SDRBench: A Software-Defined Radio Access Network Controller Benchmark

Arled Papa, Raphael Durner, Fabian Edinger, Wolfgang Kellerer
Chair of Communication Networks
Department of Electrical and Computer Engineering
Technical University of Munich, Germany
Email: {arled.papa, r.durner, f.edinger, wolfgang.kellerer}@tum.de

***Abstract*—Software-Defined Networking (SDN) has been identified as a key enabler for 5G networks to enhance the network capabilities by introducing flexibility and programmability. While SDN has been widely exploited in the core network side, it still remains an open research question in the Radio Access Network (RAN). Initial works highlight the benefits of SDN in RAN and investigate the idea of separating the control plane from the data plane of the Base Stations (BS) by means of SDN. The pioneer Software-Defined RAN (SD-RAN) controllers are available, nonetheless there exist no tools which can shed light on their performance and help to understand their limitations. We introduce SDRBench, a novel SD-RAN controller benchmark tool to fill this void. In this work, we evaluate FlexRAN, which is the first open source SD-RAN platform. In our tool, a Python-based instance of a FlexRAN agent is created and it communicates messages with the controller according to the FlexRAN protocol. The benchmark is used to uncover the limits of the SD-RAN controller.**

***Index Terms*—SD-RAN, RAN Slicing, 5G, Programmability, Flexibility.**

I. Introduction

Fifth generation (5G) networks are envisioned to cope with the tremendous growth of traffic demands and heterogeneity of future applications such as automotive, e-health or virtual reality. The main solutions rely on flexible and programmable concepts, which promise cost-efficient and seamlessly adaptable networks. To this end, novel techniques such as Software-Defined Networking (SDN) and Network Function Virtualization (NFV) have emerged.

The introduction of SDN/NFV has lead to numerous benefits in terms of cost reduction, flexibility and enhancements in the network management mainly in data centers and core networks. Analogously, the Radio Access Network (RAN) can potentially benefit from the concepts of SDN/NFV. On the one hand, the separation of the control plane from the data plane of RAN can result in improvements in spectrum efficiency or better interference management. On the other hand, there are challenges in RAN which lie in the tasks of wireless resource management and the ability to perform strict delay operations (e.g., MAC scheduling in real time).

From the research perspective, SDN in RAN advocates new emerging concepts such as Cloud RAN, Mobile Edge Computing (MEC) or Network Slicing. Recently, the aforementioned techniques have attracted increased attention both in academia and industry. Network slicing is envisioned as a key technology for next generation 5G networks for supporting the deployment of multiple heterogeneous networks with distinct Quality-of-Service (QoS) requirements in a flexible and cost-efficient manner [1]. Likewise MEC is another novel concept of 5G networks which further enhances network capabilities by enabling computing functionalities on the edge and therefore decreasing potential delays [2]. Hence, architectures to provide the logic of SDN in RAN have been proposed in the literature [3], [4].

The first open source Software-Defined RAN (SD-RAN) platform which enforces programmable control logic on the RAN side is FlexRAN [3]. Thus, enabling applications such as MEC, which are envisioned to be deployed in a centralized fashion. Additionally, an Application Programming Interface (API) is utilized for the interaction between the applications and the control plane. Leveraging this API, the application of concepts such as network slicing can be possible. This enables the deployment of multiple services over the RAN domain, utilizing available radio resources by means of virtualization in a flexible manner. Even though SD-RAN platforms have triggered vast ongoing research in the fields of RAN slicing and MEC, there exist no in depth evaluation of such platforms in the literature. Inherently, an accurate investigation on their performance and limitations has to be conducted.

To the best of our knowledge, we are the first to investigate the limitations of existing SD-RAN approaches and evaluate their performance. We present SDRBench, a novel benchmark tool for our analysis. Based on our tool, in this work we evaluate FlexRAN, one of most representative SD-RAN platforms in the literature. Nonetheless, a complementary analysis can be performed on other existing concepts. The core contribution of our work is on the one hand to highlight the importance of such tools for analyzing the behavior of SD-RAN controllers and on the other hand identify the main capabilities of SD-RAN controllers using our tool.

The rest of the paper is organized as follows. Section II describes the related work on SD-RAN platforms and controller benchmarks. We introduce the architecture of FlexRAN in Section III and define the main idea and properties of our benchmark. The main results of our analysis are described in Section IV. Moreover, section V introduces an extension of

our work towards a fully fledged benchmark. Finally Section VI concludes this paper and emphasizes the main findings.

II. Background

Similar to the fixed networks, SDN is paving the way towards a programmable and flexible Radio Access Network (RAN). From the architectural perspective, several approaches have been proposed in the literature regarding wireless resource management [5], [6], [7]. Driven by means of SDN, these approaches benefit from virtualization concepts and introduce a split between the data and control plane of RAN, by delegating the control functionalities in a single entity known as Software-Defined RAN (SD-RAN) controller. Nonetheless, the aforementioned works only present a conceptual idea of the SDN-enabled RAN architectures and have not yet been implemented.

In order to fill this void initial implementations of SD-RAN platforms have been introduced in the state-of-the-art [3], [4]. Furthermore, to prove their efficiency the aforementioned SD-RAN platforms have been integrated into existing open source LTE platforms such as OpenAirInterface [8] and srsLTE [9], thus introducing an appealing avenue of research in the area. FlexRAN [3] and EmPower [4] are two of the first open source implementations of SD-RAN platforms. FlexRAN separates the data plane from the control plane of RAN according to the SDN logic and is implemented on top of the existing open source LTE platform OpenAirInterface [8]. Therefore, it allows for implementation and demonstration of possible applications that benefit from this concept. On the other hand, EmPower was initially designed for WiFi and recently is applied on srsLTE [9] to support the concept of SD-RAN.

There exist many works in the literature regarding topics such as network slicing especially in the algorithmic part. However the design and implementation of SD-RAN platforms has provided many researchers with a tool to evaluate the performance of their approaches in a real testbed. For instance, in [10] the notion of multiplexing gain is introduced for RAN slicing. The problem is formulated as a knapsack problem, where the focus is on the QoS performance while maximizing the unused part of the spectrum. Additionally, the algorithm is implemented in the current version of FlexRAN. Moreover, in [11] the problem of RAN slicing is tackled by applying the Lyapunov optimization technique [12]. In this work, a dynamic resource allocation is proposed. The MAC scheduling is envisioned in a hierarchical fashion and the architectural concept is built upon FlexRAN.

Given the high interest in SD-RAN controllers and platforms, a qualitative study on the behavior and their performance has to be initiated. Currently, there exist no benchmarks for the evaluation of SD-RAN controllers performance. Nonetheless, the benchmarking of SDN controllers is not an unfamiliar notion on the core network, where such controllers have been well investigated. For instance, Cbench [13] and OFCBenchmark [14] are two of most representative benchmark tools for SDN controllers. In our work, we incorporate the idea implemented from the aforementioned SDN controller benchmarks and propose an equivalent benchmarking tool for FlexRAN SD-RAN controller.

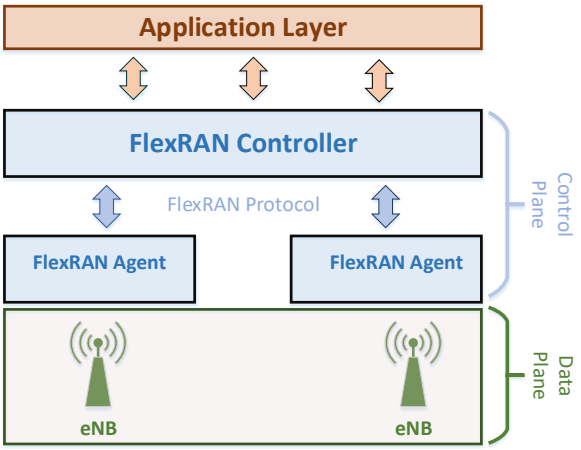

Figure 1. SD-RAN platform architecture.

III. Benchmark Design

In this section we introduce our SD-RAN controller benchmark tool SDRBench. Our benchmark is initially designed to evaluate the FlexRAN controller. We first present the architecture of FlexRAN as it is tightly coupled with the architecture of our benchmark. Moreover, we introduce the FlexRAN protocol, which enables the communication in the control plane of the SD-RAN platform. Finally, we describe the design principles of our approach and shed light on the important functionalities needed to achieve our goal.

A. FlexRAN Architecture

Similarly to the SDN principles, the control logic of the RAN is centralized in a entity known as SD-RAN controller. This introduces a functional split, where the data plane of RAN is separated from the control plane. FlexRAN [3] is one of the initial open source SD-RAN platforms which facilitates this concept. The architecture of FlexRAN is presented in Fig. 1. As denoted by the architecture, the control plane is envisioned in a hierarchical fashion. For each eNB, a FlexRAN agent is initiated. This agent is responsible for the local control over the eNB. Further, the control of multiple eNBs can be aggregated to a FlexRAN controller. The connection between these entities is realized by the FlexRAN protocol. In analogy with SDN southbound protocols (i.e., OpenFlow [15]), the FlexRAN agent API is anticipated for the communication between the control plane and data plane of the FlexRAN platform. Additionally, a northbound API enables the communication between the FlexRAN controller and the application layer. It is this interface that allows for modification of the underlying physical infrastructure depending on the current network state and application requirements, thus providing programmability and flexibility for RAN.

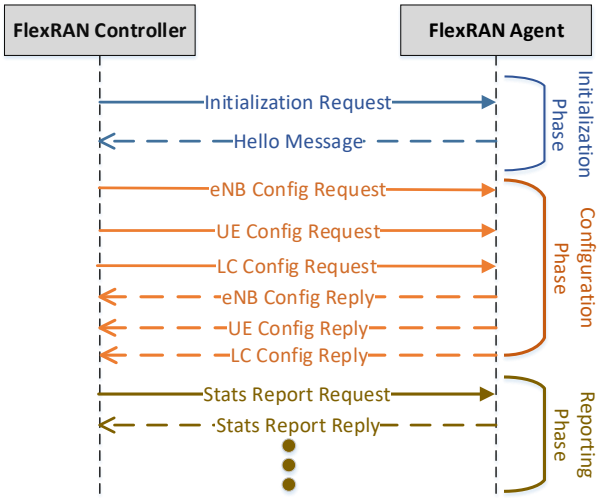

Figure 2. FlexRAN protocol consisting of three phases, namely initialization phase, configuration phase and reporting phase. The reporting phase is a repetition process consisting of stats request and replies.

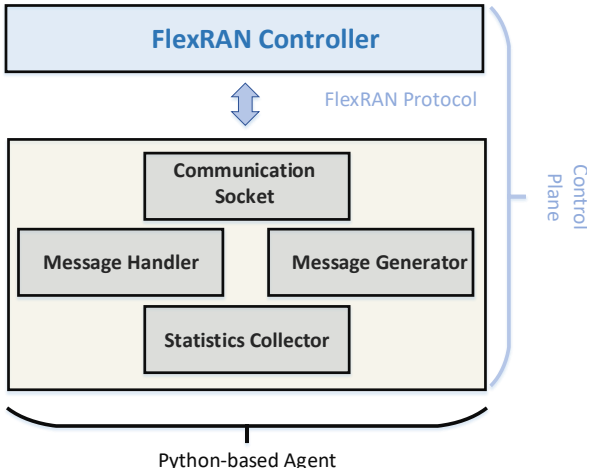

Figure 3. SD-RAN benchmark architecture. The FlexRAN agent is replaced by a Python-based implementation.

B. FlexRAN Control Plane

The control plane of FlexRAN consists of the FlexRAN controller and the FlexRAN agent. On the one hand the controller is unique, whereas on the other hand multiple FlexRAN agents can be created depending on the number of eNBs present on the underlying physical network. The communication between the FlexRAN controller and FlexRAN agent is bi-directional. The FlexRAN agent reports statistics and events regarding the respective eNB to the FlexRAN controller, whereas the controller can define rules and delegate decisions to the corresponding agent. Considering delay critical tasks such as MAC scheduling, the communication between the FlexRAN controller and FlexRAN agent should utilize low-latency or high-bandwidth paths. However, depending on the network conditions the FlexRAN platform allows for local control over the eNB from the FlexRAN agent in order to account for delay reductions, thus providing flexibility.

C. FlexRAN Protocol

The FlexRAN protocol is an important feature of the FlexRAN platform since it realizes the communication between the FlexRAN controller and FlexRAN agent in the control plane. In order to accurately replicate the FlexRAN implementation, we put high emphasis on the understanding of the FlexRAN protocol. In the current version of the FlexRAN protocol, TCP is used to enable the communication between the FlexRAN controller and the FlexRAN agent. The messages of the aforementioned protocol are encoded/decoded by utilizing Google Protocol Buffers [16]. The detailed protocol is illustrated in Fig. 2. The FlexRAN protocol consists of three parts, namely the initialization phase, configuration phase and reporting phase respectively. Initially, the FlexRAN controller initiates a FlexRAN agent once the eNB is activated. Further, the handshake process continues where the corresponding agent replies with a *Hello* message to conclude the initialization phase. Once the initialization phase has terminated, the FlexRAN controller requests statistics from the agent regarding the eNB, UE and logic channel (LC) configuration. Accordingly the agent replies with configuration information and thus terminating the configuration phase. Finally, the reporting phase is a repeated operation that consists of statistics requests from the FlexRAN controller and statistics replies from the agent. Upon these statistics the FlexRAN controller might trigger changes on the operation of the FlexRAN agents.

D. Benchmark Implementation

The main design goal of our benchmark is to evaluate the control plane of FlexRAN. In our approach we do not modify the FlexRAN controller, however we concentrate on the FlexRAN agent, which represents each eNB. Inherently, we focus on the creation of FlexRAN agent replicas and establishment of the connection with FlexRAN controller in order to be able to explore its limitations. As illustrated, in our model the control plane stills remains the same, nonetheless in contrast to the previous approach the traditional FlexRAN agent established during the initialization phase of the eNB, is replaced by a Python-based program. The architecture of our implemented FlexRAN agent replica is presented in Fig. 3. Within the Python program, several agent functionalities have been implemented. Analogous to the current implementation of the FlexRAN, a communication entity is needed for the interaction between the FlexRAN controller and the FlexRAN agent. In our case this is referred to as **communication block**, realized in a socket-based manner. Furthermore, a **message handler** is envisioned to encode/decode the messages sent/received by the program towards the controller by using the FlexRAN protocol. To support the report functionality of the FlexRAN agent, in our approach we incorporate an entity, namely **message generator**, which is responsible for generating reports with respect to requests coming from the controller (i.e., *hello* messages, *stats* reports). Finally, the

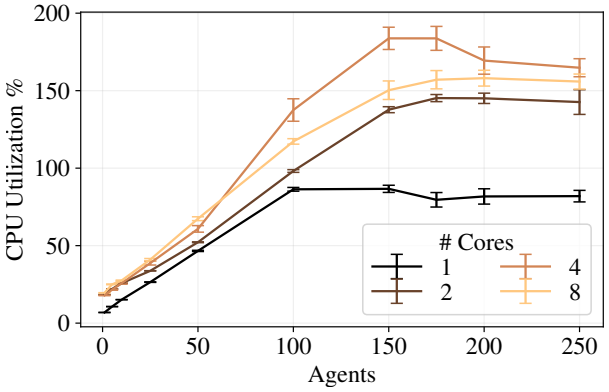

Figure 4. CPU Utilization for different numbers of CPU cores. The controller can utilize up to 2 cores, whereas 1 core supports 100 agents and more cores 150 agents.

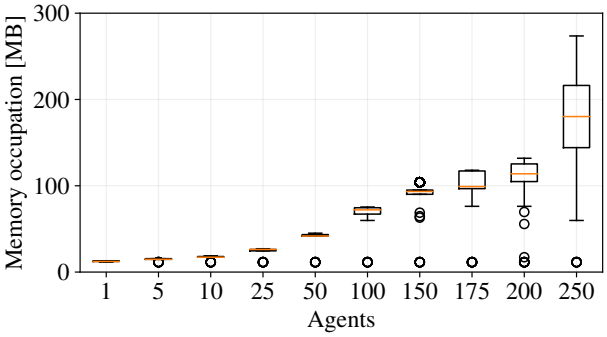

Figure 5. Memory occupation for 2 cores and different number of agents in MB. Memory leakage increases the utilization for 250 agents significantly. Results do not depend much on the number of used CPU cores.

statistics collector is responsible for gathering all the statistics from the measurements conducted with the controller (i.e., *delay* for stats reply).

In our Python-based implementation multiple instances of FlexRAN agents can be initiated and operated simultaneously. This number can be altered and therefore allows for several tests with the controller. Once the FlexRAN agent has been created, it starts the communication with the controller by sending a *Hello* message according to the FlexRAN protocol. The **message handler** is then used to receive the reply for the respective *Hello* message and therefore can record its round trip time and furthermore update the statistics stored in the **statistics collector**.

IV. Results

In this section we elaborate the initial findings from our benchmark implementation regarding the behavior of the current version of the FlexRAN controller. We investigated the performance of the FlexRAN controller in terms of CPU utilization, memory usage and delay of the initial hello message.

For the measurement we used a Desktop PC equipped with an AMD Ryzen 7 2700X and 16Gb RAM. This CPU has 8 physical cores and a frequency of 3.7 Ghz. The controller is running inside a virtual machine that is virtualized with Virtualbox. The benchmark is running on the host system and is connected with the controller through a virtual switch. We used virtualization to ease deployment of the controller on our test system. Even though the virtualization may cause some overheads, it has been shown that these overheads are not significant [17].

A. CPU utilization

When a FlexRAN agent is initiated it connects to the designated controller and sets up the connection using the FlexRAN protocol as described in Section III-C. After the initiation the agent sends stats reply in a repeated manner upon a request from the controller. Even though the controller does not reply on these messages, it still has to process the message and update its internal state. As a result the CPU utilization is increased if more agents are added.

FlexRAN is implemented using multiple threads and therefore scales with the number of CPU cores. Fig. 4 demonstrates the measurement results of the CPU utilization in percent. Each experiment is conducted 4 times with a duration of 1000 seconds each and one measurement point per second. We ploted the mean and the confidence intervals for the respective scenarios. The utilization of all cores is summed up to show the total utilization of a process. This explains the increase beyond 100% if more than one core is used. The results show that the utilization depends linearly on the number of agents until a feasible maximum is reached. Afterwards the utilization flattens out and timeouts can be observed. It can be seen that one core flattens out after 100 agents, while for more cores this point is reached after 150 agents. On the other hand we can observe that the FlexRAN controller does not scale well with the number of cores. One CPU core supports 100 agents, whereas with increasing number of cores a maximum of 150 agents is supported. Further, in total no more than 2 cores can be effectively utilized irrespective of the total number of assigned cores to the controller.

B. Memory usage

During the same experiment we also measured the memory occupation of the controller. This metric does not depend much on the number of CPU cores, consequently we only show the results with 2 cores in Fig. 5. In general the FlexRAN controller is not very demanding, consuming only 100 MByte with 150 agents connected. On the other hand in an overload situation (>150 agents) we can notice an increase of the load over time which indicates memory leakages.

C. Hello-message reply delay

The final metric we consider represents the delay of the controller. We initially connect $n-1$ agents and further add an additional n^{th} agent in order to measure the delay of the first handshake for the latest added agent. Fig. 6 presents the results with n agents. The delay for a low number of agents (1-101) is mostly below 3 ms with a mean of 1.5 ms. As expected

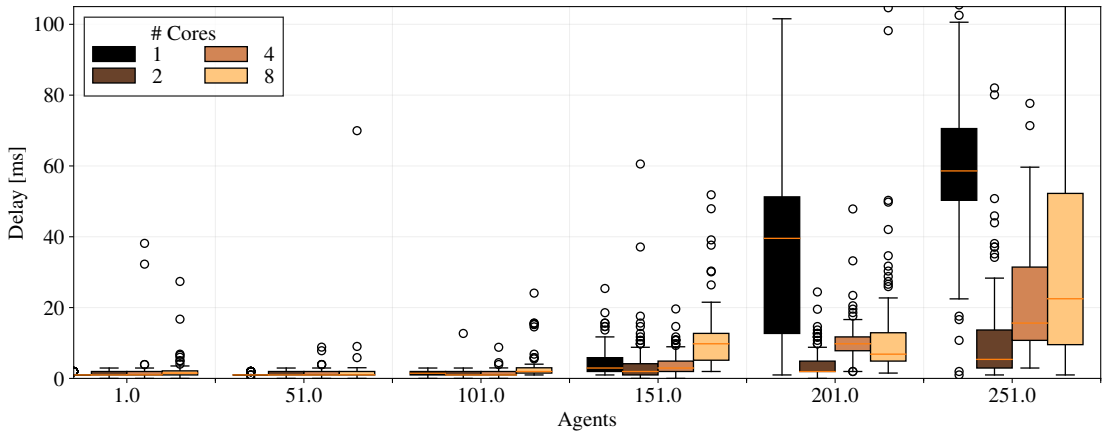

Figure 6. Delay of the Hello message for different number of CPU cores and active agents. One core can support up to 100 agents without an increase in the delays, more cores up to 150. 8 cores is worse than 2 or 4 cores due to context switches.

the delay increases rapidly if the controller is over-utilized, causing a mean delay of 60 ms using one core and 251 agents.

Additionally the results show that too many cores can even harm the delay. For example for 101 agents the delay using 8 cores has a mean of 2 ms, while lower core numbers have a mean of 1 ms. Further also 8 cores shows more outliers with comparably high delays (e.g. results for 151 agents). This can be explained by an increased number of context switches of the CPUs.

V. SD-RAN Control Plane Model

This work is a first step towards a fully fledged benchmark for SD-RAN controllers. The SD-RAN controller is the heart of each SD-RAN platform. It facilitates the logic of SDN by being in charge of the control plane of the RAN and therefore renders flexibility and programmability. The controller serves as a connection bridge between the application layer and the data plane (i.e., physical layer). Thus, for a lot of applications such as Network Slicing, MEC, Cloud RAN a fully capable controller is required. Yet, until now it is not clear if the available controllers can serve all requests coming from the data plane and the application layer in a timely manner. The load on the controller depends on a number of influence factors as depicted in Fig. 7:

1. The number of SD-RAN agents (e.g., FlexRAN agents).
2. The load of the traffic in the data plane.
3. The load of the applications and services on the application layer.
4. The type of applications running on the application layer.

In this paper our benchmark emulates the behavior of a FlexRAN agent in the static case. This includes regular stats report messages sent to the controller. As an initial step in our work we vary the number of FlexRAN agents and identify the behavior of the controller in such a scenario. Nonetheless, we do not consider any load imposed by the users in the data plane. However, in a more realistic scenario the traffic load (i.e., transmission requests) or traffic patterns of users (i.e., user arrivals/departures) in the data plane can influence the

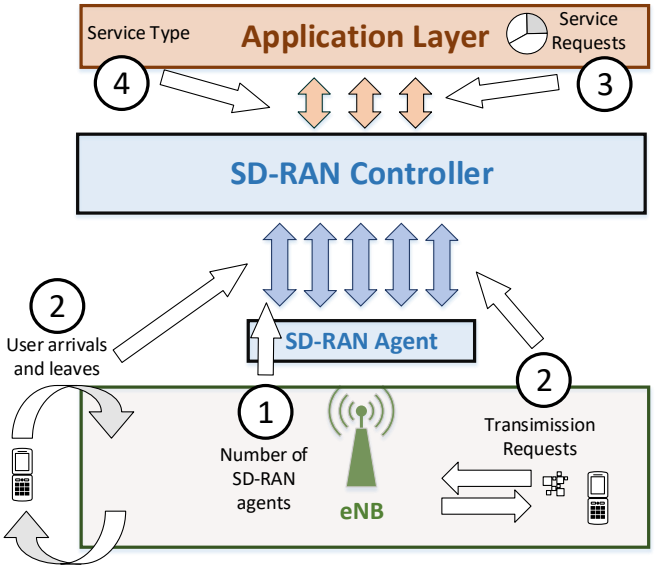

Figure 7. SD-RAN Control Plane Model. The load on the controller depends on 1. Static eNB messages, 2. User arrivals and leaves, 3.Slice Requests, 4.Transmission Requests.

performance of the SD-RAN controllers. For instance, a high traffic load can increase the frequency of the communication between the agent and the controller. Similarly, a handover might also trigger reports in order to inform the SD-RAN controller for these events.

From the application layer perspective, the SD-RAN controllers should be able to sustain a normal behavior in order to cope with the application requirements. Intuitively the load of applications or services that operate on top of such controllers directly affects their performance. Moreover, the nature of applications or services might also play a huge role on the traffic imposed on the controller. For instance, if we think of network slicing, constant slicing requests might be triggered by the application layer every time a new slice is initiated or deleted. Furthermore, to account for constant monitoring

of these slices, a considerable amount of messages between the SD-RAN controller and the application itself might be required.

Considering the above mentioned factors which can influence the behavior of SD-RAN controllers, a more realistic and complete model should be created. To this end, an interesting avenue of research is modeling the behavior of such controllers depending on all the parameters shown in Fig. 7. Furthermore, we believe that obtaining a mathematical framework to model the behavior of SD-RAN controllers might result in a compelling tool that can be used by many researchers interested in Software-Defined RAN.

VI. Conclusion

Given the high interest in programmable and software-enabled 5G Radio Access Networks (RANs) platforms, the evaluation of the performance and limits of such approaches is crucial both for academia and industry. In order to answer related questions, in this paper we introduced SDRBench, a novel benchmark tool for the analysis and evaluation of Software-Defined RAN (SD-RAN) controllers. We provided insights regarding FlexRAN, one of the most representative SD-RAN platforms in the state-of-the-art and shed light on some limitations with respect to the number of FlexRAN agents that can be supported. Moreover, we introduced a methodology towards a fully fledged benchmark. This implies an extension for our tool which takes into account both northbound and southbound API requests to obtain a full picture on the behavior of SD-RAN controllers in a realistic scenario. Therefore, we will further continue with the development of our SD-RAN controller benchmark to generate representative results for alternative SD-RAN controllers and compare their performance. Eventually, upon a consolidation of our benchmark we plan for a public release, which can serve as a benchmark for future SD-RAN platforms in the field.

Acknowledgements

This work has been partially funded by the German Research Foundation (DFG) under the grant number KE 1863/8-1 and by a research grant from Zodiac Inflight Innovations (TriaGnoSys GmbH), Germany.

References

[1] N. Alliance, "Description of network slicing concept," *NGMN 5G P*, vol. 1, 2016.

[2] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computingA key technology towards 5G," *ETSI white paper*, vol. 11, no. 11, 2015.

[3] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A flexible and programmable platform for software-defined radio access networks," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016.

[4] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kuklinski, T. Rasheed *et al.*, "Programming abstractions for software-defined wireless networks." *IEEE Trans. Network and Service Management*, vol. 12, no. 2, 2015.

[5] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software defined radio access network," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013.

[6] M. Y. Arslan, K. Sundaresan, and S. Rangarajan, "Software-defined networking in cellular radio access networks: potential and challenges," *IEEE Communications Magazine*, vol. 53, no. 1, 2015.

[7] M. Yang, Y. Li, D. Jin, L. Su, S. Ma, and L. Zeng, "OpenRAN: a software-defined ran architecture via virtualization," in *ACM SIGCOMM computer communication review*, vol. 43, no. 4, 2013.

[8] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A flexible platform for 5G research," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, 2014.

[9] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: an open-source platform for LTE evolution and experimentation," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, 2016.

[10] C.-Y. Chang, N. Nikaein, and T. Spyropoulos, "Radio access network resource slicing for flexible service execution," in *IEEE Conference on Computer Communications Workshops (INFOCOM)*, 2018.

[11] A. Papa, M. Klügel, L. Goratti, T. Rasheed, and W. Kellerer, "Optimizing Dynamic RAN Slicing in Programmable 5G Networks," in *IEEE International Conference on Communications (ICC)*, 2019.

[12] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, 2010.

[13] R. Sherwood and K.-K. Yap. CBench Controller Benchmarker. [Online]. Available: http://www.openflowswitch.org/wk/index.php/Oflops,2011.

[14] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A flexible openflow-controller benchmark," in *Software Defined Networking (EWSDN)*, 2012.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, 2008.

[16] https://developers.google.com/protocol-buffers/.

[17] A. J. Younge, R. Henschel, J. T. Brown, G. Von Laszewski, J. Qiu, and G. C. Fox, "Analysis of virtualization technologies for high performance computing environments," in *Cloud Computing (CLOUD)*, 2011.