EURECOM
Department of Communication Systems
Campus SophiaTech
CS 50193
06904 Sophia Antipolis cedex
FRANCE

Research Report RR-17-335

RAN Slicing Runtime System for Flexible and Dynamic Service Execution Environment

October 18th, 2017

Chia-Yu Chang and Navid Nikaein

Tel : (+33) 4 93 00 81 00
Fax : (+33) 4 93 00 82 00
Email : {Firstname.Lastname}@eurecom.fr

[1]EURECOM's research is partially supported by its industrial members: BMW Group Research and Technology, IABG, Monaco Telecom, Orange, Principaut de Monaco, SAP, ST Microelectronics, Symantec.

RAN Slicing Runtime System for Flexible and Dynamic Service Execution Environment

Chia-Yu Chang and Navid Nikaein

Abstract

Network slicing is one the key enabler to provide the required flexibility and to realize the service-oriented 5G vision. Unlike the core network slicing, RAN slicing is still at its infancy and several works just start to investigate the challenges and potentials to enable a mutli-tenant and multi-service RAN, toward a serviced-oriented RAN (SO-RAN) architecture. One of the major concerns in RAN slicing is to provide different levels of resource isolation and sharing as per slice requirements. Moreover, the control and user plane processing may be customized allowing a slice owner to flexibly control its service. Enabling dynamic RAN composition with flexible functional split for disaggregated RAN deployments is yet another challenge. In this paper, we propose a RAN slicing runtime system through which the operation and behavior of the underlying RAN could be customized and controlled to meet slice requirements. We present a proof-of-concept prototype of the proposed runtime system for LTE system, assess its feasibility and potentials, and demonstrate the isolation, sharing, and customization capabilities with three use cases.

Index Terms

Network slicing, RAN slicing, 5G, service orientation

Contents

List of Figures

1 Introduction

Fifth generation (5G) mobile networks is a paradigm shift beyond a new radio and spectrum with the objective of improving overall efficiency and flexibility of mobile networks. It is about evolution of computing for wireless networks (e.g. central offices become data-centers) and enabling service-oriented architecture to deliver networks on an *as-a-service* basis. Support of vertical markets is one of the main driving forces behind this evolution to empower 5G business and value creation. The underlying idea being to support multiple services and/or virtual networks on a single physical network infrastructure with different requirements is in terms of service definition and agreement, control and management, and performance. Through this service-oriented 5G vision, naturally the network infrastructure providers (e.g. operators and data-center owners), service providers (e.g. operators and verticals), and network function providers (e.g. vendors) are decoupled to allow an cost-effective network composition and sharing model to reduce both capital expenditure (CAPEX) and operating expense (OPEX). Fig. 1 illustrates the relationship between different providers and the transformation of value-chain in telecommunication industry. For example, network infrastructure may be provided by the operator as an intermediary between the vendors and data center owners or by a combination of network equipments from vendors, data centers from ITs, and transport network from operators. A service is built through a composition of *multi-vendor network functions*, physical or virtual (PNF/VNF), that not only shall meet the requirements of service providers such as performance and cost but also that of network infrastructure providers in terms of PNF/VNF interoperability and compatibility when the service is running on the infrastructure.

Network slicing is one of the key enabler to provide the required flexibility for the envisioned service-oriented 5G. It enables the composition and deployment of multiple logical networks over a shared physical infrastructure, and their delivery as a service or slice. A slice can either be completely *isolated* from the other slices down to the different sets of spectrum and cell site (as in most of current 3G/4G deployment), or be *shared* across all types of resources including radio spectrum and network functions (e.g. all layers of protocol stack), or be *customized* for a subset of data user-plane (UP) and control-plane (CP) processing with an access to a portion of radio resources in a virtualized form. In addition, a slice may span across *domain-specific resources* each with different levels of isolation and sharing with the objective of accommodating both service and infrastructure providers. Here, domain boundaries could be administrative (e.g. an operator), network segment (e.g. radio access network), technology (e.g. 4G/5G) among the others, and resources could be of different types including computing, storage, network, hardware, radio, spectrum, and network functions. To this end, *softwarization*, *virtualization*, and *disaggregation* are key slicing enablers to flexibly customize a slice, automate its life-cycle management, and ease the development of network functions and applications with the objective to accommodate the requirement of an E2E service. They constitute the foundation for a multi-service and multi-tenant

architecture, and are realized by applying software-define networking (SDN), network function virtualization (NFV), and cloud computing principles to mobile networks [1].

Several standardization bodies and organizations outline the crucial role of E2E network slicing to fulfill the *service-oriented* visions of 5G, e.g., ITU [2], 3GPP [3] and NGMN [4]. Also, prominent network architectures are proposed by 5G initiatives and projects, e.g., 5GPPP European program [5]. Many architectures and prototypes leveraging cloud computing, SDN, and NFV principles have been proposed for core network (CN) slicing [6–8] and radio access network (RAN) slicing [9–11]. The challenge of CN slicing has been also addressed by 3GPP, and realized through a dedicated core network (DECOR) [12] and evolved DECOR [13]. Nevertheless, RAN slicing remains a challenge in *providing different levels of isolation and sharing to allow a slice owner to customize its service across UP, CP, and the control logics (CL) while increasing the resource utilization of RAN infrastructure*. Note that the CL refers to the logic that makes the decisions for a particular CP/UP function, e.g., CL decides on user handover and CP performs the corresponding handover action.

To this end, the proposed RAN slicing runtime system provides following contributions:

- Review the state-of-the-art on network slicing architecture with the particular focus on RAN slicing (Section 2);

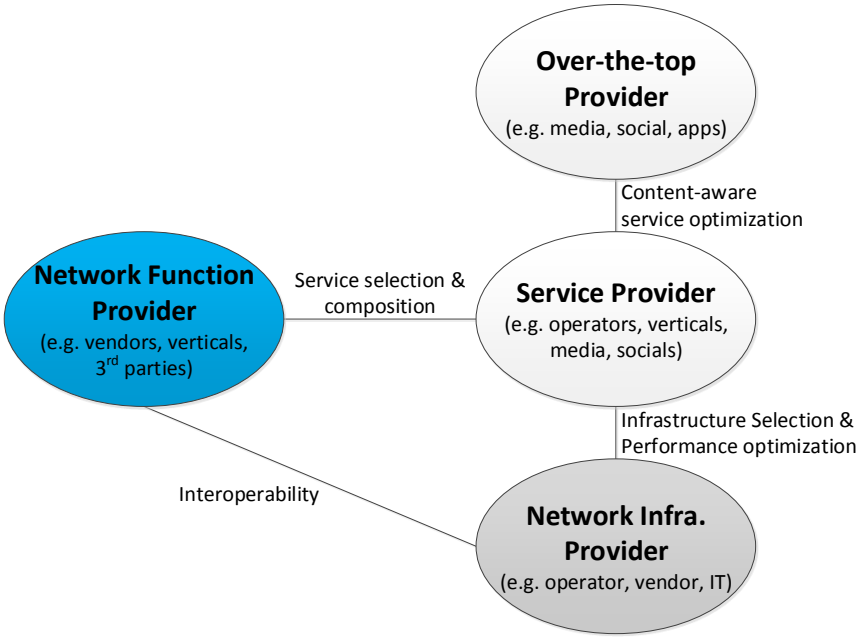


Figure 1: Impact of service-oriented architecture on telecommunication industry.

- Present a RAN slicing design in form of runtime system to enable different levels of isolation and sharing among slices accessing the underlying RAN modules and resources while allowing flexible service composition and customization across UP, CP, and CL (Section 3 and 4);
- Propose and evaluate a practical set of radio resource abstractions and inter-slice resource partition approach (Section 5);
- Build a concrete RAN slicing runtime system prototype on the top of OpenAirInterface (OAI) [14] and FlexRAN [15] platform and then characterize its performance through three case studies (Section 6).

2 Related work

The network slicing architecture has been surveyed widely and such concept can be traced back to the idea of *network sharing* like the gateway core network (GWCN) defined by 3GPP via sharing RAN and parts of CN. Additional sharing models are possible and summarized in [16, 17]. In [9], a slice-based network architecture is proposed with the "Network store" concept as a platform to facilitate dynamic network slicing based on the virtualized network functions (VNFs) on top of commodity infrastructures. The same idea is extended in [18] featuring the "Network and application store" that simplifies the procedure to define each slice. In [19], the modularized architecture is presented that is composed of several building blocks each with various sub-functions to customize functionalities on per service of slice. The network slice broker of resources is investigated in [20] enabling the on-demand multi-tenant slice resource allocation. The generic slice as a service model is presented in [21, 22] aiming to orchestrate customized network slice as a service with mapped network functions from service level agreement (SLA). A cloud-native network slicing approach presented in [23] allows to devise network architectures and deployments tailored to the needs of service. In [24], an E2E overarching architecture converged from optical to wireless is outlined to enable the cross-domain slicing.

In terms of the RAN slicing, the first approach is stemmed from the *RAN sharing* concept such as Multi-Operator RAN (MORAN) and Multi-Operator CN (MOCN). The MORAN approach can share the same RAN infrastructure but with dedicated frequency bands for each operator whereas MOCN allows to also share the spectrum among operators as standardized in LTE Release 8 [25]. These approaches can efficiently utilize available radio resources which are surveyed widely as network virtualization substrate (NVS) in several works [26–28] that aims to virtualize radio resources for different resource provisioning approaches to allow several mobile virtual network operators (MVNOs) to coexist in a single physical RAN. Authors of [29] propose application-oriented RAN resource sharing framework with Quality of Service (QoS) guarantee. On a more general basis, RAN virtualization [30, 31] provides functional isolation in terms of customized and

dedicated control plane functionalities for each MVNO. The above works consider either radio resource sharing or functional isolation whereas little attentions are given to be simultaneously satisfy both concerns.

To enable the network slicing concept in RAN, several 5G RAN design requirements and paradigms shall be fulfilled as elaborated in [32]. Future RAN design patterns are explained in [33] along aspects of cloud computing, SDN/NFV and software engineering. Moreover, 3GPP mentions RAN slicing realization principles in [34, 35] such as RAN awareness slicing, QoS support, resource isolation, SLA enforcement among the others. These principles can be enabled through the software-defined RAN (SD-RAN) concept that decouples CP and UP. Several works argue the level of centralization of CP functionalities. The fully centralized architecture is proposed in OpenRAN [36] and SoftAir [37] that will face the challenge of real-time control under the inherent delay between the controller and RAN. The SoftRAN [38] architecture statically refactors the control functions into centralized and distributed ones based on the time criticality and central view requirement. The SoftMobile approach [39] further abstracts the CP in layers based on the functionalities to form the network graphs and performs control functionalities through Application Programming Interface (API). As for the UP programmability and modularity, the OpenRadio [40] and PRAN [41] are pioneered to decompose the overall processing into several functionalities that can be chained. FlexRAN realizes a SD-RAN platform and implements a custom RAN southbound API through which programmable CL can be enforced with different levels of centralization, either by the controller or RAN agent. Several on-going 5GPPP projects are also working on the architecture design and slicing of SD-RAN, for instance, METIS-II, 5G-NORMA, COHERENT, 5G-MoNArch, 5G-Picture, and SliceNET.

With aforementioned enablers, several RAN slicing works are initiated. The blueprint proposed as RadioVisor [42] aims to isolate the control channel messages, elementary resources like CPU and radio resource to provide customized service for each slice. A fully isolation platform is provided in [43] with virtual base stations (BSs) as different slices; however, there is no multiplexing benefits in the radio resource allocation since the spectrum is disjointly partitioned. In addition, network function sharing and multiplexing are not considered in such work. In [44], the radio resource scheduling is separated into intra-slice and inter-slice scheduler; however, the resource abstraction/virtualization is not included and the inter-slice scheduler only bases on traffic flows according to the required QoS. Hence, it does share a portion of functions but without considering proper resource isolation. In [10], a RAN slicing architecture is proposed that allows radio resource management (RRM) policies to be enforced at the level of physical resource blocks (PRBs) through providing the virtualized resource blocks (vRBs) by a novel resource visor toward each slice. Such work provides certain level of isolation and sharing among resources while does not consider function isolation. Authors of [11] introduce the idea of BS hypervisor to simultaneously isolate slice-specific control logics and share the radio resources. The hypervisor groups the underly-

Table 1: RAN slicing state-of-the-arts comparison

State-of-the-arts	Radio resource sharing	CP function	UP function
Network store [9]	-	Dedicated	Dedicated
NVS [26]	Physical or virtualied resource sharing	-	-
Yasir et al. [30]	Physical resource sharing	Dedicated	Dedicated
FlexRAN [15]	Physical or virtualied resource sharing	Shared	Shared
Radiovisior [42]	Dedicated 3D resource grid allocation	Dedicated	Dedicated till programmable radio
Nakao et al. [43]	Dedicated spectrum allocation	Dedicated	Dedicated
Rost et al. [44]	Physical resource sharing	Split into cell and user-specific	Dedicated till real-time RLC
Ksentini et al. [10]	Flexible between dedication and sharing	Dedicated	Shared
Hypervisor [11]	Virtualzed resource sharing	Split into cell and user-specific	Dedicated till PHY layer

ing PRBs into vRBs through a set of abstractions and provides only relevant user information to the corresponding slice. Such work exploits the prerequisites of function isolation and resource virtualization while it does not consider customization and multiplexing of CP/UP functions in both monolithic and disaggregated RAN deployment.

TABLE 1 compares relevant related works in three dimensions: radio resource sharing model, control plane function, and user plane function. To serve various flavors of slice, the flexibility and proactiveness of these three dimensions shall be achieved through RAN slicing. In that sense, we envision the runtime system that leverages and extends the hypervisor approach [11] aiming to support various slice requirements (e.g. isolation) and elastically improve multiplexing benefits (e.g. sharing) in terms of (1) the new set of radio resource abstractions, (2) network service composition/customization for modularized RAN, and (3) flexibly adaptability to different deployment scenarios ranging from monolithic to disaggregated.

3 RAN Slicing Runtime System

We propose a RAN slicing runtime system that provides a flexible *execution environment* to run multiple virtualized RAN instances with the required level of isolation and sharing of the underlying RAN modules and resources. It allows slice owners to (a) create and manage their slices, (b) perform their custom control logics (e.g. handover decision) and/or custom UP/CP processing (e.g. PDCP and RRC functions), and (c) operate on a set of virtual resources (e.g. resource block or spectrum) or capacity (e.g. rate) and accessing to their CP/UP state (e.g. user identity) that are revealed by the RAN runtime. The isolation and customization properties provided by runtime is in favor of the slice owners allowing them to control the slice compositions and the behavior of the underlying RAN module as per service requirements, whereas the sharing is in favor of infrastructure provider allowing to efficiently and dynamically *multiplex* multiple tenants over resources, processing, and states of common RAN module to reduce the expenditures. Here, RAN module refers to a unit that comprises a subset of RAN functions and performs a

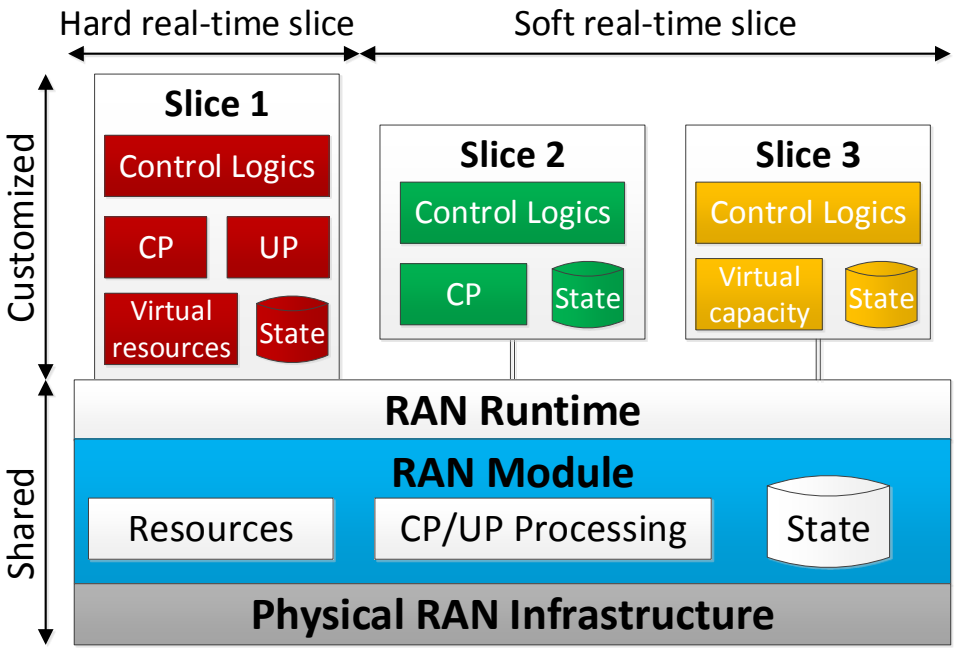


Figure 2: High-level architecture of RAN slicing runtime system

portion of RAN processing. 3GPPP defines 3 types of unit, namely (remote) radio unit (RRU/RU), distributed unit (DU), and centralized unit (CU) [45].

The proposed RAN slicing runtime system is shown in Fig. 2, with the runtime being the core component by which a running slice interacts with the RAN modules to *access resources and state, and control the underlying behavior*. From the slice owner perspective, the runtime provides an execution environment through which a slice can perform customized processing, request resources, and access state. At the same time, it enables infrastructure provider to manage the underlying RAN module, enforce slice-specific policies, and perform access and admission control. The runtime by itself is in charge of managing the life-cycle of slices, abstracting the radio resources, share states, and applying changes into the underlying RAN module to customize slices. It also implements a set of runtime APIs to enable bidirectional interactions between different slices and RAN module to monitor or control the underlying CP/UP processing, resources, and states while retaining the isolation among slices.

A slice is formally represented to the runtime by a slice descriptor that defines the slice service requirements in terms of resources, custom processing, and performance. It is generally provided by the service orchestrator during the creation of a slice, and indicates for each slice how radio resources are *allocated, reserved, preempted, or shared* by the runtime, how the processing is *pipelined*, and what are the average expected throughput and latency. The customization feature provided by the runtime allows a slice owner to only contain a portion of resources and processing within the slice boundary and multiplex the remaining into the un-

derlying RAN module. To realize a flexible tradeoff between isolation and sharing, the state of CP and UP processing is maintained in a database[1] allowing to update the processing pipeline (e.g. from customized to multiplexed or vice versa) on-the-fly while retaining the service continuity and isolation on the input/output data streams. Note that by maintaining the state, the network functions are virtually turned into stateless processing allowing to update the service and recover the state through the runtime. In addition, the overall CP processing of a BS are logically separated into slice-specific and BS-common functions to increase the multiplexed processing. Note that here we consider per function CP separation, for instance, the masterinformationblock (MIB) and systeminformationblocks (SIBs) are broadcasted commonly to all users hence are categorized into BS-common ones, whereas the random access resources could be customized by a slice so as to reduce the latency due to the common random access procedure.

In summary, in the proposed RAN slicing model, RAN functions are pipelined to compose the desired RAN module, i.e., monolithic or disaggregated RAN instances, either via multiplexed or customized CP and UP functions as per slice requirements. The runtime system acts as the intermediate between customized slices and underlying shared RAN module and infrastructure providing a unified execution environment with substantial flexibility to achieve the required level of isolation and sharing.

4 Design Elements of Runtime

This section details the main components of the RAN slicing runtime systems, namely runtime data, services, and APIs.

4.1 Design Challenge

Based on the proposed runtime RAN architecture, we identify a number of challenges that runtime should resolve:

- Allow each slice to interact with the underlying RAN and change the CP and UP behavior that dynamically determined during its execution (section 4.2 and 4.3).
- Provide different levels of resource isolation and sharing to allow a slice owner to flexibly compose slice-specific RAN resources and processing from multiplexed and customized resources and CP/UP functions while maximizing the multiplexing gain of the underlying RAN resources and modules (section 4.3).
- Provide APIs to enable the slice-specific CP, UP and control decisions to be realized for both soft and hard real-time slices (section 4.4).

[1] This is regardless of whether a network function is stateful or stateless [46, 47].

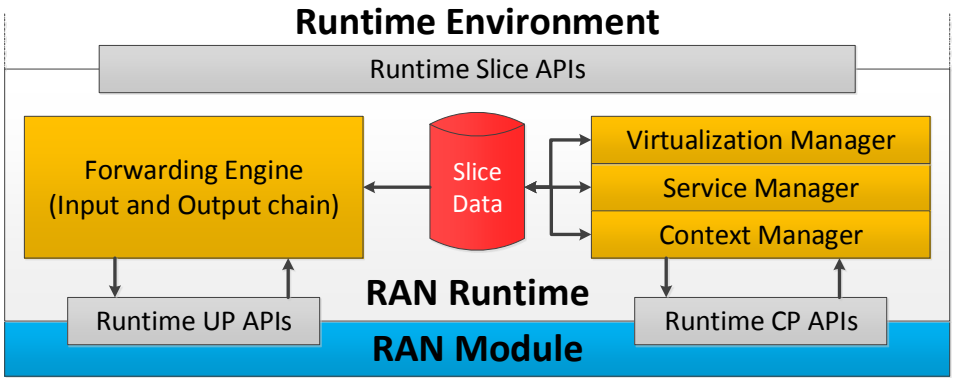


Figure 3: Architecture of the runtime system.

Table 2: Slice context maintained by runtime system.

Slice Context	Description
Slice identity	Represents a unique slice identifier
Service registry identity	Identifies to which runtime services a slice is registered to, e.g., service manager
Slice SLA and policy	Describes a business agreement between slice owner and infrastructure provider in terms of performance, resource, access control, and priority of a corresponding slice
Customized processing	Specifies the in-slice custom CP/UP processing functions If not specified explicitly, the default pipelined processing are applied to this slice.
User context	Identifies which pair of BSs and slices a user belongs to and also the mapping between traffic flow and dedicated radio bearers (DRBs)[2]

Fig. 3 illustrates the main building blocks of runtime that consist of: (a) slice data, (b) CP and UP functions to provide runtime services, and (c) API, that are described in following subsections.

4.2 Runtime slice data

Slice data is the entity that stores both *slice context* and *module context* under the control of context manager. They are used to customize and manage a slice in terms of required runtime services, resources, processing, state, and users.

The *slice context* describes the basic information and prerequisites to instantiate a slice service and manage users. It is provided by the slice orchestrator and can be updated by the corresponding slice (cf. Fig. 2) following the agreement between slice owner and infrastructure provider. TABLE 2 describes the slice context information maintained by the runtime in the slice data.

The *module context* includes CP and UP state information (belonging to slice owners), module life-cycle management primitives such as start, configure, and stop service (belonging to network function provider), and resources (belonging to infrastructure provider). Unlike input or output data streams of the RAN module

[2]The 1:n:m relation of user-to-slice-to-BS mapping will make use of runtime CP APIs for network slice selection operation.

Table 3: BS-common and user-specific functions

Process	BS-common functions	user-specific functions
Location tracking and Paging	Tracking area update, CN paging	RAN Paging
Handover and cell reselect	Cell (re-)selection criterion	User measurement configuration, Handover
Random access	Common random access	Dedicated random access
User attach procedures	-	Slice-based user association control
QoS maintenance and admission control	-	QoS flow maintenance and slice-based admission control
Security function	Common BS key management	Slice-specific CP/UP key management
Bearer management	Signaling radio bearer maintenance	Dedicated radio bearer management
Radio resource allocation	Common cell signal, e.g., CRS, PSS, SSS	Per-slice dedicated resource reservation
System information	Broadcast NAS, MIB, SIB information	-

that are pipelined, the control-data state is maintained separately by the runtime and revealed to each slice in real-time to allow efficient slice-specific processing. In addition, such state may be shared among multiple slices subject to access control, for instance when coordinated processing and/or decision making are required as in case of handover decision of a user belonging to two slices. Note that in general case, state only includes user-specific functions in CONNECTED (or INACTICE-CONNECTED [48]) mode, and not necessarily the BS-common functions that are executed independent of the number of instantiated slice even with no instantiated slice or when operating in IDLE mode (cf. TABLE 3).

4.3 Runtime services

4.3.1 Context Manager

This service is managing both slice and module context by performing CRUD[3] operation on the slice data. To create a slice context, the context manager firstly performs slice admission control based on the provided slice description that defines the required processing, resources, and states (as agreed between the slice owner and infrastructure provider). Upon admission control, module context is used by the context manager to register slice-specific life-cycle primitives to the service manager and the requested resources and/or performance to the virtualization manager. The former allows custom CP/UP processing to be applied on the input/output data streams, whereas the latter enables resource partitioning and abstraction to be performed among multiple slices. At this stage, a slice starts to consume the runtime services not only to manage its service but also to interact with the underlying RAN module through runtime CP/UP APIs, and the context manager will handle real-time CP/UP state information within the slices and the underlying RAN module so as to keep the slice data in-sync with the instantaneous state.

[3]CRUD includes four basic operations: create, read, update, and delete.

4.3.2 Service Manager

The service manager is the entity responsible for managing the life-cycle of a slice when instructed by the slice owner or service orchestrator. Through service manager, slice life-cycle operations can be triggered, which in turn enables both slice owner and infrastructure provider to control and update slice service definition as per need and agreement. Based on the service definition and slice context, the service manager determines the CP/UP processing chain for each slice and each traffic flow, and programs the forwarding engine through a set of rules allowing to direct the input and output streams across multiplexed processing operated by the underlying RAN module and customized processing performed by the slice. Unlike the context manager that handles the local slice context, service manager operates on an E2E RAN service in support of service continuity when slice service definition is changed. For example, a slice owner that performs customized UP processing can opt in for a multiplexed pipelined processing to reduce its OPEX, which cause changes in slice service definition. In addition, when the slice requirements are violated (e.g. performance degradation), the service manager may change the requested resources, allocation type, resource partitioning period, or even update the service definition to comply with the service requirements. Service manager is also in charge of taking a set of actions when detecting conflicts among multiple slices based on a set of policy rules. Such conflict can happen at the level of slice when service definition is changed or at the level of users when they belong to multiple slices (1:n or m:n relationships). For instance, reserving the resources and/or changing the allocation type of a slice may violate the performance of another slice that requires high bandwidth. Another example is when different user mobility measurements are requested by different slices that requires a coordination to reconfigure the measurement with the largest common parameters and least denominator. To this end, context manager relies on a set of policy rules to decide whether to preempt one slice, reject another slice, or multiplex the request.

4.3.3 Virtualization Manager

This service is in charge of providing the required level of isolation and sharing to each slice. It performs *partitioning* on resources and states based on the slice and module contexts, *abstraction* of physical resources and states to/from the virtualized ones, and revealing virtual views to a slice that is customized and decoupled form the exact physical resources and states. In the following paragraphs, we omit state partitioning and abstraction as they can be realized through some well-known approaches such as database partitioning and control access.

4.3.3.1 Inter-slice resource partitioning Resource partitioning is a periodic process that happens every allocation window of T [11, 26]. It allows to distribute resources among multiple slices based on the requirements expressed in the

Table 4: Mapping between resource abstraction and allocation type.

Requested resources	Abstraction types (Resource granularity)	DL Resource allocation type
Resource block	vRBG Type 0 (Non-contiguous)	Type 0, Type 1
	vRBG Type 1 (Contiguous)	Type 0, Type 2
	vRBG Type 2 (Fixed position allocation)	Type 2 localized
Capacity	vTBS Type 0 (RBGs with min granularity)	All Types

slice context and stored in slice data. Radio resource descriptor has two elements: (1) resources type that defines whether the requested resources are of type physical/virtual radio resources in time or in frequency domain (this could be extended to component carrier and space), or capacity in terms of rate, and (2) abstraction type that maps physical radio resource allocation types, namely fixed position, contiguous, non-contiguous, or minimum resource block groups (RBG), to virtual RBGs (vRBG) or virtual transport block size (vTBS). In addition, different transmission time interval (TTI) and sub-carrier spacing (SCS) can be applied depending on the deployed frequency band and/or maximum user mobility to mitigate the impact of the non-idealities (e.g. Doppler shift) for each slice.[4]

Besides aforementioned radio resource requirement provided by the slice owner, the resource allocation shall also respect the policy defined by the infrastructure provider, for instance, the allowable resource allocation type of underlying radio access technologies (RATs). Take the DL resource allocation of LTE system for instance[5], there are three types of resource allocation: (i) Type 0 allocation is based on the minimum granularity as resource block group (RBG) that comprises multiple RBs, (ii) Type 1 categorizes RBGs into different subsets and only allocate RBs within the same subsets, and (iii) Type 2 allocates contiguous virtual RBs (vRBs) that can be physically contiguous (localized vRB) or non-contiguous (distributed vRB). Then, four types of abstraction are introduced with RBG as the minimum resource granularity, and their respective mapping to the DL resource allocation type is shown in TABLE 4. It can be seen that the proposed vRBG and vTBS are a superset of legacy resource allocation types and provide the required flexibility not only for intra-slice resource allocation, but also inter-slice resource partitioning as it will be detailed in Section 5. Fig. 4 illustrates an example of resource partitioning among four slices over an allocation window of T with different types of abstraction. The proposed abstraction allows RUNTIME to dynamically change the mapping to resource allocation type, for instance changing allocation type 0/1 to allocation type 2 in slice 3 and 4. Additional level of flexibility is achieved when unused resources can be further multiplexed to increase the resource utilization (see Section 5).

[4]This defines the allowed TTI and numerologies in correspondence with the SCS [49].

[5]Note that the UL resource allocation of LTE is a subset of DL resource allocation.

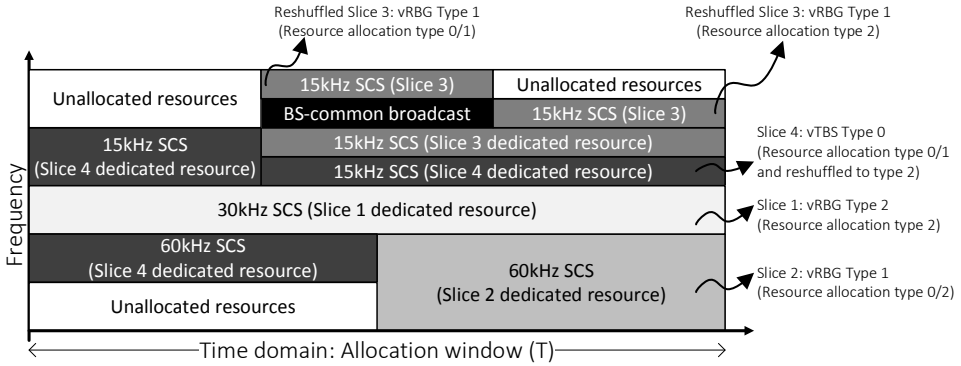


Figure 4: Radio resource partition with different types of abstraction

4.3.3.2 Radio resource abstraction Resource abstraction serves for two main purpose: (1) isolate resources by presenting a virtual view of the resources that is decoupled from its exact physical location, and (b) increase multiplexing gain by adjusting allocation types to share unused resources. The former simplifies the intra-slice resource scheduling operation and prevents other slices to access or even infer the resources allocated to others (in favor of slice owner), and the latter allows to maximize the resource efficiency and utilization (in favor of infrastructure provider). Take the 3 MHz case of LTE system as an example in Fig. 5, where the total PRB is 15 PRBs, PRBG granularity is 2 PRBs, giving a total of 8 PRBGs partitioned among 4 slices. These PRBGs are firstly aggregated from PRBs and then partitioned for each slice based on the number of required resources and the type of allocation (e.g. fixed, contiguous, non-contiguous). Then they are virtualized into vRBGs and allocated based on the abstraction type. For instance, fixed position resources is requested by slice 1 and hence no virtualization is performed (i.e. PRBG). In contrast, slice 4 only requests a capacity, and thus its PRBGs are abstracted to vTBS with the capacity value. PRBGS of slice 2 and 3 are virtualized into vRBGs via abstracting the exact frequency/time locations and dimensions; and are pooled together to maintain their relative frequency dependencies among virtualized resources but without revealing their absolute physical frequency dependencies. Take the slice 3 that uses resource allocation type 1 as an example, only PRBGs within the same subset can be scheduled at the same time. In that sense, vRBGs are *pooled* to indicate an exclusive condition between vRBG pool 1 (i.e. PRBG0, PRBG6) and vRBG pool2 (i.e. PRBG 5). Hence, the intra-slice scheduler of slice 3 will allocate resources to each user from either vRBG pool 1 or pool 2.

4.3.3.3 Radio resource accommodation and multiplexing After the radio resource partitioning and virtualization, each slice can perform intra-slice resource scheduling to users and the runtime will map the scheduling decision into PRBs and allocate corresponding control channel elements (CCEs) to transport the con-

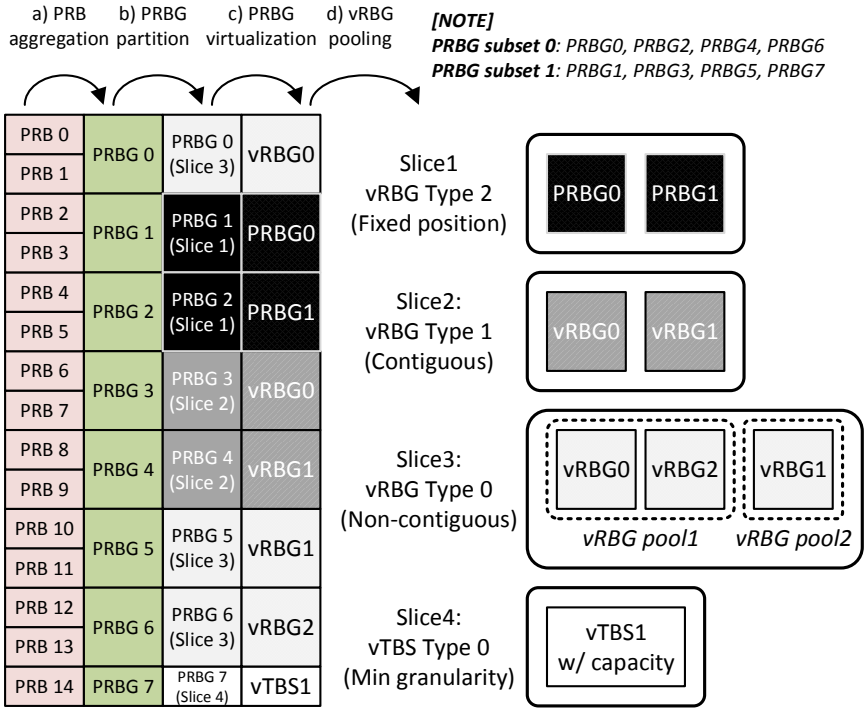


Figure 5: Different stages to form vRBG and vRBG pool

trol information (CI). Note these CIs are used to indicate the user about the positions of allocated PRB as well as necessary physical layer information (e.g. modulation and coding scheme (MCS), new data indication, HARQ process indication, etc.) for successful reception or transmission. With limited control region to accommodate CCEs[6], the runtime will leverage the unallocated resources after not only to carry the CI but also to increase the multiplexing gain in the data plane (cf. Fig. 4).

4.3.4 Runtime forwarding engine

The forwarding engine manages CP and UP input and output streams, or simply data streams, between RAN and users across multiplexed and customized processing. Fig. 6 shows an example of how the forwarding engine manages the UP processing chain in downlink (from RAN to users). Input flows of the RAN module for each slice are forwarded either to the customized (i.e. slice 1 and 2) or the multiplexed (i.e. slice 3) processing chain based on the rules applied by the service manager. After the first stage of the processing, the outputs flows are further forwarded to the corresponding entry points in the multiplexed chain or the output endpoint. Note that, more complex forwarding rules can be applied if per-function customization is required, for instance, customized MAC function to man-

[6]In LTE, up to 3 symbols are used as the control region in most cases.

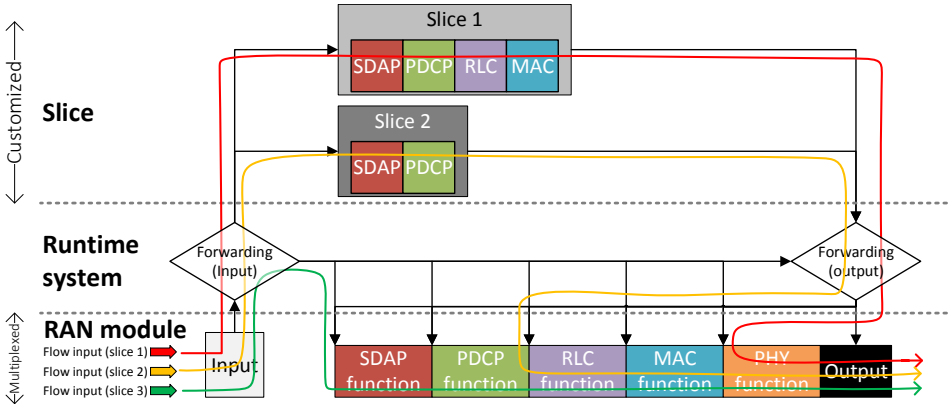


Figure 6: Forwarding engine and UP processing chain

age intra-slice scheduling while multiplexing other functions. Such forwarding engine leverages match-action abstraction following SDN principles to establish the input/output forwarding path between runtime and slice in both directions [50,51].

Furthermore, the forwarding engine not only is able to direct data in a monolithic RAN but also in a disaggregated RAN, where the RAN infrastructure is decomposed into CU, DU, and RU with several possible functional splits in between [34]. Note that in the proposed slicing model, RAN disaggregation and functional splits are controlled and maintained by the infrastructure provider, whereas RAN service customization is managed by the slice owner. Fig. 7 shows the input/output forwarding path between CU, DU, and RU to compose a distributed UP processing chain with 3GPPP function splits option 2 between CU and DU and option 6 between DU and RU. The input and output endpoints of RAN module will perform the infrastructure-dependent packet processing like encapsulation and switching/routing for fronthaul/midhaul transportation which is transparent to the slice owner. However, when adopting flexible function split and placement [52], the CP/UP state information has to be efficiently shared among disaggregated RAN infrastructures. TABLE 5 summarizes the main UP state information that must be maintained in the slice data. Also note these depicted chains are applied for the downlink direction but the same forwarding engine can be applied at the uplink with different processing chain composition.

4.4 Runtime APIs

The runtime APIs are exposed both in the north-bound towards slices and in the south-bound towards the underlying RAN module allowing to manage a slice and control the underlying RAN module. In the north-bound, the APIs provides interfaces and communication channels to connect a slice to the runtime as a separate process, whether it is local or remote, allowing to consume the services. Hence, each slice can be executed in isolation from each others either at host or guest level

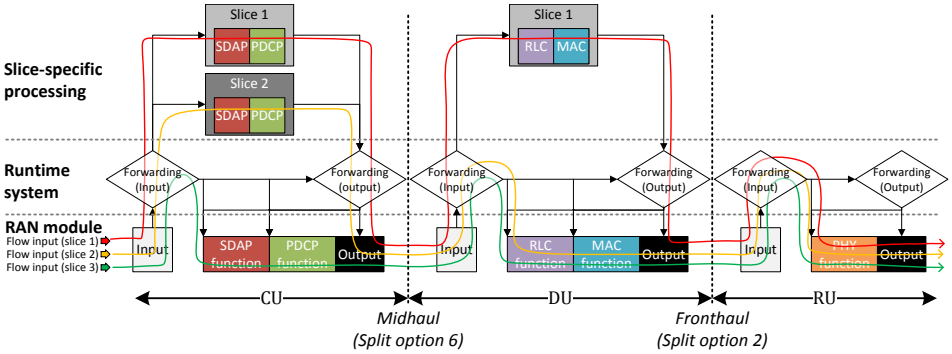


Figure 7: UP forwarding path in disaggregated RAN

Table 5: UP network functions and the decoupled states

Layer	Network function	Network state
PHY	RF processing	Carrier frequency, Spectrum bandwidth
	DFT/IDFT	Point of FFT, Output indexes
	Multi-antenna processing	Transmission mode, Beamforming matrix
	(De-)Modulation	Modulation order, Reference symbol information
	Bit-rate processing	Information of coding, Scrambling, Rate matching, CRC
MAC	HARQ process	HARQ index, User identity, Redundancy version
	(De-)Multiplexing	(De-)Mutltiplexed logic channel identities
	Dynamic scheduling and priority handling	Priorities between logic channels and users
RLC	ARQ error correction	Status report parameters, Polling information
	segmentation and reassemble	Size of corresponding PDU and SDUs
	SDU discard	Discard criterion, e.g., window information
PDCP	Header (de-)compression	Header compression profile, state and parameters
	Integrity protection/verification	Integrity protection algorithm and parameters
	(De-)Ciphering	Ciphering algorithm and parameters
	Reordering and duplicate detection	Sequence number of queued PDUs
SDAP	Mapping between QoS flow and DRB	QoS flow identity, QoS profile, mapping policy
	Marking QoS flow identity	QoS flow identity

leveraging well-know OS and virtualization technologies, such as container or virtual machine. The slice APIs allow a slice to register to the runtime services, manage its service in coordination with the runtime and high-level service orchestrator, and customize the CP/UP processing. In the south-bound, the runtime CP/UP APIs enables a slice to take control of its service by requesting resources, applying control decisions, and accessing states. When a slice is deployed locally, the runtime APIs exploits the inter-process communication mechanism to allow a slice to perform real-time operation (e.g. MAC function) with hard guarantees. Remote slices on the other hands, communicate with the runtime through the asynchronous communication interface and can perform non-time-critical operation (e.g. PDCP function).

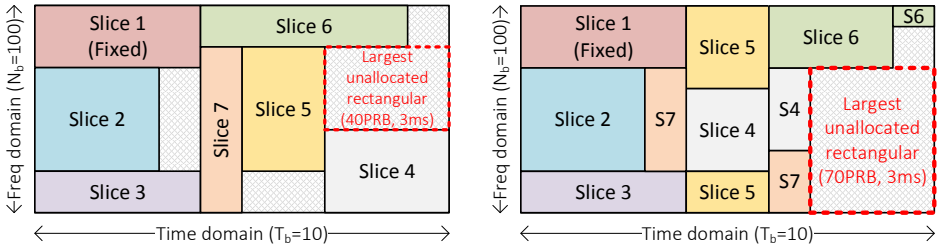
5 Inter-slice resource partitioning and accommodation

5.1 Inter-slice resource partitioning

The radio resources partitioned by the runtime in an allocation window time T (in millisecond) is denoted as N in the number of Hz of frequency domain. These resources can be specifically quantized into a resource grid with T_b TTIs in time domain and N_b PRBs in the frequency domain with respect to the base TTI (TTI_{base}) and base SCS (SCS_{base}) used by the infrastructure provider, e.g., a 20MHz LTE radio bandwidth in a 10ms allocation window is separated into 100 PRBs each with 180kHz bandwidth[7] and 10 TTIs each with 1ms duration. Then, there are $|S|$ slices requesting the radio resources as $\mathcal{S} = \{s_1, \cdots s_{|S|}\}$. For the i-th slice (i.e. s_i), its radio resource requirement includes: (a) SCS_i comprises the applicable SCSs, (b) T_i and N_i are the number of requested TTIs in time and PRBs in frequency domain, and (c) g_i is the granularity which can be contiguous, non-contiguous, fixed position (with its fixed position as $FixF_i$ and $FixT_i$) or minimum granularity as introduced in section 4. The fixed position can provide isolation and the guarantee for SLA but limit the possible allocation position. The contiguous granularity is more suitable for the constant traffic pattern (e.g. streaming) as it can reduce the latency due to contiguous allocation and minimize the overhead of signaling. The non-contiguous granularity, on the other hand, accommodates better variable/mix traffic pattern as it can combine fragmented resources into a flexible transport size. The minimum granularity is adopted to the slices that requests capacity (i.e. vTBS Type 0 abstraction) allowing any feasible allocations to be applied.

In following, an example of resource partition is provided in Fig. 8 with 7 slices (i.e. $|S| = 7$) and each slice has different requested granularities: $g_1 =$ Fixed, $g_2 = g_3 =$ Contiguous, $g_4 = g_5 =$ Non-contiguous, and $g_6 = g_7 =$ Minimum. The largest rectangular of unallocated resource is marked which is an important criterion for further resource multiplexing. Since such unallocated rectangular resource can potentially fit in any granularity of time (i.e. TTI) and frequency domain (i.e. SCS) and can be utilized for control information transportation, BS information broadcasting/multicasting, or sharing between slices. It can be observed from Fig. 8a and 8b that although both partitioning satisfy the required resources for all slices, only one achieves a larger unallocated rectangular region. Such resource packing is obtained by combining different levels of granularity when partitioning resources, e.g. s_4, s_5 are discontinuous in frequency and time separately, and both s_6 and s_7 use the minimum granularity. Hence, the inter-slice resource partition has two complementary goals: (a) satisfy as many requests as possible, and (b) maximize the size of unallocated rectangular region for further multiplexing. Such problem becomes a two-dimensional knapsack problem when the granularity of all requests is continuous, which makes the complexity NP-hard [53]. In that sense, finding an optimal inter-slice resource partition becomes inefficient and

[7] Remaining bandwidth is the guard band without transporting any information.



(a) Contiguous granularity (b) Slice-specific granularity

Figure 8: Examples of radio resource partitioning

we propose algorithms that can efficiently partition resources and provide close to optimal performances.

5.1.1 Algorithms

Our proposed approach is presented in Alg. 1 that firstly prioritizes slices according to their identities pID and then partitions resources based on their granularity, i.e., g_{pID}. As each slice can support more than one SCSs, the remapping from infrastructure base SCS (SCS_{base}) via SCSMap $(\cdot)$ function is used to derive the corresponding values of the target SCS for both requested resources ($RegF_{scs}$, $RegT_{scs}$) and fixed position ($FixF_{scs}$, $FixT_{scs}$). The proposed granularity-based partitioning includes four algorithms: fixed position (Alg. 2), contiguous (Alg. 3), non-contiguous (Alg. 4) and minimum granularity (Alg. 5). Finally, a resource grid re-mapping is done using RGReMap $(\cdot)$ function to map the grid of optimal SCS (i.e. $OptSCS$) to the ones of other SCSs.

When applying fixed position algorithm, the FindFixExact $(\cdot)$ function checks the feasibility of fixed position, whereas the FindExact $(\cdot)$ function is used for the contiguous algorithm to examine all feasible contiguous allocation positions over the resource grid. Then, we pick the position with the largest unallocated rectangular using the FindMaxUnUse $(\cdot)$ function. Among the non-contiguous case, the FindAvail $(\cdot)$ function outputs any available places for allocation without requiring a contiguous portion. Then, the priority is given to the allocation of non-contiguous portions over the time with the highest unallocated resources using the sort $(\cdot)$ function. Finally, the minimum granularity algorithm applies the same function but fills the allocation sequentially without any prioritization. The complexity of the proposed approach is proportional to the number of SCSs ($|SCS|$), number of slices ($|\mathcal{S}|$) and the size of resource grid ($N_b \times T_b$) and is with polynomial time complexity.

5.1.2 Performance comparison

As mentioned before, resource partitioning are processed sequentially based on the slice priority, and therefore high priority slices will impact the available

Algorithm 1: Inter-slice Resource Partition Algorithm

Input : T_b and N_b represents the size of resource grid in the partition window
$\mathcal{S}$ is the set of slices
Output: $RGMap$ is the resource grid allocation map
$OptSCS$ is the set of applied SCS of each slice
Sat is the set of slice satisfaction index

begin
  foreach $s_i \in \mathcal{S}$ **do**
    $Sat\,[i] = 0$; /* Initialize satisfaction index of each slice */
    $OptSCS\,[i] = 0$; /* Initialize the selected SCS of each slice */
    foreach $scs \in SCS_i$ **do**
      $[RegF_{scs}\,[i]\,, RegT_{scs}\,[i]]$ = SCSMap $(N_i, T_i, scs, SCS_{base})$; /* Map to different SCSs */
      $[FixF_{scs}\,[i]\,, FixT_{scs}\,[i]]$ = SCSMap $(FixF_i, FixT_i, scs, SCS_{base})$; /* Map to different SCSs */
  foreach $scs \in SCS$ **do**
    for $i = 1$ **to** $N_b \cdot scs/SCS_{base}$ **do**
      for $j = 1$ **to** $T_b \cdot SCS_{base}/scs$ **do**
        $RGMap_{scs}\,[i]\,[j] = 0$; /* Initialize resource grid allocation */
  while *isempty* $(\mathcal{S}) ==$ *false* **do**
    $pID =$ prioritize $(\mathcal{S}, priority)$; /* Get the most prioritized slice index within $\mathcal{S}$ */
    switch g_{pID} **do**
      case *Fixed* **do**
        $[Sat\,[pID]\,, OptSCS\,[pID]\,, RGMap]$ = FixPos_RP $(pID, \mathcal{S}, RGMap)$; (cf. Alg. 2)
      case *Contiguous* **do**
        $[Sat\,[pID]\,, OptSCS\,[pID]\,, RGMap]$ = Con_RP $(pID, \mathcal{S}, RGMap)$; (cf. Alg. 3)
      case *Non-contiguous* **do**
        $[Sat\,[pID]\,, OptSCS\,[pID]\,, RGMap]$ = NonCon_RP $(pID, \mathcal{S}, RGMap)$; (cf. Alg. 4)
      case *Minimum* **do**
        $[Sat\,[pID]\,, OptSCS\,[pID]\,, RGMap]$ = Min_RP $(pID, \mathcal{S}, RGMap)$; (cf. Alg. 5)
    if $Sat\,[pID] == 1$ **then**
      $\mathcal{S} =$ SetDiff $(\mathcal{S}, pID)$; /* Remove satisfied slice from set $\mathcal{S}$*/
      $RGMap =$ RGReMap $(RGMap, OptSCS\,[pID]\,, scs)$; /* Remap resource grid to other SCSs */

regions for low priority slices. In following, we evaluate five different priortization orderings:

1. *Optimal*: Exhaustive search all possibilities to get the best ordering.
2. *Random*: Randomize the slice ordering.
3. *Greedy*: Use the greedy method to prioritize slice that can generate the largest unallocated rectangular region.
4. *Granularity*: Sort slices based on the level of granularity as follows fixed position, contiguous, non-contiguous, and minimum granularity.
5. *Granular & Greedy*: Use two sequential sorting, firstly based on the granularity and secondly based on the greedy.

Algorithm 2: Fixed Position Resource Partition (FixPos_RP)

Input : k is target slice index of the set of slice $\mathcal{S}$
$RGMap$ is the resource grid allocation map
Output: Sat is the slice satisfaction index
$OptSCS$ is the selected SCS for the target slice
$OutMap$ is the output resource allocation map

begin
  $MaxRec = 0$;
  $OutMap = RGMap$;
  $Sat = OptSCS = 0$; /* Initialize the output satisfaction and optimal SCS index*/
  foreach $scs \in SCS_k$ **do**
    if *FindFixExact* $(RegF_{scs}[k], RegT_{scs}[k], scs, RGMap_{scs}, FixF_{scs}[k], FixT_{scs}[k])$ **then**
      $Sat = 1$; /* Slice with identity k is satisfied*/
      $tMap = RGMap_{scs}$;
      for $i = 0$ **to** $ReqF_{scs}[k] - 1$ **do**
        for $j = 0$ **to** $ReqT_{scs}[k] - 1$ **do**
          $tMap[i + FixF_{scs}[k]][j + FixT_{scs}[k]] = k$;
      $tRec =$ FindMaxUnUse $(tMap)$; /* Find the maximum unallocated rectangular resources */
      if $tRec > MaxRec$ **then**
        $OptSCS = scs$;
        $MaxRec = tRec$;
        $OutMap_{scs} = tMap$;

Algorithm 3: Contiguous Resource Partition (Con_RP)

Input : k is target slice index of the set of slice $\mathcal{S}$
$RGMap$ is the resource grid allocation map
Output: Sat is the set of slice satisfaction index
$OptSCS$ is the selected SCS for the target slice
$OutMap$ is the output resource allocation map

begin
  $MaxRec = 0$;
  $OutMap = RGMap$;
  $Sat = OptSCS = 0$; /* Initialize the output satisfaction and optimal SCS index*/
  foreach $scs \in SCS_k$ **do**
    $[PosF, PosT] =$ FindExact $(ReqF_{scs}[k], ReqT_{scs}[k], scs, RGMap_{scs})$;/* Find possible positions */
    for $pos = 1$ **to** $|PosF|$ **do**
      $Sat = 1$;
      $tMap = RGMap_{scs}$;
      for $i = 0$ **to** $ReqF_{scs}[k] - 1$ **do**
        for $j = 0$ **to** $ReqT_{scs}[k] - 1$ **do**
          $tMap[i + PosF[pos]][j + PosT[pos]] = k$;
      $tRec =$ FindMaxUnUse $(tMap)$; /* Find the maximum unallocated rectangular resources */
      if $tRec > MaxRec$ **then**
        $OutSCS = scs$;
        $MaxRec = tRec$;
        $OutMap_{scs} = tMap$;

The results are shown in Fig. 9 with 7 slices and each requests a time-varying and uniformly-distributed resources with $N_i = [10, 50]$ PRBs and $T_i = [1, 10]$

Algorithm 4: Non-Contiguous Resource Partition (NonCon_RP)

Input : k is target slice index of the set of slice $\mathcal{S}$
$RGMap$ is the resource grid allocation map
Output: Sat is the set of slice satisfaction index
$OptSCS$ is the selected SCS for the target slice
$OutMap$ is the output resource allocation map

begin
- $MaxRec = 0$;
- $tIdxCount = 0$;
- $OutMap = RGMap$;
- $Sat = OptSCS = 0$; /* Initialize the output satisfaction and optimal SCS index*/
- **foreach** $scs \in SCS_k$ **do**
  - $[PosF, PosT]$ = FindAvail $(RGMap_{scs})$; /* Find all available resource to be allocated */
  - **for** $j = 1$ *to* $T_b \cdot scs/SCS_{base}$ **do**
    - $aIdx\,[j]$ = find $(PosT\,[*] == j)$; /* Find available allocation portion at time index j */
    - **if** $|aIdx\,[j]| \geq ReqF_{scs}\,[k]$ **then**
      - $tIdxCount = tIdxCount + 1$; /* Increase possible time index count by 1 */
  - **if** $tIdxCount \geq ReqT_{scs}\,[k]$ **then**
    - $Sat = 1$;
    - $tMap = RGMap_{scs}$;
    - $Torder$ = sort $\left(1 : T_b \cdot \frac{scs}{SCS_{base}}, aIdx, \text{descend}\right)$; /* Sort time indexes base on descending order of $aIdx$ */
    - **for** $j = 1$ *to* $ReqT_{scs}\,[k]$ **do**
      - $PossibleFIdx = PosF\,[aIdx\,[Torder\,[j]]]$;
      - **for** $i = 1$ **to** $ReqF_{scs}\,[k]$ **do**
        - $tMap\,[i + PossibleFIdx\,[i]]\,[Torder\,[j]] = k$;
    - $tRec$ = FindMaxUnUse $(tMap)$; /* Find the maximum unallocated rectangular resources */
    - **if** $tRec > MaxRec$ **then**
      - $OutSCS = scs$;
      - $MaxRec = tRec$;
      - $OutMap_{scs} = tMap$;

TTIs under several levels of SCS, i.e. $SCS = SCS_i = \{15, 30, 60\}\,kHz, \forall i$. Fig. 9a shows the average satisfaction ratio of all 7 slices and of slices with different granularity levels (i.e. fix, contiguous, non-contiguous and minimum), and the optimal ordering reaches the highest satisfaction ratio (82% on average for all 7 slices), whereas the randomized ordering represents the worst case. From the figure, it can be observed that the *Granular* (79%) and the *Granular & Greedy* (81%) orderings outperform the Greedy approach and are very close to the optimal ordering. The resource grid utilization ratio is shown in Fig. 9b with percentage of partitioned resources, largest unallocated rectangular, and other unallocated resource in box plot. Note that the largest unallocated rectangular is used as a metric to characterize the multiplexing gain of inter-slice partitioning with different SCS/TTI requirement. The randomized and greedy ordering has the largest unallocated rectangular ratio (20% and 23% on average) at the cost of a lower partitioning percentage (73% and 71% on average). In contrast, the partitioned re-

Algorithm 5: Minimum granularity Resource Partition (Min_RP)

Input : k is target slice index of the set of slice $\mathcal{S}$
$RGMap$ is the resource grid allocation map
Output: Sat is the set of slice satisfaction index
$OptSCS$ is the selected SCS for the target slice
$OutMap$ is the output resource allocation map

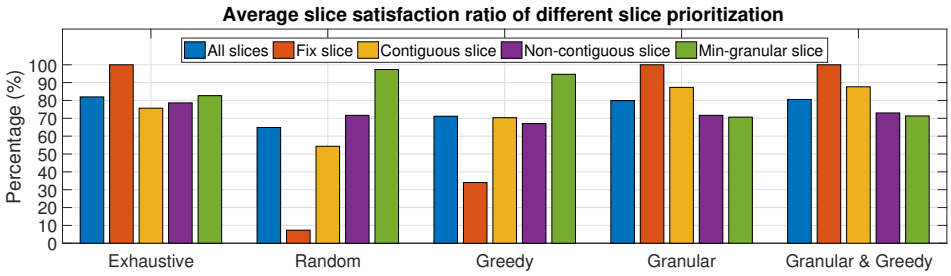
begin
  $MaxRec = 0$;
  $OutMap = RGMap$;
  $Sat = OptSCS = 0$; /* Initialize the output satisfaction and optimal SCS index*/
  foreach $scs \in SCS_k$ **do**
    $[PosF, PosT]$ = FindAvail $(RGMap_{scs})$; /* Find all possible position to be allocated */
    if $|PosF| \geq ReqF_{scs}[k] \times ReqT_{scs}[k]$ **then**
      $Sat = 1$;
      $tMap = RGMap_{scs}$;
      for $pos = 1$ **to** $ReqF_{scs}[k] \times ReqT_{scs}[k]$ **do**
        $tMap[PosF[pos]][PosT[pos]] = k$;
      $tRec$ = FindMaxUnUse $(tMap)$; /* Find the maximum unallocated rectangular resources */
      if $tRec > MaxRec$ **then**
        $OptSCS = scs$;
        $MaxRec = tRec$;
        $OutMap_{scs} = tMap$;

source utilization ratio and the largest unallocated rectangular ratio are very close for both exhaustive search (84% and 12%) and *Granular & Greedy* ordering (85% and 10%), confirming the efficiency of the proposed algorithm.

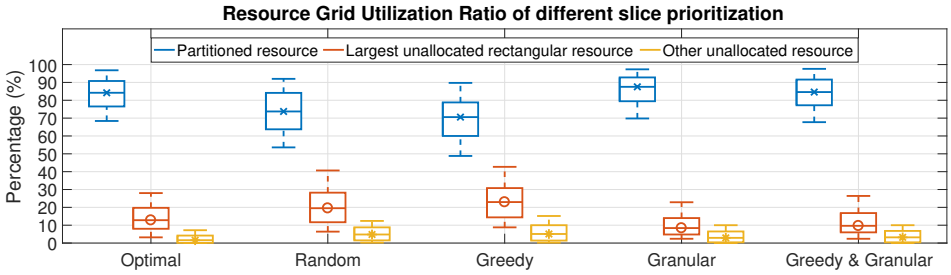
5.2 Radio resource accommodation

After inter-slice partitioning and intra-slice scheduling, the runtime then accommodates these decisions to physical resources. In Fig. 10, an example is shown based on the inter-slice partitioning outcome in Fig. 8b. The intra-slice scheduling result is shown in Fig. 10a with the gray portions refer to the scheduled parts whereas transparent portions are unscheduled, and we can see the same largest unused rectangular region. However, a larger region is formed in Fig. 10b via remapping scheduling results of s_4 and s_7 while still fulfill their abstraction types.

Like the inter-slice partitioning, we can use almost the same algorithms to pack such intra-slice scheduling results within its partitioned region considering different priority orderings. Here, we assume the traffic arrival rate of each slice is in proportional to the requested radio resource and times with an time-varying uniformly-distributed random variable $p = [0.65, 1.0]$. Then, we evaluate different orderings in Fig. 11 compare with the inter-slice partitioning result. We can see that there are approximately 1.5% (*Random*) to 3% (*Greedy*, *Granular*, *Granular & Greedy*) increasing in the largest unallocated rectangular; however, most of the increment is at the other unallocated region. Such phenomenon is due to some un-

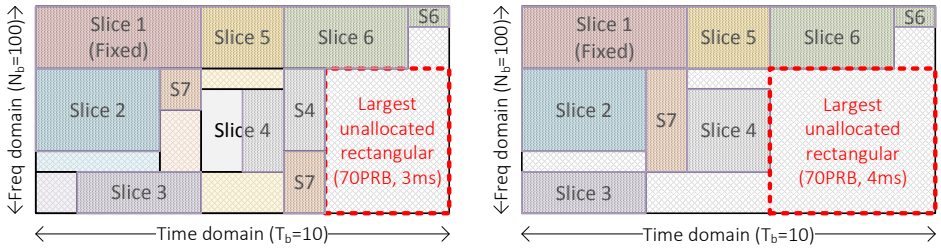


(a) Average slice satisfaction ratio



(b) Resource grid occupation ratio

Figure 9: Performance of different slice priortization in resource partitioning



(a) Inter-slice

(b) Slice-specific granularity

Figure 10: Examples of radio resource partitioning

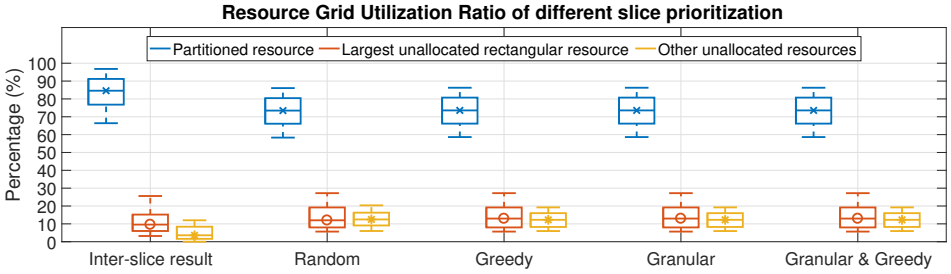
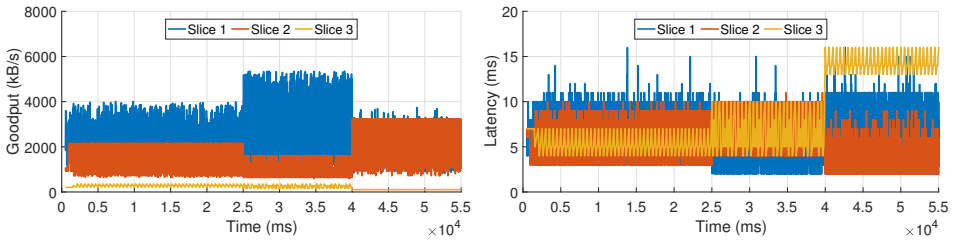


Figure 11: Performance of different slice priortization in resource accommodation

scheduled intra-slice resources are not close to the largest unallocated rectangular and can not contribute, e.g., s_2 and s_3 in Fig. 10.



(a) Aggregated good-put (b) Average latency

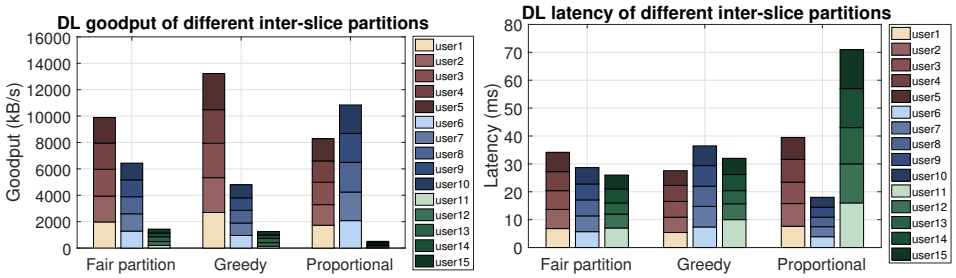
Figure 12: Impact of time-varying inter-slice partitions on slice performance

6 Proof of Concepts

To validate the concept of RAN slicing runtime system and explore different use-cases, we implemented and built an LTE-based prototype of runtime following aforementioned design in section 4. The runtime is developed based on the FlexRAN agent [15], which operates on the top of the OAI platform [14]. The main functionalities of the proposed runtime services and CP/UP APIs are implemented and integrated with the agent. To map a user to a slice and populate the slice context information, we used internal OAI user identities together with the radio network temporary identifier (RNTI) generated by the MAC layer to determine user slice identifier. Note that in a real deployment, a user communicates this information through RRC and non-access-stratum procedures. A template is used to describe slice resource requirements in terms of vRBG type 0/1 and vTBS type 0, resource isolation, and slice priority in both downlink and uplink direction. We created three remote slices using asynchronous communication channel with the runtime that embeds control logics and operates on virtualized resources and states based on the modified version of FlexRAN controller and it software development kit (SDK). In following, we present the results of three considered use cases to demonstrate the performance tradeoff between isolation and sharing and flexibility in changing the RAN service definition.

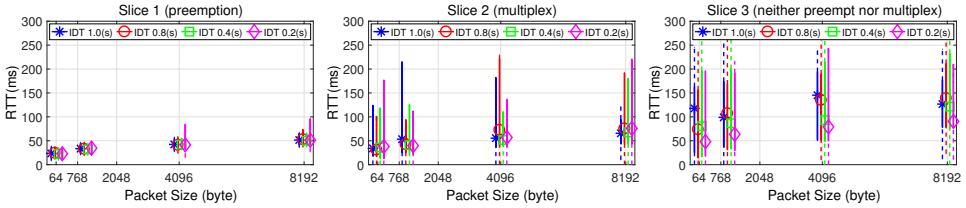
6.1 Radio Resource and Function Isolation

To demonstrate the impact of inter-slice partitioning, we deploy three slices with three different traffic patterns, high bit rate with traffic variability for slice 1, medium bit rate for slice 2, and low bit rate with periodic traffic for slice 3. Each slice contains 5 users. Different partitioning policies are applied at different time intervals: a) fair partitioning that allocates 33% of total resources to each slice before t_1, b) greedy partitioning between t_1 and t_2 that allocates 60% of resources to slice 1, and 20% to slice 2 and 3, and c) proportional partitioning after t_2 that allocates 50% to slice1, 40% to slice 2, and 10% for slice 3. For the intra-slice scheduling, we apply a simple fair scheduling among users. From the results presented in Fig. 12, it can be observed that the aggregated goodput and average



(a) Goodput (b) Latency

Figure 13: Impact of inter-slice partitioning on per-user goodput and latency



(a) Slice 1 (b) Slice 2 (c) Slice 3

Figure 14: Impact of preemption and multiplexing on RTT.

latency can significantly fluctuate when adopting different inter-slice partitions. However, the changes of the inter-slice partitioning have no impact on the decision of the intra-slice scheduling function, as shown in Fig. 13. This confirms the capability of runtime in providing isolation among slices and providing performance guarantee, as listed in section 4.1.

6.2 Radio Resource Preemption and Multiplexing

In this experiment, we demonstrate the impacts of resource multiplexing and preemption on the perceived performance in a three-slice scenario. Besides the applied abstraction/virtualization, different slice policies are explored: Slice 1 can preempt all other slices to get its desired resources when needed, slice 2 can utilize the unallocated resource to increase the multiplexing gain when available, and slice 3 can neither preempt nor multiplex resources. We firstly show the measured round trip time (RTT) in Fig. 14 with different packet size and inter-departure time (IDT). Obviously, a lower RTT is achieved for slice 1 as it has the ability to preempt other two slices; hence, such slice has the highest flexibility in adapting the resources to its instantaneous workload. As for the multiplexing, slice 2 utilizes all the unallocated resourced, and can opportunistically improve the RTT compared to slice 3 that is limited to its partitioned resources.

When examining the good-put and delay-jitter in Fig. 15, it can be seen that slice 1 can flexibly adapt its data rate as a function of workload by preempting the resources from other slices, i.e., from 3 Mbps to 6 Mbps, whereas slice 2 experi-

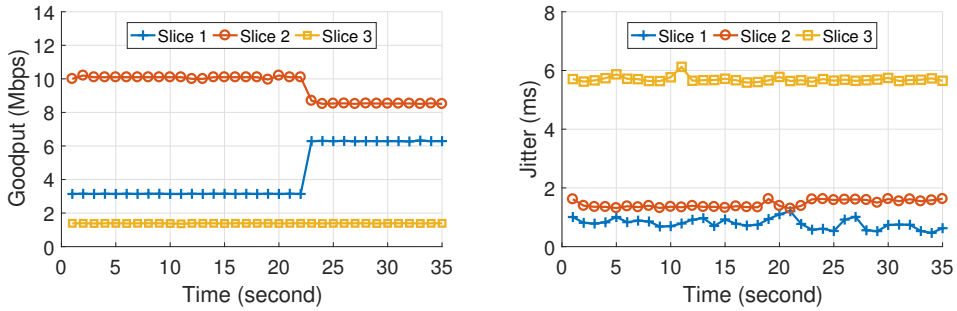


Figure 15: Impact of preemption and multiplexing on good-put and delay jitter

ences a data rate drop from its desired 10 Mbps to 8Mbps. The same trend can be observed in the delay-jitter for slice 1, 2, and 3.

6.3 Network function and state flexibility

We showcase the capability of the runtime to change the service definition of the underlying RAN module from monolithic to disaggregated deployments from the infrastructure provider perspective. For this purpose, we consider three possible RAN deployments at different time instances without creating any slices: (a) monolithic eNodeB deployment at t_1, (b) disaggregated Cloud-RAN deployment using split option 8 at t_2, and (c) using split option 7 at t_3. Our considered C-RAN deployment uses UDP/IP based Ethernet transportation over the fronthaul interface with one RRU gateway in the middle to route the traffic. The results are shown in Fig. 16 in terms of the good-put, delay jitter and RTT when a 15 Mbps traffic flow is transferred in downlink direction. Surprisingly, no good-put drop is observed when changing the split. This is because the considered split (i.e. performing cell processing at the RRU) only requires RAN module reconfiguration without any state synchronization, which explains why the good-put remains unchanged among different deployments. As for the delay jitter and RTT, they are increased at t_2 and t_3 due to some Ethernet packet loss when changing the split as well as the extra time spent for the Ethernet transportation (e.g. packetization [54], compression) along the fronthaul links and RRU gateway.

As mentioned, the functional splits is determined by the infrastructure provider rather than the slice owner. Hence, the runtime system shall make sure the SLA is maintained when there is any change in the service definition. Moreover, it shall transfer the CP/UP states when the functional split includes user-specific processing (3GPPP option 1 to 6), as listed in TABLE 5.

7 Conclusions

In this work, we propose the RAN slicing runtime system that serve as a flexible execution environment to run multiple customized slice instances with the required level of isolation while sharing underlying RAN modules and infrastructure.

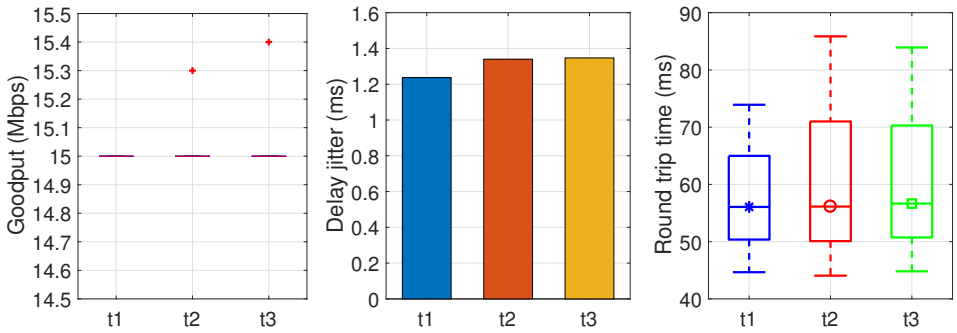


Figure 16: Flexible RAN deployment impact on good-put, delay jitter and RTT.

We elaborate on the design of such system and identify the required functionalities in both control and user plane. A new set of radio resource abstractions are defined to efficiently provide resource isolation among different slices. On the user-plane, the forwarding engine of runtime is introduced to compose the input and output data stream for a flexible processing pipeline composition. We also propose the inter-slice resource partition approach that satisfies requests of various granularities and maintains a large multiplexing region with acceptable complexity. Finally, we showcase the proposed runtime system in three use cases that exactly match aforementioned RAN slicing challenges using the OAI and FlexRAN platforms.

References

[1] P. Rost, A. Banchs, I. Berberana, M. Breitbach, M. Doll, H. Droste, C. Mannweiler, M. A. Puente, K. Samdanis, and B. Sayadi, "Mobile network architecture evolution toward 5G," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 84–91, 2016.

[2] *IMT-2020 Deliverables*, ITU-T Focus Group, 2017.

[3] *TR 23.799 Study on Architecture for Next Generation System (Release 14)*, 3GPP, Dec. 2016.

[4] NGMN Alliance, "Description of network slicing concept," Tech. Rep., 2016.

[5] 5G PPP Architecture Working Group, "View on 5G architecture," *White Paper*, 2016.

[6] Y. H. Kim *et al.*, "Slicing the next mobile packet core network," in *Wireless Communications Systems (ISWCS), 2014 11th International Symposium on*. IEEE, 2014, pp. 901–904.

[7] T. Taleb, M. Corici, C. Parada, A. Jamakovic, S. Ruffino, G. Karagiannis, and T. Magedanz, "EASE: EPC as a service to ease mobile core network deployment over cloud," *IEEE Network*, vol. 29, no. 2, pp. 78–88, 2015.

[8] Z. A. Qazi, M. Walls, A. Panda, V. Sekar, S. Ratnasamy, and S. Shenker, "A High Performance Packet Core for Next Generation Cellular Networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 348–361.

[9] N. Nikaein, E. Schiller, R. Favraud, K. Katsalis, D. Stavropoulos, I. Alyafawi, Z. Zhao, T. Braun, and T. Korakis, "Network store: Exploring slicing in future 5G networks," in *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture*. ACM, 2015, pp. 8–13.

[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] X. Foukas, M. Mahesh K., and K. Kontovasilis, "Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture," in *The 23rd Annual International Conference on Mobile Computing and Networking (MobiCom '17)*. ACM, 2017.

[12] *TR 23.707 Architecture enhancements for dedicated core networks; Stage 2 (Release 13)*, 3GPP, Dec. 2014.

[13] *TR 23.711 Architecture enhancements for dedicated core networks; Stage 2 (Release 14)*, 3GPP, Sep. 2016.

[14] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A flexible platform for 5G research," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 33–38, 2014.

[15] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. P. Kontovasilis, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks." in *CoNEXT*, 2016, pp. 427–441.

[16] A. Khan, W. Kellerer, K. Kozu, and M. Yabusaki, "Network sharing in the next mobile network: TCO reduction, management flexibility, and operational independence," *IEEE Communications Magazine*, vol. 49, no. 10, 2011.

[17] L. Doyle, J. Kibilda, T. K. Forde, and L. DaSilva, "Spectrum without bounds, networks without borders," *Proceedings of the IEEE*, vol. 102, no. 3, pp. 351–365, 2014.

[18] K. Katsalis, N. Nikaein, E. J. Schiller, A. Ksentini, and T. Braun, "Network Slices Towards 5G Communications: Slicing the LTE network," *IEEE Communications Magazine*, vol. 55, no. 8, 2017.

[19] X. An, R. Trivisonno, H. Einsiedler, D. von Hugo, K. Haensge, X. Huang, Q. Shen, D. Corujo, K. Mahmood, D. Trossen *et al.*, "End-to-End Architecture Modularisation and Slicing for Next Generation Networks," *arXiv preprint arXiv:1611.00566*, 2016.

[20] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5G network slice broker," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 32–39, 2016.

[21] I. Chih-Lin, S. Han, Z. Xu, S. Wang, Q. Sun, and Y. Chen, "New Paradigm of 5G Wireless Internet," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 474–482, 2016.

[22] X. Zhou, R. Li, T. Chen, and H. Zhang, "Network slicing as a service: enabling enterprises' own software-defined cellular networks," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 146–153, 2016.

[23] S. Sharma, R. Miller, and A. Francini, "A Cloud-Native Approach to 5G Network Slicing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 120–127, 2017.

[24] A. Tzanakaki, M. Anastasopoulos, I. Berberana, D. Syrivelis, P. Flegkas, T. Korakis, D. C. Mur, I. Demirkol, J. Gutiurrez, E. Grass *et al.*, "Wireless-Optical Network Convergence: Enabling the 5G Architecture to Support Operational and End-User Services," *IEEE Communications Magazine*, 2017.

[25] *TS 23.251 Network sharing; Architecture and functional description*, 3GPP, Jan. 2009.

[26] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, "Nvs: A substrate for virtualizing wireless resources in cellular networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 5, pp. 1333–1346, 2012.

[27] X. Costa-Pérez, J. Swetina, T. Guo, R. Mahindra, and S. Rangarajan, "Radio access network virtualization for future mobile carrier networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 27–35, 2013.

[28] R. Mahindra, M. A. Khojastepour, H. Zhang, and S. Rangarajan, "Radio access network sharing in cellular networks," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 2013, pp. 1–10.

[29] J. He and W. Song, "AppRAN: Application-oriented radio access network sharing in mobile networks," in *Communications (ICC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3788–3794.

[30] Y. Zaki, L. Zhao, C. Goerg, and A. Timm-Giel, "LTE mobile network virtualization," *Mobile Networks and Applications*, vol. 16, no. 4, pp. 424–432, 2011.

[31] C. Liang and F. R. Yu, "Wireless virtualization for next generation mobile cellular networks," *IEEE wireless communications*, vol. 22, no. 1, pp. 61–69, 2015.

[32] P. Marsch, I. Da Silva, O. Bulakci, M. Tesanovic, S. E. El Ayoubi, T. Rosowski, A. Kaloxylos, and M. Boldi, "5G radio access network architecture: design guidelines and key considerations," *IEEE Communications Magazine*, vol. 54, no. 11, pp. 24–32, 2016.

[33] K. Katsalis, N. Nikaein, E. Schiller, R. Favraud, and T. I. Braun, "5G architectural design patterns," in *Communications Workshops (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 32–37.

[34] *TR 38.801 Study on new radio access technology: Radio access architecture and interfaces (Release 14)*, 3GPP, Mar. 2017.

[35] *TR 38.804 Study on new radio access technology: Radio Interface Protocol Aspects (Release 14)*, 3GPP, Mar. 2017.

[36] M. Yang, Y. Li, D. Jin, L. Su, S. Ma, and L. Zeng, "OpenRAN: a software-defined ran architecture via virtualization," in *ACM SIGCOMM computer communication review*, vol. 43, no. 4. ACM, 2013, pp. 549–550.

[37] I. F. Akyildiz, P. Wang, and S.-C. Lin, "Softair: A software defined networking architecture for 5g wireless systems," *Computer Networks*, vol. 85, pp. 1–18, 2015.

[38] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software defined radio access network," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 25–30.

[39] T. Chen, H. Zhang, X. Chen, and O. Tirkkonen, "SoftMobile: control evolution for future heterogeneous mobile networks," *IEEE Wireless Communications*, vol. 21, no. 6, pp. 70–78, 2014.

[40] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "Openradio: a programmable wireless dataplane," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 109–114.

[41] W. Wu, L. E. Li, A. Panda, and S. Shenker, "Pran: Programmable radio access networks," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. ACM, 2014, p. 6.

[42] A. Gudipati, L. E. Li, and S. Katti, "Radiovisor: A slicing plane for radio access networks," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 237–238.

[43] A. Nakao, P. Du, Y. Kiriha, F. Granelli, A. A. Gebremariam, T. Taleb, and M. Bagaa, "End-to-end network slicing for 5G mobile networks," *Journal of Information Processing*, vol. 25, pp. 153–163, 2017.

[44] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega *et al.*, "Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, 2017.

[45] *TS 38.401 NG-RAN; Architecture description (Release 15)*, 3GPP, Aug. 2017.

[46] J. Matias, J. Garay, N. Toledo, J. Unzilla, and E. Jacob, "Toward an SDN-enabled NFV architecture," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 187–193, 2015.

[47] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless network functions: Breaking the tight coupling of state and processing." in *NSDI*, 2017, pp. 97–112.

[48] J. Kim, D. Kim, and S. Choi, "3GPP SA2 architecture and functions for 5G mobile communication system," *ICT Express*, 2017.

[49] J. Vihriälä, A. A. Zaidi, V. Venkatasubramanian, N. He, E. Tiirola, J. Medbo, E. Lähetkangas, K. Werner, K. Pajukoski, A. Cedergren *et al.*, "Numerology and frame structure for 5G radio access," in *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on*. IEEE, 2016, pp. 1–5.

[50] OpenvSwitch, http://openvswitch.org/.

[51] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 99–110.

[52] C.-Y. Chang, N. Nikaein, R. Knopp, T. Spyropoulos, and S. S. Kumar, "FlexCRAN: A Flexible Functional Split Framework over Ethernet Fronthaul in Cloud-RAN," in *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–7.

[53] A. Caprara and M. Monaci, "On the two-dimensional knapsack problem," *Operations Research Letters*, vol. 32, no. 1, pp. 5–14, 2004.

[54] C.-Y. Chang, R. Schiavi, N. Nikaein, T. Spyropoulos, and C. Bonnet, "Impact of packetization and functional split on C-RAN fronthaul performance," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–7.