FlexSlice: Flexible and real-time programmable RAN slicing framework

Chieh-Chun Chen*, Chia-Yu Chang†, Navid Nikaein*
*EURECOM, Sophia-Antipolis, France, Email: {chieh-chun.chen, navid.nikaein}@eurecom.fr
†Nokia Bell Labs, Antwerp, Belgium, email: chia-yu.chang@nokia-bell-labs.com

***Abstract*—Radio Access Network (RAN) slicing aims to roll out various services in the same network deployment while ensuring performance guarantee and resource isolation, it is identified as one work item of 3GPP Release 18 and as a specific O-RAN use case. However, the current O-RAN architecture lacks control flexibility for real-time programmability of less than 10ms. In this work, we propose the FlexSlice framework, which entails the realization of flexible control logic topologies (i.e., centralized, decentralized, and distributed) by evolving the O-RAN architecture to achieve lower control loop latency. In addition, the radio resource scheduler at the Medium Access Control (MAC) layer is redesigned for recursive operations to facilitate virtualization for multi-level resource allocation. Finally, a concrete prototype is developed to demonstrate its efficiency, real-time programmability and control flexibility.**

I. Introduction

Radio Access Network (RAN) slicing is a key enabler for 5G and beyond, unlocking the multi-service offering within the same network deployment while ensuring performance guarantees and resource isolation. To meet a variety of service requirements, many control options are standardized, leading to an increase in the control complexity of RAN User Plans (UPs).

In the envisioned RAN openness sketched by the O-RAN Alliance, the complexity of control grows further in that an App (e.g., xApp in the O-RAN architecture) running on top of the RAN controller (e.g., near Real-Time RAN Intelligent Controller [nearRT-RIC]) needs to control multiple RAN-Nodes, e.g., Centralized Unit (CU) and Distributed Unit (DU) that are called E2-Nodes. Additionally, when controlling the underlying RAN-Nodes, the RAN controller coordinates several Apps and mitigates any conflict. Another challenge is the lack of flexibility for real-time control and monitoring of sub-10ms control loops to accomplish RAN slicing in 5G-Advanced services.

To present a generalized approach and delve into the details of our proposed FlexSlice framework, three components are illustrated on the left side of Fig. 1: (1) business logic, (2) control logic, and (3) RAN UPs. The business logic is defined by service providers in terms of the required Key Performance Indicators (KPIs), e.g., expected throughput and latency, in the Service Level Agreement (SLA). By invoking the exposed service Application Programming Interfaces (APIs), business logic is converted into control logic, e.g., scheduling algorithms and parameters. Finally, the control logic is applied to RAN UPs, e.g., radio resource scheduler at the Medium Access Control (MAC) layer, by using the exposed RAN APIs.

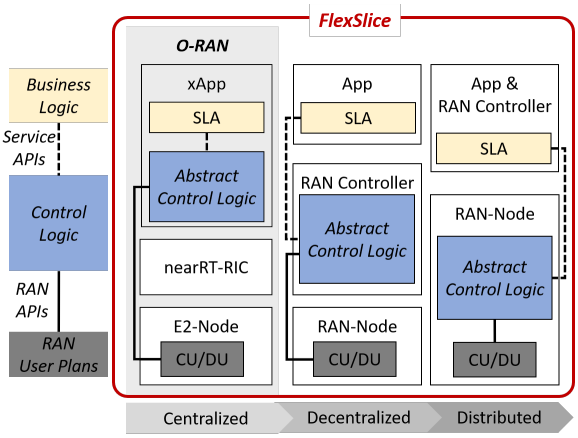


Fig. 1: Hierarchical control for RAN user plans (left) and three potential topologies for control logic (right).

From the viewpoint of service providers, the control logic can be centralized in the App, decentralized in the RAN controller, or distributed in RAN-Nodes, as shown on the right side of Fig. 1. Their trade-off is mainly in terms of control loop latency and RAN-Node complexity. Take the centralized one (i.e., current O-RAN approach) as an example; the control logic is mostly located in the App; thus, it has the lowest RAN complexity but the highest control loop latency, which limits its real-time control capability. In contrast, decentralized and distributed topologies can not only reduce such control loop latency, but also enable App interoperability across various RAN controller platforms.

In this work, we propose FlexSlice, a flexible RAN slicing framework supporting all three topologies that offers both control flexibility and real-time programmability. We also redesign the radio resource scheduler to show how it controls RAN UPs. In short, this paper makes the following contributions: (1) real-time control down to microseconds, evolving the O-RAN architecture, (2) a flexible RAN slicing framework with loosely coupled control logic locations, and (3) a recursive scheduler for a multi-level resource allocation.

TABLE I: State-of-the-art comparison for RAN slicing framework.

Name	Programmable Granularity	Control Logic Location(s)	Control Logic Topologies	Minimum Control Latency	Average RAN Complexity
O-RAN [1]	Service	App	Centralized	High	Low
NexRAN [2]	Service	App	Centralized	High	Low
HexRAN [3]	RAN function	App	Centralized	High	Low
FlexApp [4]	RAN function	RAN Controller	Decentralized	Medium	Low
RAN Engine [5]	RAN function	RAN-Node	Distributed	Low	High
dApp [6]	RAN function	RAN-Node	Distributed	Low	High
JANUS [7]	RAN function	RAN-Node	Distributed	Low	High
FlexSlice	RAN function	App, RAN Controller, and RAN-Node	Centralized, Decentralized, and Distributed	Low	Medium

II. Related Works

In the following, we first compare the FlexSlice framework with other works, and then outline the radio resource scheduler to control RAN UPs.

A. RAN slicing framework

A comparison of the FlexSlice framework with current O-RAN approach and other works can be found in TABLE I. As mentioned earlier, it supports all three control logic topologies shown in Fig. 1, so a balance can be struck between control latency and RAN complexity. However, other works rely on a specific control logic topology and thus can only be suitable for certain deployments. For example, centralized control logic reduces RAN complexity, but each App must translate business logic into control logic, which adds control latency in heterogeneous RAN deployments. In contrast, distributed control logic can reduce control latency by incorporating control logic and RAN UPs at RAN-Nodes, but at the cost of a more complicated RAN implementation. Then, the decentralized one relies on the abstraction layer at the RAN controller to handle business logic to be applied on the RAN-Nodes. Note that the FlexSlice framework supports programmability down to the RAN function for finer control granularity compared to prior state-of-the-arts [1], [2].

B. Radio Resource Scheduler

Radio resource scheduling aims to use limited spectrum for distinct criteria, e.g., Proportional Fairness (PF). Several works explore further performance metrics, like the ones introduced in NVS [8] and Earliest Deadline First (EDF) [9] algorithms, by taking particular service requirements into account. Another aspect is how radio resources are allocated, e.g., the work in [10] provides a two-level radio resource scheduler (slice- and user-level) by abstracting radio resources from Physical Resource Blocks (PRBs) to virtual RBs (vRBs). In this regard, our objective is to increase the modularity and extensibility of the radio resource scheduler, so that the control logic in the FlexSlice framework can flexibly construct a multi-level scheduling function. To achieve it, we redesign the MAC scheduler recursively in Section IV.

III. FlexSlice Overview

Our proposed FlexSlice framework evolves the current O-RAN architecture by flexibly deploying control logic in centralized, decentralized, or distributed topologies (see Fig. 1) to achieve a shorter control loop latency. Also, the radio resource scheduling function is realized recursively to achieve adaptability and extensibility. This concept of recursion can be applied to other RAN functions, enabling RAN nodes to support multi-service and multi-tenancy.

In Fig. 2, a high-level architecture of FlexSlice is shown, containing business logic (yellow), control logic (blue), and RAN UPs (gray). The service provider defines the business logic, which specifies the required KPIs for a particular service, such as throughput and latency, as part of the SLA[1]. Control logic is then responsible for meeting these SLAs by instantiating appropriate network slices in RAN UPs, and it can be located in the App, the RAN controller, or the RAN-Node, depending on the topology used (cf. Fig. 1):

- *Centralized*: Apps handle business logic and control logic. As shown in Fig. 2, App-I abstracts the expected throughput of Service-1 into control logic, e.g., PRB number, in a comprehensible manner to the radio resource scheduler at RAN-Node A.
- *Decentralized*: Apps define business logic and use service APIs exposed by the RAN controller. In Fig. 2, App-II and App-III request the expected throughput for Services 2 and 3, respectively, and Service 2 asks for extra expected latency for its two sub-services (2a and 2b). The RAN controller abstracts all these requests into the control logic and uses RAN APIs toward RAN-Node B.
- *Distributed*: RAN-Nodes have RAN UPs and control logic, allowing shorter control loops in real-time. Consider App-III and App-IV in Fig. 2: RAN-Node C embeds the abstracted control logic from the expected latency for Services 4 and 5 directly into the radio resource scheduler.

[1]Business logic is used to encode real-world business rules as a sequence of commands or actions, and can be defined in a nested way.

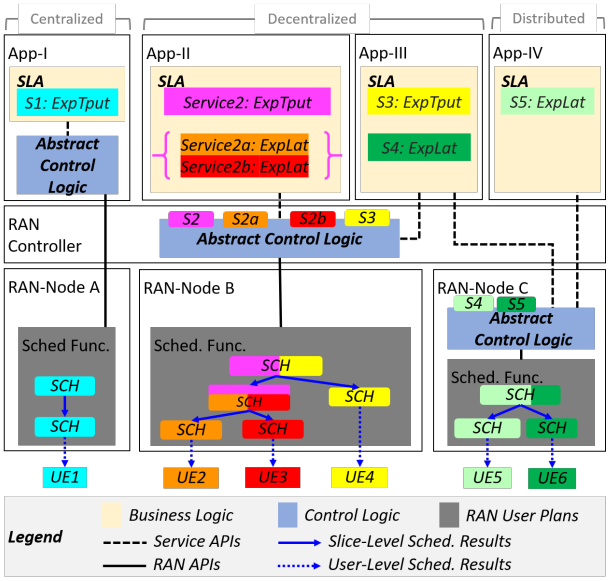


Fig. 2: High-level architecture of FlexSlice framework with three abstract control logic examples.

Moreover, the abstract control logic shown in Fig. 2 unifies the underlying RAN UP functions and Service Models (SMs) across various dimensions, including multi-vendor and multi-Radio Access Technology (RAT) RAN-Nodes. Thus, it allows tailoring the control logic to the service requirements in terms of the requested KPIs. Such an "abstraction layer" is essential to FlexSlice and will be detailed in Section IV-A.

Entering RAN UPs in Fig. 2, three scheduling functions are shown in RAN-Nodes A, B, and C for Services 1, Services 2 and 3, as well as Service 4 and 5, respectively. These functions are realized by employing the recursive approach (denoted as SCH in Fig. 2 and will be detailed in Section IV) which allows for multi-level scheduling. At the slice level, radio resources are partitioned based on (sub-)slice information (e.g., SLA). These per-slice resources are then allocated to the associated UEs based on UE information, e.g., Quality of Service (QoS).

IV. Control Logic Abstraction and Redesigned Radio Resource Scheduler

In this section, we elaborate on two key enablers of FlexSlice: (a) control logic abstraction, and (b) redesigned RAN function to support recursive operations.

A. Control Logic Abstraction

The aim of control logic abstraction is to allow network operators to define the characteristics of each slice in a flexible manner based on the KPIs of each service without disclosing specific deployment details (e.g., multi-vendor and multi-RAT at various sites). As depicted in Fig. 3, the process of controlling RAN slices

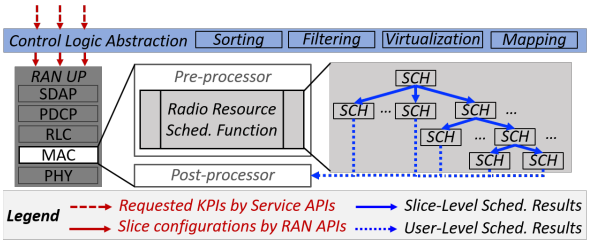


Fig. 3: Control logic abstraction and Redesigned radio resource scheduler for recursive operation.

is done by the "mix-and-match" control logic abstraction between the requested KPIs for each service and the respective slice configuration (e.g., radio resource share), which includes setting up slices, and deciding the scheduling algorithm and parameters. Specifically, four tasks are included as shown in Fig 3:

1) *Sorting*: Arrange services based on their requested KPIs or non-technical criteria,
2) *Virtualization*: Virtualize radio resources of RAN-Nodes with tailored approach (e.g. PRB to vRB),
3) *Filtering*: Adjust the requested KPIs in accordance with the (virtualized) resources of RAN-Nodes,
4) *Mapping*: Create slice configurations by mapping the requested KPIs to use respective RAN functions (e.g., scheduling algorithm and parameters).

B. Redesigned Radio Resource Scheduler

RAN function redesign is key to FlexSlice, and we focus on the radio resource scheduler. First, it is divided into two parts in Fig. 3: (a) pre-processor and (b) post-processor. The former incorporates the radio resource scheduling function and can be controlled as a RAN function, while the latter allocates resources to physical channels based on the results from the pre-processor. Then, the radio resource scheduling function is redesigned recursively (denoted as SCH in Fig. 3) to dynamically apply different algorithms and parameters based on performance requirements and scheduling constraints, effectively realizing both slice- and user-level scheduling. By doing so, the original scheduling problem can be decomposed, further increasing flexibility and extensibility.

Practically, this recursive approach is provided in details in Algorithm 1, and we first define two sets: $\mathcal{C} := \{c_1, \cdots, c_{|\mathcal{C}|}\}$ contains all slices to be scheduled, and $\mathcal{U}_i := \{u_1, \cdots, u_{|\mathcal{U}_i|}\}, \forall c_i \in \mathcal{C}$ includes all associated UEs with the i-th slice. Then, two global variables are formed in Algorithm 1: $\mathbf{C}$ is a vector of size $|\mathcal{C}|$ comprising all slice configurations, and $\mathbf{S}_i$ is a vector of size $|\mathcal{U}_i|+1$ including the number of scheduled resources for the i-th slice (i.e., $\mathbf{S}_i[0]$ as the first element of $\mathbf{S}_i$) and for its associated UEs (i.e., $\mathbf{S}_i[j]$, $\forall j \in [1, |\mathcal{U}_i|]$). Finally, $\mathbf{S}_i$, $\forall c_i \in \mathcal{C}$ will be passed to the post-processor as the scheduling results.

Then, three inputs are defined: l denotes the scheduling level, r is the number of available RBs, and set $\mathcal{N}$ contains the unscheduled slices. To start the recursive operation, the following arguments are used: $l \leftarrow slice$, $\mathcal{N} \leftarrow \mathcal{C}$, and $r \leftarrow R$, where R is the maximum number of RBs can be scheduled. At the slice-level from line 4 in Algorithm 1, we first select an unscheduled slice using the $SelectSlice\,(\cdot)$ function, allocate RBs to this slice using the $AllocRB\,(\cdot)$ function, recursively call SCH to assign RBs to its associated UEs (see line 9), and move to the next unscheduled slice (see line 12). At the user-level from line 15, every associated UE is looped through to schedule all slice-level resources, i.e., $\mathbf{S}_i\,[0]$.

Algorithm 1 Recursive radio resource scheduling

Global variable
$\mathbf{C}$ is a vector of size $\mathcal{C}$ with slice configuration (scheduling algorithm and associated user information).
$\mathbf{S}_i$ is a vector containing the number of scheduled resources to i-th slice slice and its associated UEs.

Input
l is the scheduling level, equal to $slice$ or $user$.
r is the number of available resource blocks.
$\mathcal{N}$ is the set of unscheduled slices.

```
1: procedure SCH(l, r, N)
2:     if r > 0 then
3:         if l = slice then
4:             i ← SelectSlice(C, N)
5:             S_i[0] ← min(r, AllocRB(r, C[i], 0))
6:             r ← r − S_i[0]
7:             N ← N \ {c_i}
8:             if length(S_i) > 1 then
9:                 SCH(user, S_i[0], N)
10:            end if
11:            if |N| > 0 then
12:                SCH(slice, r, N)
13:            end if
14:        else
15:            j ← 1
16:            while r > 0 and j < length(S_i) do
17:                S_i[j] ← min(r, AllocRB(r, C[i], j))
18:                r ← r − S_i[j]
19:                j ← j + 1
20:            end while
21:        end if
22:    end if
23: end procedure
```

V. Performance Evaluation

In this section, we prototype the FlexSlice framework on top of a platform comprising components from OpenAirInerface (core network, gNB as E2-Node, UEs) and FlexRIC (nearRT-RIC, xAPP). Specifically, the control logic abstraction is developed in xApp, nearRT-RIC, and E2-Node, and messages in between are encapsulated into SLA SM and Slice SM [11]. Our evaluation includes two aspects: (1) network and user performance when RAN is dynamically sliced, and (2) quantitative comparison of three control logic topologies.

A. Network and user performance

In this part, we aim to show the capability of FlexSlice for dynamic RAN slicing when applying different algorithms and changing parameters on-the-fly. Specifically, a single E2-Node is used to serve UE1 and UE2, and its maximum number of schedulable RBs is R=106. Moreover, due to radio condition, the maximum cell capacity is about 130Mbps, and we send fixed 120Mbps downlink User Datagram Protocol (UDP) traffic to each UE. It is worth emphasizing that this scenario is created to observe the impacts on dynamic RAN slicing even when RAN UPs is fully loaded.

Moreover, six scenarios are described in TABLE II to serve both UE1 and UE2, with the number of slices varying between zero (Scenario 1), one (Scenarios 2 and 5) and two (Scenarios 3, 4, and 6). Also, both NVS [8] and EDF [9] algorithms apply different parameters in these scenarios: NVS uses the cap parameter to identify the slice radio resource share in percentage, while EDF uses the d parameter to indicate the maximum delay and the n parameter to denote the number of RBs provided during this period[2].

As follows, we go through the results of each scenario in terms of RB utilization percentage and MAC scheduler processing time at E2-Node (Fig. 4a), the perceived throughput at UE (Fig. 4b), and both one-way delay and packet loss percentage (Fig. 4c):

1) Scenario 1: At 5 sec, both UE1 and UE2 consume 50% of RBs because no slicing is employed and the PF algorithm is used at the user-level. Since the sum of UP traffic (240 Mbps) is much higher than the cell capacity (130 Mbps), the one-way delay and loss rate are high, but seem fair to both UEs.

2) Scenario 2: At 22 sec, we use the NVS algorithm and set 70% of the radio resources to slice 1, so that each UE takes 35% of the radio resources. Moreover, both one-way delay and loss rate increase accordingly (around 1.3x) due to traffic overload of a finite queue size (i.e., first-in, first-out blocking queue).

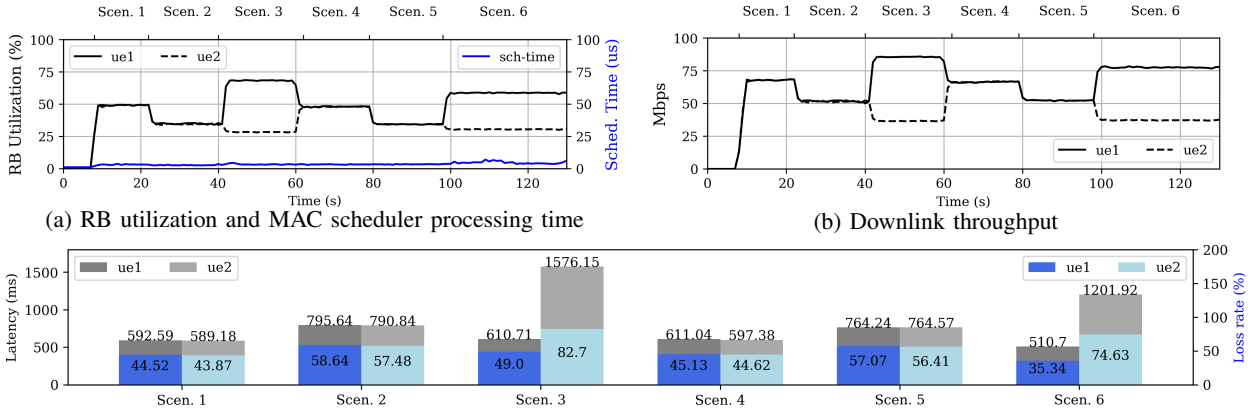
3) Scenario 3: At 40 sec, the second slice is set up with 30% of the radio resource for UE2; thus, UE1 will occupy all 70% of the radio resource for slice 1. As for the one-way delay in Fig. 4c, it is inversely proportional to cap values, but UE1 has a larger latency than in Scenario 1. This is because the NVS algorithm selects one slice at a time, so higher throughput (cf. Fig. 4b) does not imply lower delay (cf. Fig. 4c).

4) Scenario 4: At 62 sec, both slices use cap=50, so compared to Scenario 1, both UEs use a similar number of RBs and have similar throughput and loss rates. Due to the above characteristic of the NVS algorithm, the one-way delay is slightly higher than Scenario 1.

[2]The EDF algorithm creates a priority list of network slices based on their current deadline, but the number of RBs will not exceed n.

TABLE II: Description of six scenarios (Here s1 and s2 represent slice 1 and slice 2, respectively).

Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6
No Slicing	s1(NVS, *cap*=70): UE1, UE2	s1(NVS, *cap*=70): UE1 s2(NVS, *cap*=30): UE2	s1(NVS, *cap*=50): UE1 s2(NVS, *cap*=50): UE2	s1(EDF, *d*=2, *n*=150): UE1, UE2	s1(EDF, *d*=2, *n*=150):UE1 s2(EDF, *d*=20, *n*=620):UE2



(a) RB utilization and MAC scheduler processing time

(b) Downlink throughput

(c) One-way delay and packet loss percentage

Fig. 4: Network and User performance in a fully-loaded scenario with a centralized topology.

5) Scenario 5: At 79 sec, the EDF algorithm is applied and the second slice is removed. Note that the EDF algorithm will preserve n=150 RBs within the maximum delay of d=2 slots. So about 70% of RBs will be allocated to Slice 1, similar to Scenario 2; however, this scenario has a lower one-way delay because the preserved resources are fixed periodically, which is not the case for the NVS algorithm.

6) Scenario 6: At 98 sec, the second slice is set up, and it requests about 30% of RBs in a relaxed deadline d=20. In Fig. 4a, UE1 takes about 60% of RBs, while UE2 occupies 30% of RBs. It is worth noting that about 10% of RBs are unused, since we do not over-provide the values of n, which is smaller than the maximum number of schedulable RBs per slot. We also see that UE1 now has a lower latency and loss rate than Scenario 1, because it has a smaller deadline and thus takes precedence over UE2 most of the time, while both UEs are only handled by the PF algorithm in Scenario 1.

In short, the FlexSlice framework enables dynamic RAN slicing no matter what algorithms or parameters are changed on-the-fly. Also, the redesigned radio resource scheduler has a very small footprint (cf. Fig. 4a), even when RAN UPs is fully loaded.

B. Trade-off between different control logic topologies

In this part, we quantify the trade-off between three distinct control logic topologies (i.e., centralized, decentralized, and distributed) in terms of control loop latency and resource consumption. In their respective cases, control logic abstraction is deployed in the xApp, nearRT-RIC, and E2-Node to handle the service requirements in SLA SM (received through service APIs), which are then mapped to the slice configuration in Slice SM (sent via RAN APIs).

First, the control loop latency is measured between the control logic abstraction location and the corresponding RAN function in Fig. 5. We can see that the distributed topology has the lowest mean value and least variation, the reason behind this is that there is no need to leverage the protocol stack, i.e., Stream Control Transmission Protocol (SCTP), between E2-Node and nearRT-RIC as well as between xAPP and nearRT-RIC [4] for communication. Such benefit is essential to realize time-critical business logic into the real-time programmable RAN UP functions for deterministic behavior. In contrast, the decentralized and centralized topologies take 8x and 17x more latency, due to the extra one-hop and two-hop operations, respectively.

We then measure the memory usage of xApp, nearRT-RIC, and E2-Nodes in TABLE III before and after introducing the control logic abstraction. First, the extra memory incurred by realizing the control logic abstraction in nearRT-RIC (decentralized topology) is minimal. This is because the nearRT-RIC was originally designed to communicate with xApp and E2-Node via SLA SM and Slice SM, so no extra message handling is required. Also, when we compute the ratio of the extra memory usage to the baseline memory usage, the smallest result occurs in the distributed topology. This is because an E2-Node already requires a larger baseline memory to accommodate all RAN functions. But when a single business logic would like to control N E2-Nodes (N>1), this extra memory usage for control logic abstraction needs to be deployed in every E2-Node due to the distributed nature.

To sum up, there are pros and cons to deploying control logic in different locations. First, the centralized topology enables the xApp to directly define its

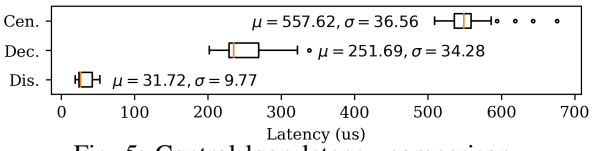


Fig. 5: Control loop latency comparison.

TABLE III: Memory consumption comparison.

Control logic topology	Control logic location	Baseline memory usage	Extra memory usage	Scaled to N E2-Nodes
Centralized	xApp	3.41MB	0.23MB	1
Decentralized	nearRT-RIC	3.50MB	0.10MB	1
Distributed	E2-Nodes	1.19GB	0.26MB	N

requirements for all RAN-Nodes, and it can naturally control multiple E2-Nodes without additional resources (cf. Table III), but at the cost of a higher control loop latency (cf. Fig. 5). In contrast, the decentralized topology requires minimal resources and can control multiple underlying E2-Nodes without further resources (cf. Table III). Also, the control loop latency is reduced compared to the centralized one (cf. Fig. 5). Finally, the distributed topology shows minimal control loop latency with the smallest variance to enable deterministic business logic; however, control logic abstraction needs to be performed at each E2-Node.

C. Summary and Discussions

In summary, our FlexSlice prototype is evaluated in two aspects. First, we show its capability to facilitate dynamic RAN slicing over the redesigned radio resource scheduler. In detail, there is no performance degradation or system interruption even when we change the number of slices, slicing algorithm, and UEs-slice associations on-the-fly. Next, three distinct control logic topologies are compared. The centralized topology is currently used in the O-RAN architecture, giving xApps full control over how their business logic is realized, at the cost of higher control loop latency and App/platform dependency. The decentralized topology can control multiple E2-Nodes with lower control loop latency and smaller resource usage. Lastly, the distributed topology is effective for the real-time control of deterministic business logic on few E2-Nodes.

In addition, we see that FlexSlice is paving the way for 5G-Advanced and 6G in several ways. First, it provides <50us (cf. distributed topology in Fig. 5) control loop latency, which can not only cope with shorter Transmission Time Interval (TTI) but also handle real-time control for rapid channel variation. Moreover, the enhanced interoperability of xApps after extending the centralized topology can naturally establish a cross-platform App marketplace to serve neutral hosts [12] and new verticals.

VI. Conclusions

Our proposed FlexSlice framework provides flexible control and real-time programmability for RAN UPs beyond the current O-RAN architecture to enable RAN slicing. Specifically, three control logic topologies are supported, and to realize this framework, both control logic abstraction and redesigned RAN functions are crucial. Finally, the FlexSlice prototype demonstrates its capability in supporting dynamic RAN slicing and adaptable control logic topologies.

Acknowledgments

This work has been funded by the European Commission as part of the H2020 program 6GBrain project, under the grant agreement 101017226, the Horizon Europe 2022 ImagineB5G project, under grand agreement 101096452, and the imec.icon project 5GECO, which is co-financed by imec and receives financial support from Flanders Innovation & Entrepreneurship (project nr. HBC.2021.0673).

References

[1] O-RAN Working Group 1, "Slicing Architecture," Tech. Rep., 2023, Technical Specification O-RAN.WG1.Slicing-Architecture-R003-v09.00.

[2] D. Johnson *et al.*, "NexRAN: Closed-loop RAN slicing in POWDER-A top-to-bottom open-source open-RAN use case," in *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterization (WiNTECH)*, 2022, pp. 17–23.

[3] A. Kak *et al.*, "HexRAN: A Programmable Multi-RAT Platform for Network Slicing in the Open RAN Ecosystem," *arXiv preprint arXiv:2304.12560*, 2023.

[4] C.-C. Chen *et al.*, "FlexApp: Flexible and low-latency xApp framework for RAN intelligent controller," in *ICC 2023 - IEEE International Conference on Communications*, 2023, pp. 1–7.

[5] R. Schmidt and N. Nikaein, "RAN Engine: Service-Oriented RAN Through Containerized Micro-Services," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 469–481, 2021.

[6] S. D'Oro *et al.*, "dApps: Distributed Applications for Real-Time Inference and Control in O-RAN," *IEEE Communications Magazine*, vol. 60, no. 11, pp. 52–58, 2022.

[7] X. Foukas *et al.*, "Taking 5G RAN Analytics and Control to a New Level," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23)*, 2023, pp. 1–16.

[8] R. Kokku *et al.*, "NVS: A Substrate for Virtualizing Wireless Resources in Cellular Networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1333–1346, 2012.

[9] T. Guo and A. Suárez, "Enabling 5G RAN Slicing With EDF Slice Scheduling," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 3, pp. 2865–2877, 2019.

[10] A. Ksentini and N. Nikaein, "Toward Enforcing Network Slicing on RAN: Flexibility and Resources Abstraction," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 102–108, 2017.

[11] R. Schmidt *et al.*, "FlexRIC: An SDK for next-generation SD-RANs," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '21)*, 2021, p. 411–425.

[12] J. F. N. Pinheiro *et al.*, "5GECO: A Cross-domain Intelligent Neutral Host Architecture for 5G and Beyond," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2023, pp. 1–7.