RAN Engine: Service-Oriented RAN through Containerized Micro-Services

Robert Schmidt and Navid Nikaein
EURECOM, Sophia-Antipolis, France
E-Mail: robert.schmidt@eurecom.fr, navid.nikaein@eurecom.fr

***Abstract*—Network slicing is considered to be the enabler for a coexistence of a multitude of services with heterogeneous requirements on a multi-tenant 5G infrastructure. In the core network, it has shown its potential in customizing and extending service-specific functionality beyond a mere configuration. In the radio access network (RAN) however, service customization and functionality extension remain a challenge due to the rigid and complex nature of the RAN and the fact that all services have to be mapped onto the scarce radio resources. In this article, we present the RAN service engine that allows services to customize and extend RAN functionality using containerized micro-services. This is achieved through micro-SDKs that abstract key RAN control endpoints, and which can then be used by the services to flexibly customize and extend the RAN in order to steer control plane behavior. Through these micro-SDKs, the engine can enforce isolation between services while multiplexing them efficiently onto the infrastructure. We also present a concrete implementation of the engine with its key micro-SDKs, and demonstrate the feasibility through a prototype for the MAC scheduling control endpoint, showing the versatility of the RAN engine.**

***Index Terms*—5G, RAN, Service, Customization, Extension, Slicing, Cloud-Native.**

I. Introduction

The forthcoming fifth generation (5G) mobile communication standards is a paradigm shift beyond the new radio and wider spectrum. It is about the evolution of computing for wireless networks in support of a variety of services and capabilities in diverse usage scenarios, such as low-latency and ultra-reliable communication (URLLC) [1]. These use cases exhibit characteristics that are partially incompatible to a mere enhancement of network performance or protocols. Given that the vertical market is the main driving force of 5G, the network shall be tailored to meet the respective service requirements in terms of (a) openness to allow services to control and customize the network towards their needs, and (b) network delivery on an as-a-service basis [2].

To tackle this problem, the concept of network slicing [3] emerged to overlay a common infrastructure with multiple, logically separated spaces for each service. In such an architecture, an infrastructure provider owns the network and requires to efficiently multiplex many services onto this multi-tenant infrastructure. On the other hand, service providers such as verticals wish to provide their service without having to run a full-fledged network. However, they need to customize the network to tailor it towards their needs, and ideally maintain control over the network running their service.

For the radio access network (RAN), the concept of RAN-as-a-service (RANaaS) [4] emerged as a way of running multiple services concurrently and efficiently, while having a relative freedom of customization. However, to the best of our knowledge, the current state of the art does not fully address how RANaaS could be realized. Instead, different concepts have been applied to realize aforementioned service-oriented vision: software-defined networking (SDN) and its close relative software-defined RAN (SD-RAN) have been used to bring programmability to the network for monitoring, control and coordination of the network. Network function virtualization (NFV) has been used to facilitate the life cycle management of the RAN, and cloud-native principles such as containerization and micro-services have been applied to quickly deliver and update new features. This calls for a holistic approach combining mentioned principles when realizing RANaaS. In particular, it is of paramount importance to use 5G's flexibility and enable a lightweight and customizable RAN service definition on top of a common infrastructure, rather than overloading service providers by running quasi-complete virtualized RANs (e.g., through a hypervisor [5]).

To address these issues, we analyze the existing RAN architecture and propose a system that allows to specify services and their behavior with minimal overhead while guaranteeing functional isolation and efficient multiplexing. In particular, the proposed solution does not depend on a full-fledged hypervisor approach, rendering the service implementation a complicated task. Instead, it abstracts (*synthesizes*) a base station, virtualizes control plane endpoints as micro Service Development Kits (micro-SDKs) and delivers it on an as-a-service basis. The micro-SDKs are abstractions of atomic RAN control plane functionality, such as scheduling, and enable services to configure, customize and extend the desired processing behavior independently on top of a shared base station processing pipeline, i.e., chained RAN processing functions, using service-specific RAN controllers while capturing the service characteristics and preserving security. Furthermore, similar to NFV, services can register service-specific control logic in the form of micro-services, effectively opening up the RAN towards service-specific customization and extensions.

In summary, this paper makes the following contributions:

- A RAN service engine that multiplexes services onto a common RAN infrastructure;
- A synthesized description of a base station with micro-SDKs that allow to modify RAN control plane behavior;
- A description of services as a set of containerized micro-

TABLE I
MAIN ACRONYMS USED IN THIS ARTICLE.

Name	Approach
CP	Control Plane
CU-CP	Centralized Unit-Control Plane
CU-UP	Centralized Unit-User Plane
DU	Distributed Unit
eMBB	enhanced Mobile BroadBand
MAC	Medium Access Control
micro-SDK	micro Service Development Kit
NFV	Network Function Virtualization
PDCP	Packet Data Convergence Protocol
PHY	PHYsical layer
RANaaS	RAN-as-a-Service
RAN	Radio Access Network
RLC	Radio Link Control
RRC	Radio Resource Control
RRM	Radio Resource Management
RU	Remote Unit
SDAP	Service Data Adaptation Protocol
SDN	Software-Defined Networking
SD-RAN	Software-Defined RAN
UP	User Plane
URLLC	Ultra-Reliable Low-Latency Communications

services to customize the RAN's control plane behavior, allowing to implement a service-specific RAN controller;
- A concrete prototype implementation of a selected set of micro-SDKs and micro-services based on the OpenAirInterface [6] and Mosaic5G [7] platforms, and experimental results highlighting its performance and capabilities.

The paper is organized as follows: in Section II, we review related work on network slicing and RAN customization. In Section III, we explain the current RAN architecture and the challenges that lie on the way to create a service-oriented network. Section IV gives an overview of how to customize the existing RAN through our envisioned architecture. The description of a service as well as how processing can be customized is explained in Section V, and the micro-SDKs are discussed in Section VI. Section VII evaluates the architecture, Section VIII briefly highlights the lesson learned, and Section IX concludes.

II. Related Work

Network slicing is an actively researched field. In the core network, network slices can be deployed and *customized* through virtual network functions running on a common infrastructure [8] through the concept of network function virtualization (NFV). The slice-based "network store" architecture [9] proposes a collection of 5G network functions to chain service-specific functionality, similar to a mobile app store. Furthermore, the application of cloud-native principles [10], such as containerized micro-services, improves the life cycle management of slices. The aspects of agility and scalability, i.e., fast deployment of network slices and scaling to network load, are further desirable aspects of cloud-native architectures[1].

Regarding the RAN domain, many works focus on the virtualization of radio resources. The network virtualization substrate (NVS) [11] virtualizes radio resources at the level of the medium access control (MAC) layer for different resource provisioning resources for an arbitrary number of service providers, but without considering functional isolation between them. The work in [12] generalizes this concept to a two-level scheduler that abstracts radio resources from physical resource blocks (PRB) to virtual resource blocks (vRB) to accommodate slices slices. However, no further customization of the RAN is considered. The two-level scheduler approach is further formalized in [13], showing that a parameterization can strike a balance between isolation and multiplexing gains.

In recent years, RAN slicing with a focus on customization and extensibility is investigated to enable a multi-tenant RAN, e.g., to enable service delivery for verticals. Early works [14] focused on the flexible RAN, i.e., using functional splits, to realize network slices. A parameterization of the upper RAN layers was proposed to realize different slices. The parameterization concept was further investigated by using slice descriptors [15] to configure the RAN towards different slices. However, in both cases the authors do not consider dynamic service-specific functional customizations of the slices. The work in [16] analyzes gaps in the current 5G architecture and proposes a slice-aware programmable architecture including a cloud-enabled protocol stack for slice differentiation in the RAN. Furthermore, the authors of [2] investigate enablers and open challenges for slicing with a focus on the openness of the RAN for vertical service customization. The customization capabilities and trade-offs of the 5G RAN on slicing are investigated in [17] with a particular focus on physical layer considerations for different slices. However, the authors do not consider the customization of higher layers.

With the disaggregation of the RAN into network functions (distributed unit, DU, centralized unit-control plane, CU-CP, and centralized unit-user plane, CU-UP) as specified by 3GPP [18], the RAN enables finer per-network function customization following NFV principles. In [19], the authors analyze the 5G RAN network functions with respect to the different sharing options, and investigate the level of customization and isolation of these options. Reference [20] criticizes that the current 5G-RAN standard does not specify the slice behavior, and harmonizes 3GPP and NFV viewpoints to enable the translation of slice requirements into customized RAN functionality through virtual network functions.

Programmability has been introduced through the application of software-defined networking (SDN) which decouples control plane from user plane processing to facilitate the customization of RAN processing. Early works [21], [22], [23] centralized control plane functionality to benefit from cloudification and virtualization for customization and coordination purposes. SDN in the RAN has been explored with the software-defined RAN (SD-RAN) controller FlexRAN [24], [25] that implements a customized south-bound API for such control plane (CP) and user plane (UP) separation. This allows to push programmable logic into the base station, but no service isolation has been studied. Recently, such software-defined RAN control has also attracted interest from a commercial perspective with the emergence of the O-RAN RAN Intelligent Controller (RIC) [26].

[1]Cf. the Cloud Native Computing Foundation CNCF: https://www.cncf.io/

TABLE II
COMPARISON OF SOME BASE STATION AND SERVICE CUSTOMIZATION APPROACHES IN THE LITERATURE.

Name	Approach	Isolation Level	Cust. Granularity	Cust. Overhead	Processing State
Network Store [9]	NFV	Service	Network Function	Fat	Stateful
FlexRAN [24], [25]	SDN Controller	Base station	Push control logic	Light	N/A
O-RAN RIC [26]	SDN Controller	Base station	No	N/A	N/A
Ferrús [15]	Configuration/SDN	Service	No	N/A	N/A
Orion [5]	Hypervisor	Service	Complete control plane	Fat	Stateful Slice Controller
RAN Runtime [27]	Multi-level SDN Controller	Service	RAN Layer	Medium	Stateful RAN layer
Engine (this paper)	*Micro-SDK* Composition	Service	Per control endpoint	Light	Stateless Micro-services

The authors of [27] propose a RAN runtime system that can dynamically slice and customize a RAN to meet slice requirements, such as isolation, sharing, and customization options. The system fulfills RAN customization requirements by replacing RAN layers, but does not address micro-customizations for individual control plane logic and possible state conflicts between the slice-specific RAN layers. Full slice virtualization has been explored through Orion [5]. The idea is that a hypervisor abstracts the RAN on a per-service basis and multiplexes the radio resources of each service onto the common resources. However, this approach has the drawback of a "fat virtualization" approach where a stateful network function implements the control plane of a complete service on top of the hypervisor.

Table II summarizes a selection of the existing work on RAN customization and extension and compares it with respect to the used approach, the level of isolation, what processing elements can be customized and the corresponding overhead, and whether these processing elements are stateful or not. We observe that the existing work only partially combine and apply SDN, NFV, and cloud-native principles in the design of a multi-service RAN architecture. Simultaneously enabling programmability, virtualization, customizability and extensibility, as well as agility and scalability on the fundamentally shared multi-tenant RAN infrastructure, is the main, yet unresolved, challenge addressed by this paper.

III. BACKGROUND AND RAN CUSTOMIZATION CHALLENGES

From a conceptual perspective, the base station can be divided into multiple RAN network functions with distinct functionalities [18], [28], forming a RAN processing pipeline (we will use the terms "base station" and "(RAN) pipeline" interchangeably): the CU-CP contains the radio resource control (RRC) for radio resource management (RRM), the CU-UP handles user plane-related functionality, and both are connected via the F1 midhaul [29] to the DU, which handles MAC functionality, such as radio resource scheduling. The DU in turn connects to one or more remote units (RU) via a fronthaul split such as eCPRI [30] or the O-RAN fronthaul [31], which also defines the split point of the physical (PHY) layer.

To link these network functions, 3GPP defined functional splits. It is therefore possible to *disaggregate* a base station and place parts of a base station in a central cloud (e.g., the CU), whereas others remain closer to the edge of the network (e.g., DU in a cell aggregation site, the RU is at the cell towers). With RAN disaggregation, a base station may be partially shared and partially dedicated, e.g. multiple CU-UPs can be created to (a) balance load and (b) implement service-specific functionalities on top of the shared RAN. However, the architecture is also quite coarse in the sense that service-specific customizations can only be carried out on a per-network function base, and typically only one CU-CP exists for many DUs and CU-UPs. Furthermore, the customization of RAN network functions to meet service needs is implementation-specific. For instance, each cell is typically composed of one DU and no standard way for service-specific customizations exists. Scalability and agility are limited, since no clear framework for instantiation and migration of services is in place.

SDN is supported through the decoupling of the CU into a CP and UP part. However, we note that each CU-UP might host service-specific control plane functions (e.g., scheduling of multiple applications within the service data adaptation protocol [SDAP]/packet data convergence protocol [PDCP] layers), and that the CU-CP hosts control functions for different services which need to be isolated from each other. Furthermore, no control and user plane separation exists on the level of the DU, despite the importance of the MAC-level scheduling on the base station performance in general and service-specific aspects of scheduling and selection of PHY-layer functionality, and that the F1-C midhaul is not sufficient to control the DU on a real-time basis. Also, 3GPP does not foresee a cross-layer communication path between the RLC (DU) and SDAP (CU-UP) to minimize queuing delays, which furthermore might be governed on a per-service basis. Therefore, we conclude that the separation of control and user plane has to be considered incomplete in the current state, and cross-RAN network function optimization is limited.

Additionally, a single SDN controller for RAN control brings its own security problems like denial-of-service attacks or confidentiality issues [32]. Distributing the control among services, limiting access through access control, and enforcing isolation between them through service-specific controllers can give service-providers access to their network while limiting potential security-related impacts.

Finally, the current pipeline has the underlying notion of layers instead of tasks. This complicates the description of services and their requirements (e.g., specific rate or latency requirements) as well as the composition and customization of service-specific functionality (e.g., hand-over logic for load balancing) according to these service-specific requirements, especially when considering many services, since a whole layer might need to be customized, instead of just a particular control end-point. Implementing service customizations involves understanding and modifying all RAN network functions in the pipeline, which typically is outside of the expertise of a service provider. Instead, a more structured approach of chaining and meshing the control and user plane functionalities for multiple services is required. Such structure should include a consolidation of control plane functionality across RAN network functions and the isolation of specific RAN functionalities that are typically modified by a service, opening up each pipeline for service modification.

The above observations call for a truly service-oriented RAN architecture that allows services to be mapped onto an underlying infrastructure, while abstracting network specificities such as disaggregation or deployment type. A consolidated control plane would allow a simpler access to control endpoints for service modification, contain state of different services in a central place, and manage the base station pipeline through an API that allows to isolate multiple services from each other, while giving access to user plane (state) modification and control. Such architecture could be enabled by a base station-local service container engine, allowing a more modular description of service processing extensions.

IV. Overview of RAN Service Engine

To realize the service-oriented (SO) RAN, we consider a base station as a RAN infrastructure pipeline, shown in Fig. 1. The pipeline consists of multiple RAN network functions that may be shared across multiple services. Through a service container engine, services are mapped and multiplexed onto this infrastructure allowing them to monitor, configure, customize, and extend (parts of) the pipeline subject to access control.

A **service** is characterized by a service descriptor consisting of specific SO resources (R), processing (P), and state (S). This triple defines the amount of (radio) resources this service is allowed to use, a number of customizations and extensions for this service in the pipeline's processing stages (execution logic), and associated state, such as configuration or user association. The descriptor delineates the service level agreement, including customizations, and corresponds to the service instance layer in NGMN terminology [3].

To map each service onto the infrastructure pipeline, we employ a **service container engine** to extend the capabilities and customization options of the infrastructure pipeline. The engine synthesizes a descriptor based on the pipeline descriptor to (i) reserve the appropriate amount of (radio) resources, (ii) integrate the processing customization in or apply control commands to the pipeline, and (iii) keep and expose the state of each service. The engine maps multiple services to the pipeline by effectively generating and exposing the topology of the underlying RAN network functions for each service with the desired level of granularity. The engine corresponds to the network slice instance layer in NGMN terminology, instantiating the network services.

The **infrastructure** is a 3GPP-compliant 5G-RAN base station, consisting of a CU-CP, multiple CU-UPs, the DU-CP and a DU-UP as the resource layer. In the proposed SO-RAN infrastructure, the DU is further decoupled into control plane and user plane to allow consolidation of the control plane at the service engine. This enables the coordination among CUs and DUs and increases the flexibility for network services.

As an example, consider the two enhanced mobile broadband (eMBB) [1] services in Fig. 1. Service "eMBB 1" uses all macro- and micro-cell sites, and deploys additional CU-UPs to efficiently handle the traffic; due to this reason, topology information is also exposed to the service itself. It also customizes handover logic (for mobility load balancing) and scheduling (efficient cross-site scheduling). Service "eMBB 2" merely reserves some resources; therefore, no topology information is exposed. No processing is customized either.

The service engine's task is to *multiplex* services onto the common RAN infrastructure while allowing per-service control plane customization and extension and ensuring isolation between multiple services. The service engine allows pipeline customization through micro service development kits (*micro-SDKs*), encapsulating pipeline control endpoints through which the desired behavior of the control plane is synthesized before being applied to the user plane, such as mobility management. Services can customize the behavior of the pipeline by providing isolated, stateless **micro-services** that implement the interface towards micro-SDKs, such as a specific handover policy for mobility load balancing. Service-specific state information is partitioned through a monitoring micro-SDK to ensure isolation while common state information, e.g., base station info, are shared across multiple services.

The **micro-SDKs** open up the RAN for per-service customizations and extensions and are used by micro-services to reprogram and reconfigure the infrastructure pipeline. They are deployed through an internal runtime to manage service lifecycles. The micro-SDKs allow a light-weight virtualization of the base station's control plane *when needed* as well as a composition of a service-specific RAN controller. Existing work such as Orion [5] requires the full implementation of a virtual control plane above the specified abstraction (e.g., S1 and X2 protocols, as mentioned in the paper), leading to a "fat" hypervisor approach, regardless of whether services need to customize this control plane or not, whereas the engine only abstracts specific control end-points, enabling more light-weight controllers. Orion might then be implemented as a micro-SDK. From an information exposure point of view, the service container engine exposes the same information as does the E2 interface towards O-RAN's RAN Intelligent Controller [26], or the FlexRAN controller [24].

The engine "re-aggregates" (i.e., consolidates) the control plane of the infrastructure towards the services through the micro-SDKs, which only expose granted RAN customization and extension facilities as requested by a service and regardless of the physical deployment type (e.g. monolithic, cloud-RAN, or disaggregated). Such re-aggregation facilitates the life-cycle management of services, since each specific control point is encapsulated on a per-micro-SDK basis. Furthermore, it allows to decouple the control plane behavior and configuration, which might be customized on a per-service basis, from the control plane protocol and user plane handling, governed by

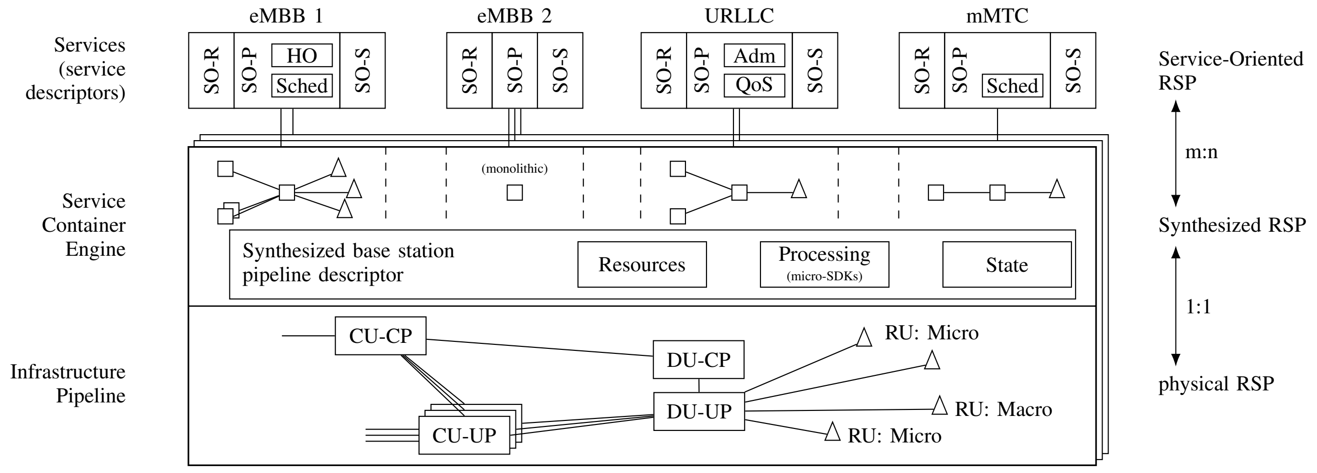

Fig. 1. Overview of our architecture, consisting of an infrastructure, the service engine, and services that are multiplexed onto the infrastructure. The service engine exposes deployment and topology information to the services as depicted in the little RAN representations within the service engine.

the specification. This decoupling is done through a protocol between the service container engine and the services, ensuring strong functional isolation, while the micro-SDKs have the task of efficient multiplexing of control decisions onto the pipeline. Each service can then be easily deployed and modified, and control the part of the infrastructure belonging to this service independent from other services.

As also shown in Fig. 1, one service container engine governs one infrastructure pipeline, whereas multiple services connect to multiple infrastructure pipelines in the network:

- The engines *synthesizes* the infrastructure pipeline: it represents the available resources, and aggregates the state of multiple RAN functions, similar to the first-level physical-to-logical base station abstraction as done in a RAN controller [25]. It also consolidates processing control endpoints through micro-SDKs, allowing to re-program control plane behavior for RAN customization on a per-service basis.
- The engines *maps* multiple services onto a single infrastructure pipeline by partitioning state on a service basis, mapping service-specific resources onto the physical resources, and linking micro-services for RAN customization through the micro-SDKs. Services control a sub-network of the complete RAN by being mapped to multiple pipelines (see Fig. 1). This corresponds to the second-level logical-to-virtual base station abstraction [25].

The above abstraction allows (i) the coexistence of multiple services on a shared infrastructure, (ii) an efficient multiplexing of services onto the base station resources, and (iii) a full isolation of services. Micro-services enable service controllers that coordinate within both a single and multiple (disaggregated) base stations through the composition of micro-SDKs: across different resource dimensions in the RAN, i.e., (1) radio resources (MAC), (2) connection and configuration control (RRC), and (3) data flow management (SDAP/PDCP), as well as across multiple base stations, e.g. mobility load balancing. Each of these controllers might operate on different levels of the abstraction, e.g., service providers control on top of the second level of abstraction (e.g., specific micro-SDKs) whereas the infrastructure provider has full control on the consolidated control plane of the whole infrastructure (full stack). This increases the security and isolation, since there is no centralized (service) controller that might become a "single point of failure", unlike the existing solutions such as O-RAN's RIC [26] or FlexVRAN [25].

V. Service Engine and Service Embedding

The Service Engine abstracts the underlying infrastructure in a network-deployment-independent fashion to provide a unified execution environment. It has the tasks of mapping the pipeline's (i) resources, (ii) state, and (iii) customizable processing control endpoints into a synthesized description that at a time allows to reconstruct the pipeline, and that leans toward a service-oriented description. Starting from this abstraction, a unified description of services (second abstraction) allows to describe service specificities and requirements that can then be mapped onto the infrastructure by the engine, effectively referring to a sub-pipeline specifically for this service.

To conceptualize the representation of a RAN pipeline and the embedded services, we introduce descriptors for both the pipeline and the service (c.f. Fig. 1). The *pipeline descriptor* defines the actual infrastructure pipeline and available control endpoints for independent base station control. The *service descriptor* defines the mapping and the required customization to tailor the pipeline to service requirements. The descriptors consists of three elements:

- **Resources** denote the radio resources of the base station, typically including operating bands, numerologies, and operating bandwidths. Combined with processing, a number of performance indicators can be inferred from the resources such as sustainable data rate, number of devices, or minimum latencies. Given that the radio resource allocation among services is governed through a slicing algorithm according to service objectives (rate, delay, etc.), the service resources shall include such objectives.
- **State** consists of control and user plane parts that are used by the service control logic to steer the behavior

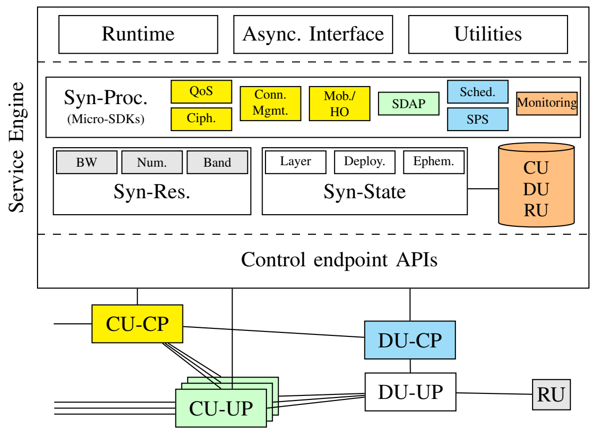

Fig. 2. The service engine synthesizes the pipeline. Processing provides micro-SDKs for control endpoint abstraction.

of the pipeline. This typically includes the deployment configuration (e.g., separate CU-UP), RAN layer-related configuration like user-to-service association and other per-UE configuration like identifiers and bearers, and (potentially ephemeral) user plane statistics.

- **Processing** denotes the functional control endpoints of the pipeline and service customizations. Processing is tightly bound to the resources and the state, since any processing decision (steering control plane behavior) operates on top of the current state, and has access to certain resources and shall respect the constraints imposed by the resources. In particular, the processing part lists the customization of the pipeline that entails the embedding of a particular service.

A. Synthesized Pipeline Representation

To capture the control plane of the pipeline, the engine interfaces with the pipeline through an API as shown in Fig. 2. As has been mentioned, the synthesized resources capture properties such as the bandwidth and numerologies allowing to infer a sustainable data rate during runtime for this pipeline and to reserve resources in RAN layers (e.g., in terms of CU-UP throughput requirements). Note that this corresponds to a description of the resources of the pipeline and not an abstraction which is ensured by processing customizations.

The synthesized state captures RAN configuration as well as user context information and forwards it to a shared storage that gathers this information, consolidating the state information. Such state is then exposed to the services as generic base station information or in relation to a micro-SDK, i.e., a particular RAN control endpoint, to simplify control handling within the services.

The synthesized processing of the pipeline descriptor abstracts the access to specific control plane endpoints in the form of micro-SDKs. From a service perspective, the engine provides access to the pipeline via an API, and a runtime manages the lifecycle of the micro-SDKs, which can be deployed and released dynamically within the service engine like applications [33]. The micro-SDKs provide atomic, independent, extendable CP abstractions through the APIs, and implement an abstract interface towards the services for RAN customization and extension. They provide operations to map, deploy and release services for a control endpoint, apply service actions, and resolve conflicts. Additionally, micro-SDKs might compose existing micro-SDKs to reflect complex operations.

The set of micro-SDKs of a RAN pipeline marks the consolidation of the actual control plane APIs for services in a deployment-independent and vendor-independent fashion. These control plane APIs can be grouped into (1) UP handling, e.g., packet scheduling of data flows at the SDAP, (2) radio control handling, e.g., mobility control at the RRC, or (3) radio resource handling, e.g., MAC scheduling. The micro-SDKs thus reflect the pipeline's modification possibilities in the form of a *capability* that might be customized by a service. For radio resource allocation for instance, a micro-SDK could give access to a part of the spectrum for custom scheduling. The implementation of the micro-SDKs further depends on the actual endpoint, as described in Section VI.

This synthesis of the pipeline allows a consolidation of the control plane of the base station. The processing descriptor groups control endpoints for customization and configuration, and the resources in the descriptor are used to reserve RAN resources within the pipeline. The state of the pipeline is logically centralized, allowing to execute control plane behavior logic independently of the pipeline and rendering parts of the pipeline "stateless", i.e., control decisions may not need to be taken in the pipeline. By definition, the pipeline cannot be made completely stateless, since (instantaneous) state is necessary for consistency in control plane operation (such as HARQ) and interaction among the layers (e.g., amount of buffered data). However, information such as user context, identifiers, runtime configuration such as bearer information, the scheduling policies, or active bandwidth parts, is stored to perform failover or replication in the CU-UP and DU-UP. Also, this means that the base station description is fundamentally independent of the deployment scenario (Cloud-RAN, monolithic, or disaggregated), and a redeployment of control logic on top of different base station pipelines within a multi-vendor context becomes feasible, since the micro-SDKs expose a consistent interface towards the services.

Through CP consolidation, a tighter control coordination (e.g. joint signal processing) among different RAN network functions becomes possible while retaining the benefit of multiplexing gain and cross-layer optimization. As an example, CP consolidation enables a joint SDAP (CU-UP) and RLC (DU-UP) packet scheduling opportunity to reduce the buffer bloat [34], which otherwise can not be supported in a disaggregated RAN, since such information is not available via the F1 interface.

B. Service Description

With the help of the synthesized pipeline descriptor, services can now be multiplexed onto one or more pipelines based on the agreement between service provider and infrastructure provider, as shown in Fig. 3, providing a service-specific and network-wide view. A service descriptor describes the

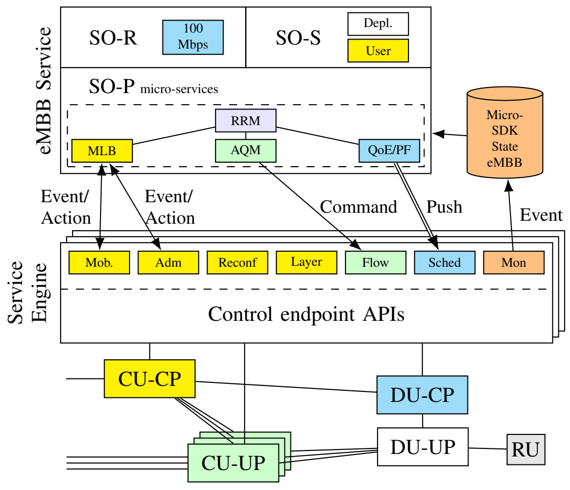

Fig. 3. Services multiplex their micro-service customizations through the service engines onto the underlying pipelines by matching micro-SDKs. Based on the micro-SDK, a message-based exchange might be employed (e.g., for MLB or AQM) or functionality will be pushed into the engine for timing constraint reasons (Sched). Rate information is used to reserve resources in the RAN, and state is partitioned per service.

resources that are to be reserved, service-specific configuration (state), and provides service-specific processing customizations and extension to describe the behavior of the 5G-RAN pipeline as per service requirements.

The service-oriented resources (SO-R) specify the desired performance indicators that need to be supported in the pipeline (e.g., the radio resources by an underlying slice algorithm). Upon admission control, the resource is mapped into a rate by the service engine in order to reserve resources in RAN layers and select appropriate network functions, as described further below. The service-oriented state (SO-S) describes the non-ephemeral configuration of the service: next to the obvious user association, this would include the usage of specific features, e.g., enabling multi-path transmission, disabling retransmission for latency-critical services, or specific RAN network function configuration such as configuration for a separate CU-UP, if applicable. Further, the state includes ephemeral data such as statistics for service optimization. For a configuration example, reconsider the URLLC service in Fig. 1 (overview figure). Such service might use a specialized CU-UP for faster processing, limit the used RUs to macro cells for better coverage, and perform special admission control to ensure reliable transmissions at all times. State is partitioned by a monitoring micro-SDK on a per-service basis in order to provide isolation and privacy.

The service-oriented processing (SO-P) allows to customize and extend processing functionality by combining micro-SDKs at the service engine. The service processing descriptor marks customization intentions for this service, and either supplies the execution logic in binary form as a micro-service, or an end-point for message-based interaction. Here, customization refers not only to a reconfiguration or replacement of specific RAN control end-points for a service, but also possible extensions of control plane behavior logic, for instance using machine learning to predict user traffic and adjust the scheduling algorithms. This allows the service provider to quickly deploy and modify service-specific functionality, and extend basic RAN functionality by composing multiple micro-services or through specialized micro-SDKs in the engine. For instance, a specialized micro-SDK reuses monitoring and MAC scheduling micro-SDKs: through traffic analysis of a group of users and reconfiguration of resource allocation, scheduling latencies are reduced and quality of service (QoS) improved, as in the “burst analysis” micro-SDK in Section VII-E.

Based on the actual RAN control endpoint and its timing requirements, the interaction between a service and its associated micro-SDKs inside the engine shall support both hard and soft real-time operations to meet deadlines. The micro-SDKs link the infrastructure to the services, and thus control triggers might come from the pipeline (e.g., event for necessary handover) or the services (command for policy update to improve service). This is supported by general event-action messages, direct service-originated commands, or the embedding of custom logic within or close to the engine (“push”). As shown exemplary in Fig. 3, a service can supply micro-services for mobility load balancing (MLB), active queue management (AQM), and scheduling (Sched) to extend pipeline processing. For the MLB, the service execution logic might be triggered through an event which is handled remotely. Similarly, the AQM micro-service continuously monitors the RLC queue state to update the SDAP processing. For scheduling, a co-location of execution logic is required to keep the tight deadlines imposed by 5G, e.g., through shared memory on the same execution environment.

The admission control process enforced through the service engine is a three-step process: (1) the resource needs of the service need to be fulfilled. This is dependent on a separate admission control in at least a subset of micro-SDKs, such as the scheduling, which needs to verify that radio resources are available, potentially translating latency-based or other requirements into rate requirements. (2) The specific configuration of the services, including rate requirements, need to be checked for consistency with the current pipeline state, and configuration conflicts need to be resolved. Consider an eMBB service with high rate requirements, requiring the set-up of a new CU-UP; or semi-persistent scheduling (SPS) for a machine-to-machine service that needs to be aligned with existing SPS configurations. (3) The service engine’s micro-SDKs need to accommodate the micro-services of the new services, e.g., start MAC scheduling micro-services. If admission control passes, the service engine will finally map the service onto the existing RAN network functions.

If a service has been admitted, a set of micro-SDKs are revealed, allowing this service to implement a service-specific controller as shown in Figure 3. Each controller is only aware of state pertaining to the corresponding service, and controls, reconfigures and/or reprograms its subnetwork (i.e., the service). This is fundamentally different from current RAN controllers such as O-RAN’s RIC or FlexRAN, which control the whole base station, whereas the proposed service-specific controller remains isolated from other controllers and a failure in micro-services of one service leave operation of other services unaffected.

VI. Micro-SDKs

The service-specific execution logic are containerized, stateless micro-services with a limited, service-specific view on parts of the control plane, which is enabled through micro-SDKs. The micro-SDKs encapsulate an API to a control endpoint in the RAN, effectively *programming* the pipeline. In the following, we describe exemplary micro-SDKs for monitoring as well as in CU-CP and DU-CP, and how they enable the micro-services to act on control points in the RAN network functions.

A. Monitoring Micro-SDK

The monitoring micro-SDK is possibly one of the most fundamental micro-SDKs: monitoring of the RAN allows a service to observe the RAN, react on changes in the user plane, and reprogram the control plane through other micro-SDKs. Therefore, it is an integral part of the service engine of all RAN network functions. As shown previously in Fig. 3, it partitions the state information of the pipeline on a per-service basis and exports RAN layer, deployment, and ephemeral user plane information in the form of messages. Due to the possibly high number of users in the pipeline and the high frequency of information generation (every TTI), scalability is a major characteristic, which we evaluate in Section VII-D. The reporting frequency (i.e., how often) and granularity (i.e., which information) is configurable to reflect the differing requirements among services.

B. Micro-SDKs in the CU-CP

The RRC contained in the CU-CP hosts functionality for connection management, mobility, radio resources management, and QoS, among others. The functionality can be divided into cell-common and user-specific functions [27]. Common functionality includes information not destined to individual users, such as cell information broadcast or emergency services, and remains at the CU-CP, as shown in Fig 3.

User-specific functionality can be handled by services in an isolated fashion, exposed through a micro-SDK that encapsulates the corresponding control endpoint. User-specific functionality includes layer configuration, UE connection reconfiguration (bearers, measurements, ...), as well as admission, mobility and secondary cell control. As processing in the CU-CP has soft real-time requirements, service processing is executed remotely, allowing to (a) customize specific control plane aspects while extending the possibilities of control plane actions, and (b) deploy at scale with isolated context among services through state partitioning. Micro-SDK and micro-service exchange through event/action messages, where events trigger a decision process resulting in an emitted action. Finally, the micro-SDK checks the result and handles conflicts.

For instance, a mobility management micro-SDK might handle handover commands and check for valid source and target base stations, rejecting the request otherwise. Similarly, upon the admission of a new (dedicated) bearer, the engine might adjust the guaranteed bitrate (GBR) to accommodate all the services, e.g., when the sum of GBRs of all UEs for this service exceeds the amount of resources allowed in the service descriptor.

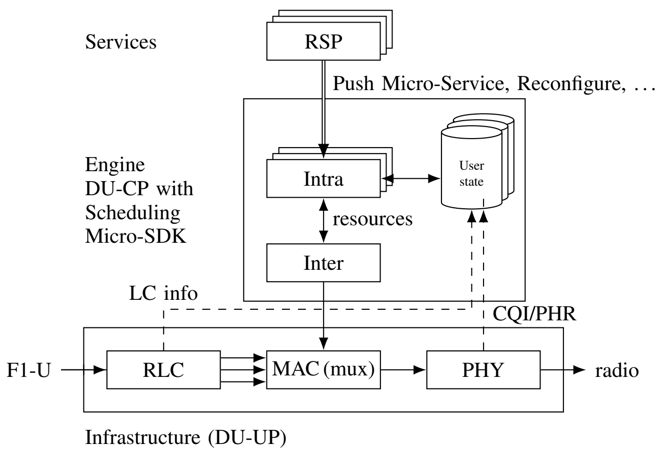

Fig. 4. DU CP & UP deconstruction. The DU-CP is part of the Service Engine and implements the “Scheduling” Micro-SDK. Micro-Services are pushed into the engine, or might be connected through a low-latency interface.

C. Micro-SDKs in the DU-CP

The DU-CP provides a micro-SDK for the MAC scheduling control endpoint of the RAN infrastructure. This micro-SDK is realized by means of a two-level scheduler as an extension to our previous work [12] with an additional feature that guarantees isolation and conflict resolution among the services. The proposed architecture is shown in Fig. 4. It consists of an inter-slice scheduler for resource slicing, and a per-service intra-slice scheduler for user scheduling. The inter scheduler implements the micro-SDK to accommodate multiple micro-services in the form of the intra schedulers. Both schedulers belong to the control plane part of the DU. The MAC multiplexing functionality (mux) in the user plane receives the control plane actions and applies them in the pipeline. The scheduler indirectly controls RLC and PHY: the mux fetches data from RLC and multiplexing it onto the PHY layer as a result of control plane actions.

In every slot, the inter-slice scheduler partitions the eligible resources for the slices. Note that the actual resource partitioning between multiple slices is still under active research, and out of scope of this work. The inter scheduler informs the intra-slice scheduler about the allowed RBs using a bitmap (i.e., bandwidth parts), and a list of slice-associated users that this slice should schedule. This list also includes per-UE information required for scheduling, such as buffer status, QoS, or channel quality indicators (CQI). The intra-scheduler then schedules active users on the allowed time-frequency resources. Here, any scheduling strategy might be implemented, e.g., a simple round-robin, more advanced channel-aware or QoS-aware scheduling schemes, or beamforming-based scheduling schemes, among others. Finally, the inter-slice scheduler resolves possible conflicts in the final scheduling decision, such as an attempt to schedule resources outside the assigned radio resources, before applying it via the MAC multiplexing functionality of the pipeline.

The decoupling of service and user scheduling has a dual advantage of (i) enforcement of various slice scheduling schemes or policies within the inter-slice scheduler, i.e., the micro-SDK, and (ii) customizability of user scheduling within the micro-service. The intra scheduler either runs within the engine for real time purposes or in a separate execution environment for isolation purposes, is implemented as a micro-service, and its current state (e.g. user channel quality, buffer status) is fetched with every request. Such a stateless behavior allows to dynamically manage the life cycle of the scheduler as per service needs or when a failure occurs while retaining service continuity and performance requirements. To guarantee a secure, functional isolation between service and user scheduling, the inter-slice scheduler uses shared memory per service: services implement their own intra-slice scheduler, which is run in a separate process, and shielded within a container.

We claim that such separation between inter- and intra-slice schedulers in the micro-SDK allows increased customizability of the system, and decreases complexity by separating the scheduling problems of slices and users. This allows to push a service-specific radio resource allocation that exploits the future channel quality as a function of predicted user mobility to ensure QoS by relying on remote machine-learning within the system.

VII. Evaluation of the Architecture

To verify our design for an extensible and customizable RAN, we implemented prototypes of the scheduling, monitoring, and "burst analysis" micro-SDKs, and scheduling micro-services, within OpenAirInterface [6] (OAI), an open-source LTE implementation, and FlexRAN [25], an SD-RAN controller. Also, we implemented multiple slice scheduling algorithms as well as user scheduling algorithms corresponding to the inter and intra schedulers of the scheduling micro-SDK.

For our experiments, we used OAI in the "L2 simulator" flavor: in this deployment, the eNB directly exchanges MAC-layer nFAPI [35] messages with an UE emulator, also based on OAI. Thus, no PHY layer is present, and no PHY channels (with associated noise, interference, or other models) are emulated. The remaining layers are, however, fully standard-compliant, and we claim that such deployment (1) suffices to demonstrate our RAN customization and extension prototype and (2) has the advantage of scalability, since a large number of UEs can be emulated, which would be infeasible with full PHY channel emulation. Besides, the prototype includes an OAI-based core network, connected to the eNB, and a modified FlexRAN controller to support the newly developed micro-SDKs. The eNB uses a 5 MHz-wide (25 RB) channel that has a maximum capacity of 17,36 Mbps and minimum application plane (ping) round-trip time of roughly 15 ms. Table III summarizes the experimental parameters.

In the following, we present results for the scheduling micro-SDK: (Section VII-A) on slicing in the inter scheduler, (VII-B) its scalability in terms of scheduling processing time, and (VII-C) the extensibility of the RAN with respect to the intra scheduler; further, (VII-D) the signaling overhead using the monitoring micro-SDK; and, finally, (VII-E) a case study of the "burst analysis" micro-SDK. Note that while these results have been obtained using an emulator setup, the prototype can also be used together with custom-of-the-shelf UEs.

TABLE III
Experimental parameters

Parameter	Value
Access Technology	LTE
Bandwidth	5 MHz/25 RBs
Channel Model	Static (CQI 15) unless noted otherwise
No. of Users	2–128
Mobility	Static
Traffic	Full buffer unless noted otherwise
Slice schedulers	Static, NVS [11], SCN19 [36]
User schedulers	Round robin, prop. fair, max. throughput [38]

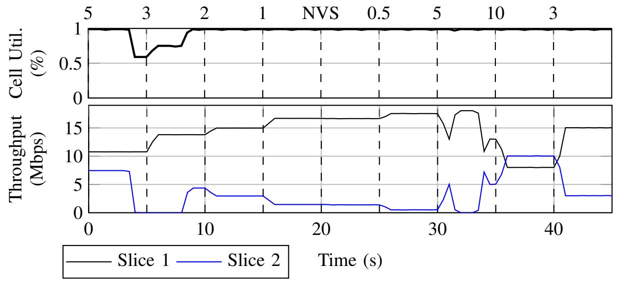

Fig. 5. Cell utilization and slice throughput during dynamic slice algorithm changes. Both share the same x-axis (time). At first, a static slice algorithm is set with decreasing amount of RBGs for slice 2, as indicated at the top. After a switch to NVS ($t = 20$ s), resources are shared automatically ($t = 33$ s), and precise slice throughputs can be set.

A. Slicing in the Service-Oriented Architecture

The infrastructure provider might change the active slice algorithm in inter phase of the scheduling micro-SDK (cf. Fig. 4) to fulfill slice requirements when deploying new services, or adapting to the network behavior. Such automation might happen at any time during runtime of the pipeline.

We implemented two slicing strategies: static and NVS [11] within the inter scheduler of the micro-SDK. In static slicing, each slice receives a fixed share of the radio resource in every subframe as per slice requirements, and the slice users are scheduled freely. The advantages are low scheduling delays and a strict isolation in frequency, making this scheme more adequate for public safety networks, interference scenarios, or constant traffic load. For fluctuating traffic demands, this means less sharing opportunities (unless a frequent re-configuration takes place), and only a couple of slices can be active due to the limited number of control channel elements (CCE) in LTE. Also, the minimum sliceable resource granularity is the resource block group (RBG). NVS performs slicing by tracking the amount of resources that each slice received through a weight, and scheduling slices in a complete subframe. Therefore, an arbitrary number of slices can be present, and both rate-based (Mbps) and resource-based (percentage of aggregate RBs) slices with arbitrary precision are available. Also, the algorithm adapts to traffic fluctuations.

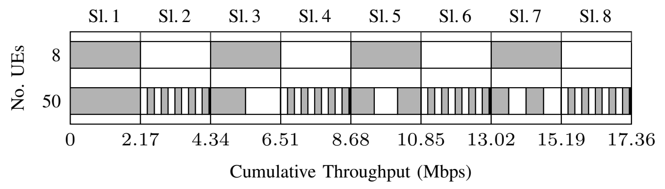

Fig. 6. Functional isolation and scalability for slicing using the micro-SDK. Above: Eight slices with eight users have the same throughput. Below: With additional users, the slice throughputs remain the same.

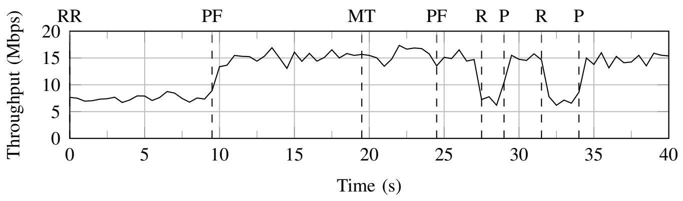

Fig. 8. Cell throughput over time and switching schedulers dynamically (RR, R: round robin; PF, P: proportional fair; MT: maximum throughput).

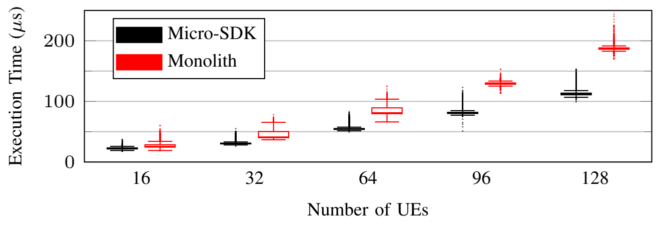

Fig. 7. Boxplots of the execution time of the scheduling phase of scheduling micro-SDK vs. the non-service-oriented scheduler for varying number of UEs.

Consider the exemplary time series in Fig. 5. Starting at $t = 0$, two slices are present, and slice 2 has 5 RBGs as noted at the top of the graph. Note how slice 2 has no traffic at $t = 4\,\text{s}$. Since static slicing has no notion of sharing resources, the cell remains underloaded. The resources of slice 2 are increasingly reduced by setting the number of RBGs down to 3, 2, and 1 RBGs. Note that the resource granularity is coarse.

By dynamically switching the slice algorithm, the architecture is able to address the problems regarding sharing and resource granularity. At $t = 20\,\text{s}$, the slicing algorithm is changed to NVS without any impact on cell performance. At $t = 30\,\text{s}$, the throughput of slice 2 is reconfigured to 5 Mbps, but uses the assigned resources only briefly before stopping transmission. The resources are shared with slice 1 automatically by NVS, leading to a multiplexing gain of 27,7 %. It is furthermore possible to set exact rates as indicated, e.g., 10 Mbps at $t = 35\,\text{s}$, or 3 Mbps at $t = 40\,\text{s}$. The delays for reaching the assigned throughput are due to the exponential moving average used by NVS to track the resource usage of the slices.

We configured eight slices with equal throughput and each one user, as shown in Fig. 6 (application throughput with iperf over 10 s). To assess both the isolation between the slices as well as the scalability for many slices, we connected additional UEs: slices 1, 3, 5, and 7 have 1, 2, 3, and 4 UEs, respectively, and all other slices have 10 UEs. Note how the throughput per slice does not change. This shows, that (a) isolation between slices is guaranteed, and (b) the scheduling SDK is scalable enough to handle eight concurrent slices with 50 UEs.

B. Execution Time of the Scheduling Micro-SDK

To verify that the scheduling micro-SDK is indeed capable of handling multiple service providers, we compare the scheduling execution time of the micro-SDK (cf. Fig. 4) against the non-service-oriented, monolithic scheduler as a baseline. For the micro-SDK, we measure all phases, i.e., intra, inter scheduler, and mux functionality to have a direct comparison to the baseline. Since the inter scheduler typically partitions resources in a fixed manner (Static) or according to a weight (NVS), we assume a limited overhead for the inter scheduler. However, the intra slice schedulers scale with the number of users, frequency resources, and space.

Fig. 7 shows the scheduling execution time as a boxplot, measured for every subframe over 30 s. For the micro-SDK, we use one slice and the default round-robin scheduler; the monolithic implementation uses round-robin, too. The service-oriented scheduling micro-SDK shows an approximately linear overhead with the number of users. This indicates that running multiple intra-schedulers sequentially will incur an additional execution time that is linear to the number of users (roughly 1 μs/UE), plus a small fixed overhead. On the other hand, the monolithic implementation's execution complexity not only seems to be sup-linear, but the execution time also shows a higher variance, making it impractical for sequential execution. Additionally, the monolithic version is a multi-threaded implementation. Since the micro-SDK is not multi-threaded yet, we see potential for an even lower execution time.

C. Extensibility through Cloud-Native Behavior

The scheduling algorithm of a service is a stateless micro-service that can be set by the slice owner (cf. Fig. 3). This micro-service corresponds to the intra scheduler (see Fig. 4) that is pushed into the scheduling micro-SDK. Since the scheduling state such as UE information (CQI, etc.) is kept outside of the intra scheduler, the intra scheduler can be changed on-the-fly by service owners to adapt to user behavior.

We implemented the classic round-robin, proportional fair and maximum throughput scheduling algorithms [38]. We consider one slice with eight users with fast fading, including sporadic deep fading, and a full buffer traffic model. Note that in this evaluation, we do not attempt an exact assessment of the scheduling schemes already present in the literature, but rather highlight the extension capabilities through the micro-SDK. For simplicity, we use one intra scheduler, but multiple might have been used together with the presented slice algorithms.

Fig. 8 shows a sample cell throughput at the MAC layer over time when changing scheduling schemes dynamically. In the beginning, due to the channel-unawareness of round-robin (RR) scheduling, the cell throughput is low, but all UEs get a fair, equal amount of resources. When changing to proportional fair (PF) scheduling, the cell throughput doubles, since PF scheduling opportunistically schedules high-CQI UEs. At the time of change, no performance degradation

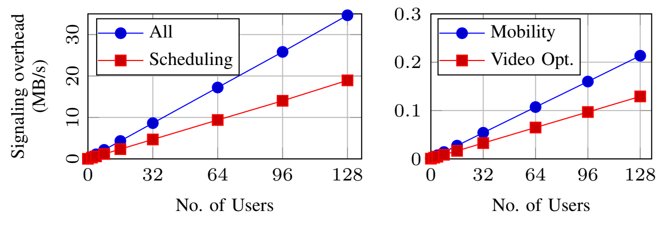

Fig. 9. The incurred overhead for monitoring traces for four sample services and up to 128 UEs. Left: exporting all data and MAC scheduling every ms. Right: mobility management and video optimization receive data every 30 and 100 ms, resp. Note the different scales of both plots.

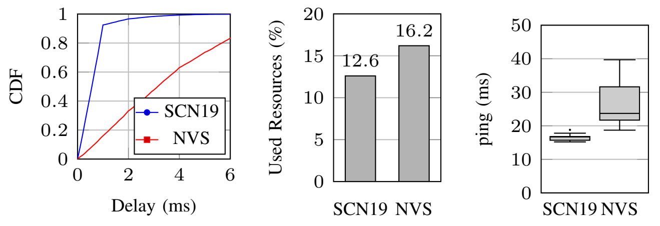

Fig. 10. Comparison of the slice algorithms SCN19 [36] and NVS [11] w.r.t. scheduling delays [36], used resources [36], and measured ping latency [33]. SCN19 keeps scheduling delays lower than NVS while using less resources.

is visible, since the proposed design and implementation is able to switch the scheduler atomically between subframes. Switching to maximum throughput (MT) yields only marginal gains, with no fairness being ensured.

Note the dynamicity of the scheduling scheme switches, starting around 25 s in Fig. 8. The slice owner can arbitrarily change the schedulers without any impact on service continuity, which is guaranteed at all times.

D. Overhead of Monitoring micro-SDK

Monitoring is a separate micro-SDK that exports configurable, service-specific traces of pipeline statistics. Service providers might use these traces for performance monitoring, service optimization, or other service-related tasks, and scalability is of paramount importance in order to serve all of them. For isolation purposes, the micro-SDK supports different services by partitioning state per service and exporting data only towards the corresponding service, as shown in Fig. 3.

We studied the incurred monitoring overhead for four different services. (1) The infrastructure provider traces all data every ms for debug purposes. (2) Another service performs remote MAC scheduling and needs data of the PHY, MAC, and radio link control (RLC) sublayers every ms. Furthermore, (3) a service performs mobility management: it receives all RRC-related measurements every 30 ms. Finally, (4) a video optimization service receives packet data convergence protocol (PDCP) and CQI traces every 100 ms.

Fig. 9 shows the signaling overhead, measured over 10 s. When exporting all data, every UE causes a monitoring overhead of approx. 250 kB/s. The overhead drops by roughly 50% for the MAC scheduling use case. Considering that the user plane throughput of an active UE will be much higher, this seems reasonable. The overhead of the mobility and video optimization use cases is two order of magnitudes lower and thus negligible, even for many UEs. Thus, the results confirm the scalability of the monitoring micro-SDK.

E. Use case: "Burst Analysis" Micro-SDK

NVS allows to translate the spectrum into a notion of rates and can effectively share the resources, but its design can lead to increased scheduling delays, since a low-rate slice might be scheduled only seldom. In our previous work [36], [37], we designed a slice algorithm ("SCN19") that extends NVS with delay-bound slices while enforcing a maximum aggregate resource share over a time window. This allows delay-bound slices to temporarily use more resources when needed, effectively sharing resources in favor of low-latency slices, but resources are lost if not used. Thus, for UE groups with mixed traffic patterns, it is beneficial to create delay-bound slices if low-latency users are active, and share those resources with the remaining users otherwise.

The recognition of the low-latency users necessitates a real-time analysis of the traffic of such users, which can be resource intensive when done frequently (see Section VII-D). We thus developed the "burst analysis" micro-SDK that composes monitoring and scheduling micro-SDKs: it analyzes traffic of users in a specific slice, and if an intermittent, bursty traffic pattern specific to low-latency traffic is recognized, it creates an SCN19 delay-bound slice for such user.

Fig. 10 compares SCN19 and NVS in terms of scheduling delays, used resources, and application round-trip time: SCN19 guarantees service providers short scheduling delays while using less resources, since only a part of a subframe is used when scheduling a slice. It thus strikes a balance between the multiplexing needs (here, a gain of 3.6%) of the infrastructure provider and the latency-sensitive service requirements of service providers. On the application level, this can mean up to 20 ms shorter round-trip time.

VIII. Lesson Learned

We found that modularity and flexibility principles are two enablers to implement a service-oriented architecture, especially for time-critical network functions such as scheduling and resource allocation: (1) by clearly identifiying responsibilities and, as a result, refactoring existing code, the scheduling functionality became both modular and efficient; and (2) the resulting flexibility allowed to seemlessly integrate multiple services including the switch between multiple slice and user scheduling algorithm implementations (dynamically). Furthermore, we hope that this work will inspire other researchers to equally contribute to open, available platforms.

IX. Conclusion

We presented the RAN service engine, a versatile system that allows to customize and extend the functionality of the RAN. The service engine presents RAN control endpoints for control plane behavior steering through platform-independent micro-SDKs which services use to customize and extend RAN functionality through containerized micro-services. We showed the definition of a synthesized base station at the level of the service engine and the corresponding definition of services, matching micro-services onto micro-SDKs. We furthermore presented key micro-SDKs such as MAC scheduling and monitoring. The former allows to change slicing and user scheduling algorithms, efficiently multiplexing services while retaining functional isolation between services, with no additional execution overhead. The latter incurs only small monitoring overhead. Thus, we demonstrated the feasibility of light-weight RAN service customizations, indicating that micro-SDKs are a promising way to achieve the RAN-as-a-service concept. The service-oriented base station approach coupled with micro-services allows verticals to easily define and integrate their RAN customizations, tailored to their requirements with fine-grained control over their service.

We believe that service-specific RAN controllers will be the next step in RAN slicing, since they allow service providers such as verticals to customize and optimize their network without the intervention of the infrastructure provider, and increase security by shielding services from each other. As has been shown, such a controller can be realized through a composition of dedicated micro-SDKs, and we consider implementing more micro-SDKs in our future work.

Acknowledgment

This work received funding from the European Union's Horizon 2020 Framework Programme under grant agreements No. 857201 (5G-VICTORI), No. 957317 (Affordable5G), and No. 101017226 (6G-BRAINS).

References

[1] IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond, International Telecommunications Union, ITU-R M.2083-0, September 2015.

[2] S. E. Elayoubi, S. B. Jemaa, Z. Altman and A. Galindo-Serrano, "5G RAN Slicing for Verticals: Enablers and Challenges," in *IEEE Communications Magazine*, vol. 57, no. 1, pp. 28–34, January 2019, DOI: 10.1109/MCOM.2018.1701319.

[3] P. Hedman (ed.), "Description of Network Slicing Concept", NGMN Alliance, v. 1.0, 2016. [Online] Available: https://www.ngmn.org/wp-content/uploads/160113_NGMN_Network_Slicing_v1_0.pdf

[4] D. Sabella *et al.*, "RAN as a service: Challenges of designing a flexible RAN architecture in a cloud-based heterogeneous mobile network," *2013 Future Network & Mobile Summit*, Lisboa, 2013, pp. 1–8.

[5] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture," in *Proc. of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom '17)*, New York, NY, USA, 2017, pp. 127–140, DOI: 10.1145/3117811.3117831.

[6] N. Nikaein *et al.*, OpenAirInterface: A Flexible Platform for 5G Research. *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, October 2014, pp. 33–38, DOI: 10.1145/2677046.2677053.

[7] N. Nikaein, C.-Y. Chang, and K. Alexandris, "Mosaic5G: agile and flexible service platforms for 5G research," *SIGCOMM Comput. Commun. Rev.* vol. 48, no. 3, pp. 29–34, July 2018, DOI: 10.1145/3276799.3276803.

[8] K. Katsalis *et al.*, "Network Slices toward 5G Communications: Slicing the LTE Network," in *IEEE Communications Magazine*, vol. 55, no. 8, pp. 146–154, August 2017, DOI: 10.1109/MCOM.2017.1600936.

[9] Navid Nikaein *et al.*, "Network Store: Exploring Slicing in Future 5G Networks," in *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture (MobiArch '15)*, New York, NY, USA, 2015, pp. 8–13, DOI: 10.1145/2795381.2795390.

[10] S. Sharma, R. Miller and A. Francini, "A Cloud-Native Approach to 5G Network Slicing," in *IEEE Communications Magazine*, vol. 55, no. 8, pp. 120–127, August 2017, DOI: 10.1109/MCOM.2017.1600942.

[11] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, "NVS: a virtualization substrate for WiMAX networks," in *Proc. of the sixteenth annual international conference on Mobile computing and networking (MobiCom '10)*, New York, NY, USA, 2010, pp. 233–244, DOI: 10.1145/1859995.1860023.

[12] A. Ksentini and N. Nikaein, "Toward Enforcing Network Slicing on RAN: Flexibility and Resources Abstraction," in *IEEE Communications Magazine*, vol. 55, no. 6, pp. 102–108, June 2017, DOI: 10.1109/MCOM.2017.1601119.

[13] D. Marabissi and R. Fantacci, "Highly Flexible RAN Slicing Approach to Manage Isolation, Priority, Efficiency," in *IEEE Access*, vol. 7, pp. 97130–97142, 2019, DOI: 10.1109/ACCESS.2019.2929732.

[14] P. Rost *et al.*, "Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks," in *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, May 2017, DOI: 10.1109/MCOM.2017.1600920.

[15] R. Ferrus, O. Sallent, J. Perez-Romero and R. Agusti, "On 5G Radio Access Network Slicing: Radio Interface Protocol Features and Configuration," in *IEEE Communications Magazine*, vol. 56, no. 5, pp. 184–192, May 2018, DOI: 10.1109/MCOM.2017.1700268.

[16] M. Shariat *et al.*, "A Flexible Network Architecture for 5G Systems," in *Wireless Communications and Mobile Computing*, vol. 2019, DOI: 10.1155/2019/5264012.

[17] C. Sexton, N. Marchetti and L. A. DaSilva, "Customization and Trade-offs in 5G RAN Slicing," in *IEEE Communications Magazine*, vol. 57, no. 4, pp. 116–122, April 2019, DOI: 10.1109/MCOM.2019.1800360.

[18] 5G NG-RAN Architecture description, 3GPP, TS 38.401, version 15.7.0, January 2020.

[19] O. Adamuz-Hinojosa, P. Muñoz, P. Ameigeiras and J. M. Lopez-Soler, "Sharing gNB components in RAN slicing: A perspective from 3GPP/NFV standards," *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, Granada, Spain, 2019, pp. 1–7, DOI: 10.1109/CSCN.2019.8931318.

[20] O. Adamuz-Hinojosa *et al.*, "Harmonizing 3GPP and NFV Description Models: Providing Customized RAN Slices in 5G Networks," in *IEEE Vehicular Technology Magazine*, vol. 14, no. 4, pp. 64–75, December 2019, DOI: 10.1109/MVT.2019.2936168.

[21] A. Gudipati, D. Perry, L. E. Li and S. Katti, "SoftRAN: software defined radio access network," in *Proc. of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN '13)*, New York, NY, USA, 2013, pp. 25–30, DOI: 10.1145/2491185.2491207.

[22] T. Chen, H. Zhang, X. Chen and O. Tirkkonen, "SoftMobile: control evolution for future heterogeneous mobile networks," in *IEEE Wireless Communications*, vol. 21, no. 6, pp. 70–78, December 2014, DOI: 10.1109/MWC.2014.7000974.

[23] I. F. Akyildiz, P. Wang, Shih-Chun Lin, "SoftAir: A software defined networking architecture for 5G wireless systems," in *Computer Networks*, vol. 85, pp. 1–18, 2015, DOI: 10.1016/j.comnet.2015.05.007.

[24] X. Foukas *et al.*, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks," in *Proc. of the 12th International on Conference on emerging Networking EXperiments and Technologies (CoNEXT '16)*, New York, NY, USA, 2016, pp. 427–441, DOI: 10.1145/2999572.2999599.

[25] R. Schmidt, C. Chang and N. Nikaein, "FlexVRAN: A Flexible Controller for Virtualized RAN Over Heterogeneous Deployments," *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, 2019, pp. 1–7, DOI: 10.1109/ICC.2019.8761222.

[26] Near-Real-time RAN Intelligent ControllerNear-RT RIC Architecture, O-RAN Alliance, O-RAN.WG3.RICARCH-v01.00, February 2020.

[27] C. Chang and N. Nikaein, "RAN Runtime Slicing System for Flexible and Dynamic Service Execution Environment," in *IEEE Access*, vol. 6, pp. 34018–34042, 2018, DOI: 10.1109/ACCESS.2018.2847610.

[28] E. Dahlmann, S. Parkvall, and J. Sköld, "Radio Interface Architecture,", in 5G NR: the next generation wireless access technology, 1st ed., London, United Kingdom: Academic Press, 2018 pp. 73–102.

[29] F1 general aspects and principles, 3GPP, TS 38.470, version 15.7.0, January 2020.

[30] Common Public Radio Interface: eCPRI Interface Specification, eCPRI Specification, V1.2, June 2018.

[31] O-RAN Fronthaul Cooperative Transport Interface Transport Control Plane Specification, O-RAN Alliance, O-RAN.WG4.CTI-TCP.0-v01.00, April 2020.

[32] Z. Zaidi, V. Friderikos, Z. Yousaf, S. Fletcher, M. Dohler and H. Aghvami, "Will SDN Be Part of 5G?," in *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3220–3258, Fourthquarter 2018, DOI: 10.1109/COMST.2018.2836315.

[33] Robert Schmidt and Navid Nikaein, “Service-oriented intelligent and extensible RAN”, presented at the *26th Annual International Conference on Mobile Computing and Networking (MobiCom ’20)* (virtual conference), London, United Kingdom, 2020, pp. 1–3, DOI: 10.1145/3372224.3417321.

[34] M. Irazabal, E. Lopez-Aguilera and I. Demirkol, “Active Queue Management as Quality of Service Enabler for 5G Networks,” *2019 European Conference on Networks and Communications (EuCNC)*, Valencia, Spain, 2019, pp. 421–426, DOI: 10.1109/EuCNC.2019.8802027.

[35] FAPI and nFAPI specifications, Small Cell Forum, SCF 082, version 9.0, May 2017.

[36] R. Schmidt, C. Chang and N. Nikaein, “Slice Scheduling with QoS-Guarantee Towards 5G,” *2019 IEEE Global Communications Conference (GLOBECOM)*, Waikoloa, HI, USA, 2019, pp. 1–7, DOI: 10.1109/GLOBECOM38437.2019.9013258.

[37] R. Schmidt and N. Nikaein, “Demo: Efficient Multi-Service RAN Slice Management and Orchestration”, presented at the *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium* (virtual conference), Budapest, Hungary, 2020, pp. 1–2, DOI: 10.1109/NOMS47738.2020.9110253.

[38] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia and P. Camarda, “Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey,” in *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 678–700, Second Quarter 2013, DOI: 10.1109/SURV.2012.060912.00100.

Robert Schmidt obtained a diploma with distinction in Information Systems Engineering from Dresden University of Technology, Germany, and a diploma in Engineering from Ecole Centrale Paris/CentraleSupélec, France, in 2017. He currently pursues a Ph.D. in communications within the Communication Systems Department of Eurecom, France. Robert is involved in collaborative research projects in the context of the EU H2020 framework programme and an active contributor to the OpenAirInterface and Mosaic5G projects. His main research interests include 4G and 5G wireless cellular networks, heterogeneous software-defined (radio access) networks, network slicing and MAC layer scheduling.

Navid Nikaein is a Professor in Communication System Department at Eurecom. He received his Ph.D. degree in communication systems from the Swiss Federal Institute of Technology EPFL in 2003. Broadly, his research contributions are in the areas of experimental 4G-5G system research related to radio access, edge, and core networks with a blend of communication, computing, and data analysis with a particular focus on industry-driven use-cases. He is a board member of OpenAirInterface.org software alliance as well as the founder of the Mosaic-5G.io initiative whose goal is to provide software-based 4G/5G service delivery platforms.