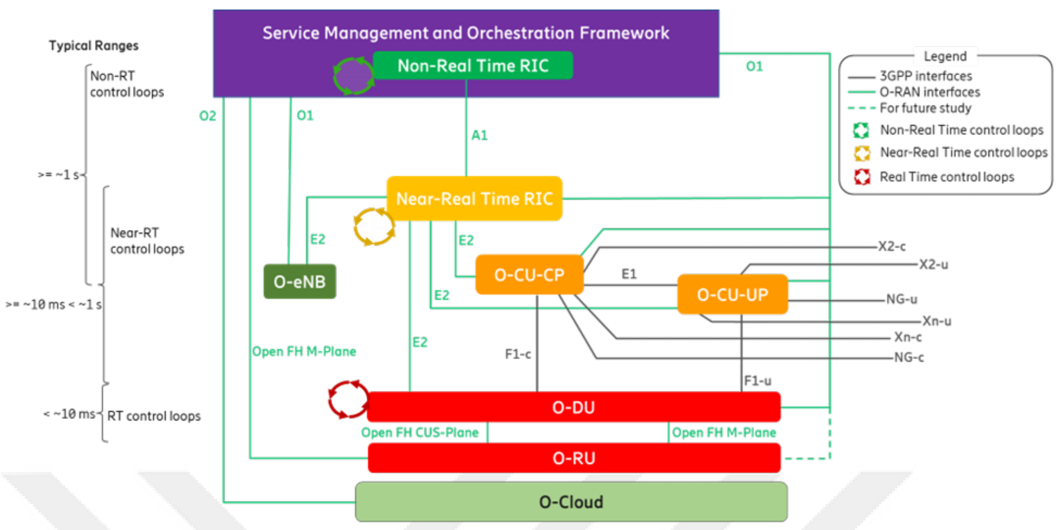

Figure 2.5: Control Loops of O-RAN [9]

or scheduling. The timing of the loop is less than 10ms, hence called real-time. Control loops in E2 nodes can operate below 10ms, O-DU radio scheduling maybe given as an example fore RT control loop.

2.4 Non-RT RIC

The main goal of Non-RT RIC is to support non-real-time radio resource management, higher layer procedure optimization, RAN policy optimization. Non-RT RIC also assists Near-real-time RIC functions' operation by guiding, providing parameters, policies, and AI/ML models. In addition, service and policy management, RAN analytics, and model training for Near-real-time RICs are Non-RT RIC functions. Concepts, specifications, architecture, and reference implementations are defined and described by O-RAN Alliance architecture and specifically the Non-RT RIC project [14]. The architecture of O-RAN with the illustration of relation between Non-RT RIC and A1 is given in Figure 2.6:

Non-RT RIC components are:

- Non-RT RIC Control Panel/Dashboard as a graphical user interface - views and manages in the RAN

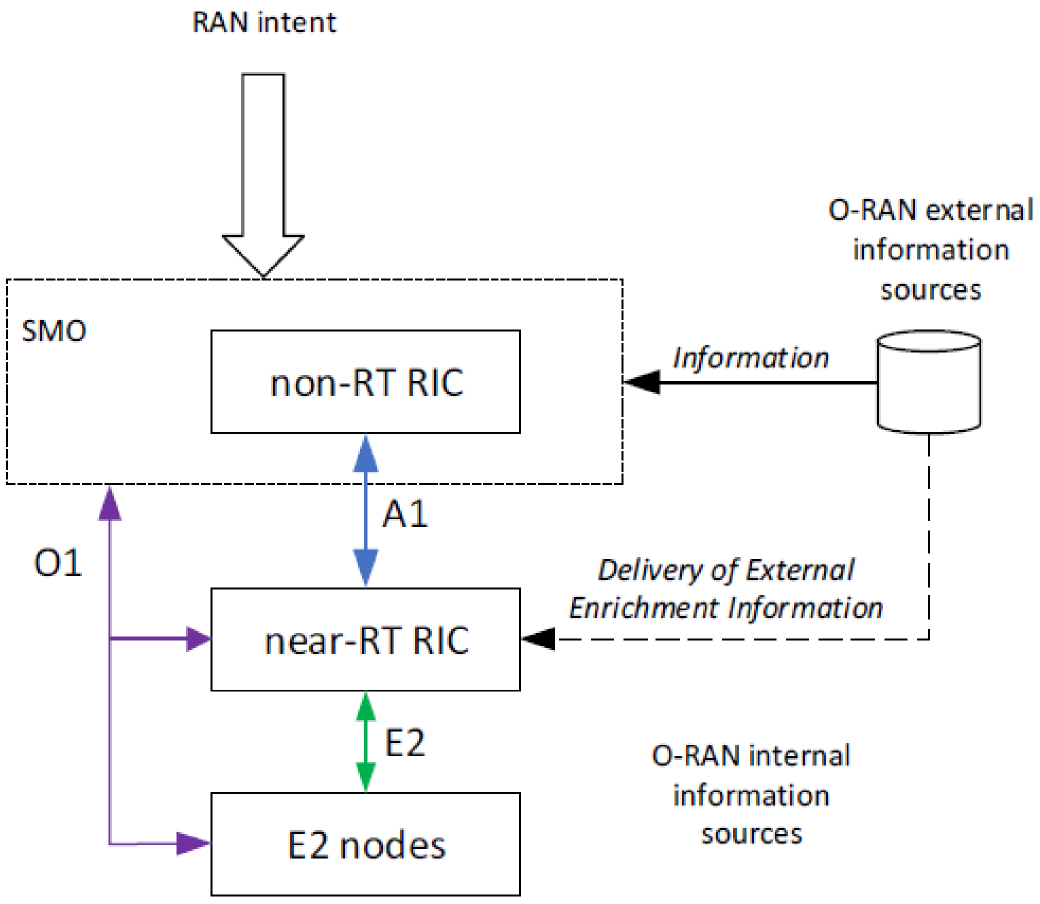
RAN intent
O-RAN external information sources
SMO
non-RT RIC
Information
A1
O1
Delivery of External Enrichment Information
near-RT RIC
E2
O-RAN internal information sources
E2 nodes

Figure 2.6: The Illustration of Non-RT RIC and the relation with A1 [16]

- A1 Policy Management Service which is a microservice that maintains a transient repository of all configured A1 policies instances in the network, developed in ONAP

- Enrichment Information Coordinator coordinating and registers A1-EI types, producers, consumers and jobs

- r-App Catalogue that is register for r-Apps

- A1 Controller which mediation point for A1 interface termination in SMO/NONRTRIC based on ONAP SDNC

- A1 simulator simulating the A1 aspects of a Near-RT RIC, Stateful A1 test stub

Non-RT RIC includes two sub-functions:

- Non-RT RIC Framework is the functionality as a part of SMO which logically terminates the A1 interface and provide the required services to rApps via R1 interface.

- rApps are modular applications of Non-RT RIC with a goal to perform RAN optimization. They are enabled by the services through R1 enable to obtain information and perform actions such as policies, and reconfiguration via related services.

As stated in Section 2.2, Non-RT RIC communicates with Near-RT RIC via A1 interface. A1 interface is independent of implementations of SMO, Non-RT RIC and Near-RT RIC, also it provides the multi-vendor environment. A1 policies are created, modified and deleted by Non-RT RIC and yhe main purpose of the policies is to guide the RAN performance in order to reach a better fulfillment of RAN Intent. Non-RT RIC handles A1 policies based on A1 policy feedback, and the network status. Network status is provided by O1 interface and is used to evaluate the influence of A1 policies. Non-RT RIC can also provide enrichment information through A1 in for policy enforcement in Near-RT RIC.

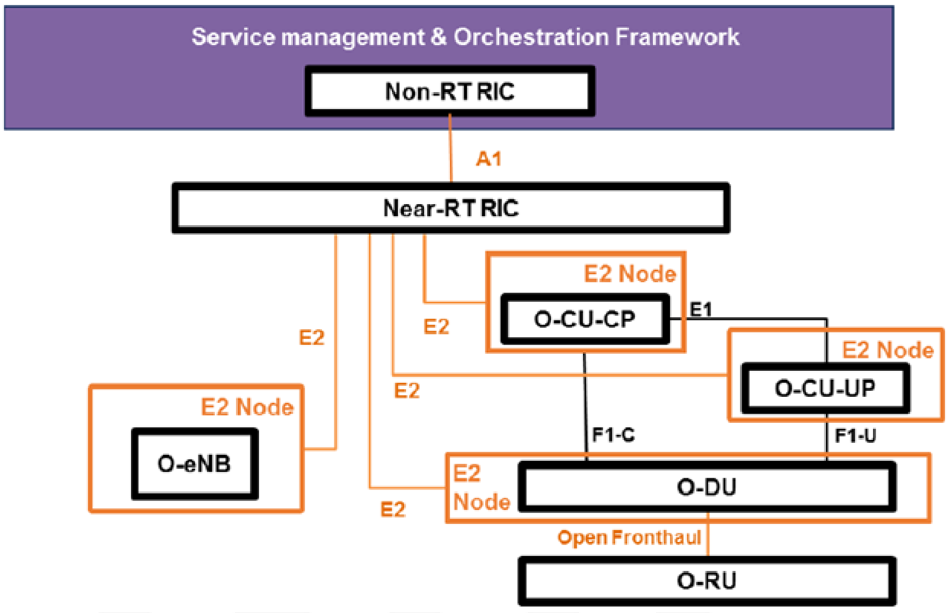

Figure 2.7: O-RAN Architecture overview with Near-RT RIC Interfaces [17]

2.5 Near-RT RIC

Near-RT RIC is a logical function enabling near-real time control optimization of E2 nodes functions and resources through fine-grained data collection and actions over E2 interface. It is placed between SMO and E2 nodes. E2 nodes are connected with Near-RT RIC by E2 logical interface. Near-RT RIC hosts one or more xApps to get near-real time information and provide value-added services. xApps are software tools to manage network functions in near-real time. As a part of Near-RT RIC, xApps handle controlling and optimizing RAN functions and resources. O-RAN architecture focused on Near-RT RIC interfaces is shown in Figure 2.7. As it can be seen, Near-RT RıC is connected to O-CU-CP, O-CU-UP, O-DU, and O- eNB. An E2 node can be connected to only one Near-RT RIC whereas Near-RT RIC can be connected to multiple E2 nodes, hence multiple O-CU-CPs, O-CU-UPs, O-DUs, and O- eNBs. Near-RT RIC can be connected to only one Non-RT RIC, via A1 interface.

The advantage of using Near-RT RIC deployment is flexibility, as it allows combining the entities or split them, and being deployed and delivered by different companies.Near-RT RIC is one of the main elements in the O-

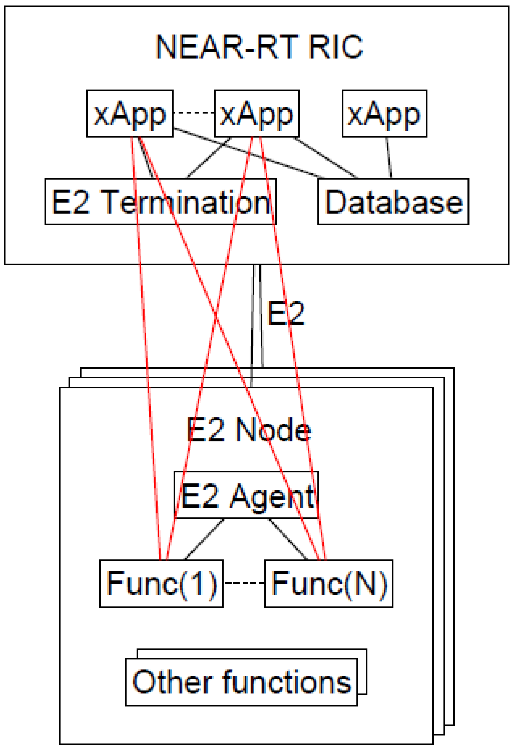

Figure 2.8: The Relationship between Near-RT RIC and E2 node [17]

RAN architecture, allowing external information into the operations of radio network. In addition, Near-RT RIC provides a platform for the vendors and developer to adapt and optimize radio resources per-use case RRM algorithms for specific scenarios.

RRM functional allocation between Near-RT RIC and E2 node is based on E2 node's capability over E2 interface according to E2 service model. E2 service model describes the functions in E2 node that is controlled by Near-RT RIC, and defines functional RRM split between them. Relationship between Near-RT RIC and E2 node is given in Figure 2.8.

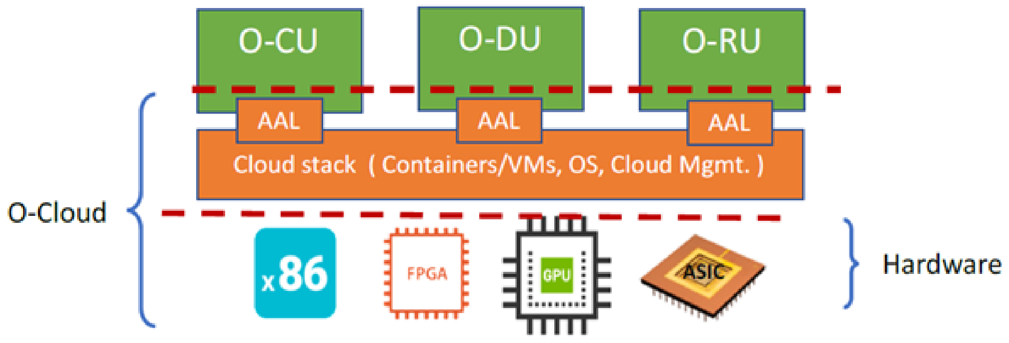

Figure 2.9: Decoupling and Illustration of O-Cloud [18]

2.6 O-Cloud

O-Cloud is a cloud computing platform consisting of a collection of physical infrastructure nodes that satisfy O-RAN requirements to host O-RAN functions, supporting software components and management and orchestration functions.

One of the characteristics of O-Cloud is that the hardware and software is decoupled. There are three layers to consider considering decoupling of hardware and software:

- Hardware layer, shown at the bottom in Figure 2.9
- Middle layer including Cloud Stack functions and Acceleration Abstraction Layer (AAL) functions.
- Top layer supporting the virtual RAN (VRAN) functions

Each layer comes from a different supplier, hence the first thing to check is whether Cloud Stack can work on multiple suppliers' hardware. Another aspect of decoupling is to ensure that Cloud Platform can support RAN virtualized function from multiple RAN software suppliers. If both satisfied, then the Cloud Platform is to be said an O-RAN Cloud Platform i.e., O-Cloud. Layers of O-Cloud can be seen in in Figure 2.9.

O-Cloud includes functionality to support both Deployment-plane and Management services. It provides a single logical reference point for resource pools within O-Cloud boundary [18]. The characteristics of the O-Cloud Cloud Platform are:

- It is a set of hardware and software components providing cloud computing capabilities for execution of RAN network functions
- The hardware of the platform includes computation, storage and networking component, there might be also acceleration technologies to meet RAN performance requirements
- O-Cloud software exposes open and well-defined API that manages entire life cycle of the network functions
- The Cloud Platform software is decoupled from the hardware so that they can be sourced from different vendors which is one of the main goals of O-RAN

An example of a Cloud Platform can be given as a Kubernetes Deployment on a set of Commercial-of-the-shelf (COTS) servers (including FPGA and GPU cards) interconnected by a spine or leaf networking fabric. An important aspect that needs to be considered is the exchange between virtualized RAN functions and hardware. Performance requirements should be met and functionality should be supported economically.

Key components and management of O-Cloud can be seen in Figure 2.10. These key concepts are defined as:

- O-Cloud instance refers to a set of resources at one or more locations and the software that manages nodes and deployments hosted on them. An O-cloud instance should have the functionality supporting both Deployment-plane i.e., UP and Management services.
- O-Cloud Resource Pool is a collection of O-Cloud nodes in one location to be used for either UP or Management services.
- O-Cloud Node can be thought of as a server which is a collection of CPUs, Mem, Storage,NICs, Accelerators, etc.. These nodes support one or more roles.
- O-Cloud Node Roles are the functionalities that a given node may support, including Compute, Storage and Networking for UP.
- O-Cloud Deployment Plane is a logical structure referring to O-Cloud nodes accross the resource pools

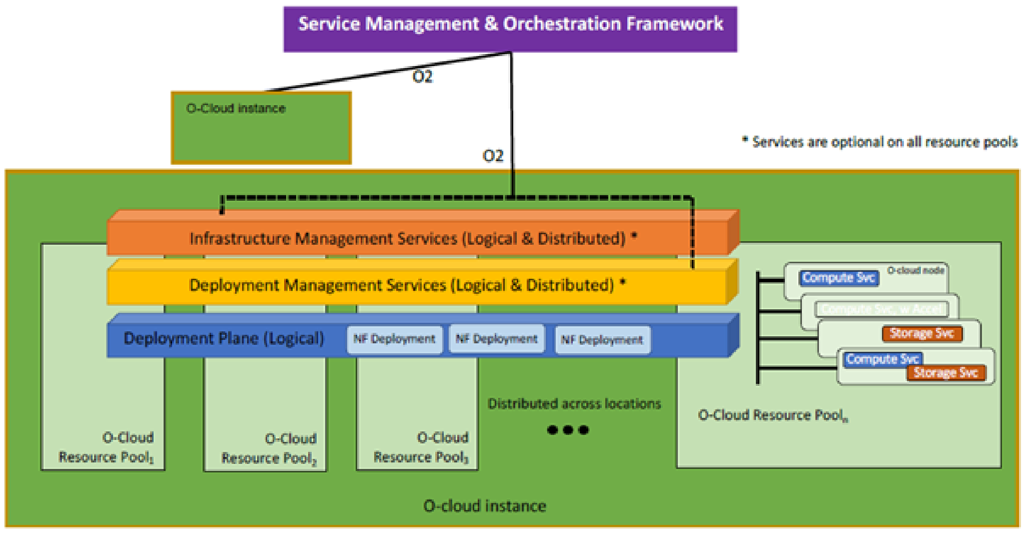
Service Management & Orchestration Framework
O2
O-Cloud instance
O2
* Services are optional on all resource pools
Infrastructure Management Services (Logical & Distributed) *
Deployment Management Services (Logical & Distributed) *
Deployment Plane (Logical)
NF Deployment
NF Deployment
NF Deployment
Compute Svc
O-cloud node
Storage Svc
Compute Svc
Storage Svc
Distributed across locations
O-Cloud Resource Pool$_1$
O-Cloud Resource Pool$_2$
O-Cloud Resource Pool$_3$
O-Cloud Resource Pool$_n$
O-cloud instance

Figure 2.10: Key Components of O-Cloud [18]

- O-Cloud Deployment is a deployment of a cloud native Network Function (NF), resources shared within or accros NF

- O2 Intergace is a the services and associated interfaces provided by O-Cloud to SMO. These services are divided into two Infrastructure Management Services (IMS) and Deployment Management Services (DMS). IMS is responsible for deploying and managing cloud infrastructure whereas DMS aims to handle the life cycle of virtualized and containerized deployments on the cloud infrastructure.

2.7 Use Cases

There are currently many use cases that are studied, and/or supported by O-RAN specifications. O-RAN Alliance specifies use cases and defines the policy frameworks. WG documents for use cases define use cases by examining the background and aim of the use case, entities and resources involved, solutions and required data. An O-RAN use case is a use case that leverages the O-RAN architecture and demonstrates its unique benefits. These benefits are the ability to use ML and AI systems and modules to empower intelligence in a multi-vendor network. O-RAN Alliance released a white paper titled "O-RAN Use Cases and Deployment Scenarios" in which an initial set of use cases were introduced. The said paper includes white-box hardware, open hardware reference design, cloud native deployment through O-RAN cloudification and orchestration platform, O-Cloud [19].

Design of low cost RAN white-box hardware is Phase I for uses cases. White-box hardware is a way to reduce the cost of 5G deployment, and O-RAN aims to release a complete design of white-box base station to help the O-RAN ecosystem get scale effect of hardware selection and reduction of cost in the whole industry. The design for white-box hardware requires the support of O-RAN open fronthaul interface, O-DU software and facilitates network functions deployed as virtualized applications in the cloud. Phase I use cases are as follows:

- Traffic Steering: it is an evolution of mobile load balancing and is widely used in network solution to optimize traffic distribution.
- QoE Optimization: 5G native applications are bandwidth consuming and latency sensitive hence cannot be supported efficiently by QoS framework. However, QoE service could not be always guaranteed either. With the usage of software defined RIC and interfaces of O-RAN, Therefore AI models can be deployed to optimize OoE.
- QoS Based Resource Optimization: it can be used by Non-RT RIC to make sure that certain prioritized users can reach a specific level of QoS in a congestion situation. Moreover, it can be used for Non-RT RIC and Near-RT RIC to optimize the allocation of RAN resources between users where deployed RAN resources and the configuration do not satisfy the requirements for all users.
- Massive MIMO Optimization: MAssive MIMO being as one of the key technologies in 5G, the main goal of this use case is to improve cell-centric network QoS and user-centric QoE in either a multi-cell or multi-vendor deployment.

O-RAN Use Cases for Phase II are given as:

- RAN Slice SLA Assurance: ML/AI based architecture of O-RAN enables RAN SLA assurance mechanisms. Based on RAN SLA requirements, Non-RT RIC and Near-RT RIC improve RAN behaviour to make sure these requirements are satisfied dynamically.

- Context-Based Dynamic Handover Management for V2X: V2X (vehicle to anything) communications is one of the main topics for car manufacturers and network technology companies. Due to high speed and heterogeneous environment of vehicles there might be handover (HO) anomalies. In such O-RAN architecture helps the collection and management of radio and HO data, deployment and evaluation of ML/AI application predicting HO anomalies, and real-time monitoring of traffic and radio conditions, deployment and maintenance of real-time applications to prevent anomalies on UE level in Near-RT RIC.

- Flight Path Based Dynamic UAV Resource Allocation: multi-dimensional data can be obtained in O-RAN architecture. Near-RT RIC supports deployment and execution of ML/AI models from Non-RT RIC, and based on this radio resource allocation for on-demand coverage of UAV can be performed by Near-RT RIC taken into account radio channel condition, flight path and other application information.

- Radio Resource Allocation for UAV Applications: radio resource requirements are sent to O-CU and O-DU for execution by Near-RT RIC for optimization of parameters.

- RAN Sharing: in this case O-RAN provides a set of interfaces (E2, 01, 02) to monitor remote users' performance in order to optimize radio allocation and configuration of QoS parameters, therefore brings more freedom to the RAN sharing process.

These use cases and corresponding O-RAN components of each use, for both phase are given in Figure 2.11.

Another white paper was released with title "O-RAN Minimum Viable Plan and Acceleration towards Commercialization" focused on how to accelerate and enable the deployment of O-RAN architecture in commercial networks.

O-RAN Minimum Viable Plan (MVP) approach uses operator members' input to prioritize use cases and focuses on WGs to deliver the required specifications for these use cases. Members of O-RAN Alliance continue to innovate and define additional use cases. The O-RAN Software Community (OSC) continues developing open source code for speeding up fulfilment of the O-RAN

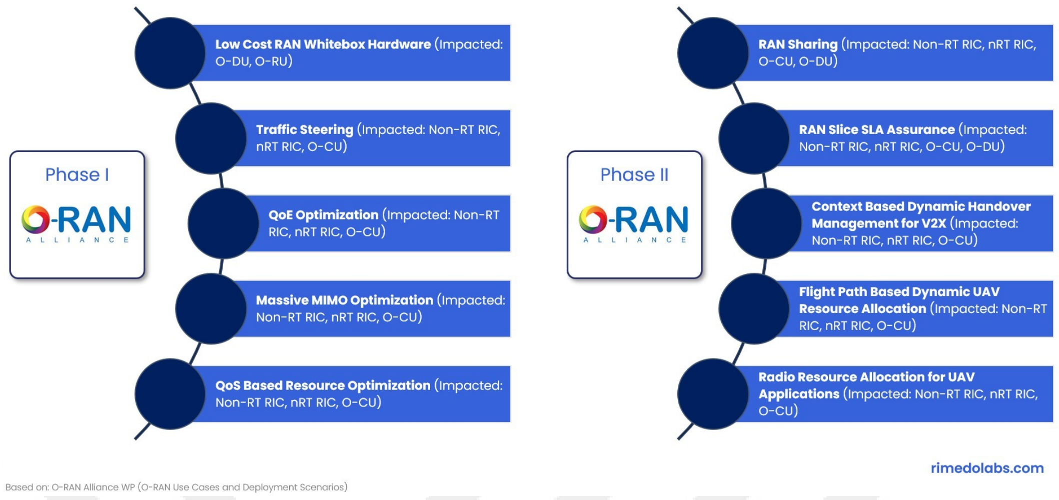
Phase I
O-RAN ALLIANCE
Low Cost RAN Whitebox Hardware (Impacted: O-DU, O-RU)
Traffic Steering (Impacted: Non-RT RIC, nRT RIC, O-CU)
QoE Optimization (Impacted: Non-RT RIC, nRT RIC, O-CU)
Massive MIMO Optimization (Impacted: Non-RT RIC, nRT RIC, O-CU)
QoS Based Resource Optimization (Impacted: Non-RT RIC, nRT RIC, O-CU)
Phase II
O-RAN ALLIANCE
RAN Sharing (Impacted: Non-RT RIC, nRT RIC, O-CU, O-DU)
RAN Slice SLA Assurance (Impacted: Non-RT RIC, nRT RIC, O-CU, O-DU)
Context Based Dynamic Handover Management for V2X (Impacted: Non-RT RIC, nRT RIC, O-CU)
Flight Path Based Dynamic UAV Resource Allocation (Impacted: Non-RT RIC, nRT RIC, O-CU)
Radio Resource Allocation for UAV Applications (Impacted: Non-RT RIC, nRT RIC, O-CU)
rimedolabs.com
Based on: O-RAN Alliance WP (O-RAN Use Cases and Deployment Scenarios)

Figure 2.11: O-RAN Use Cases of Phase I and II [20]

vision. The Test and Integration Focus Group (TIFG) works for defining end-to-end test specifications along with guidelines for Open Test and Integration Centers to verify multi-vendor O-RAN networks. The Security Focus Group (SFG) was formed for ensuring secure deployment of O-RAN networks through all WGs. In this paper prioritized uses cases were defined as follows [21]:

- Traffic Steering: seen as a high priority of MVP, this use case addresses the problem of adapting to diverse user demands and variable optimization objectives by allowing operators to have flexibility in configuration of optimization policies, to utilize convenient performance criteria and incorporate ML for intelligent and proactive traffic management and control.

- QoS and QoE Optimization: O-RAN RIC facilitates real-time RAN and application QoS and OoE optimization with closed-loop network optimization. O-RAN architecture also helps the deployment of QoS and QoE as automated policy control in the network, so that operator can rapidly deploy and fine tune the behaviour.

- RAN Slicing and SLA Assurance: combined with AI/ML based architecture, O-RAN allows the implementation of challenging RAN slice SLA assurance mechanisms which are critical for operators to realize the network slicing opportunities. In order to assure RAN slice SLAs, Non-RT RIC and Near-RT RIC can fine-tune RAN behaviour dynamically based on RAN specific SLA requirements.

- SMO: O-RAN aims to deliver an end-to-end SMO solution and enhance RAN automation, management, and deployment of cloud native infrastructure and applications.

- Massive MIMO Optimization: being identified as one of the most important, this use case aims to optimize the network over entire coverage area and according to operator-defined objectives by appropriate Massive MIMO configuration and policy.

Chapter 3

O-RAN Scenarios

Having gained background information regarding O-RAN technologies in previous chapter, this chapter aims to investigate solutions and projects proposed and provided by O-RAN in more detail. these are O-RAN Non-RT RIC, ETSI OSM, ONAP and ONF SD-RAN projects.

3.1 O-RAN SC Non-RT RIC Project

In this section the integration of O-RAN architecture and Non-RT RICs will be examined. Non-RT RIC supports non-real-time intelligent managements of open RAN functions as described in Chapter2 in detail. Along with this, the main goal of Non-RT RIC is to support higher layer optimization, policy optimization in RAN and to provide guidance, parameters and ML/AI models to Near-RT RIC in order to achieve RAN obejctives [22]. Non-RT RIC project provides concepts, specifications, architecture, and reference implementations in accordance with O-RAN architecture [23].

Figure 3.1 shows the functional view of OSC Non-RT RIC implementation where the functios will be supported by SMO.

OSC Non-RT RIC implementation communicates with Near-RT RIC in RAN via A1 interface. Using A1 interface OSC Non-RT RIC aims to:

provide A1 policies

monitor and give basic feedback on policy state coming from Near-RT RICs

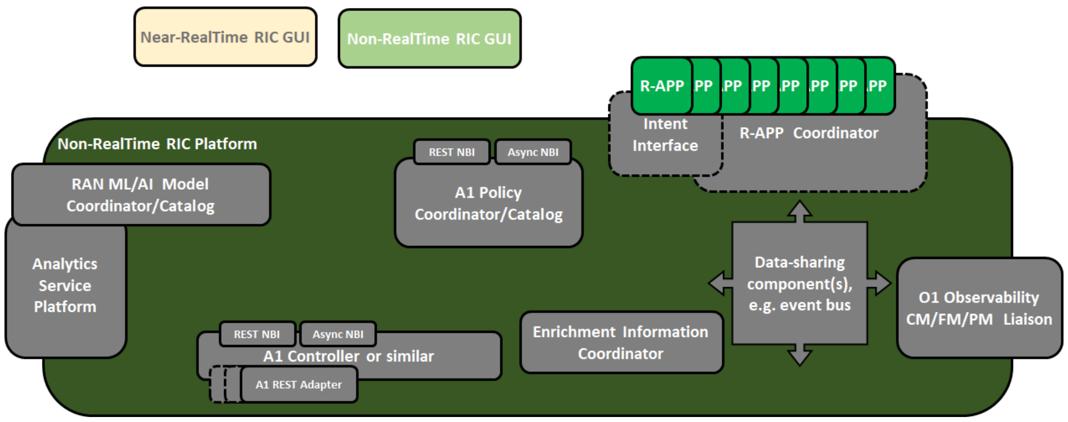
Near-RealTime RIC GUI
Non-RealTime RIC GUI
R-APP
PP
PP
PP
PP
PP
PP
PP
Intent Interface
R-APP Coordinator
Non-RealTime RIC Platform
RAN ML/AI Model Coordinator/Catalog
REST NBI
Async NBI
A1 Policy Coordinator/Catalog
Analytics Service Platform
Data-sharing component(s), e.g. event bus
O1 Observability CM/FM/PM Liaison
REST NBI
Async NBI
A1 Controller or similar
A1 REST Adapter
Enrichment Information Coordinator

Figure 3.1: Functional View of Non-RT RIC function in OSC [23]

provide A1 enrichment information required by Near-RT RICs

act as a host platform for rApps

host R1 interface between SMO, Non-RT RIC and rApps

manage the exposure towards rApps

All the implementations will be demonstrated to prove functionalities and provide feedback to O-RAN WGs. Also note that Non-RT RIC functions partly use some existing infrastructure from ONAP which will be examined in section 3.2.

Figure 3.2 shows the functional view of latest released OSC Non-RT RIC functionality.

As seen in Figure 3.2 components of Non-RT RIC are:

- Non-RT RIC Control Panel / Dashboard: a graphical user interface (GUI) for viewing and managing A1 policies in Near-RT RICs, interacting with REST API.

- A1 Policy Management Service (developed in ONAP): a microservice which maintains a repository of all configured A1 policy instances, Near-RT RICs, all policies for Near-RT RICs and all configured policy instance in the network.

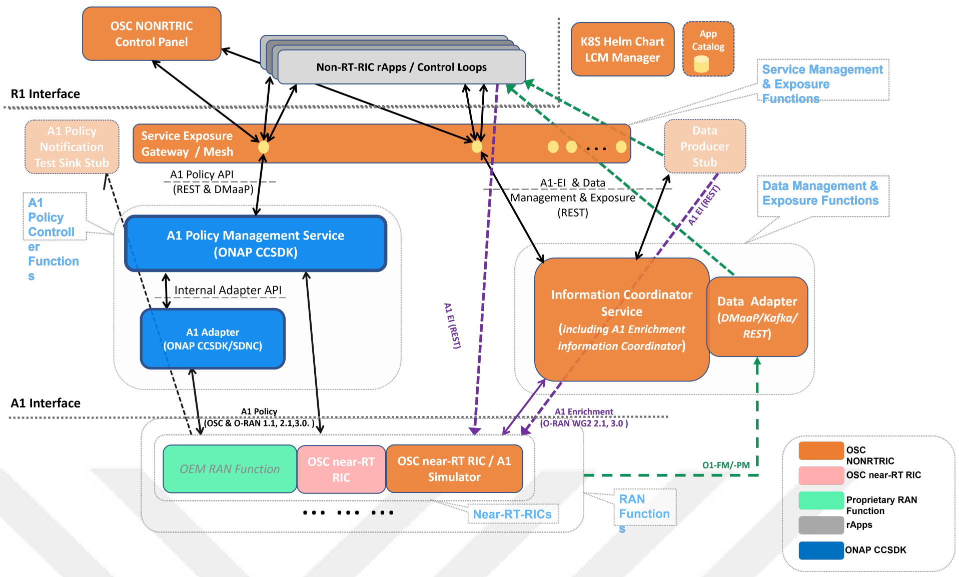
OSC NONRTRIC Control Panel
Non-RT-RIC rApps / Control Loops
K8S Helm Chart LCM Manager
App Catalog
Service Management & Exposure Functions
R1 Interface
A1 Policy Notification Test Sink Stub
Service Exposure Gateway / Mesh
Data Producer Stub
A1 Policy API (REST & DMaaP)
A1-EI & Data Management & Exposure (REST)
A1 Policy Controller Functions
Data Management & Exposure Functions
A1 Policy Management Service (ONAP CCSDK)
Internal Adapter API
A1 Adapter (ONAP CCSDK/SDNC)
A1 EI (REST)
A1 EI (REST)
Information Coordinator Service (including A1 Enrichment information Coordinator)
Data Adapter (DMaaP/Kafka/ REST)
A1 Interface
A1 Policy (OSC & O-RAN 1.1, 2.1,3.0.)
A1 Enrichment (O-RAN WG2 2.1, 3.0)
O1-FM/-PM
OEM RAN Function
OSC near-RT RIC
OSC near-RT RIC / A1 Simulator
Near-RT-RICs
RAN Functions
OSC NONRTRIC
OSC near-RT RIC
Proprietary RAN Function
rApps
ONAP CCSDK

Figure 3.2: Functional View of OSC Non-RT RIC Funcionality, Latest Release (F) [22]

- Enrichment Information (EI) Coordinator: a services that coordinates and registers A1-EI types, produces, consumers and jobs. It also maintains a registry of A1-EI query API, query status of A1-EI jobs and monitors Near-RT RICs.

- rApp Catalogue: a register for rApps as a first step towards rApp management in Non-RT RIC.

- A1 Controller-based ONAP SDNC: mediation point for A1 interface termination in SMO/Non-RT RIC.

- A1 Simulator: a stateful A1 test stub used to create several stateful A1 providers.

As said before, Non-RT RIC project reuses ONAP functions partly where needed. Integration of two project can be seen in Figure 3.3. ONAP functions are used by Non-RT again when and where necessary support needed. Non-RT RIC project states that it is strongly believed that open source software should be reused as much as possible, when appropriate in order to maximize the participation of the community. For instance, A1 Policy Management

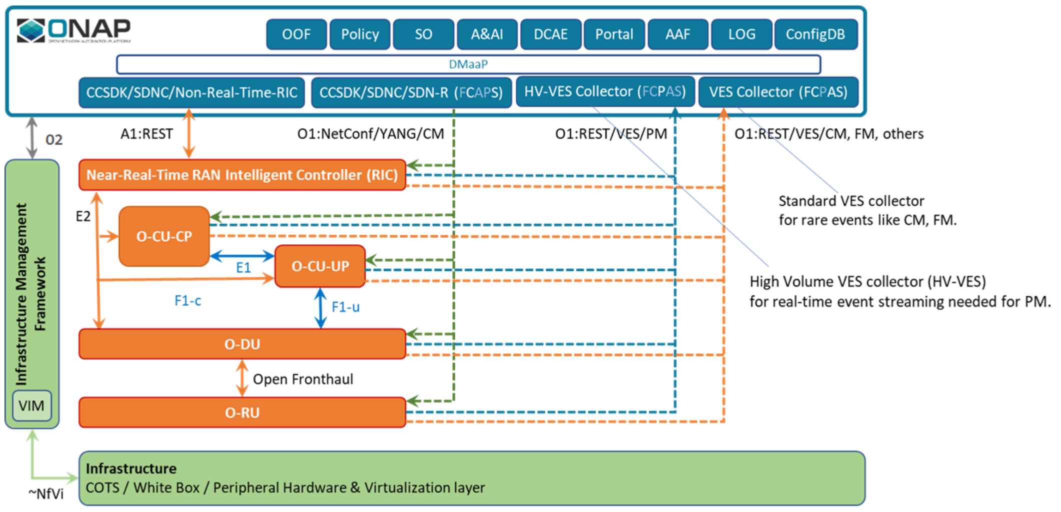
ONAP
OOF
Policy
SO
A&AI
DCAE
Portal
AAF
LOG
ConfigDB
DMaaP
CCSDK/SDNC/Non-Real-Time-RIC
CCSDK/SDNC/SDN-R (FCAPS)
HV-VES Collector (FCPAS)
VES Collector (FCPAS)
02
A1:REST
O1:NetConf/YANG/CM
O1:REST/VES/PM
O1:REST/VES/CM, FM, others
Near-Real-Time RAN Intelligent Controller (RIC)
Standard VES collector
for rare events like CM, FM.
E2
O-CU-CP
E1
O-CU-UP
High Volume VES collector (HV-VES)
for real-time event streaming needed for PM.
F1-c
F1-u
O-DU
Open Fronthaul
O-RU
Infrastructure Management Framework
VIM
Infrastructure
COTS / White Box / Peripheral Hardware & Virtualization layer
~NfVi

Figure 3.3: Integration of ONAP and O-RAN Non-RT RIC [23]

functions of ONAP are used in Non-RT RIC project as emerged from merging related activities of both.

3.2 ONAP

ONAP stands for Open Network Automation Platform, an open source software platform for orchestration, management and automation of network and edge computing services for network operators and cloud providers.It allows rapid automation of services and management of life cycle that is critical for 5G networks by its real-time, policy-driven orchestration and automation mechanism [24]. ONAP is designed as a microservice-based system in which all the components are released as Docker containers aiming to optimize image size by following best building practices. ONAP by definition needs to be a highly reliable, scalable, secure and easy to manage platform. This project aims to orchestrate and manage physical and virtual network services in order to reach agility, customer satisfaction and lower cost.

Operators can orchestrate physical and virtual network functions synchronously with ONAP which allows operators to leverage existing networks. There are several releases of ONAP, Istanbul was release on November 18, 2021 and the newest one, Jakarta, was released on July 5, 2022. Jakarta release aims to

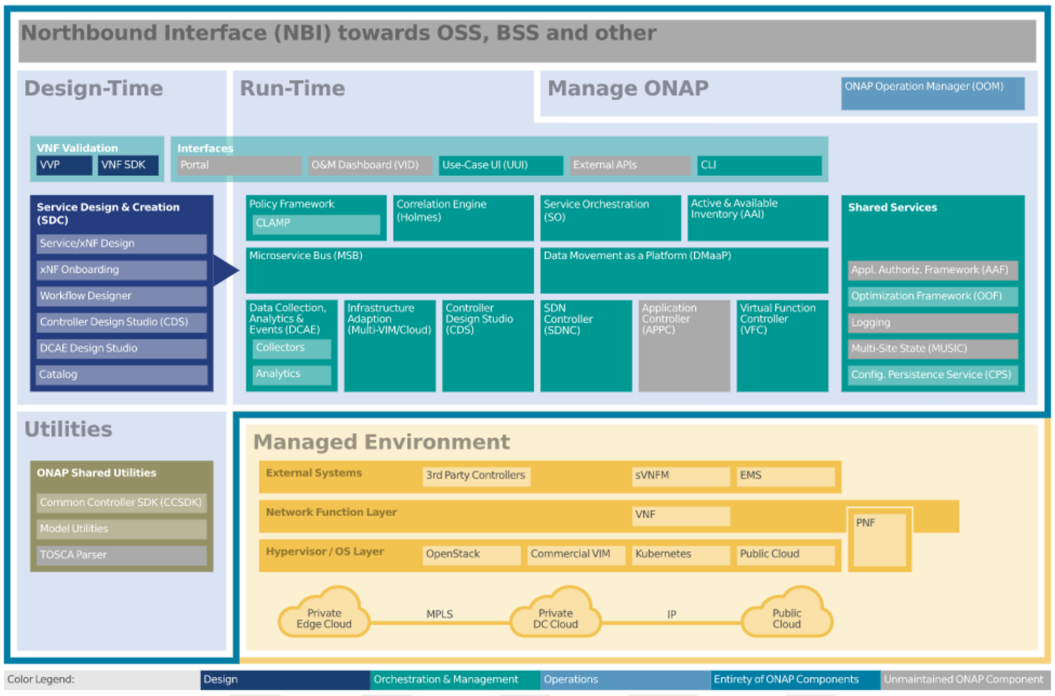
Northbound Interface (NBI) towards OSS, BSS and other
Design-Time
Run-Time
Manage ONAP
ONAP Operation Manager (OOM)
VNF Validation
VVP
VNF SDK
Interfaces
Portal
O&M Dashboard (VID)
Use-Case UI (UUI)
External APIs
CLI
Service Design & Creation (SDC)
Service/xNF Design
xNF Onboarding
Workflow Designer
Controller Design Studio (CDS)
DCAE Design Studio
Catalog
Policy Framework
CLAMP
Correlation Engine (Holmes)
Service Orchestration (SO)
Active & Available Inventory (AAI)
Shared Services
Microservice Bus (MSB)
Data Movement as a Platform (DMaaP)
Appl. Authoriz. Framework (AAF)
Optimization Framework (OOF)
Logging
Multi-Site State (MUSIC)
Config. Persistence Service (CPS)
Data Collection, Analytics & Events (DCAE)
Collectors
Analytics
Infrastructure Adaption (Multi-VIM/Cloud)
Controller Design Studio (CDS)
SDN Controller (SDNC)
Application Controller (APPC)
Virtual Function Controller (VFC)
Utilities
ONAP Shared Utilities
Common Controller SDK (CCSDK)
Model Utilities
TOSCA Parser
Managed Environment
External Systems
3rd Party Controllers
sVNFM
EMS
Network Function Layer
VNF
PNF
Hypervisor / OS Layer
OpenStack
Commercial VIM
Kubernetes
Public Cloud
Private Edge Cloud
MPLS
Private DC Cloud
IP
Public Cloud
Color Legend:
Design
Orchestration & Management
Operations
Entirety of ONAP Components
Unmaintained ONAP Component

Figure 3.4: High-level view of ONAP Architecture [26]

advance security, deepen OSC alignment, enhance cloud native around Kubernetes and continue 5G Blueprint alignment [25].

Architecture of ONAP consists of a design time and runtime functions, ONAP Shared Utilities and ONAP-managing functions. OVerview of ONAP Architecture can is given in Figure 3.4.

Five key components of ONAP [26] are given as:

- ONAP Design time environment: provides onboarding services and resources into ONAP and designs required services

- External API: provides northbound interoperability (NBI) for ONAP platform

- ONAP Runtime environment: provides a framework for automated insantiation and configuration of services and resources

- ONAP Operation Manager (OOM): manages cloud native installation and deployments to Kubernetes cloud environments

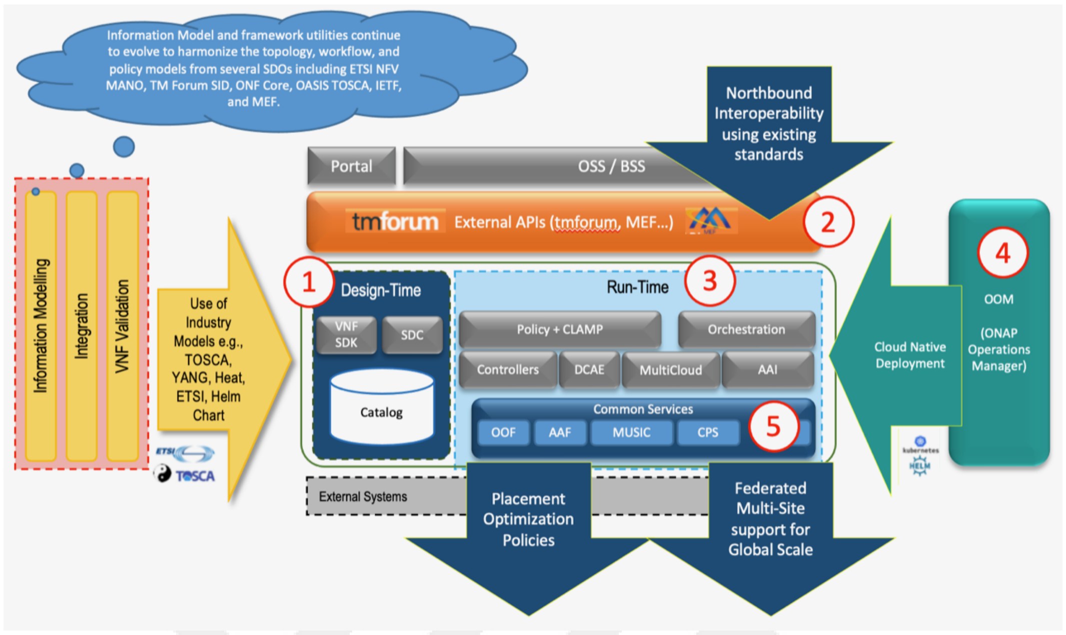
Information Model and framework utilities continue to evolve to harmonize the topology, workflow, and policy models from several SDOs including ETSI NFV MANO, TM Forum SID, ONF Core, OASIS TOSCA, IETF, and MEF.
Information Modelling
Integration
VNF Validation
Use of Industry Models e.g., TOSCA, YANG, Heat, ETSI, Helm Chart
TOSCA
Portal
OSS / BSS
Northbound Interoperability using existing standards
tmforum External APIs (tmforum, MEF...)
1
2
3
4
5
Design-Time
VNF SDK
SDC
Catalog
Run-Time
Policy + CLAMP
Orchestration
Controllers
DCAE
MultiCloud
AAI
Common Services
OOF
AAF
MUSIC
CPS
Cloud Native Deployment
OOM (ONAP Operations Manager)
External Systems
Placement Optimization Policies
Federated Multi-Site support for Global Scale

Figure 3.5: Functional view of ONAP Architecture [26]

- ONAP Shared Services: provides shared capabilities for ONAP modules

Highlights of these key components and the functional view of ONAP can be seen in Figure 3.5. Information Model and framework utilities keep on evolving to ingegrate and enhance the topolpgy, workflow, and policy models from several Standards Development Organizations (SDOs) such as ETSI NFV MANO, ETSI/3GPP, O-RAN, TM Forum SID, ONF Core, OASIS TOSCA, IETF, and MEF.

ONAP provides a wide range of use cases for Communication Service Providers (CSP) and open source networking community. After reaching a certain level of maturity use cases can be called ONAP Blueprints. Some of the uses cases defined in Honolulu for 5G release are 5G Service Modeling, 5G SON Use Case, ONA/3GPP & O-RAN Alignment, End to End Network Slicing. ONAP bluprints provided in Honolulu release are 5G Blueprint, Virtual CPE (vCPE), Broadband Service (BBS), Voice over LTE (VoLTE), Cross Domain and Cross Layer VPN (CCVPN) and Multi-Domain Optical Network Service (MDONS). In the subsection 3.2.1 5G Blueprint will be examined.

3.2.1 ONAP Akraino 5G Blueprint

5G aims to increase the speed and capacity, and decrease the latency at the same time while dynamically supporting multiple services. Key technologies of 5G are enhanced mobile broadband (eMMB), ultra-reliable low-latency communications for a remote device (uRLLC), and massive machine-type communications. These new technologies require a level of dynamic network behaviour different than the previous generations of wireless technologies. However, the need of this dynamic behaviour comes from different applications that require different levels of latency, reliability, availability, bandwidth, mobility and cost. In order to meet these diverse requirements and support 5G networks, network automation is a key consideration and a new set of requirements emerges: network slicing, RAN support, cloud native architecture support and real time analytics. ONAP project handles automation of 5G using SDN and NFV technologies and 5G Blueprint is a focused on five key initiatives around end to end service orchestration, network slicing, PNF/VNF life cycle management, PNF integration and network optimization.

The modeling work in ONAP for end to end network service orchestration allows creating an end to end 5G network service, aligned with ETSI and 3GPP. An end to end service consists of RAN elements (RU, DU and CU), and core elements as seen in Figure 3.6. Moreover the service later can be orchestrated by runtime components of ONAP.

Comprehensive support for all life cycle management activities is provided by ONAP. For configuration traditional mechanisms such as YANG/NETCONF/RESTCONF or REST API can be used. Life cycle management can be performed by using REST APIs, Ansible playbooks, or Python scripts. ONAP already supports the O1 interface with support for initial implementation of A1 interface to Near-RT RIC.

ONAP includes modeling and orchestration funcionality of a slice including 5G RAN, core and transport network slice subnets as well as workflows and user interfaces for functions that enable design, orchestration, activation, deactivation and termination of a slice [27].

3.3 OSM

Open Source MANO (OSM) is an ETSI-hosted open source initiative for developing MANO (Management and Orchestration) stack aligned with ETSI NFV.

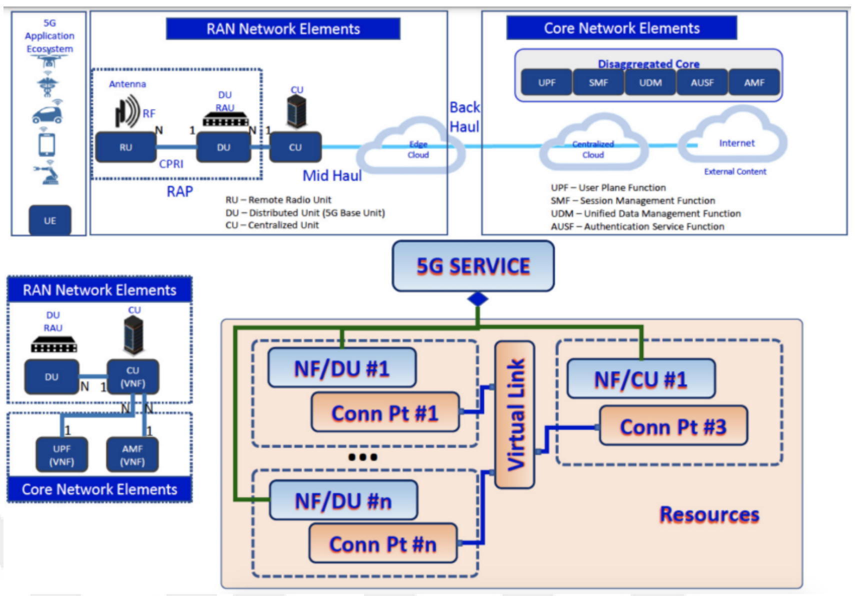

Figure 3.6: End to end 5G Service Design and Orchestration [27]

ETSI is a European Standards Organization (ESO) which produce globally applicable standards for ICT systems, applications and services. OSM aims to develop a community-driven E2E Network Service Orchestrator (E2E NSO) capable of modelling and automating telco services. OSM accelerates maturation of NFV technologies and standards, enables an ecosystem of VNF vendors, test and validate the interaction of orchestrator with the other components.

There are for key aspects of OSM to minimize integration efforts:

- Information Model (IM) responsible for modelling and automation of life cycle of network functions, network services, and network slices from initial deployment to operation and monitoring. IM enhances ETSI NFV SOL006, and it can be used to instantiate a given element in a wide range of VIM types and transport technologies which makes VNF models deployable everywhere.

- Northbound Interface (NBI) is provided by OSM to enable the full operation of system, network services and network slices under control. NBI offers managing the life cycle of these services and slices.

- Extended concept of Network Service in OSM provides that a network

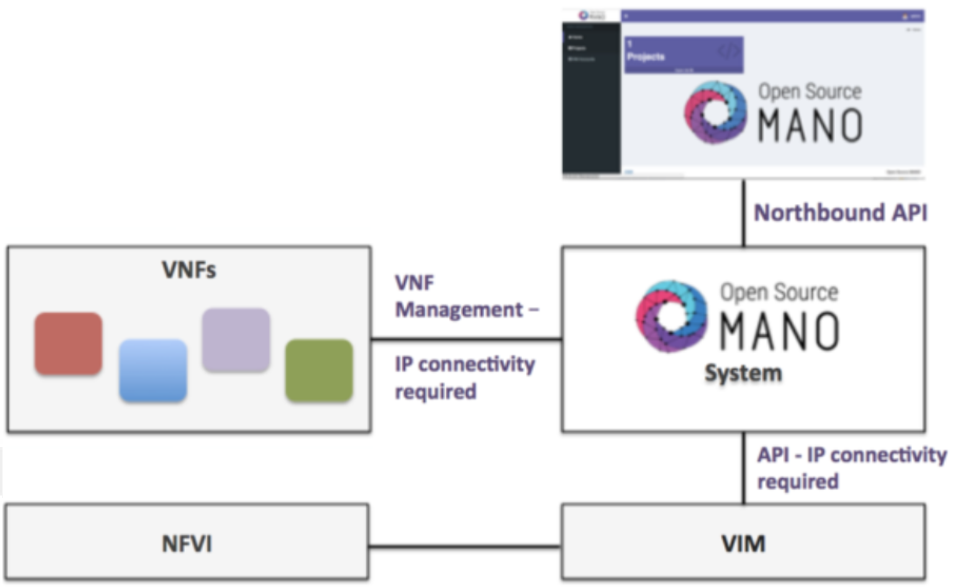

Figure 3.7: OSM interaction with VIMs and VNFs [28]

service can span across different domains such as virtual, physical, transport and control the life cycle by interacting with VNF, PNF and HNFs.

- Management of life cycles of network slices can be done by OSM when an integrated operation or slice management are requires.

OSM is capable of consuming openly published IMs, suitable for all VNFs and independent of VIM. Figure 3.7

OSM talks to the VIM for the deployment of VNFs and virtual links (VLs) connecting them, OSM also talks to the VNFs deployed in a VIM to run basic instantiation (day-0), service initialization (day-1) and runtime operations (day-2) configuration primitives. The onboarded VNF should fulfill the life cycle stages required for functioning properly, also the NFV MANO layer should be able to automate. Hence, the resulting package, should include all the requirements, instructions and elements to achieve these life cycle stages. For OSM to work there are three assumptions:

- Each VIM has an API endpoint reachable from OSM

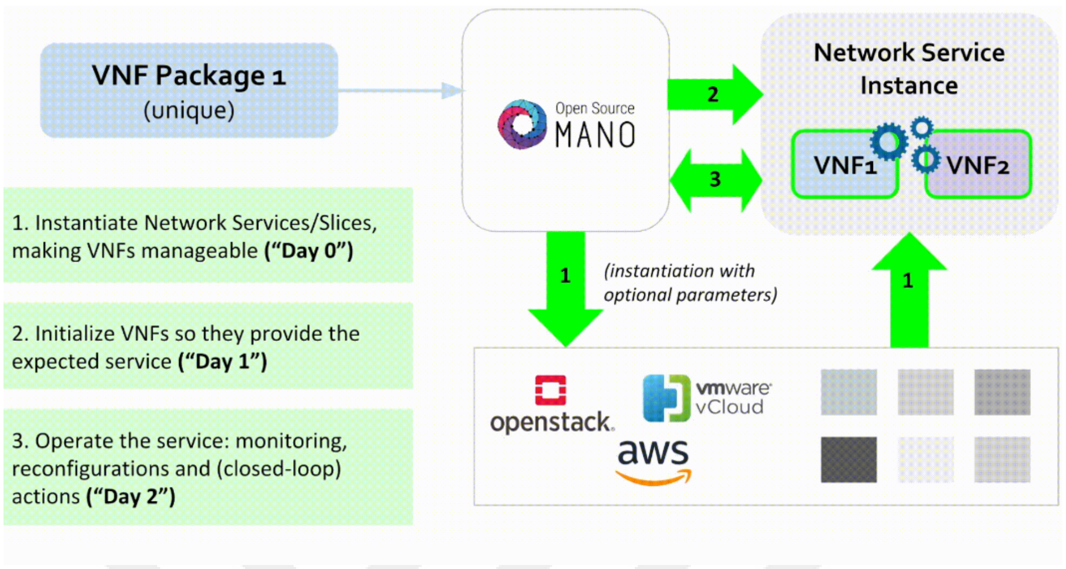
VNF Package 1
(unique)
1. Instantiate Network Services/Slices, making VNFs manageable ("Day 0")
2. Initialize VNFs so they provide the expected service ("Day 1")
3. Operate the service: monitoring, reconfigurations and (closed-loop) actions ("Day 2")
Open Source MANO
2
3
Network Service Instance
VNF1
VNF2
1
(instantiation with optional parameters)
1
openstack
vmware vCloud
aws

Figure 3.8: VNF Onboarding [28]

- Each VIM has a management network which provides IP address to VNFs
- OSM is able to reach the management network

Figure 3.8 represents the onboarding of VNF.
Deployment of a network service can be done by following the guide in [29]:

1. Onboarding of VNF
2. Adding VIM accounts such as OpenStack, OpenVIM
3. Onboarding of NS package
4. Instantiating the NS
5. Updating VNF instance in NS
6. Advanced instantiation using instantiation parameters
7. Monitoring and autoscaling

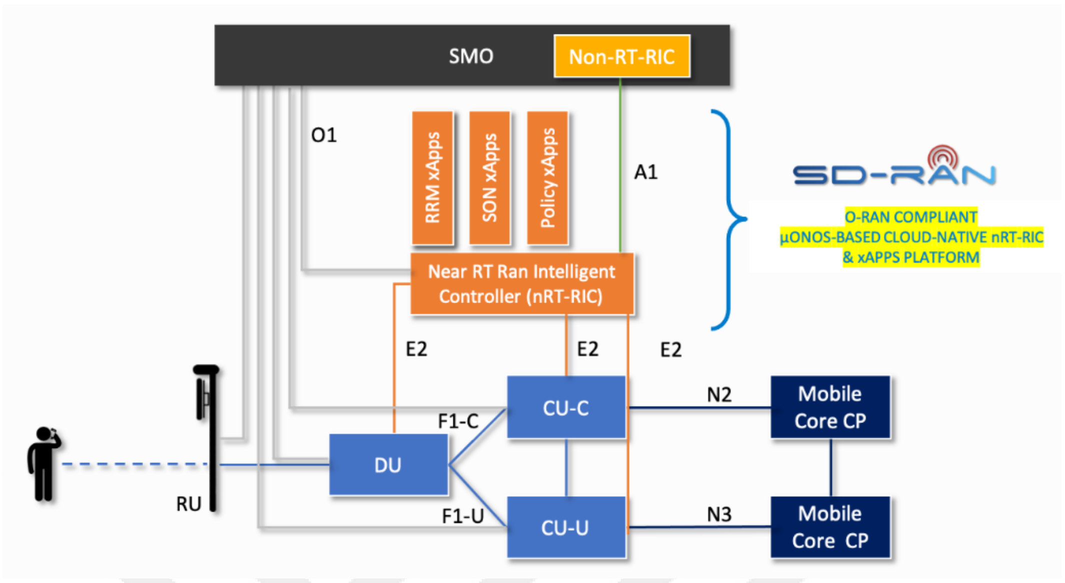

Figure 3.9: O-RAN Compliant SD-RAN [31]

3.4 ONF SD-RAN

Open Network Foundation (ONF) is an open source model aiming to promote Software-Defined Networking (SDN) and standardize related protocols and technologies. ONF initiative's goal is to speed innovation via software changes in networking areas.

ONF SD-RAN is building open source components for RAN space to complement O-RAN architecture and interfaces which will allow the creation of multi-vendor solutions and accelerate innovation of RAN systems. SD-RAN is developing a Near-RT RIC and a set of xApps. Near-RT RIC is cloud-native and built on a couple of ONF platforms such as ONOS SDN Controller. This new architecture will leverage O-RAN architecture and vision. As O-RAN specifications expands and develops SD-RAN follows them. While SD-RAN produces new functionalities, the extensions and experience stemming from the creation of the system will be contributed back to O-RAN to help advance O-RAN further [30].

SD-RAN platform is compliant with 3GPP and consistent with O-RAN architecture, an overview of SD-RAN is shown in Figure 3.10.

Starting with Near-RT RIC based on micro-ONOS, SD-RAN platform aims

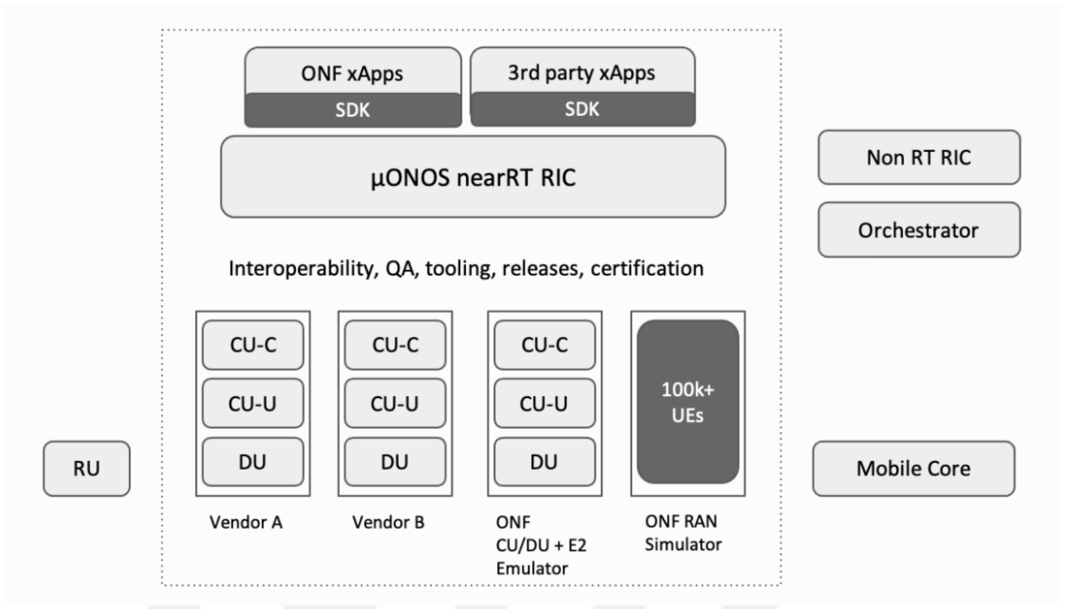

Figure 3.10: SD-RAN Solution [31]

to develop components for both control and users planes of CU and DU in disaggregated architecture of O-RAN. micro-ONOS is a name for the next generation architecture of ONOS which is an open source project architected to reach high availability, scalability and performance for service provider networks. ONOS stands for Open Network Opertating System.

Proposed SD-RAN solution can be seen in Figure 3.10.

This solution involves:

- O-RAN compliant interfaces and protocols
- Clustered micro-ONOS architecture
- Integration with third party xApp vendors
- Development of O-RAN compatible RU/CU/DU
- RAN-Simulator for scale testing
- SD-RAN-in-a-box (RiaB) for developing, testing and reference
- Contribution of SMs and app-sdk to O-RAN and OSC

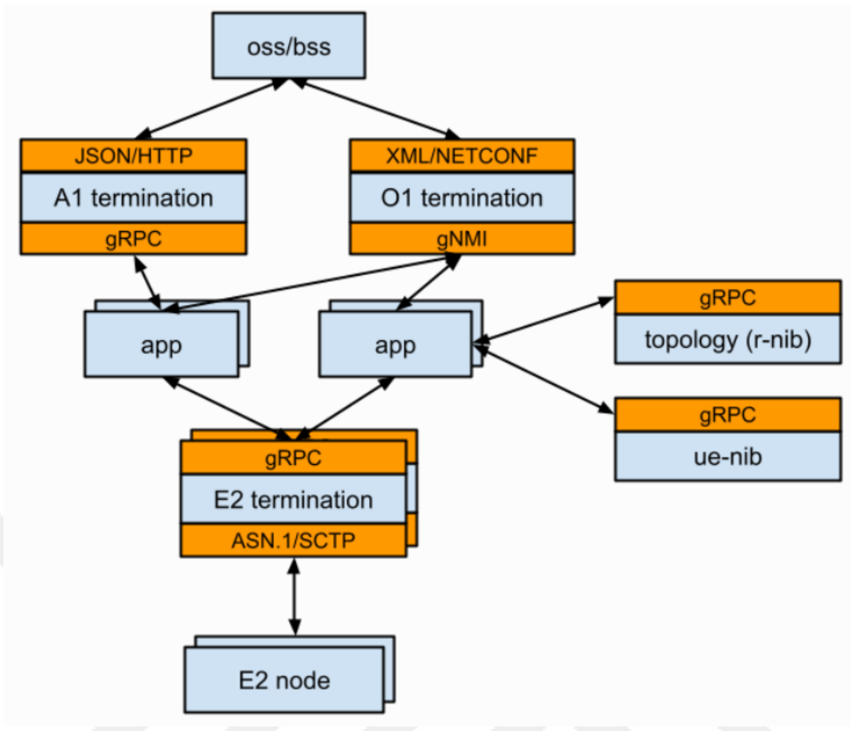

Figure 3.11: High-level structure of RIC [31]

- Releases with regression test suites
- Quality Assurance (QA) and interoperability lab with O-RAN Open Test and Integration Centers (OTIC)
- Hardening and operation for laboratory and field trials

The heart of ONF SD-RAN architecture is micro-ONOS RIC which allows ease in scalability, high performance and availability, and multi-vendor equipment. This structure is centered on micro-services and gRPC APIs for communication between processes. The overall project relies on micro-ONOS components such as onos-topo, onos-cli, onos-config, onos-gui. micro-ONOS RIC has a similar structure to other micro-ONOS subsystems such that it clearly describes the roles of southbound, northbound and core components. According to O-RAN specifications Near-RT RIC apps i.e., xApps are a part of overall RIC. The high-level structure of RIC is given in Figure 3.11.

The RIC system can be thought of as a chassis that includes several key subsystems establishing the platform and NEar-RT applications can be installed into this chassis and use gRPC-based APIs to communicate with RIC platform services. These services and their operations are:

- E2 Termination manages external ASN.1/SCTP connections to E2 nodes and helps receive and send E2SM messages
- A1 Termination handles external JSON/HTTP REST API requests from northbound orchestration systems and Non-RT applications
- O1 Termination manages external XML(YANG)/NETCONF configuration
- E2 Subscription Management handles subscription requests for E2SM messages and assists in routing messages between E2 termination nodes and application
- R-NIB Service tracks and distributes information about the E2 nodes and their relationships
- UE-NIB Service tracks and disseminates information about the end-nodes i.e., UEs
- Storage Facilities assists applications to maintain their state in a distributed, scalable and highly-available way

These services are being implemented as separate micro-services and allow multiple instances for high-availability and scalability. It is planned to use Golang for service implementation, in order to be consistent with other micro-ONOS services and make use of patterns and libraries. The termination services provide an exterior interface for interacting with outside environment specified by O-RAN standards, they also supply an interior gRPC interface for RIC use.

Software Development Kit (SDK) is a set of software-building tools for a specific platform including a framework, code libraries, building blocks. In this case SDK provided for developing RIC applications and preventing generation of similar code patterns. SDK is responsible for dealing with complexities of gRPC services and client-side logic implementation. Inıtial version of SD-RAN architecture includes:

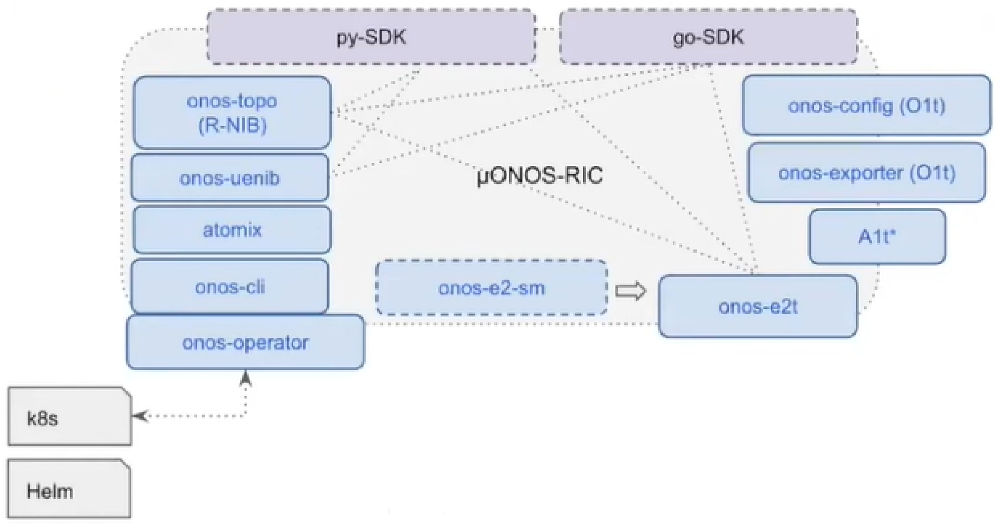

Figure 3.12: High-level structure of RIC [32]

- E2 Termination implemented via onos-e2t, tracking E2 node connections and handles subscriptions
- A1 Termination, not implemented yet
- O1 Termination partially implemented via onos-config and gNMI, rather than via NETCONF
- R-NIB Service implemented by onos-topo
- UE-NIB Service implemented by onos-uenib
- Storage Facilities backed by atomix and related libraries

SD-RAN's micro-ONOS based Near-RT RIC architecture can be seen in Figure 3.12. The app-SDKs are a collection of APIs and tools in different languages that enable xApps to interact with RIC platform services. onos-topo R-NIB provides R-NIB functionality for RAN entities and their relations that are configured and discovered while onos-uenib trakcs information about RAN user elements. onos-cli is a single CLI executable for multiple sub-components, onos-operator extends Kubernetes API with custom resources. onos-config is transactional initial or runtime configuration of targets such as xApps, and onos-exporter is composed of scrapes, formats and exports metrics to backend. onos-e2t is an intelligent proxy and adapter for handling the interactions between RIC components and RAN nodes by implementing E2-AP southbound and coverting ASN. onos-e2-sm is deployed as a plugin to e2t to provide protobuf translations for O-RAN Service Models (SMs) and Go code for ASN.1 PER encoding. atomix includes data structures and synchronization pritimitives for building and distributed ststems such as backend databases, replication protocols.

RIC platform and app-SDKs together allow flexibility in xApp development thanks to multiple KPM applications, applications that can subscribe to multiple SMs and choose not to subscribe to any SMs. Note that this structure is used in DT Berlin trial. ONF and Deutsche Telekom (DT) hosted a live event for launching the first fully disaggregated O-RAN 5G field trial. Interconnected components from eight vendors with ONF SD-RAN's open RAN Near-RT RIC were used in the trial aiming to leverage the O-RAN architecture. This was a huge achievement for open source, disaggregated RAN and multi-vendor advancement. SD-RAN participants include AirHop, Edgecore, Facebook, Foxconn, Intel, Radisys, Supermicro, TIP and Wiwynn, and the trial is continuing to expand and diversify as additional vendor components are deployed. Aether hosts the Radisys containerized CU while the Intel Smart Edge Open software toolkit hosts the Radisys DU to enable cloud-native deployment of the RAN workload with optimization on the 3rd Gen Intel Xeon Scalable processor and Intel vRAN Dedicated Accelerator ACC100. The Radisys CU and DU are integrated with ONF's nRT-RIC, xApps and SD-Core 5G core. Aether's components are interconnected with a P4-programmable Intel Tofino Intelligent Fabric Processor switch [**bertri**].

3.4.1 SD-RAN-in-a-box

SDRAN-in-a-Box (RiaB) is a SD-RAN cluster which is able to operate within a single host machine, providing a development and test environment for ONF SD-RAN community. RiaB deploys SD-RAN infrastructure, the EPC (OMEC), emulated RAN (CU/DU/UE), and ONOS RAN Intelligent Controller (ONOS RIC) services, over Kubernetes. On top of the SD-RAN infrastructure, end-to-end tests in terms of the user plane and the SD-RAN control

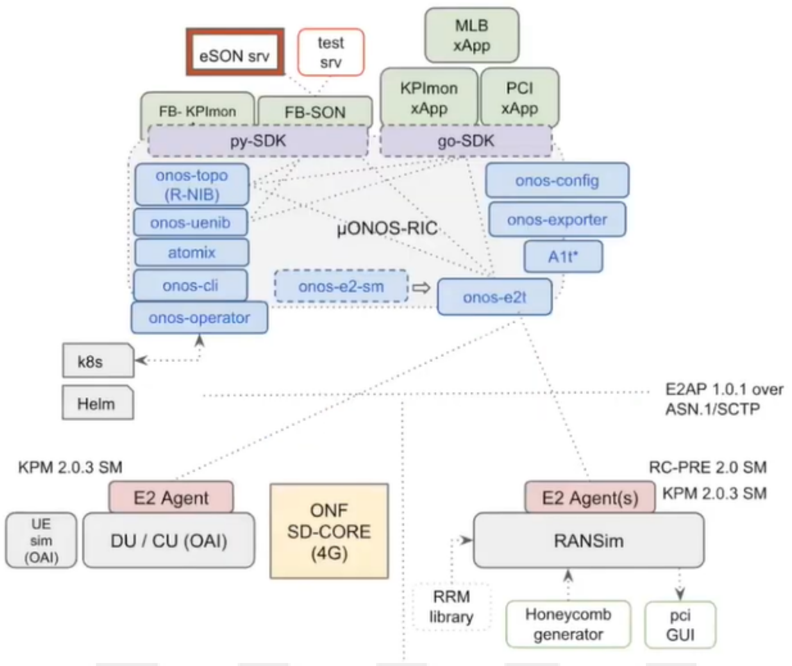

Figure 3.13: SD-RAN-in-a-box (RiaB) [32]

plane can be conducted [33]. Features of RiaB are Kubernetes and Helm required for SD-RAN services, an emulator or simulator for RAN, and support of end-to-end connectivity test. E2E connectivity test can be done either for user plane pr SD-RAN control plane, the latter using xAPPs such as onos-kpimon, onos-pci, onos-mlb and onos-mho. There are three choices to emulate or simulate RAN:

- RAN-Simulator
- OMEC / CU-CP / OAI nFAPI emulator for DU/UE
- OMEC / CU-CP / OAI DU and UE with USRP hardware and/or LTE smartphones

SD-RAN-in-a-box can be seen in Figure 3.13.

RiaB is available with many different versions (latest, master-stable, release/tag, dev), and options. There are four options: ONOS RIC with OAI nFAPI emulator, ONOS RIC with RAN-Simulator, ONOS RIC with OAI and

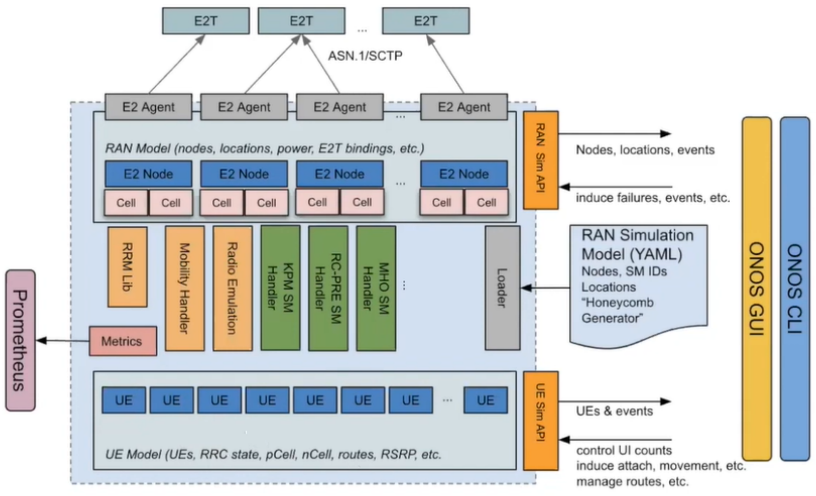

Figure 3.14: Architecture of RAN Simulator [31]

USRP devices, and Facebook-AirHop xAPP use case. The ONOS RIC with RAN-Simulator option deploys ONOS RIC services with RAN-Simulator, supporting an E2E connectivity on the SD-RAN control plane. Architecture of it can be seen in Figure 3.14. It involves E2 nodes, RAN environment, data stores, RAN simulator APIs and RAN simulator CLI.

RAN Simulator allows simulation of a number of RAN CU/DU nodes and RU cells via the O-RAN E2AP standard. The simulated RAN environment is described using a YAML model file loaded at the start. The simulator offers a gRPC API that to be used at runtime to induce changes in order to simulate a dynamically changing environment. The software is accompanied by a number of utilities that allow generation of YAML files that describe RAN topologies and various environmental metrics. CLI for the RAN simulator is available via onos-cli ransim usage and allows access to all major features of the simulator gRPC API, for controlling and monitoring the changes to the simulated environment [31].

3.4.2 micro-ONOS

micro-ONOS is the next generation architecture of ONOS, and natively based on new generation control and configuration interfaces and standards. It is

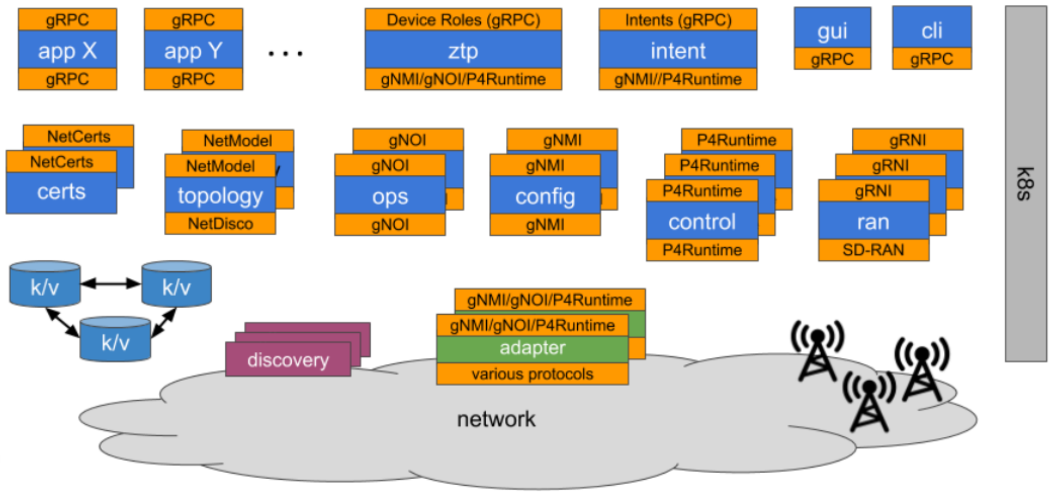

Figure 3.15: micro-ONOS Deployment Architecture [32]

implemented in systems-level languages, mostly Go, also modular and based on gRPC. micro-onos is composed of micro-services and deployable on Kubernetes. The platform enables network operations such as configuration, maintenance, and monitoring of network devices, validation of network topology, and collecting fine-grained performance metrics. Architecture of deploying micro-ONOS is given in Figure 3.15.

Development work on optimizing the micro-services based ONOS, micro-ONOS, for RAN control has already started. Two sample RRM applications have also been developed for handover control and mobility load balancing. A first-generation implementation of ONOS has been in use for more than five years and is the leading open-source SDN control plane in terms of high availability, performance and scalability. However, a next generation of ONOS i.e., micro-ONOS RIC will run as a logically centralized SD-RAN controller, and adopt a micro-services architecture [1]. High level architecture of micro-ONOS RIC can be seen in Figure 3.16.

Deployment configurations can be found in the onos-helm-chart repository [34] of onosproject. Each service has a chart directory, for instance, onos-topo chart is in onos-helm-charts/onos-topo. In order to deploy micro-ONOS microservices on Kubernetes, a package manager is needed and for that Helm is used. Helm is a package manager for Kubernetes to share, find and use software. Helm allow projects to provide a set of templates for the resources

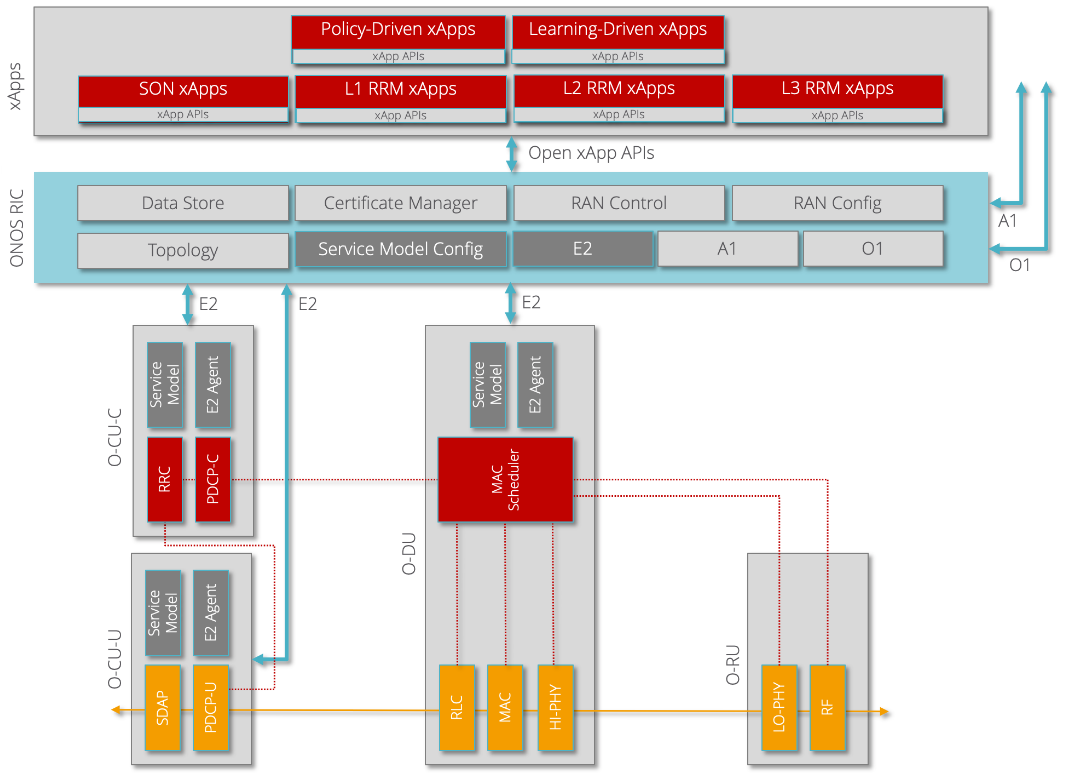

Figure 3.16: High Level micro-ONOS RIC Architecture [1]

deployed on Kubernetes. There are individual and over-arching charts i.e., umbrella charts. Components of the project can be either deployed individually or altogether via an umbrella chart, or combination of both. In order to deploy these components, a namespace must be created and Atomix contoller must be deployed in the corresponding namespace. Atomix is an open source tool that provides the components to manage and coordinate Kubernetes applications. In ONOS project, there is an umbrella chart called onos-umbrella that contains:

- onos-topo chart provides topology management for micro-ONOS core services and applications and also deploys custom Atomic resources to store all the topology information in a fail-safe way.
- onos-config is the ONOS configuration subsystem built using micro-ONOS architecture
- onos-cli is the implementation of ONOS command line interface providing a single CLI such that different ONOS subsystems can be reached
- onos-gui is the GUI for micro-ONOS

Helm chart provides resources for deploying services on Kubernetes:

- Deployment that provides a template for ONOS Config pods
- ConfigMap that provides test configurations for the application
- Service which exposes ONOS Config to other applications
- Secret that provides TLS certificates for end-to-end encryption
- Ingress that optionally provides support for external load balancing

Deployment can be done locally by Kind or bare metal. For local deployment with Kind, Docker must be installed for building and deploying images. Then Kind and Helm must be installed. Kind is a tool for running Kubernetes clusters using Docker. The installation steps are explained in detail in Chapter 4. There are prerequisites for any deployment scenario, these are adding CORD Helm char repository, Atomix repository and onosproject repository. After adding the Helm chart repositories, a namespace called micro-onos should be created. Note that micro-onos name is not compulsory but preferred for consistency in documentation. Next, Atomix controller and ONOS operator must be deployed. Finally a complete set of micro-ONOS can be deployed either individually or using the umbrella chart [35].

Chapter 4

Deployment

After investigating O-RAN scenarios in Chapter 3, it was decided to deploy micro-ONOS, test it and then monitor KPI using the xApp for ONOS SD-RAN, onos-kpimon. In this chapter, the deployment procedure will be presented.

The onos-kpimon is the xApp that runs on ONOS SD-RAN to monitor the KPI. onos-kpimon collects KPIs reported by E2 nodes through the KPM service model version 2.0. Since ONOS SD-RAN has multiple micro-services running on the Kubernetes platform, onos-kpimon should run on the Kubernetes along with the other ONOS SD-RAN micro-services. In order to deploy onos-kpimon on the Kubernetes, a Helm chart is necessary, which is in the sdran-helm-charts repository. onos-kpimon should run together with other micro-services, such as Atomix, onos-operator, onos-e2t, onos-uenib, onos-topo, and onos-cli. Luckily, sd-ran umbrella chart can be used to deploy all essential micro-services and onos-kpimon.

For the practical part of this work, a Virtual Machine (VM) was provided with Ubuntu 22.04. Host name was added and checked:

```
sudo nano /etc/hosts
cat /etc/hosts

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.0.1 host . minikube . internal
192.168.122.60 u22-k8s-master control-plane.minikube.internal
```

Then firewall was turned-off and IP packet filter rules can be seen with following command:

```
sudo iptables -nL
```

The VM was cleaned and updatet:

```
sudo apt clean
sudo apt update
sudo apt -y full - upgrade
```

4.1 Docker Installation

Docker is a platform designed for developers to build, share, and run modern applications. Docker enables the separation of applications infrastructure so that the software can be delivered quickly. As needed for Kubernetes to run, Docker was installed by following the guide [36]:

To uninstall Docker Engine, CLI and containerd:

```
sudo apt-get purge docker-ce docker-ce-cli containerd.io
```

To delete images, containers and volumes:

```
sudo rm -rf /var/lib/docker
sudo rm -rf /var/lib/containerd
```

To refresh the repositories:

```
sudo apt - get update
```

For needed dependencies:

```
sudo apt install apt-transport-https curl gnupg-agent ca-certificates software
    -properties-common -y
```

To add GPG key for Docker Community Edition (CE):

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
```

To install Docker and the additional packages, libraries and dependencies:

```
sudo apt install docker-ce docker-ce-cli containerd.io -y
```

In order to run docker without invoking sudo each time, add logged-in user to the docker group:

```
sudo usermod -aG docker \
```

To verify Docker is installed:

```
docker version

Client: Docker Engine - Community
 Version:           20.10.21
 API version:       1.41
 Go version:        go1.18.7
 Git commit:        baeda1f
 Built:             Tue Oct 25 18:01:58 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.21
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.18.7
  Git commit:       3056208
  Built:            Tue Oct 25 17:59:49 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.9
  GitCommit:        1c90a442489720eec95342e1789ee8a5e1b9536f
 runc:
  Version:          1.1.4
  GitCommit:        v1.1.4-0-g5fd4c4d
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

From this output, it can bee seen that Docker 20.10.21 was installed successfully.

4.2 Kubernetes Installation

Kubernetes is open-source orchestration software that provides an API to control how and where containers will run. It allows to run Docker containers and workloads and helps tackling some of the operating complexities when moving to scale multiple containers, deployed across multiple servers. Being one of the main prerequisites of micro-ONOS deployment, Kubernetes was installed following the guide [37]:

```
To check if br_netfilter is loaded:
lsmod | grep br_netfilter

br_netfilter           32768  0
bridge                307200  1 br_netfilter
```

To configure sysctl:

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF

net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1

sudo sysctl   system
* Applying /etc/sysctl.d/10-console-messages.conf ...
kernel.printk = 4 4 1 7
* Applying /etc/sysctl.d/10-ipv6-privacy.conf ...
net.ipv6.conf.all.use_tempaddr = 2
net.ipv6.conf.default.use_tempaddr = 2
* Applying /etc/sysctl.d/10-kernel-hardening.conf ...
kernel.kptr_restrict = 1
* Applying /etc/sysctl.d/10-magic-sysrq.conf ...
kernel.sysrq = 176
* Applying /etc/sysctl.d/10-network-security.conf ...
net.ipv4.conf.default.rp_filter = 2
net.ipv4.conf.all.rp_filter = 2
* Applying /etc/sysctl.d/10-ptrace.conf ...
kernel.yama.ptrace_scope = 1
* Applying /etc/sysctl.d/10-zeropage.conf ...
vm.mmap_min_addr = 65536
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.core_uses_pid = 1
net.ipv4.conf.default.rp_filter = 2
net.ipv4.conf.default.accept_source_route = 0
sysctl: setting key "net.ipv4.conf.all.accept_source_route": Invalid argument
net.ipv4.conf.default.promote_secondaries = 1
sysctl: setting key "net.ipv4.conf.all.promote_secondaries": Invalid argument
net.ipv4.ping_group_range = 0 2147483647
net.core.default_qdisc = fq_codel
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
fs.protected_regular = 1
fs.protected_fifos = 1
* Applying /usr/lib/sysctl.d/50-pid-max.conf ...
kernel.pid_max = 4194304
* Applying /usr/lib/sysctl.d/99-protect-links.conf ...
fs.protected_fifos = 1
fs.protected_hardlinks = 1
fs.protected_regular = 2
fs.protected_symlinks = 1
* Applying /etc/sysctl.d/99-sysctl.conf ...
* Applying /etc/sysctl.d/k8s.conf ...
```

```
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
* Applying /etc/sysctl.conf ...
```

To install kubeadm, kubelet, kubectl, first updating:

```
sudo apt update
sudo apt -y full-upgrade
```

To add Kubernetes repository for Ubuntu 22.04:

```
sudo apt install curl apt-transport-https -y
curl -fsSL  https://packages.cloud.google.com/apt/doc/apt-key.gpg|sudo gpg --
    dearmor -o /etc/apt/trusted.gpg.d/k8s.gpg
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key
    add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/
    apt/sources.list.d/kubernetes.list
```

To install required packages:

```
sudo apt update
sudo apt install wget curl vim git kubelet kubeadm kubectl -y
sudo apt-mark hold kubelet kubeadm kubectl
```

To check installation via kubectl version:

```
kubectl version --client && kubeadm version
```

To disable swaps:

```
sudo swapoff -a
```

To check that swap has been disabled:

```
free -h
total        used        free      shared   buff/cache   available
Mem:           7.8Gi       267Mi       5.5Gi       1.0Mi       2.0Gi       7.2
    Gi
Swap:             0B          0B          0B
```

To confirm setting is correct:

```
sudo mount -a
free -h
```

To enable kernel modules and configure sysctl:

```
sudo modprobe overlay
sudo modprobe br_netfilter
sudo tee /etc/sysctl.d/kubernetes.conf<<EOF
```

```
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF
sudo sysctl   system
```

Downgrade:

```
apt install kubeadm=1.22.0-00
apt install kubelet=1.22.0-00
apt install kubectl=1.22.0-00
```

This downgrade was performed because otherwise images cannot be pulled since Kubernetes stopped supporting Docker with newer versions. The reason is explained in more detail in Appendix A.

Pulling images:

```
sudo kubeadm config images pull
I1106 17:36:01.697440   26375 version.go:255] remote version is much newer: v1
    .25.3; falling back to: stable-1.22
[config/images] Pulled k8s.gcr.io/kube-apiserver:v1.22.15
[config/images] Pulled k8s.gcr.io/kube-controller-manager:v1.22.15
[config/images] Pulled k8s.gcr.io/kube-scheduler:v1.22.15
[config/images] Pulled k8s.gcr.io/kube-proxy:v1.22.15
[config/images] Pulled k8s.gcr.io/pause:3.5
[config/images] Pulled k8s.gcr.io/etcd:3.5.0-0
[config/images] Pulled k8s.gcr.io/coredns/coredns:v1.8.4

sudo kubeadm init
I1106 17:39:49.470517   26849 version.go:255] remote version is much newer: v1
    .25.3; falling back to: stable-1.22
[init] Using Kubernetes version: v1.22.15
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your
    internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm
    config images pull'
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [kubernetes kubernetes.
    default kubernetes.default.svc kubernetes.default.svc.cluster.local u22-
    k8s-master] and IPs [10.96.0.1 192.168.122.61]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [localhost u22-k8s-
    master] and IPs [192.168.122.61 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [localhost u22-k8s-
```

```
    master] and IPs [192.168.122.61 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/
    kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config
    .yaml"
[kubelet-start] Starting the kubelet
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/
    manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control plane as
    static Pods from directory "/etc/kubernetes/manifests". This can take up
    to 4m0s
[apiclient] All control plane components are healthy after 5.003064 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config"
    in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.22" in namespace kube-system
    with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node u22-k8s-master as control-plane by
    adding the labels: [node-role.kubernetes.io/master(deprecated) node-role.
    kubernetes.io/control-plane node.kubernetes.io/exclude-from-external-load-
    balancers]
[mark-control-plane] Marking the node u22-k8s-master as control-plane by
    adding the taints [node-role.kubernetes.io/master:NoSchedule]
[bootstrap-token] Using token: 1y7f0i.1876ottqvo3rusns
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC
    Roles
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to get
    nodes
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post
     CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] configured RBAC rules to allow the csrapprover controller
    automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] configured RBAC rules to allow certificate rotation for all
    node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public"
    namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a
    rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:
  mkdir -p \
```

To configure kubectl:

```
mkdir -p HOME/.kube
sudo cp -f /etc/kubernetes/admin.conf HOME/.kube/config
sudo chown (id -u):(id -g) HOME/.kube/config
To check cluster status:
kubectl cluster-info
Kubernetes control plane is running at https://192.168.122.61:6443
CoreDNS is running at https://192.168.122.61:6443/api/v1/namespaces/kube-
    system/services/kube-dns:dns/proxy
```

To install Kubernetes network plugin (Flannel network was chosen):

```
wget https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation
    /kube-flannel.yml
```

To install Flannel:

```
kubectl apply -f kube-flannel.yml
```

For confirming pods are running:

```
kubectl get pods -n kube-flannel
NAME                    READY   STATUS    RESTARTS   AGE
kube-flannel-ds-dm52t   1/1     Running   0          13s
```

To confirm master node is ready:

```
kubectl get nodes -o wide
NAME             STATUS   ROLES                  AGE   VERSION   INTERNAL-IP
         EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION      CONTAINER-
    RUNTIME
u22-k8s-master   Ready    control-plane,master   11m   v1.22.0
    192.168.122.61   <none>        Ubuntu 22.04.1 LTS   5.15.0-52-generic
    docker://20.10.21
```

To confirm if the cluster is functional, deploy a test application:

```
kubectl apply -f https://k8s.io/examples/pods/commands.yaml
```

To see that pod started:

```
kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
command-demo   0/1     Pending   0
```

Finally, to see the kubeadm agents actually deployed:

```
kubectl get pods -A
NAMESPACE      NAME                                     READY   STATUS
                 RESTARTS      AGE
default        command-demo                             1/1     Running
                0              93s
kube-flannel   kube-flannel-ds-dm52t                    1/1     Running
                0              3m4s
kube-system    coredns-78fcd69978-82fzt                 1/1     Running
                0              13m
kube-system    coredns-78fcd69978-s9tgr                 1/1     Running
                0              13m
kube-system    etcd-u22-k8s-master                      1/1     Running
                0              13m
kube-system    kube-apiserver-u22-k8s-master            1/1     Running
                0              13m
kube-system    kube-controller-manager-u22-k8s-master   1/1     Running
                0              13m
kube-system    kube-proxy-q7155                         1/1     Running
                0              13m
kube-system    kube-scheduler-u22-k8s-master            1/1     Running
                0              13m
```

4.3 micro-ONOS

One of the goals of the micro-ONOS project is to provide simple deployment options that integrate with modern technologies. Deployment configurations can be found in the onos-helm-charts repository. Each onos service has a directory containing its chart.

Before deploying micro-ONOS services, there are several prerequisites that must be met: installation of Docker, Kubernetes, Kind, and Atomix controller. Since Docker and Kubernetes were already installed, the next step is Kind.

For Kind installation, Go is needed. Go installation:

```
sudo snap install go -classic
confirmation -> go 1.18.7 from Michael Hudson-Doyle (mwhudson) installed
```

To test if installation done correctly:

```
vim hello.go
// Hello Word in Go by Vivek Gite
package main

// Import OS and fmt packages
import (
        "fmt"
        "os"
)
// Let us start
```

```
func main() {
    fmt.Println("Hello, world!")  // Print simple text on screen
    fmt.Println(os.Getenv("USER"), ", Let's be friends!") // Read Linux \
```

```
go run hello.go
Hello, world!
ubuntu , Let's be friends!
```

To download Kind:

```
go install sigs.k8s.io/kind@v0.17.0
kind create cluster
```

Then “kind: command not found” error was encountered, So it was suggested that the directory was added to PATH. While doing that a reboot was performed, and with reboot Kubernetes stopped working properly because kubectl did not work anymore. In order to solve it, several methods were tried but none of them worked. Hence, Kubernetes was uninstalled totally, cleaned, and installed again following the same steps in Section 4.2.

New token was noted:

```
kubeadm join 192.168.122.61:6443 --token 6ob2az.egwww68k7gni96jz \
        --discovery-token-ca-cert-hash sha256:
            bba941a7353232554eb1a7b3caa4020843ddb8953b9e35269fa2f1ead62d8225
```

To check if it works correctly:

```
kubectl cluster-info
Kubernetes control plane is running at https://192.168.122.61:6443
CoreDNS is running at https://192.168.122.61:6443/api/v1/namespaces/kube-
    system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump
    '.
```

To install Kind, an online guide [38] was followed:

```
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.14.0/kind-linux-amd64
chmod +x ./kind
sudo mv ./kind /bin/kind
kind create cluster
Creating cluster "kind" ...
 Ensuring node image (kindest/node:v1.24.0)
 Preparing nodes
 Writing configuration
 Starting control-plane
 Installing CNI
 Installing StorageClass
 Set kubectl context to "kind-kind"
You can now use your cluster with:
```

```
kubectl cluster-info --context kind-kind

Have a question, bug, or feature request? Let us know! https://kind.sigs.k8s.
    io/#community
```

Check cluster created with kind create: (starting kind)

```
kind get clusters
kind
```

Export the configuration:

```
kind get kubeconfig > ~/.kube/kind
```

Export new environment to access the Kubernetes cluster:

```
export KUBECONFIG=~/.kube/kind
```

Add Atomix Helm chart repo:

```
helm repo add atomix https://charts.atomix.io
"atomix" has been added to your repositories
```

Add onos project Helm chart repo:

```
helm repo add onosproject https://charts.onosproject.org
"onosproject" has been added to your repositories
```

Update the local cache with charts from these repos:

```
helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "atomix" chart repository
...Successfully got an update from the "onosproject" chart repository
...Successfully got an update from the "cord" chart repository
Update Complete.  Happy  Helming!
To see chart and app versions:
helm search repo onos
NAME                                            CHART VERSION   APP VERSION
                DESCRIPTION
cord/onos                                       3.0.2           2.2.1
                      Open Network Operating System
cord/onos-progran                               1.2.7           0.1.8
                      ONOS with progran APP
cord/onos-service                               3.0.1           3.0.0
                      A Helm chart for XOS's "onos-service" service, ...
onosproject/onos-classic                        0.1.31          2.5.7-rc1
                  ONOS cluster
onosproject/onos-cli                            1.3.11          v0.9.25
                    ONOS Command Line Interface
onosproject/onos-config                         1.8.0           v0.11.0
```

```
                    ONOS Config Manager
onosproject/onos-gui                        1.0.9           v0.7.2
                    ONOS Graphical User Interface
onosproject/onos-operator                   0.5.6           v0.5.2
                      ONOS  Operator
onosproject/onos-topo                       1.4.0           v0.10.3
                    ONOS Topology service
onosproject/onos-tost                       0.1.43          stable
    -2020-12-15        ONOS helm chart for TOST
onosproject/onos-umbrella                   1.3.0           v1.1.0
                     Umbrella chart to deploy all  ONOS
onosproject/onos-ztp                        0.0.5           v0.6.0
                     ONOS Zero Touch Provisioning Subsystem
onosproject/389ds                           0.1.1           fedora-32
                  389 Directory Server
onosproject/config-model-aether             2.1.1           2.0.0
                      Aether config model
onosproject/config-model-devicesim          1.0.5           1.0.0
                      Devicesim model
onosproject/config-model-devicesim-1-0-0    1.0.6           1.0.0
                      Devicesim model
onosproject/config-model-ietf               1.0.1           1.0.0
                      IETF config model
onosproject/config-model-openconfig         1.0.1           1.0.0
                      OpenConfig model
onosproject/config-model-rbac               1.0.3           1.0.0
                      RBAC config model
onosproject/config-model-stratum            1.0.3           1.0.0
                      Stratum model
onosproject/config-model-stratum-1-0-0      1.0.8           1.0.0
                      Stratum model
onosproject/config-model-testdevice         2.0.4           2.0.0
                      TestDevice model
onosproject/config-model-testdevice-1-0-0   1.0.6           1.0.0
                      TestDevice model
onosproject/config-model-testdevice-2-0-0   2.0.5           2.0.0
                      TestDevice model
onosproject/device-provisioner              0.0.7           v0.0.13
                    device-provisioner Helm chart for Kubernetes
onosproject/device-simulator                0.0.7           v0.6.3
                      ONOS Config Device Simulator
onosproject/dex-ldap-umbrella               0.2.0           v0.0.0
                     Umbrella chart to deploy Dex, OpenLDAP & phpLDA...
onosproject/fabric-sim                      0.1.14          v0.1.14
                    ONOS Fabric Simulator service
onosproject/keycloak-389-umbrella           0.1.5           v0.0.0
                     Umbrella chart to deploy Keycloak, 389ds & phpL...
onosproject/ran-simulator                   1.0.12          v0.7.3
                     ONOS RAN Simulator
onosproject/sdcore-adapter                  0.0.9           v0.1.10
                    ONOS SD-Core Adapter
onosproject/stratum-simulator               0.0.3           v0.1.0
                     Stratum Simulator
onosproject/tlaplus-monitor                 0.0.2           1
                          TLA+ Conformance Monitor
onosproject/topo-discovery                  0.0.4           v0.0.7
```

```
                     topo-discovery Helm chart for Kubernetes
onosproject/wcmp-app                           0.1.2           v0.0.4
                     wcmp-app Helm chart for Kubernetes
cord/voltha-infra                              2.10.8          2.10
                       A Helm chart to install the required infrastruc...
```

Configure the micro-onos namespace:

```
kubectl create namespace micro-onos
namespace/micro-onos created
```

The first thing that needs to be deployed in any ONOS deployment is the Atomix Custom Resource Definitions (CRDs) and Go controller and they should be deployed in the kube-system namespace:

```
ubuntu@u22-k8s-master:~\
```

onos-operator ensures that ONOS CRDs and their controllers for onos-topo and onos-config are deployed in to the cluster. To deploy onos-operator:

```
helm install -n kube-system onos-operator onosproject/onos-operator
NAME: onos-operator
LAST DEPLOYED: Fri Nov 11 21:45:17 2022
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

A complete set of micro-ONOS services can be deployed with just the overarching onos-umbrella chart. To deploy micro-ONOS services:

```
micro-onos install onos-umbrella onosproject/onos-umbrella
Error: INSTALLATION FAILED: unable to build kubernetes objects from release
    manifest: [resource mapping not found for name: "onos-consensus-store"
    namespace: "micro-onos" from "": no matches for kind "ConsensusStore" in
    version "consensus.atomix.io/v1beta1"
ensure CRDs are installed first, resource mapping not found for name: "onos-
    config" namespace: "micro-onos" from "": no matches for kind "
    StorageProfile" in version "atomix.io/v3beta3"
ensure CRDs are installed first, resource mapping not found for name: "onos-
    topo" namespace: "micro-onos" from "": no matches for kind "StorageProfile
    " in version "atomix.io/v3beta3"
ensure CRDs are installed first]
```

The error above is due to incompatible apiVersion and missing ConsensusStore which is related to Custom Resource Definitions (CRDs). Note that this problem was encountered for only onos-config and onos-topo. onos-gui and onos-cli were deployed without any issues. After checking and making sure micro-onos exists, the apiVersion that is used by Atomix was checked by:

```
kubectl api-versions | grep -i atomix
atomix.io/v2beta1
primitives.atomix.io/v2beta1
sidecar.atomix.io/v2beta1
storage.atomix.io/v2beta2
storage.cloud.atomix.io/v1beta1
```

Hence it was understood that apiVersion of ConsensusStore and Storage-Profile did not match with Atomix. For StorageProfile the correct version should be ”v2beta1” and for ConsensusStore it is completely missing. The reason was that CRD for ConsensusStore was never installed. First, missing CRD was installed. The actual repository of consensus-storage controller was a different one, so the correct repository was downloaded and the chart was installed from the root:

```
helm install atomix-consensus-controller .
```

Next, StorageProfile apiVersion was turned back to correct version by downloading the corresponding helm charts from repo:

```
git clone https://github.com/atomix/consensus-storage
helm install  -n kube-system atomix-consensus-controller .
```

And modifying corresponding YAML files i.e., storageprofile.yaml present in onos-config/templates/ and onos-topo/templates/, then installing onos-config from the locally modified helm chart and not from repo:

```
helm -n micro-onos install onos-config onosproject/onos-config
Error: INSTALLATION FAILED: unable to build kubernetes objects from release
    manifest: error validating "": error validating data: [ValidationError(
    StorageProfile.spec): unknown field "bindings" in io.atomix.v2beta1.
    StorageProfile.spec, ValidationError(StorageProfile.spec): unknown field "
    proxy" in io.atomix.v2beta1.StorageProfile.spec, ValidationError(
    StorageProfile.spec): missing required field "selector" in io.atomix.
    v2beta1.StorageProfile.spec, ValidationError(StorageProfile.spec): missing
     required field "drivers" in io.atomix.v2beta1.StorageProfile.spec]
```

Another error was encountered, the same thing happened for onos-topo as well:

```
helm -n micro-onos install onos-topo onosproject/onos-topo
Error: INSTALLATION FAILED: unable to build kubernetes objects from release
    manifest: error validating "": error validating data: [ValidationError(
    StorageProfile.spec): unknown field "bindings" in io.atomix.v2beta1.
    StorageProfile.spec, ValidationError(StorageProfile.spec): unknown field "
    proxy" in io.atomix.v2beta1.StorageProfile.spec, ValidationError(
    StorageProfile.spec): missing required field "selector" in io.atomix.
    v2beta1.StorageProfile.spec, ValidationError(StorageProfile.spec): missing
     required field "drivers" in io.atomix.v2beta1.StorageProfile.spec]
```

It was checked if onos-operator is running:

```
kubectl get po -n kube-system
NAME                                              READY   STATUS    RESTARTS
            AGE
atomix-consensus-controller-6d9dbf95bd-297vd      1/1     Running   13 (6m26s
    ago)   69m
atomix-controller-6558c98fd7-g7f5c                1/1     Running   0
                  11d
atomix-raft-storage-controller-697d48799c-sl67l   1/1     Running   0
                  11d
coredns-6d4b75cb6d-4684g                          1/1     Running   0
                  12d
coredns-6d4b75cb6d-5zsf6                          1/1     Running   0
                  12d
etcd-kind-control-plane                           1/1     Running   0
                  12d
kindnet-m7jkq                                     1/1     Running   0
                  12d
kube-apiserver-kind-control-plane                 1/1     Running   0
                  12d
kube-controller-manager-kind-control-plane        1/1     Running   0
                  12d
kube-proxy-h6n6m                                  1/1     Running   0
                  12d
kube-scheduler-kind-control-plane                 1/1     Running   0
                  12d
onos-operator-app-57486b4466-jkvr2                1/1     Running   0
                  8d
onos-operator-topo-66c84b84fc-7p94b               1/1     Running   0
                  8d
```

Then it was understood that errors were due to missing fields in SPEC files. Spec files are plain-text files that are used to construct spec strings. The problem required further research but unfortunately there was not enough time to conduct that. Since the deployment of micro-ONOS was not complete, testing and observing of KPIs was not realized either.

Chapter 5

Discussion

This chapter aims to discuss about the deployment of micro-ONOS and the reasons why it was not finalized completely. Many obstacles were faced while installing Kubernetes and micro-ONOS. First of all, as also explained in Appendix A, Kubernetes was downgraded to version 1.22.0 in order to be compatible with Docker. The original version that was installed previously was 1.24.0.

Next, while deploying micro-ONOS, an issue was encountered related to Custom Resource Definitions, CRDs. A custom resource is an extension of the Kubernetes API that is not necessarily available in a default Kubernetes installation. It represents a customization of a particular Kubernetes installation. However, many core Kubernetes functions are now built using custom resources, making Kubernetes more modular. The CustomResourceDefinition API resource enables to define custom resources. Defining a CRD object creates a new custom resource with a name and schema that can be specified. The Kubernetes API serves and handles the storage of custom resource. The name of a CRD object must be a valid DNS subdomain name [39].

There was a CRD missing which was in the end downloaded from another, correct, helm chart belonging to a different repository. However, there was another challenge, some fields were missing: "bindings", "proxy", "selector", and "drivers" in SPEC files. Unfortunately, this problem could not be solved, hence, deployment of micro-ONOS was not finalized.

In summary, it can be said that the documentation for deploying micro-ONOS can be improved since there are a few incompatibilities between Helm charts, apiVersions and missing CRDs. Some of these problems are due to upgrades and new releases, for instance, apiVersion incompatibility can occur when an upgrade or new release happens. Nevertheless, installation of Docker and Kubernetes, along with other elements needed for micro-ONOS such as Helm, Go, Kind were done successfully in spite of the obstacles encountered.

Chapter 6

Conclusion

This thesis presented network automation scenarios and solutions proposed by members of O-RAN Alliance, starting from explaining the architectural aspects of O-RAN and its key components. O-RAN Alliance is working on evolving Radio Access Networks towards being intelligent an open. With its specification efforts, Open Software Community and Testing & Integration Centers O-RAN aims to provide an open community for standardizing and building a virtualized RAN built on open hardware and cloud.

The aim of this thesis was to investigate the projects and platforms proposed and built as solutions to current network automation challenges which are O-RAN SC Non-RT RIC, ONAP, OSM, and ONF SD-RAN. Architecture, components, and current status of each scenario was studied in detail, along with a few use cases or field cases when applicable. O-RAN SC Non-RT RIC project implements different parts of the Non-RT RIC with all the documentation related to releases available for public use. ONAP 5G Blueprint is being developed in close collaboration with 3GPP, ETSI, and the O-RAN Alliance including orchestration, life cycle management, network slicing, radio area network support, and network optimization, and showcasing ONAP's 5G capabilities. ETSI OSM is currently developing an open source MANO stack aligned with ETSI NFV Information Models where it is possible to onboard VNF and NS packages. ONF SD-RAN is developing Near-RT RIC and a set of exemplar xApps built on ONF platform in order to leverage O-RAN architecture and vision. For instance, in 2021 the first fully disaggregated, open RAN 5G field trial has been launched by ONF and Deutsche Telekom in Berlin which is a huge step towards realizing the vision of fully disaggregated and intelligent RAN. This trial is also a milestone for open RAN and demonstrates the maturity of the SD-RAN open source RIC and xApp development platform.

The deployment procedure was to set up a virtual infrastructure management framework, Kubernetes, on a VM followed by micro-services of an ONOS, micro-ONOS. During the installation of these software components, many challenges and issues were observed. Especially for micro-ONOS deployment, there were many issues encountered stemming from incompatibilities between Helm charts, versions and releases. Due to these technical obstacles, unfortunately it was not possible to complete the deployment of micro-ONOS.

In conclusion, it can be said that the platforms and projects proposed by O-RAN are collectively providing the standards and specifications envisioned. A global effort has been observed towards standardizing and specifying the building blocks and principles of O-RAN. There are hardware and software components that have already been deployed and tested. There are also many research and deployment activities going on currently. It can be suggested that O-RAN satisfies the openness that it promises, and the developments and advancements going on in the community already start shaping the future of RAN. However, the software components still need time and improvement to fully mature.

Bibliography

[1] *ONF's Software-Defined RAN Platform Consistent with the O-RAN Architecture.* (Accessed on 10/31/2022). URL: https://opennetworking.org/open-ran/.

[2] Massimo Condoluci and Toktam Mahmoodi. "Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges". In: *Computer Networks* 146 (2018), pp. 65–84. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.09.005. URL: https://www.sciencedirect.com/science/article/pii/S1389128618302500.

[3] *vRAN (Virtual Radio Access Network).* (Accessed on 06/15/2022). URL: https://www.gigabyte.com/Glossary/vran-virtual-radio-access-network.

[4] Andres Garcia-Saavedra and Xavier Costa-Pérez. "O-RAN: Disrupting the Virtualized RAN Ecosystem". In: *IEEE Communications Standards Magazine* 5.4 (2021), pp. 96–103. DOI: 10.1109/MCOMSTD.101.2000014.

[5] Quang Huy Duong, Ibrahim Tamim, Brigitte Jaumard, and Abdallah Shami. "A Column Generation Algorithm for Dedicated-Protection O-RAN VNF Deployment". In: *2022 International Wireless Communications and Mobile Computing (IWCMC).* 2022, pp. 1206–1211. DOI: 10.1109/IWCMC55113.2022.9825080.

[6] *Open RAN: A game-changer in mobile communications.* (Accessed on 10/31/2022). URL: https://www.thehansindia.com/business/open-ran-a-game-changer-in-mobile-communications-713293.

[7] *5G Networks.* (Accessed on 10/31/2022). URL: https://www.5g-networks.net/5g-technology/openran-o-ran-for-5g-explained/.

[8] *About O-RAN ALLIANCE.* (Accessed on 06/15/2022). URL: https://www.o-ran.org/about.

[9] *O-RAN Architecture Description.* v06.00. O-RAN Alliance. Mar. 2022. URL: https://www.o-ran.org/specification-access.

[10] *Open RAN.* (Accessed on 07/04/2022). URL: https://www.ericsson.com/en/openness-innovation/open-ran-explained.

[11] *STL.* (Accessed on 07/04/2022). URL: https://www.stl.tech.

[12] Aly S. Abdalla, Pratheek S. Upadhyaya, Vijay K. Shah, and Vuk Marojevic. "Toward Next Generation Open Radio Access Networks–What O-RAN Can and Cannot Do!" In: *IEEE Network* (2022), pp. 1–8. DOI: 10.1109/MNET.108.2100659.

[13] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. *Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges.* 2022. DOI: 10.48550/ARXIV.2202.01032. URL: https://arxiv.org/abs/2202.01032.

[14] *O-RAN SC Non-RT RIC project.* (Accessed on 08/04/2022). URL: https://docs.o-ran-sc.org/projects/o-ran-sc-nonrtric/en/cherry/overview.html.

[15] Zahid Ghadialy. *An Overview of O-RAN Architecture.* (Accessed on 10/21/2022). URL: https://www.parallelwireless.com/blog/an-overview-of-o-ran-architecture/.

[16] *O-RAN Architecture Description.* v06.00. O-RAN Alliance. Mar. 2022. URL: https://www.o-ran.org/specification-access.

[17] *O-RAN Near-Real-time RAN Intelligent Controller Architecture E2 General Aspects and Principles.* v02.01. O-RAN Alliance. Mar. 2022. URL: https://www.o-ran.org/specification-access.

[18] *Cloud Architecture and Deployment Scenarios for O-RAN Virtualized RAN.* v02.21. O-RAN Alliance. Oct. 2021. URL: https://www.o-ran.org/specification-access.

[19] *O-RAN Use Cases and Deployment Scenarios[White Paper].* v07.00. O-RAN Alliance. Feb. 2020. URL: https://www.o-ran.org/specification-access.

[20] family=Dryjanski given i=M. given=Marcin. *O-RAN Overview, Architecture, near-Real-Time RIC and Use Cases.* Oct. 2021. URL: https://www.linkedin.com/pulse/o-ran-overview-architecture-near-real-time-ric-use-cases-dryjanski/.

[21] *O-RAN Use Cases Detailed Specification.* v07.00. O-RAN Alliance. Mar. 2022. URL: https://www.o-ran.org/specification-access.

[22] *O-RAN SC Non-RT RIC project.* (Accessed on 10/31/2022). URL: https://docs.o-ran-sc.org/projects/o-ran-sc-nonrtric/en/cherry/overview.html.

[23] *Non-RealTime RIC (NONRTRIC).* (Accessed on 10/31/2022). URL: https://wiki.o-ran-sc.org/display/RICNR/.

[24] *ONAP Developer Wiki.* (Accessed on 10/31/2022). URL: https://wiki.onap.org//.

[25] *ONAP Issues Jakarta Release with expanded Security, O-RAN alignment, 5G enhancements, and more.* (Accessed on 10/31/2022). URL: https://www.onap.org/blog/2022/07/05/onap-issues-jakarta-release-with-expanded-security-o-ran-alignment-5g-enhancements-and-more.

[26] *ONAP Architecture.* (Accessed on 10/31/2022). URL: https://docs.onap.org/en/latest/platform/architecture/index.html.

[27] *ONAP 5G Blueprint Overview.* Open Network Automation Platform. URL: https://www.onap.org/architecture/use-cases-blueprints.

[28] *Open Source MANO Documentation.* (Accessed on 10/31/2022). URL: https://osm.etsi.org/docs/user-guide/latest/index.html.

[29] *OSM Usage.* (Accessed on 10/31/2022). URL: https://osm.etsi.org/docs/user-guide/latest/05-osm-usage.html.

[30] *ONF Areas Projects.* (Accessed on 10/31/2022). URL: https://opennetworking.org/open-ran/.

[31] *SD-RAN Documentation.* (Accessed on 10/31/2022). URL: https://docs.sd-ran.org/master/index.html.

[32] *SD-RAN Berlin Trial – Event Recap.* (Accessed on 10/31/2022). URL: https://opennetworking.org/news-and-events/blog/sd-ran-berlin-trial-event-recap/.

[33] *SDRAN-in-a-Box (RiaB).* (Accessed on 10/31/2022). URL: https://docs.sd-ran.org/master/sdran-in-a-box/README.html#sdran-in-a-box-riab.

[34] *Helm charts for ONOS (μONOS Architecture).* (Accessed on 10/31/2022). URL: https://github.com/onosproject/onos-helm-charts.

[35] *Deploying μONOS micro-services with HELM.* (Accessed on 10/31/2022). URL: https://docs.onosproject.org/onos-docs/docs/content/developers/deploy_with_helm/.

[36] *How To Install Docker On Ubuntu 22.04 — 20.04.* (Accessed on 10/21/2022). URL: https://cloudcone.com/docs/article/how-to-install-docker-on-ubuntu-22-04-20-04/.

[37] *Install Kubernetes Cluster on Ubuntu 22.04 with kubeadm.* (Accessed on 10/21/2022). URL: https://computingforgeeks.com/install-kubernetes-cluster-ubuntu-jammy/.

[38] *Guide to Running Kubernetes with Kind.* (Accessed on 10/21/2022). URL: https://phoenixnap.com/kb/kubernetes-kind.

[39] *Custom Resources.* (Accessed on 10/31/2022). URL: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/.

[40] *Don't Panic: Kubernetes and Docker.* (Accessed on 06/16/2022). URL: https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/.

[41] *Updated: Dockershim Removal FAQ.* (Accessed on 08/16/2022). URL: https://kubernetes.io/blog/2022/02/17/dockershim-faq/.

Appendix A

Appendix

During the procedure of setting up VM there were many unexpected problems and challenges with the implementation of Kubernetes. Even though following the installation of Kubernetes exactly, the master node configuration could not be done due to different errors. The first one was due to container runtime interface, since both Docker Engine and containerd were installed, the following error was given:

```
[init] Using Kubernetes version: v1.24.0 [preflight] Running pre-flight checks
     error execution phase preflight: [preflight] Some fatal errors occurred:
    [ERROR CRI]: container runtime is not running: output: time="2022-05-11T14
    :35:52+02:00" level=fatal msg="getting status of runtime: rpc error: code
    = Unimplemented desc = unknown service runtime.v1alpha2.RuntimeService" ,
    error: exit status 1 [preflight] If you know what you are doing, you can
    make a check non-fatal with ‘--ignore-preflight-errors=...‘ To see the
    stack trace of this error execute with --v=5 or higher
```

This problem solved by uninstalling Docker and installing again, this time going with Docker Engine only. After following the instructions for Kubernetes installment there was still an error which is:

```
[ERROR ImagePull]: failed to pull image k8s.gcr.io/kube-apiserver:v1.24.0:
    output: time="2022-05-11T14:49:41+02:00" level=fatal msg="connect: connect
     endpoint 'unix:///var/run/dockershim.sock', make sure you are running as
    root and the endpoint has been started: context deadline exceeded" , error
    : exit status 1
```

The reason behind the error was first thought to be due to still CRI. After uninstalling and installing docker again, the Docker Engine was installed once more. But the problem was not solved yet. Then it was come to realization that there was some problem related to “dockershim”. Finally, the reason behind was found: Kubernetes support for Docker via dockershim was being removed [40]. While installing Kubernetes, the latest version was downloaded automatically which was 1.24 [41]. Hence it did not support Docker through dockershim anymore. So when the master node was trying to be configured by following:

```
sudo kubeadm init
```

Kubernetes could not communicate to Docker therefore the image could not be pulled by Docker for Kubernetes.

There were two possible solutions:

- Downgrading the version of Kubernetes to the version which was used by the company (JMA)
- Keep the same Kubernetes version and use containerd instead of Docker

The second choice was not preferred because usage of containerd might be challenging and tricky in the beginning. Therefore, the solution chosen was to downgrade Kubernetes to version 1.22 which was used by the company. kubeadm, kubelet and kubectl were all downgraded to version 1.22 by:

```
apt install kubeadm=1.22.0-00
apt install kubelet=1.22.0-00
apt install kubectl=1.22.0-00
```

After this step the problem was solved and cluster was made.

List of Figures