Load balancing algorithm running on Open RAN RIC

Hyeon-Min Yoo
dept. Electronics and Information Convergence Engineering
KyungHee University
Yong-In, Republic of Korea
yhm1620@khu.ac.kr

Jong-Seok Rhee
dept. Electronics and Information Convergence Engineering
KyungHee University
Yong-In, Republic of Korea
howrhee@khu.ac.kr

Seok-Young Bang
dept. Electronics and Information Convergence Engineering
KyungHee University
Yong-In, Republic of Korea
qkdtjr97@khu.ac.kr

Een-Kee Hong
dept. Electronics and Information Convergence Engineering
KyungHee University
Yong-In, Republic of Korea
ekhong@khu.ac.kr

2022 13th International Conference on Information and Communication Technology Convergence (ICTC) | 978-1-6654-9939-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICTC55196.2022.9952635

***Abstract*— The O-RAN alliance has received attention by presenting an O-RAN architecture. They standardizes wireless interfaces to solve the compatibility problem between multi-vendor of existing radio access network (RAN) architecture. They also disclose an open source framework, which is applicable to programmable base station equipment. In this paper, we analyze the xApp and the near-real time RAN intelligent controller (RIC) serviced by the O-RAN-compatible SD-RAN platform, developed by the open networking foundation (ONF), and address simple simulation results.**

Keywords*— *O-RAN, SD-RAN, xApp, Near-RT RIC

I. Introduction

In February 2018, five global telecom operators (AT&T, China Mobile, Deutsche Telekom, NTT DOCOMO and Orange) launched O-RAN alliance to realize an open radio access network (RAN) including openness and intelligent for the 5G era [1]. O-RAN alliance has designed O-RAN architecture and encourages the equipment vendors to develop their equipment in accordance with this standard for enabling the interoperability. The components and functions of the O-RAN architecture include [2]:

• *RAN intelligent controller (RIC)*: A controller that manages and controls network. The non-real time RIC (non-RT RIC) is responsible for functions with a control latency of more than 1 second, and the near-real time RIC (near-RT RIC) for functions with 10 millisecond to 1 second. The non-RT RIC is managed by service management and organization (SMO), and mainly performs functions based on artificial intelligence and big data such as traffic pattern and QoS prediction. The near-RT RIC performs real-time radio resource management functions such as handover and load balancing. It is connected to a number of RANs (i.e., O-CUs), and transfers collected data to non-RT RIC.

• *O-RU/O-DU/O-CU*: In O-RAN architecture, the base station (eNB, gNB) was disaggregated into O-CU, O-DU, and O-RU. The O-CU is directly connected to the near-RT RIC. The O-DU and O-RU are connected by open fronthaul interfaces, each performing the functions of the upper physical layer and the lower physical layer [3].

• *xApp*: An application running on Near-RT RIC. An xApp performs the specific intelligent function. In other words, some RAN functions such as handover and load balancing can be executed in the near-RT RIC as an xApp.

To expand an O-RAN architecture ecosystem, various projects design RANs that follow this structure and disclose them as open sources [4]. Among the projects, the Open Networking Foundation (ONF) has developed and released the SD-RAN platform including cloud-native near-RT RIC and xApp that follow O-RAN architecture [5]. They also support software development kits (SDKs) so that users can develop the xApp themselves. In this paper, we analyze the open source xApp, near-RT RIC, and RAN simulator provided by SD-RAN platform, and introduce the development method of xApp and the simulation results.

II. Near-RT RIC, xApps, and RAN Simulator

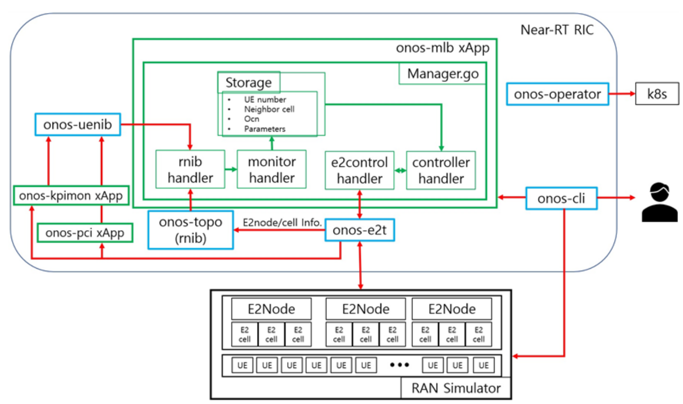


Fig. 1. Structure of xApp, RAN simulator, and near-RT RIC

Fig. 1 shows the structure of onos-mlb (mobility load balancing) xApp, RAN simulator, near-RT RIC released by ONF as an open source. It also includes the interface of onos-mlb xApp for communicating with other xApps, RAN simulator, and packages in the Near-RT RIC. Fig. 1 represents xApp in green, Near-RT RIC packages in blue, and RAN simulator in black line.

The ONF's xApp is written in Go language. There is no class in Go, so the method must be declared separately after declaring the structure. It has *Go routine*, which plays a similar role to C++'s *thread*. It also has *channel*, which transfers data between routines. In xApp, *manager* controls the behavior of xApp and declares *handlers* with structure. The handlers communicate with packages in near-RT RIC and RAN simulator using *Go routine* and *channel*. RAN simulator is implemented in software and replaces the actual hardware, O-CUs, O-DUs, and O-RUs. O-DU and O-CU are integrated to E2Node, O-RU is substitutes for E2Cell.

Near-RT RIC includes some packages for executing xApps. onos-e2t (e2 termination) is a package located at the bottom of the near-RT RIC, which serves as a proxy for the interface between RAN simulator's E2Node and other near-RT RIC packages, xApp. E2t exchanges messages with E2Node of RAN simulator and forwards the information of RAN simulator to the database package and e2control handler of the xApp. The databases defined in O-RAN architecture include onos-uenib and onos-topo (also called *rnib*, radio network information base). onos-topo stores information about E2Nodes and E2Cells, and onos-uenib stores information about the user equipments (UEs) and its neighbor cells. These databases can obtain the information from onos-kpimon (key performance indicator monitoring) xApp and onos-pci (physical cell id) xApp. onos-kpimon collects information on the number of UEs connected to each E2Cell, and onos-pci gets information on the neighbor cell of each UE by interacting with RAN simulator. In other words, for the execution of onos-mlb xApp, it is necessary to execute these xApps.

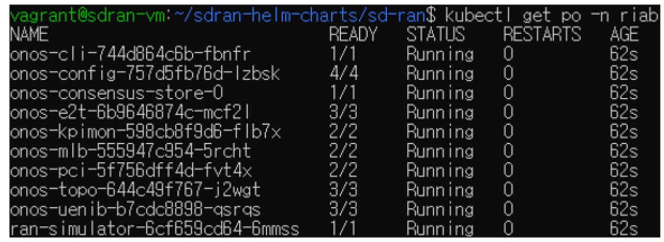
The two databases transfer the information on number of UEs and neighbor cell of each UE to the onos-mlb xApp's *rnib handler*, and *monitoring handler* monitors this information and stores it in the xApp's storage. O_{cn} is a handover parameter for load balancing. *Controller handler* reports on the situation of each E2Node and E2cell from the *e2control handler*, and O_{cn} value is changed depending on their situation. If a E2cell serves more UEs than currently set overload threshold (%), load balancing is performed by increasing the O_{cn} value to take place more frequent handover to the neighbor cell. *Controller handler* transmits a message about the changed value to E2Node of RAN simulator through *e2control handler*.

Parameters are also required for load balancing. They include overload threshold, *delta* O_{cn}, which is the interval of increment and decrement of O_{cn}, and *interval*, which is the update time interval of the O_{cn}. These values will be stored directly in the storage by the *manager*. The onos-cli (command line interface) is an interface that allows users to input commands in onos-mlb, The onos-operator builds cloud native xApp and packages of near-RT RIC on Kubernetes. It also monitors the built elements and performs functions such as scheduling resources or entering the necessary configuration for deploying.

III. Simulation Result

The xApp is built to an image through the Docker file, and this image is deployed and executed in the Kubernetes (k8s) (Fig. 2). The image of onos-mlb xApp is deployed in Kubernetes by helm command, and images of RAN simulator, onos-pci, and onos-kpimon xApp are also deployed together. *Helm* is a command for installing *charts* for running xApp and deploys images to user-specified namespaces. We built elements in namespace named 'riab', and Fig. 3 shows the results. In Fig. 3, we can identify that near-RT RIC packages, xApps, and RAN simulators have been deployed successfully in the riab namespace.

Fig. 2. Deployment and execution of xApp



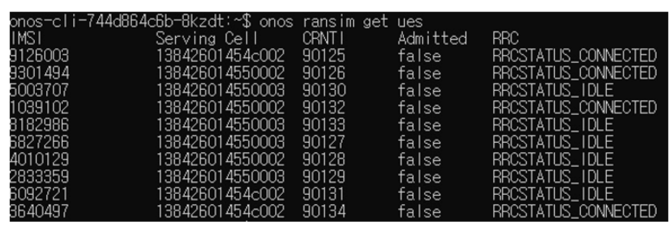
```
vagrant@sdran-vm:~/sdran-helm-charts/sd-ran$ kubectl get po -n riab
NAME                              READY   STATUS    RESTARTS   AGE
onos-cli-744d864c6b-fbnfr         1/1     Running   0          62s
onos-config-757d5fb76d-lzbsk      4/4     Running   0          62s
onos-consensus-store-0            1/1     Running   0          62s
onos-e2t-6b9646874c-mcf2l         3/3     Running   0          62s
onos-kpimon-598cb8f9d6-flb7x      2/2     Running   0          62s
onos-mlb-555947c954-5rcht         2/2     Running   0          62s
onos-pci-5f756dff4d-fvt4x         2/2     Running   0          62s
onos-topo-644c49f767-j2wgt        3/3     Running   0          62s
onos-uenib-b7cdc8898-qsrqs        3/3     Running   0          62s
ran-simulator-6cf659cd64-6mmss    1/1     Running   0          62s
```

Fig. 3. The xApps and packages of near-RT RIC deployed on riab namespace

TABLE I. SIMULATION PARAMETERS

Parameters	Value
Number of UEs	10
Number of E2Nodes	2
Number of E2Cells	6
overload threshold	80 %
Delta O_{cn}	3 dB
interval	10 s

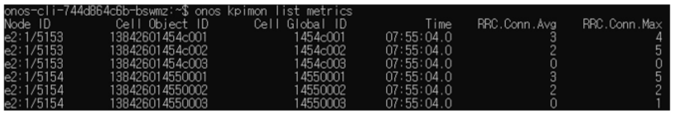
The parameter and its value used in the simulation are summarized in Table 1. We run a simulation in onos-cli bash shell. The RAN simulator has ten UEs, and two E2Nodes with Entity IDs 5153 and 5154, respectively. Each E2Node manages three E2Cells. In Fig. 4, Each UE is identified by IMSI (International Mobile Subscriber Identity) and CRNTI (Cell Radio Network Temporary Identifier).



```
onos-cli-744d864c6b-8kzdt:~$ onos ransim get ues
IMSI       Serving Cell      CRNTI   Admitted   RRC
9126003    13842601454c002   90125   false      RRCSTATUS_CONNECTED
9301494    138426014550002   90126   false      RRCSTATUS_CONNECTED
5003707    138426014550003   90130   false      RRCSTATUS_IDLE
1039102    138426014550002   90132   false      RRCSTATUS_CONNECTED
8182986    138426014550003   90133   false      RRCSTATUS_IDLE
6827266    138426014550003   90127   false      RRCSTATUS_IDLE
4010129    138426014550002   90128   false      RRCSTATUS_IDLE
2833359    138426014550003   90129   false      RRCSTATUS_IDLE
6092721    13842601454c002   90131   false      RRCSTATUS_IDLE
3640497    13842601454c002   90134   false      RRCSTATUS_CONNECTED
```

Fig. 4 IMSI and CRNTI of UEs

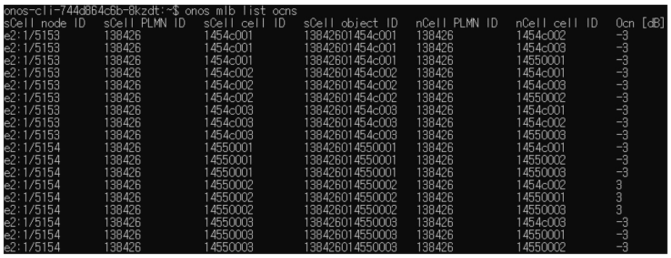
Fig. 5 shows the information provided by onos-kpimon xApp. The RRC.Conn.Avg is the average number of UEs serviced by each E2Cell over time. The RCC.Conn.Max means the maximum number of UEs that can be accommodated by each E2Cell. If the ratio of RRC.Conn.Avg to the RRC.Conn.Max of a E2Cell is higher than the overload threshold, the E2Cell increases the O_{cn} value to handover UEs to other cells more frequent.



```
onos-cli-744d864c6b-bswmz:~$ onos kpimon list metrics
Node ID          Cell Object ID        Cell Global ID          Time   RRC.Conn.Avg    RRC.Conn.Max
e2:1/5153        13842601454c001              1454c001    07:55:04.0             3               4
e2:1/5153        13842601454c002              1454c002    07:55:04.0             2               5
e2:1/5153        13842601454c003              1454c003    07:55:04.0             0               0
e2:1/5154        138426014550001              14550001    07:55:04.0             3               5
e2:1/5154        138426014550002              14550002    07:55:04.0             2               2
e2:1/5154        138426014550003              14550003    07:55:04.0             0               1
```

Fig. 5. Simulation result of onos-kpimon xApp

Fig. 6 shows the change in the O_{cn} value in onos-mlb xApp. A 14550002 E2cell belonging to 5154 E2Node is serving more than 80 percent of allowed UE. So, it increases the O_{cn} value between its neighbor cells by 3 dB to make handover more frequent. Since the other cells are serving fewer UEs, we can confirm that they lower their O_{cn} by 3 dB to reduce the handover.



```
onos-cli-744d864c6b-8kzdt:~$ onos mlb list ocns
sCell node ID   sCell PLMN ID   sCell cell ID   sCell object ID   nCell PLMN ID   nCell cell ID   Ocn [dB]
e2:1/5153       138426          1454c001        13842601454c001   138426          1454c002        -3
e2:1/5153       138426          1454c001        13842601454c001   138426          1454c003        -3
e2:1/5153       138426          1454c001        13842601454c001   138426          14550001        -3
e2:1/5153       138426          1454c002        13842601454c002   138426          1454c001        -3
e2:1/5153       138426          1454c002        13842601454c002   138426          1454c003        -3
e2:1/5153       138426          1454c002        13842601454c002   138426          14550002        -3
e2:1/5153       138426          1454c003        13842601454c003   138426          1454c001        -3
e2:1/5153       138426          1454c003        13842601454c003   138426          1454c002        -3
e2:1/5153       138426          1454c003        13842601454c003   138426          14550003        -3
e2:1/5154       138426          14550001        138426014550001   138426          1454c001        -3
e2:1/5154       138426          14550001        138426014550001   138426          14550002        -3
e2:1/5154       138426          14550001        138426014550001   138426          14550003        -3
e2:1/5154       138426          14550002        138426014550002   138426          1454c002        3
e2:1/5154       138426          14550002        138426014550002   138426          14550001        3
e2:1/5154       138426          14550002        138426014550002   138426          14550003        3
e2:1/5154       138426          14550003        138426014550003   138426          1454c003        -3
e2:1/5154       138426          14550003        138426014550003   138426          14550001        -3
e2:1/5154       138426          14550003        138426014550003   138426          14550002        -3
```

Fig. 6. Simulation result of onos-mlb xApp

IV. Conclusion

In this paper, we introduce the SD-RAN platform and structure of xApp following the O-RAN architecture. We also show the execution method and simulation results of xApps. SD-RAN has not yet developed SMO, so the function of non-RT RIC with big data/machine learning is not yet available. It is expected that more functions can be simulated after SMO is developed in the future. In addition, the development of xApps that perform new functions and contribution to the open source project would be interesting studies, and it can promote the commercialization of O-RAN architecture in the future.

Acknowledgment

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2021-0-02046) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation)

References

[1] O-RAN Alliacne White Paper, "O-RAN: Towards an Open and Smart RAN," https://www.o-ran.org/resources, October 2018.

[2] L. Bonati, S. D'Oro, M. Polese, S. Basagni and T. Melodia, "Intelligence and Learning in O-RAN for Data-Driven NextG Cellular Networks," in IEEE Communications Magazine, vol. 59, no. 10, pp. 21-27, October 2021.

[3] 3GPP TR 38.801, Study on new radio access technology: Radio access architecture and interfaces (Release 14).

[4] Michele Polese et al, "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges", 2022, arXiv:2202.01032. [Online]. Available: https://arxiv.org/abs/2202.01032.

[5] Open Networking Foundation. SD-RAN. https://docs.sd-ran.org. Accessed August 2022.