This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

RAN Runtime Slicing System for Flexible and Dynamic Service Execution Environment

Chia-Yu Chang and Navid Nikaein
Communication Systems Department, EURECOM, France
Email: firstname.lastname@eurecom.fr

***Abstract*—Network slicing is one key enabler to provide the required flexibility and to realize the service-oriented 5G vision. Unlike the core network slicing, radio access network (RAN) slicing is still at its infancy and several works just start to investigate the challenges and potentials to enable the 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 isolation and sharing as per slice requirement. Moreover, both 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 another challenge. In this paper, we propose a RAN runtime slicing 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 RAN runtime slicing system for LTE, assess its feasibility and potentials, and demonstrate the isolation, sharing, and customization capabilities with three representative use cases.**

***Index Terms*—Network slicing, RAN slicing, 5G, service orientation**

I. Introduction

Fifth generation (5G) mobile network is a paradigm shift beyond the new radio and wider spectrum with the objective of improving the overall efficiency and flexibility of mobile networks. It is about the evolution of computing for wireless networks (e.g., central offices become data centers) and enabling the service-oriented architecture to deliver networks on an *as-a-service* basis. Support of vertical markets is one of the main driving factors behind this evolution to empower the business and value creation for 5G. The underlying idea being to support multiple services and/or virtual networks on a single physical network with different service requirements is in terms of the definition and agreement, the control and management, and also the performance.

Through this service-oriented 5G vision, naturally the network infrastructure providers (e.g., operators and data center owners), service providers (e.g., over-the-top and verticals), and network function/application providers (e.g., vendors) are decoupled to allow a 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 the value-chain in telecommunication industry being aligned with the high-level role models presented by the third generation partnership project (3GPP) in [1]. 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

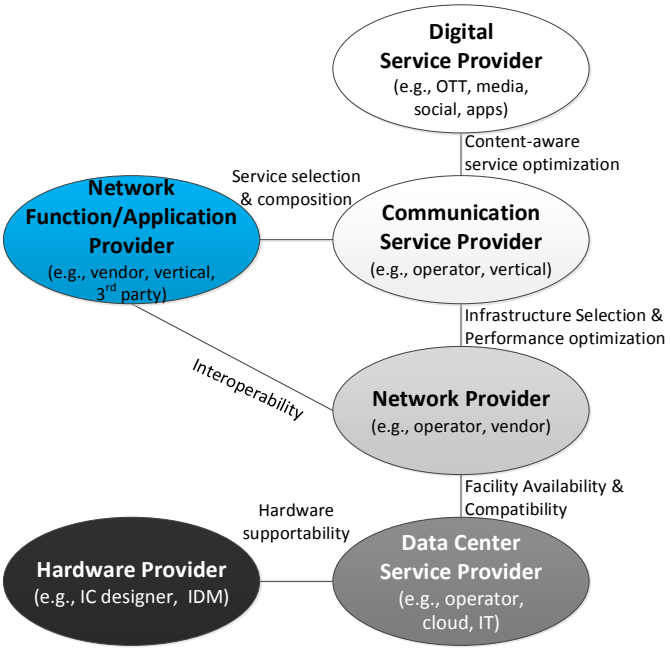


Fig. 1: Service-orientation impact on the evolution of telecommunication industry.

information technologies (ITs), and transport network from operators. A service is built through the 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 thats of network infrastructure providers in terms of PNF/VNF interoperability and compatibility when the service is running on different underlying infrastructures.

Network slicing is one of the key enablers 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 other slices down to the different sets of spectrum and cell site (as in most of current 3G and 4G deployment), or be *shared* across all types of resources including radio spectrum and network functions (e.g., all network layers of protocol stack), or be *customized* for a subset of user-plane (UP) and control-plane (CP) processing with an access to a portion of radio resources in a *virtualized* form. To enable these options, a flexible execution environment is needed to host slice service instance over the resources provided by the underlying infrastructures. Hence, different levels of *isolation* and *sharing* across the *domain-specific*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

resources spanned by a slice shall be naturally supported. Note that the concept of different isolation levels not only provides the dedication over resources but also brings the independency among network functions and applications. Further, domain boundaries could be administrative (e.g., between operators), network segment (e.g., radio access network, core network, transport network), radio access technology (e.g., 4G, 5G) among the others, and resources could be of different types including computing, storage, network, hardware, radio, spectrum, network functions and applications. For instance, a slice can be composed with dedicated core network and isolated control functions, while also leverages the virtualized radio resource in a shared radio spectrum. However, another slice can be composed of fully isolated resources (except computing resources) and network functions like the FLARE solution provided in [2].

Hence, *softwarization*, *virtualization*, and *disaggregation* are the 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 requirements of an end-to-end (E2E) service. They constitute the foundation for a multi-service and multi-tenant architecture, and are realized by applying the principles of software-define networking (SDN), network function virtualization (NFV), and cloud computing to the mobile networks [3].

Several standardization bodies and industry forums outline the crucial role of an E2E network slicing to fulfill the *service-oriented* visions of 5G, e.g., International Telecommunication Union (ITU) [4], 3GPP [5] and next generation mobile networks (NGMN) alliance [6]. Also, prominent network architectures are proposed by 5G initiatives and projects, e.g., 5G infrastructure public private partnership (5GPPP) European program [7]. Many architectures and prototypes have been proposed for core network (CN) slicing [8]–[10] and radio access network (RAN) slicing [11], [12]. The challenge of CN slicing has been also addressed by 3GPP, and realized through a dedicated core network (DECOR) [13] and evolved DECOR (eDECOR) [14]. 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 control logic (CL) while increasing the resource utilization of RAN infrastructure*. 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 following the standardized protocol stack. Note that the RAN slicing is different from legacy RAN sharing notion in which the focus is only on the efficient sharing on cell sites, passive (e.g., antenna mast) and active (e.g., transport network infrastructure) network elements, radio spectrum, network function and application, and baseband processing. Section II will delve in more details on the evolution from RAN sharing toward RAN slicing.

To this end, the novel RAN runtime slicing system is proposed and we summarize our contributions as follows:

- We review the state-of-the-art on network slicing architecture with the particular focus on the RAN slicing (Section II);
- We present a RAN slicing architecture design in form of the RAN runtime slicing system to enable different levels of isolation and sharing for each slice in terms of the underlying RAN modules and resources, while allowing flexible service composition and customization across UP, CP, and CL (Section III and IV);
- We introduce a practical set of radio resource abstractions and evaluate the multiplexing gain provided by our designed algorithms for inter-slice resource partitioning and accommodation (Section V);
- We implement a concrete RAN runtime slicing system prototype on top of OpenAirInterface (OAI) [15] and FlexRAN [16] platforms and then characterize its performance through three case studies (Section VI).

II. 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 in [17]. Additional network sharing models are surveyed and summarized in [18], [19]. In [20], a slice-based network architecture is proposed with the "Network store" concept as a platform to facilitate the dynamic network slicing based on the VNFs on top of commodity infrastructures. The same idea is extended in [21] featuring the "Network and application store" that simplifies the procedure to define each slice. In [22], the proposed modularized architecture is composed of several building blocks, each with various sub-functions to customize the functionalities on per service of slice. The 5G network slice broker notion is investigated in [23] that resides inside the infrastructure provider and enables the on-demand multi-tenant slice resource allocation. The generic slice as a service model is presented in [24], [25] and it aims to orchestrate customized network slice as a service with the mapped network functions based on the service level agreement (SLA). A cloud-native network slicing approach presented in [26] allows to devise network architectures and deployments tailored to the needs of service. The authors of [27] present the E2E network slicing architecture and elaborate on the mobility management and resource allocation mechanisms for three major 5G service types, i.e., enhanced mobile broadband (eMBB), ultra-reliable and low-latency communication (uRLLC) and massive machine type communication (mMTC). Also, a joint RAN and transport network slicing approach facilitating the programmable control and orchestration plane is provided in [28].

In terms of the RAN slicing, it is stemmed from the *RAN sharing* concept such as Multi-Operator RAN (MORAN) and Multi-Operator CN (MOCN). The MORAN approach shares the same RAN infrastructure but with dedicated frequency bands for different operators, while MOCN allows to also share the spectrum among operators as standardized by 3GPP in [17]. These approaches can efficiently utilize available radio resources which are surveyed widely as network virtualization substrate (NVS) in [29], [30] that can virtualize radio resources for different resource provisioning approaches in order to

TABLE I: Acronym Table

Abbreviation	Full name
3GPP	3rd Generation Partnership Project
5G	Fifth Generation
5GPPP	5G Infrastructure Public Private Partnership
API	Application Programming Interface
BS	Base Station
CI	Control Information
CL	Control Logic
CN	Core Network
CP	Control-Plane
CU	Centralized Unit
DU	Distributed Unit
E2E	End-to-End
IDT	Inter-Departure Time
ITU	International Telecommunication Union
MAC	Medium Access Control
MCS	Modulation and Coding Scheme
MVNO	Mobile Virtual Network Operator
NAS	Non-Access Stratum
NFV	Network Function Virtualization
NGMN	Next Generation Mobile Networks
NSSAI	Network Slice Selection Assistance Information
PDCP	Packet Data Convergence Protocol
PNF	Physical Network Function
PRB	Physical Resource Block
PRBG	Physical Resource Block Group
QoS	Quality of Service
QoE	Quality of Experience
RAN	Radio Access Network
RBG	Resource Block Group
RLC	Radio Link Control
RRC	Radio Resource Control
RTT	Round Trip Time
RU	Radio Unit
SCS	Sub-Carrier Spacing
SD-RAN	Software-Defined Radio Access Network
SDAP	Service Data Adaptation Protocol
SDK	Software Development Kit
SDN	Software-Defined Networking
SLA	Service Level Agreement
TTI	Transmission Time Interval
VNF	Virtual Network Function
vRB	virtualized Resource Block
vRBG	virtualized Resource Block Group
vTBS	virtualized Transport Block Size
UP	User-Plane

coexist several mobile virtual network operators (MVNOs) in a single physical RAN. The NetShare approach in [31] extends the NVS approach and applies a central gateway-level component to ensure resource isolation and to optimize resource distribution for each entity. In [32], the authors propose the CellSlice architecture as a gateway-level solution that can indirectly impact individual BS scheduling decision for slice-specific resource virtualization. Authors of [33] provide the AppRAN as the application-oriented framework that defines a serial of abstract applications with distinct quality of service (QoS) guarantees. The Hap-SliceR radio resource slicing framework proposed in [34] is based on the reinforcement learning approach considering resource utilization and slice utility requirements; however, its main focus is on the resource customization for haptic communication. On a more general basis, RAN virtualization [35], [36] provides functional isolation in terms of customized and dedicated control plane functionalities for each MVNO. These aforementioned works consider either radio resource sharing or functional isolation, while few attentions are given to simultaneously satisfy both concerns.

To enable the RAN slicing concept, several 5G RAN design requirements and paradigms shall be fulfilled as elaborated in [40]. Future RAN design patterns are explained in [41] along the aspects of cloud computing, SDN/NFV and software engineering. Moreover, 3GPP mentions the RAN slicing realization principles in [42], [43] including 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 processing from the UP processing. Several works argue the level of centralization of CP functionalities. The fully centralized architecture is proposed such as OpenRAN in [44] and as SoftAir in [45] that may face the challenge of real-time control given the inherent delay between the controller and underlying RAN. The SoftRAN [46] architecture statically refactors the control functions into the centralized and distributed ones based on the time criticality and the central view requirement. The SoftMobile approach [47] further abstracts the CP processing in several layers based on the functionalities in order to perform the control functionalities through the application programming interfaces (APIs). As for the UP programmability and modularity, the OpenRadio [48] and PRAN [49] are pioneered to decompose the overall processing into several functionalities that can be chained. FlexRAN [16] realizes a SD-RAN platform and implements a custom RAN south-bound API through which programmable CL can be enforced with different levels of centralization, either by the controller or RAN agent.

With aforementioned enablers, several RAN slicing works are initiated. The blueprint proposed as RadioVisor in [37] can isolate the control channel messages, elementary resources such as CPU and radio resource to provide the customized service for each slice. A fully isolation solution as FLARE is provided in [2] with different virtual base stations (BSs) representing 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 this work. In [38], the radio resource scheduling of a BS is separated into the intra-slice scheduler and inter-slice scheduler; however, the resource abstraction/virtualization is not included and only a portion of functions are isolated. In [11], 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. Nevertheless, this work neither considers function isolation nor resource customization/abstraction per slice request. In [50], different approaches to split radio resources are compared in terms of the resource granularity and the degrees of isolation and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

TABLE II: RAN slicing state-of-the-arts comparison

Authors	Solution level	Radio resource	CP function	UP function
Nikaein *et al.* [20]	Network-wide level	-	Dedicated	Dedicated
Kokku *et al.* [29]	BS level	Physical or virtualized resource sharing	-	-
Mahindra *et al.* [31]	Gateway and BS level	Physical or virtualized resource sharing	-	-
Kokku *et al.* [32]	Gateway level	Virtualized resource sharing	-	-
He *et al.* [33]	Gateway level	App-oriented virtualized resource sharing	-	-
Aijaz [34]	Gateway level	Learning-based virtual resource sharing	-	-
Zaki *et al.* [35]	BS level	Physical resource sharing	Dedicated	Dedicated
Foukas *et al.* [16]	BS level	Physical or virtualized resource sharing	Shared	Shared
Gudipati *et al.* [37]	BS level	Physical 3D resource sharing	Dedicated	Dedicated till programmable radio
Nakao *et al.* [2]	BS level	Dedicated spectrum allocation	Dedicated	Dedicated
Rost *et al.* [38]	BS level	Physical resource sharing	Split into cell and user-specific	Dedicated till real-time RLC
Ksentini and Nikaein [11]	BS level	Flexible between dedication and sharing	Dedicated	Shared
Foukas *et al.* [12]	BS level	Virtualized resource sharing	Split into cell and user-specific	Dedicated till PHY layer
Ferrús *et al.* [39]	BS level	Physical resource sharing	Dedicated	Dedicated or Shared till PHY

customization; however, the resource multiplexing capability among slices is not considered. Authors of [12] introduce the BS hypervisor concept to simultaneously isolate slice-specific control logics and share the radio resources. Moreover, it can group the underlying 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 deployments. In [39], the proposed RAN slicing framework can base on the service descriptions to flexibly share RAN functions over different network layers; however, it only considers physical resource partitioning without any resource virtualization and multiplexing.

TABLE II summaries the solution level and compares several related works in three dimensions: radio resource allocation model, control plane function, and user plane function. To serve various flavors of slice, the flexibility and effectiveness of these three dimensions shall be achieved simultaneously through a unified RAN slicing solution. To this end, our proposed RAN runtime slicing system can flexibly 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 and customization for modularized RAN, and (3) flexibility and adaptability to different RAN deployment scenarios ranging from monolithic to disaggregated.

III. RAN Runtime Slicing System

We propose a RAN runtime slicing system that provides a flexible *execution environment* to run multiple virtualized RAN instances with the requested levels of isolation and sharing of the underlying RAN modules and resources. It allows the slice owners to (a) create and manage their slices, (b) perform their customized CLs (e.g., handover decision) and/or customized UP/CP processing (e.g., packet data convergence protocol [PDCP] and radio resource control [RRC] functions), and (c) operate on a set of virtual resources (e.g., resource block or frequency spectrum) or capacity (e.g., rate) and access to their CP/UP state (e.g., user identity) that are revealed by the RAN runtime. The isolation and customization properties provided by the RAN runtime is in favor of the slice owners allowing

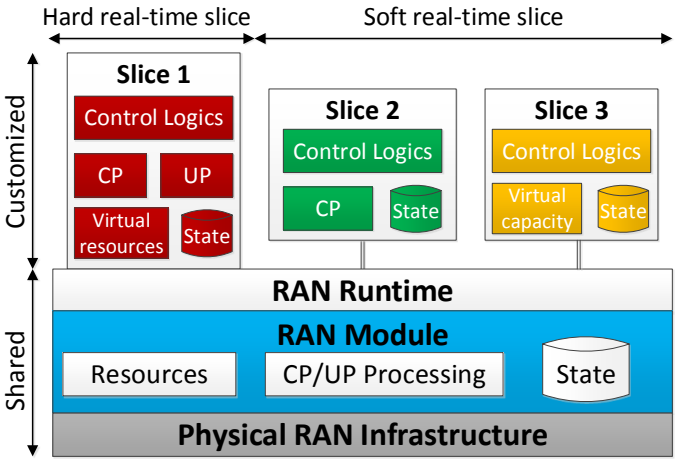


Fig. 2: High-level architecture of RAN runtime slicing system.

them to control the slice compositions and the behavior of the underlying RAN module as per service requirements, while the sharing is in favor of the infrastructure provider that enables the efficient and dynamic *multiplexing* among multiple tenants over resources, processing, and states in terms of common RAN modules to reduce the expenditures. The RAN module refers to a unit that comprises a subset of RAN functions and performs a portion of RAN processing. 3GPP decomposes the monolithic BS architecture into a three-level disaggregated manner, namely the radio unit (RU), the distributed unit (DU) and the centralized unit (CU) as introduced in [42].

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

monitor or control the CP/UP processing, resources, and states while retaining the isolation among slices.

A slice is formally represented to the RAN runtime by a slice descriptor that defines the slice service requirements in terms of the resources, custom processing, and performance. It is generally provided by the service orchestrator during the creation or update of a slice, and indicates for each slice how radio resources are *allocated, reserved, preempted, or shared*, how the CP/UP processing is *pipelined*, and what are the average expected throughput and latency. The customization feature provided by the RAN runtime allows a slice owner to only contain a portion of resources and processing within the slice boundary and to multiplex the remaining ones into the underlying RAN module. To realize a flexible tradeoff between the isolation and the sharing, the states of CP and UP processing are maintained in a database[1] allowing to update the processing pipeline (e.g., from the customized one to the multiplexed one 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 the stateless processing which allows to update the service and to recover the state through the RAN runtime.

In addition, the overall CP processing of a BS is logically separated into the slice-specific functions and the BS-common ones to exploit the function multiplexing benefits. Note that the CP processing is separated in terms of the functionalities. For instance, the master information block (MIB) and system information blocks (SIBs) are broadcasted commonly to all users with in a cell and are categorized into the BS-common one, while the random access process may be customized by each slice to reduce the latency generated by the BS-common random access procedure. Moreover, the control logics of each slice can be developed/deployed independently tailored to the service requirement. For example, the handover control decisions can be programmed to improve slice-specific quality of experience (QoE) and the RAN runtime will provide a feasible policy toward underlying RAN module.

In summary, in the proposed RAN runtime 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/UP functions and CLs as per slice requirement. The RAN runtime acts as the intermediate between the customized slices and the underlying shared RAN module and infrastructure providing a unified execution environment with substantial flexibility to achieve the required level of isolation and sharing. Finally, we provide an example with three slices as shown in Fig. 3. For slice 1, both CP and UP processing are separated into customized (RRC, service data adaptation protocol [SDAP] radio link control [RLC], medium access control [MAC] layers) and shared ones (Physical [PHY] layers), while slice 2 only customizes its SDAP function for UP processing. In contrast, slice 3 relies on the shared CP/UP processing without any customization. Moreover, the control logics of each slice can be programmed in a customized manner, e.g., handover (slice 1), RRM (slice 1 and slice 2) and monitoring (all three slices). These customized control logics will be accommodated by the shared control logics of the RAN runtime that will be elaborated in Section IV.

[1]This is regardless of whether the network function is stateful or stateless [51], [52].

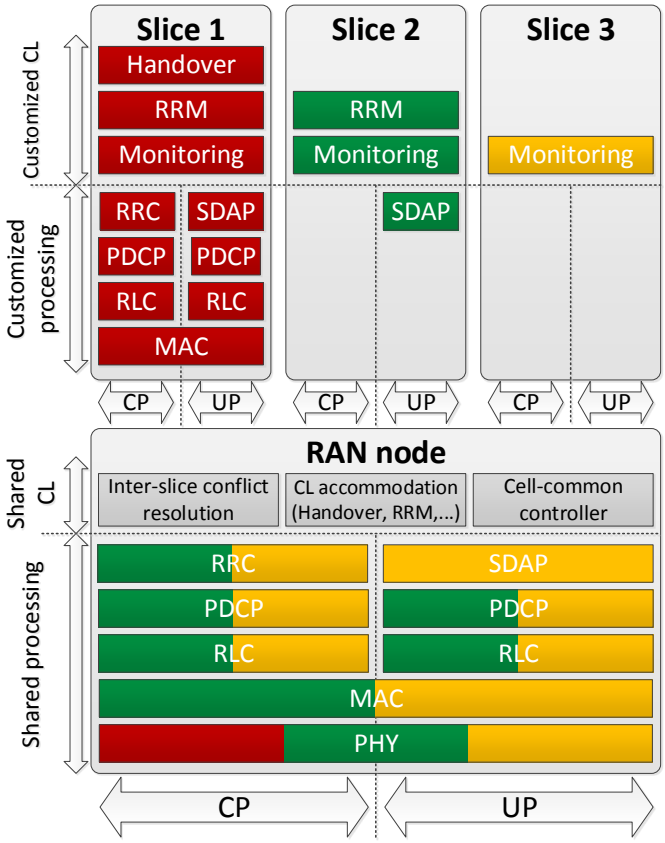


Fig. 3: Examples for three instantiated slices.

IV. Design Elements of Runtime

This section provides more details on the main components of the RAN runtime slicing system, namely the slice data, RAN runtime services, and RAN runtime APIs.

A. Design Challenge

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

- Allow each slice to interact with the underlying RAN and change the CP and UP behaviors that are dynamically determined during its execution (section IV-B and IV-C).
- Provide different levels of isolation and sharing to allow a slice owner to flexibly compose the slice-specific RAN resources and processing from the multiplexed or customized resources and CP/UP functions, respectively. Note that the multiplexing gain is also considered for the underlying radio resources and RAN modules (section IV-C).
- Provide the APIs to enable the slice-specific CP, UP and control decisions to be realized for both soft and hard real-time requirements (section IV-D).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

Fig. 4 illustrates the three main building blocks of the RAN runtime: (a) slice data, (b) CP and UP functions to provide the RAN runtime services, and (c) RAN runtime API, that are described in following paragraphs.

B. Slice data

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

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

The *module context* includes the CP and UP state information (belongs to the slice owners), module life-cycle management primitives such as start, configure, and stop service (belongs to the network function/application provider), and resources (belongs to the infrastructure provider). Unlike input or output data streams of the RAN module that can be pipelined, the control and data state are maintained separately by the RAN runtime and revealed to each slice in a real-time manner to allow the efficient and isolated slice-specific processing. In addition, such state may be shared among multiple slices subject to the access control, for instance, when coordinated processing and/or decision making are required in the case of the handover decision of a user belonging to two or more slices. Note that in general case, states only include the user-specific functions in RRC CONNECTED (and new RRC INACTICE-CONNECTED [53]) state, and not necessarily the BS-common functions that are executed independently from the number of instantiated slice, i.e., even with no instantiated slices or when operating in RRC IDLE mode (cf. TABLE IV).

C. RAN runtime services

In the following, we elaborate on five RAN runtime service that can be provisioned for each slice shown in Fig. 4, i.e., context manager, slice manager, virtualization manager, common control applications and forwarding engine. To utilize these RAN runtime services, each slice is registered and identified with its identity over the RAN runtime among disaggregated RAN entities

1) Context Manager: This service manages both slice context and module context by performing the CRUD[3] operation on the slice data. To create a slice context, the context manager firstly performs the slice admission control based on the provided network slice descriptor (NSD) that defines the required processing, resources, and states (as agreed between the slice owner and the infrastructure provider). Upon the slice admission control, module context is used by the context manager to register slice-specific life-cycle primitives to the slice 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, while the latter enables the resource partitioning and abstraction to be performed among multiple slices. At this stage, a slice can start to consume the RAN runtime services not only to manage its service but also to interact with the underlying RAN module through the RAN runtime CP/UP APIs. Then, the context manager can handle the 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.

Note that many slices can be deployed at a single RAN runtime following the multi-tenancy approach to enable scalable service deployment. However, the maximum number of slices that can be deployed depends on (1) the overhead of the RAN runtime, (2) the available resource in terms of compute, memory and link, (3) the requested SLA and resource by each slice, (4) the assurance percentage to over-provision resources, and (5) the workload of each slice.

2) Slice Manager: The slice manager entity is responsible for managing the life-cycle of a slice when instructed by the slice owner or the service orchestrator. Through the slice 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 slice 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 the multiplexed processing operated by underlying RAN module and the customized processing performed by the slice. Unlike the context manager that handles the local slice context, the slice manager operates on an E2E RAN service in support of service continuity when the slice service definition is updated. For example, a slice owner that performs the customized UP processing can opt in for the multiplexed pipelined processing to reduce its OPEX, which causes changes in its slice service definition. In addition, when the slice requirements are violated (e.g., performance degradation), the slice manager may change the number of requested resources, resource allocation type, resource partitioning period, or even update the service definition to comply with the service requirements.

Slice manager is also in charge of taking a set of actions when detecting any 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 user when it belongs to multiple slices (e.g., 1:n or m:n user-slice relationships). For instance, reserving the resources and/or changing the resource allocation type of a slice may violate the performance of another slice that requires a high bandwidth. Another example is when different user measurement events are requested by different slices which will require a coordination to reconfigure the measurement

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

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

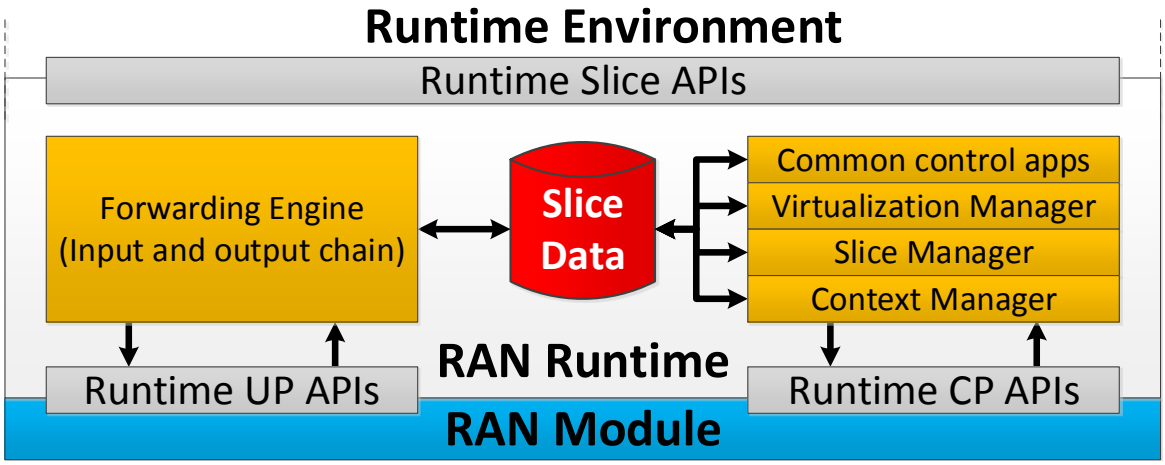


Fig. 4: Architecture of the RAN runtime slicing system.

TABLE III: Slice context maintained by the RAN runtime.

Slice Context	Description
Slice identity	Represents a unique slice identifier
Service registry identity	Identifies to which RAN runtime services a slice is registered, e.g., slice manager context manager, virtualization manager, common control applications, forwarding engine
Slice SLA and policy	Describes a business agreement between slice owner and infrastructure provider in terms of performance, resource, access control, and priority level of a corresponding slice
Customized processing	Specifies the customized CP/UP processing functions of such slice. 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]

TABLE IV: 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 re-selection	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 BS signal, e.g., cell-specific reference signal (CRS), primary/secondary synchronization signal (PSS/SSS)	Per-slice dedicated resource partitioning and accommodation
System information	Broadcast non-access stratum (NAS), MIB, and SIB information	-

with the largest common parameters and the least denominator. To this end, such manager relies on a set of policy rules defined by the infrastructure provider to decide whether to preempt one slice, reject another slice, or multiplex the requests.

3) Virtualization Manager: This RAN runtime service is in charge of providing the required level of isolation and sharing to each slice. It *partitions* on resources and states based on the slice and module contexts, *abstracts* the physical resources and states to/from the virtualized ones, and reveals the *virtual* views to a slice that is customized and decoupled from the exact physical resources and states. In the following paragraphs, we focus on the resource aspect and omit state partitioning and abstraction as they can be realized through some well-known approaches such as database partitioning and control access.

a) Inter-slice resource partitioning: Resource partitioning is a periodic process that happens in every allocation window of T [12], [29]. It allows to distribute resources among multiple slices based on the resource requirements expressed in the slice context that is stored in the slice data. Radio resource descriptor has three elements: (1) *resources type* defines whether the requested resources are of type physical/virtual radio resources in time and frequency domain[4], or capacity in terms of the data rate, (2) *abstraction type* that maps physical radio resource allocation types, namely fixed position, contiguous, non-contiguous, or minimum resource block groups (RBG), to the virtual RBGs (vRBG) or virtual transport block size (vTBS), and (3) *resource structure* that contains the applicable frame structure numerologies in time

[4]It can be extended to the dimensions of component carrier and antenna.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

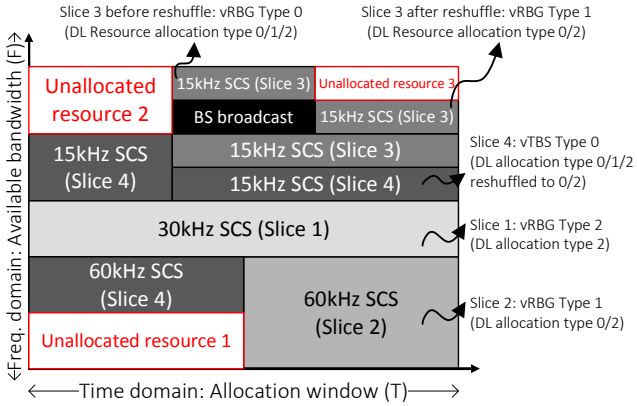
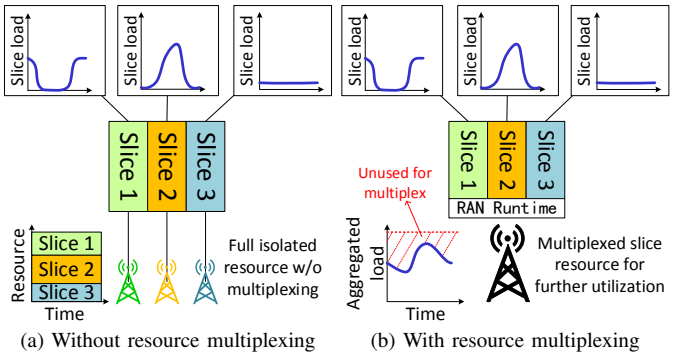


Fig. 5: Resource partition with different abstraction types.



(a) Without resource multiplexing (b) With resource multiplexing

Fig. 6: Multiplexing of slice resources.

and frequency domains. Specifically, different numerologies in terms of the transmission time interval (TTI) and the sub-carrier spacing (SCS) can be applied depending on the deployed frequency band and/or maximum user mobility in order to mitigate the impacts of wireless channel non-idealities (e.g., Doppler shift due to the user mobility) for each slice service [54]. For instance, only one type of SCS, i.e., 15 kHZ, is applied in LTE system, while there are five applicable SCSs, i.e., 15, 30, 60, 120, and 240 kHz, defined by 3GPP in [55] with their corresponding frame structures.

Besides aforementioned radio resource requirements provided by the slice owner, the resource allocation shall also respect the policy defined by the infrastructure provider, for instance, the allowable resource allocation types of underlying radio access technologies (RATs). Take the downlink (DL) resource allocation of LTE system for instance, 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 allocates 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). For uplink (UL), there are two resource allocation types: (a) UL type 0 allocates PRBG in a contiguous manner, and (b) UL type 1 allocates non-contiguous RBGs within two distinct clusters. Then, four abstraction types are introduced with RBG as the minimum resource granularity, and their respective mapping to the DL/UL resource allocation types are shown in TABLE V. Note that these DL/UL resource allocation types of the LTE system are the basis of the ones in 5G; hence, TABLE V will be further expanded to new allocation types. Moreover, like the virtual capacity in TABLE V, other types of resource can also be abstracted, e.g., virtual latency, to match further service requirements in terms of maximum allowed latency. Note that such virtual latency abstraction can be mapped to any resource allocation types (i.e., DL type 0/1/2 and UL type 0/1 of LTE system) but with a higher priority and a shorter allocation window than vTBS Type 0. In summary, the proposed vRBG and vTBS form a superset of legacy resource allocation types and they provide the required flexibility for both inter-slice resource partitioning and accommodation as detailed in Section V.

Fig. 5 illustrates an example of resource partitioning among four slices over an allocation window T with different types of abstraction. The proposed resource abstraction scheme allows the RAN runtime to dynamically change the mapping between different resource allocation types, for instance, changing allocation type 0/1/2 to allocation type 0/2 for slice 3 and 4 in Fig. 5. Such change can increase the flexibility for the infrastructure provider in resource allocation and do not impact the requested abstraction type by the slice owner.

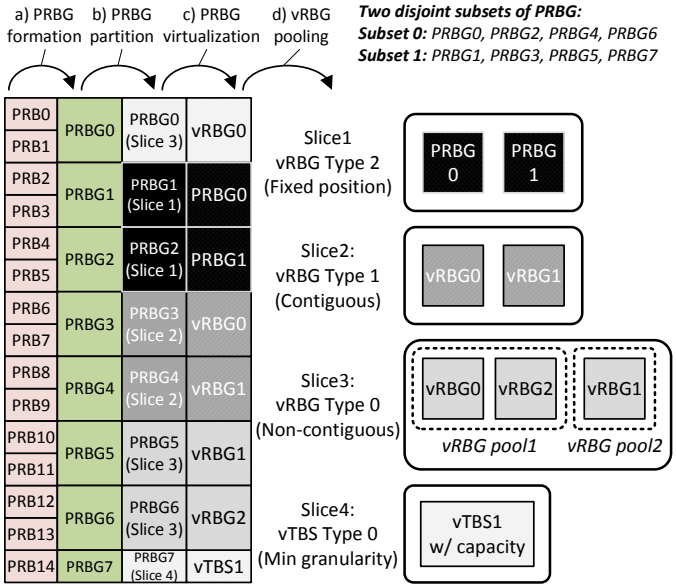
b) Radio resource abstraction: Based on aforementioned inter-slice resource partitioning, all available radio resources can be fragmented following different abstraction types. Such resource abstraction serves for two purposes: (1) isolate resources by presenting a virtual view of the resources that is decoupled from the exact physical locations, and (2) increase multiplexing gain by adjusting allocation types to share the unused resources. The former simplifies the inter-slice resource partitioning 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 increase the resource utilization efficiency (in favor of infrastructure provider). More specifically, we can observe in Fig. 6a that no other slices can utilize the unallocated resources even the traffic load variates from time to time when slice resources are not multiplexed. However, in Fig. 6b, the unallocated resources due to the time-varying load can be multiplexed to deploy more services at a single RAN infrastructure. Such multiplexing gain across tenants is anticipated by the infrastructure provider when deploying scalable numbers of slices.

Take the 3 MHz case of LTE system as an example in Fig. 7a, where the total PRB is 15 and the physical RBG (PRBG) granularity is 2 PRBs, giving a total of 8 PRBGs and the last PRBG only contains 1 PRB. These PRBGs are firstly partitioned for each slice based on the number of required resources and then they are abstracted according to the abstraction types, i.e., fixed, contiguous, non-contiguous, minimum granularity. Afterwards, the resulted PRBGs, vRBGs and vTBSs are provided to each slice for the intra-slice resource scheduling. For instance, fixed position resources is requested by slice 1 and hence no virtualization is performed (i.e., PRBG). While slice 4 requests a number of capacity, and thus its PRBGs are abstracted into vTBS with the capacity value computed from the measured channel state information.

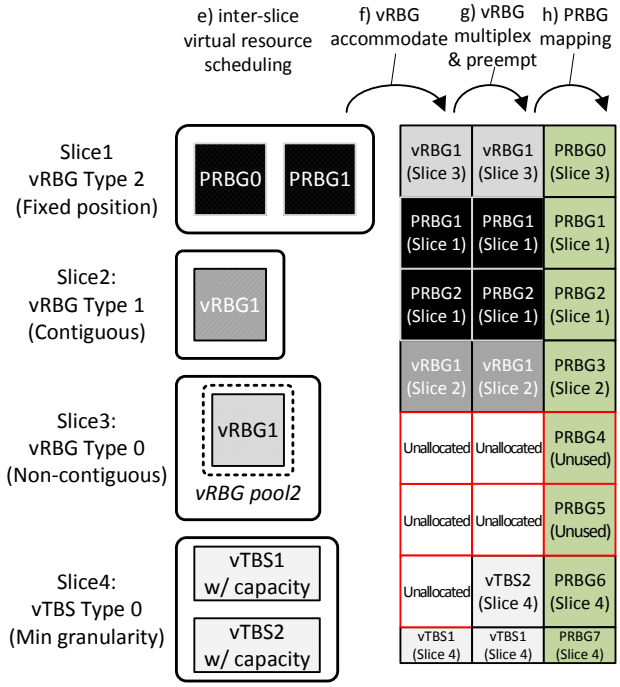
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

TABLE V: Mapping between resource abstraction type and allocation type.

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



(a) Stages form vRBG and vRBG pool



(b) Stages accommodate vRBG to PRBG

Fig. 7: Different stages for virtualized resources slicing.

The PRBGs of slice 2 and 3 are virtualized into vRBGs via abstracting the exact frequency/time locations as well as other dimensions (e.g., carrier frequency) and are pooled together to maintain the relative frequency dependencies among the virtualized resources without revealing any absolute physical relations. Take the slice 3 that uses resource allocation type 0 as an example, only PRBGs within the same subset can be scheduled at the same time. In that sense, vRBGs are *pooled* in order to indicate such exclusive condition between vRBG pool 1 (i.e., PRBG0, PRBG6) and vRBG pool 2 (i.e., PRBG 5), and thus the intra-slice resource scheduler of slice 3 will allocate resources to each user from either vRBG pool 1 or vRBG pool 2.

c) Radio resource accommodation and multiplexing: After the radio resource partitioning and abstraction, each slice can perform the intra-slice resource scheduling to its associated users and the scheduling decisions will be accommodated into PRBs as shown in Fig. 7b. Such accommodation does not necessary follow the partitioned resources of the inter-slice resource partitioning (cf. Fig. 7a) to better utilize available resources. For instance, the vRBG1 for both slice 2 and slice 3 are accommodated to their vRBG0 in the partitioning stage respectively so as to have a larger contiguous unallocated region (i.e., PRBG4 to PRBG6) that can be mapped to a larger SCS (e.g., 30 Hz) and be shared to other slices. For instance, the unallocated resource can be shared to other slices (e.g., vTBS2 of slice 4) that request more resources[5] or to some new services. Moreover, the preemption mechanism can also be applied by removing the inter-slice scheduling decisions of other low-priority slices to boost the perceived performance of high-priority slices[6]. Finally, the RAN runtime will allocate the corresponding control channel elements (CCEs) to transport the DL/UL control information (CI) based on aforementioned DL/UL resource allocation types. These CIs are used to indicate the user about the positions of allocated PRB as well as the necessary physical layer information (e.g., modulation and coding scheme [MCS], new data indication) for successful user data reception or transmission. With a limited control region to accommodate CCEs, the RAN runtime can also leverage the unallocated resources to carry these CIs.

4) Common control application: The common control applications provide a shared control logics for multiple slices. It can accommodate the customized control logics from different slice-specific control applications, resolve their conflicts, and enforce a feasible policy to underlying RAN module. For instance, the control logics of two customized RRM applications of slice 1 and slice 2 in Fig. 3 will leverage the inter-slice conflict resolution and control logics accommodation to provide their slice-specie control logics through the cell-common controller. Note that the policy for inter-slice conflict resolution is provided by the slice manager. Finally, the customized control logics of each slice will be applied in a unified manner toward the underlying RAN.

[5]Such multiplexing may not be allowed by slices with fixed position.

[6]The preemption characteristic shall be described in the slice context beforehand.

5) Forwarding engine: The forwarding engine manages the input and output streams of CP and UP, or simply data streams, between RAN and users across multiplexed and/or customized processing. Fig. 8 shows an example of how the forwarding engine manages the UP processing chain in the DL direction (i.e., from RAN to user) across several network layers: service data adaptation protocol (SDAP), PDCP, radio link control (RLC), medium access control (MAC), and physical (PHY)[7]. 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 slice manager. After the first stage of processing, the output flows are further forwarded to the corresponding entry points in the multiplexed chain (i.e., slice 2) or the output endpoint (i.e., slice 1). Note that more complex forwarding rules can be applied if the per-function customization is required, for instance, the customized MAC function to manage the intra-slice scheduling while multiplexing other functions. Further, the per-flow customization within a slice can be applied in order to differentiate the customized processing for flows with different QoS requirements. Such forwarding engine can leverage the match-action abstraction following SDN principles to establish the input/output forwarding path between the RAN runtime and slices in both directions [56], [57].

Furthermore, the forwarding engine is able to direct data not only in a monolithic RAN but also in a disaggregated RAN, where a single RAN module is decomposed into CU, DU, and RU with several possible functional splits in between [42]. Note that in the proposed RAN slicing model, RAN disaggregation and functional splits are controlled and maintained by the infrastructure provider, whereas the RAN service customization is managed by the slice owner. Fig. 9 shows the input/output forwarding path between CU, DU, and RU to compose a distributed UP processing chain using 3GPP function split [42] 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 for the slice owner[8]. Moreover, when adopting the flexible function split and placement [58], [59], the CP/UP state information has to be efficiently shared among disaggregated RANs to flexibly deploy and chain functions in between. TABLE VI summarizes the main UP state information that shall be maintained and shared in the slice data. Also note that these aforementioned chains are applied for the downlink direction, while the same forwarding engine can be utilized for uplink direction with different chain compositions.

D. RAN runtime APIs

The RAN runtime APIs are exposed both in the north-bound toward each slice and in the south-bound toward the underlying RAN module, allowing to manage a slice and control the underlying RAN module (cf. Fig. 4). In the north-bound, the RAN runtime slice APIs provides interfaces and communication channels to connect a slice to the RAN runtime as a separate process, whether it is local or remote. Hence, each slice can be executed in isolation from each other either at the host or guest level leveraging the well-know operating system (OS) and virtualization technologies, such as container or virtual machine. Such north-bound APIs allow the slice owner to register and consume the aforementioned RAN runtime services, manage its service in coordination with the RAN runtime and service orchestrator, and customize the CP/UP processing. In the south-bound, the RAN runtime CP/UP APIs enable a slice to take the control of its service by requesting virtualized/physical resources, applying its control decisions, and accessing the virtualized state information. When a slice is deployed locally, the RAN runtime APIs may exploit the inter-process communication mechanism to allow a slice to perform the real-time operation (e.g., MAC scheduling function) with hard guarantees (cf. slice 1 in Fig. 2). Remote slices, on the other hands, communicate with the RAN runtime through the asynchronous communication interface and can perform the non-time-critical operation (e.g., PDCP function) like slice 2 and 3 in Fig. 2.

E. Summary

In summary, the five proposed RAN runtime services can provide different levels of isolation and sharing and the correlated message flows between them are depicted in Fig. 10 Note that the message flows between these services and the slice data are omitted for simplicity. These message flows can be combined with other known mobile network messages (e.g., RAN or CN domain) to provide a complete set of slice-specific processing, e.g., the customized handover process between BSs tailored to slice service requirements. Further, they can be utilized for the service orchestration and management purpose to orchestrate virtualized infrastructures and VNFs for newly-instantiated services. More specifically, they can be utilized by ETSI Management and Orchestration (MANO) architectural framework to collect functional blocks, data repositories and related interfaces.

V. Resource Partitioning and Accommodation

In this section, we focus on the inter-slice radio resource partitioning and accommodation, as the intra-slice resource scheduling can utilize several known scheduling algorithms, such as proportional fair or round robin, configured by the slice orchestrator [11] to provide slice-specific customization. Specifically, in this section, we provide the algorithm of the inter-slice partitioning and accommodation, evaluate its performance, and formulate the overall multiplexing gain.

A. Inter-slice resource partitioning

The radio resources partitioned by the RAN runtime is periodically within an allocation window T (in millisecond [ms]) in the time domain and F (in Hz) in the frequency domain. These resources can be specifically quantized into

[7]Further function decompositions within the layer are possible like splitting the PHY into high-PHY and low-PHY.

[8]It can be customized for each service but needs the agreement between the slice owner and the infrastructure provider.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

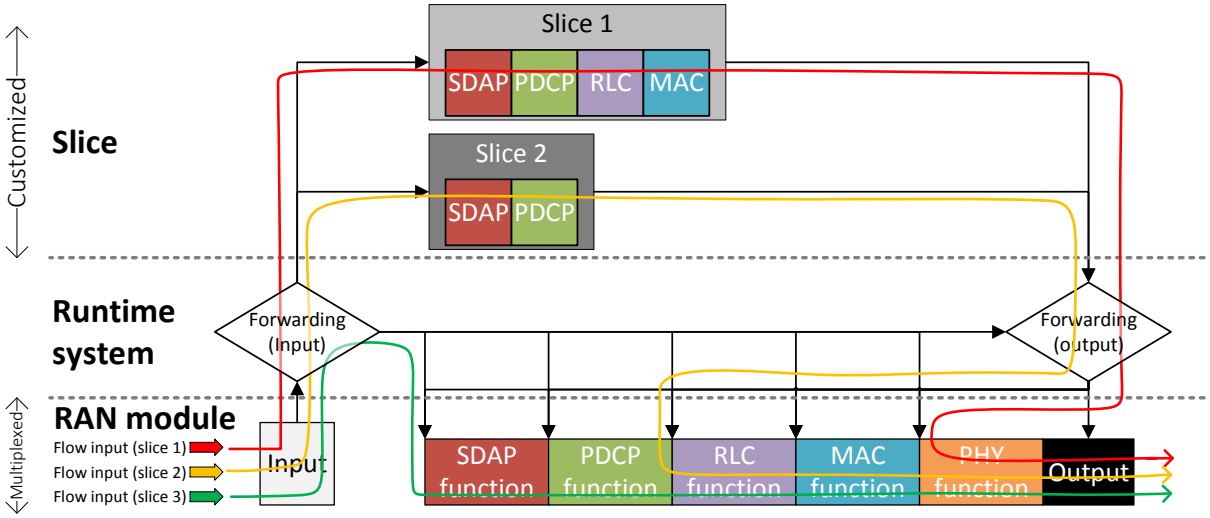


Fig. 8: Forwarding engine and UP processing chain.

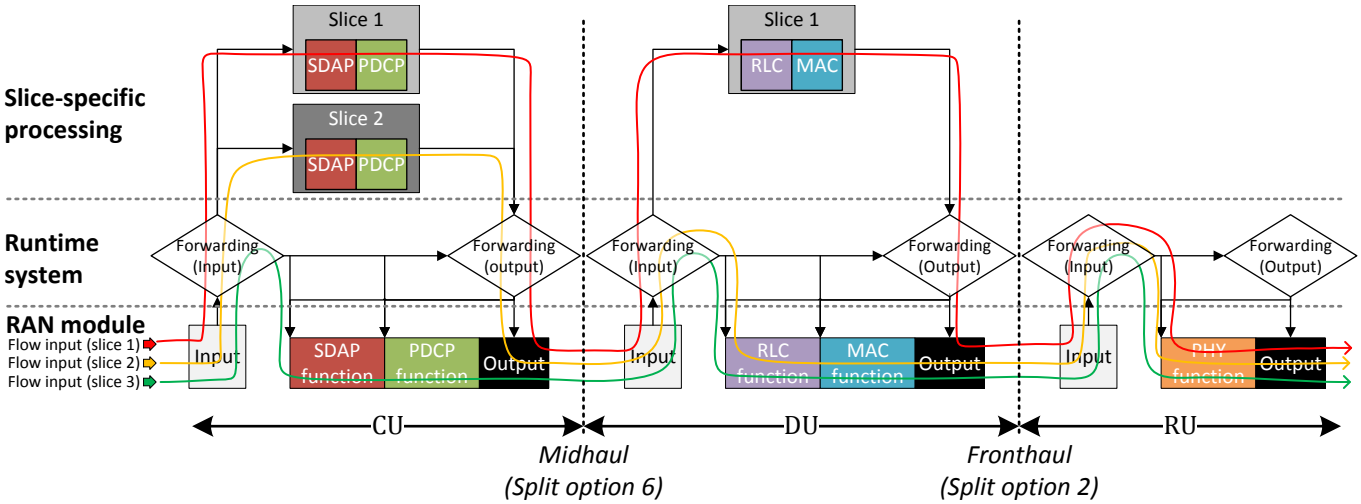


Fig. 9: UP forwarding path in three-tier disaggregated RAN (CU, DU, RU).

TABLE VI: UP network functions and the decoupled states

Layer	Network function	Network state
PHY	Radio frequency (RF) processing	Carrier frequency, Spectrum bandwidth
	(Inverse) Discrete Fourier Transform (DFT/IDFT)	Point of DFT, 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, and cycle redundancy check (CRC)
MAC	Hybrid automated repeated request (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 protocol data unit (PDU) and service data units (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 flows and DRBs Marking QoS flow identity	QoS flow identity, QoS profile, mapping policy QoS flow identity

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

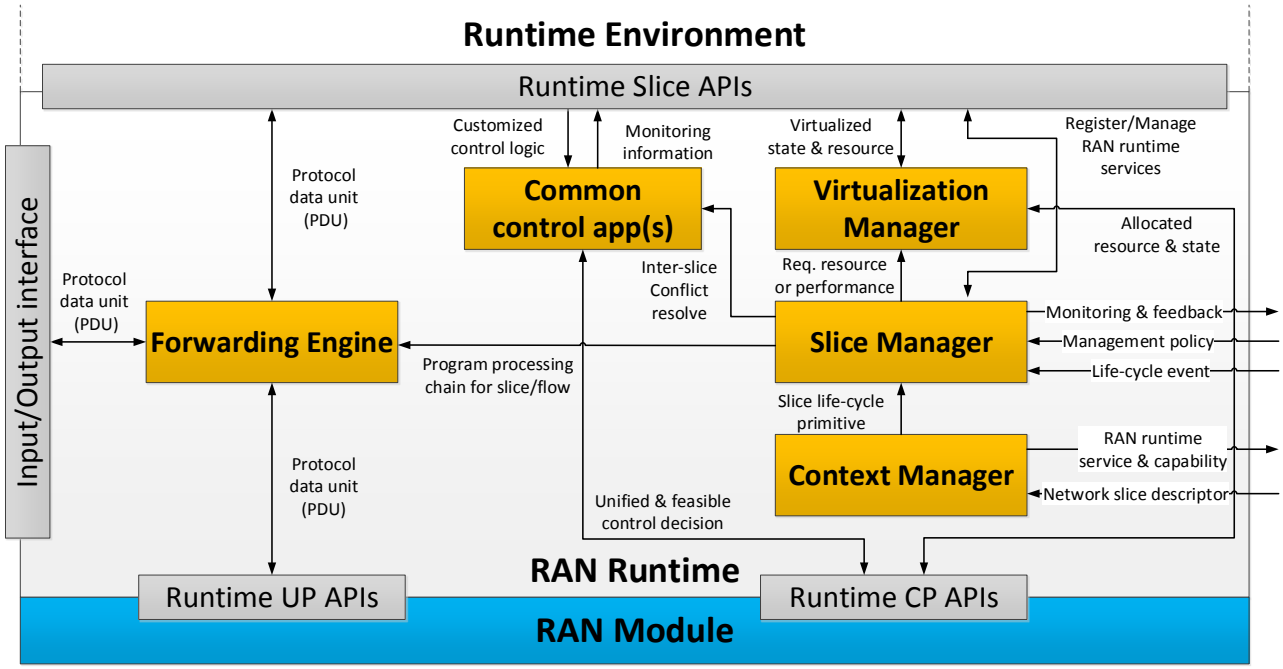


Fig. 10: Message flows between RAN runtime services.

a resource grid map Map with T_b TTIs in time domain and N_b PRBs in the frequency domain with respect to the base SCS (SCS_b) used by the infrastructure provider, e.g., a 20 MHz LTE radio bandwidth in a 10 ms allocation window is separated into 100 PRBs in frequency domain and 10 TTIs in time domain. There are $|S|$ slices that request the radio resources within the set $\mathcal{S} = \{s_1, \cdots, s_{|S|}\}$. For the k-th slice (i.e., s_k), its radio resource requirements include: (a) SCS_k set comprises the applicable SCSs, (b) T_k and N_k are the number of requested resource in time (ms) and frequency (Hz) domain respectively, and (c) g_k is the granularity which can be contiguous, non-contiguous, fixed position (with its fixed starting position denoted as FF_k and FT_k in frequency and time domain) or minimum granularity (with its request data rate as R_k) as mentioned in TABLE V. The fixed position granularity inherently isolates resources as its partitioned resources are physical ones without any virtualization. The contiguous one is more suitable for quasi-constant traffic patterns (e.g., streaming) since it can reduce the latency and minimize the CI signaling overhead. The non-contiguous one, on the other hand, accommodates better for variable traffic patterns as it can allocate fragmented resources. The minimum granularity can be utilized by those slices that request only capacity (i.e., vTBS), which allowing for all feasible partitioning.

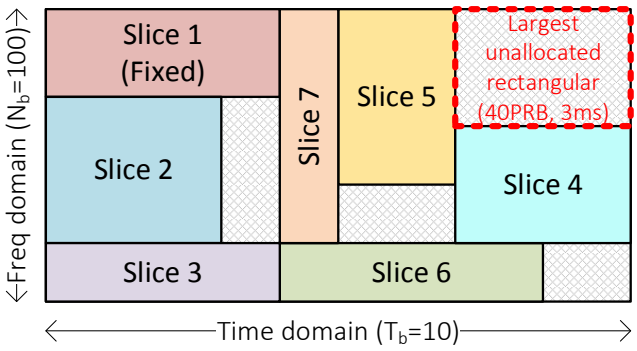
An example of the resource partitioning is depicted in Fig. 11 with 7 slices (i.e., $|S| = 7$). Each slice has different resource granularities: $g_1 = \text{Fix}$, $g_2 = g_3 = \text{Con}$, $g_4 = g_5 = \text{NonCon}$, and $g_6 = g_7 = \text{Min}$. The largest rectangular of the unallocated resource is highlighted, which is an important criterion for further resource multiplexing. Since such largest unallocated rectangular region may potentially fit in any data transportation numerology in time (i.e., TTI) and frequency domain (i.e., SCS), and can be either shared by different slices or utilized for CI transportation and BS broadcast information. Additionally, such radio resource defragmentation can provide a better slice performance in terms of delay and throughput. It is observed from Fig. 11a and Fig. 11b that although both resource partitions can satisfy the requested resources among all seven slices, while only the latter one can achieve a larger unallocated rectangular region. Such compact resource packing in Fig. 11b utilizes different resource granularities, i.e., s_4 and s_5 can be discontinuous in frequency and time separately, and s_6 and s_7 can leverage the minimum granularity.

Through such observation, the inter-slice resource partitioning has two complementary goals: (a) satisfy as many slice resource requests as possible, and (b) maximize the size of largest unallocated rectangular region. Practically, we can form such combinatorial objective function as

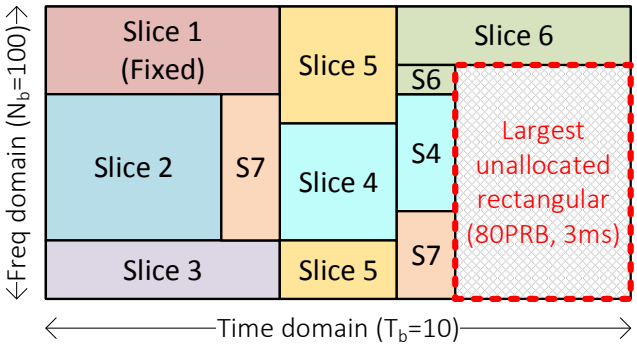
$$\max \left(\sum_{s_k \in \mathcal{S}} Sat\left[k\right] + w \cdot \text{MaxUn}\left(Map\right) \right), \tag{1}$$

where $Sat\left[k\right] \in \{0, 1\}$ is the satisfactory binary indicator for the k-th slice (e.g., $Sat\left[k\right] = 1, \forall s_k \in \mathcal{S}$ in Fig. 11a and 11b as all slices are satisfied), $\text{MaxUn}\left(\cdot\right)$ function outputs the largest unallocated rectangular in the resource grid allocation map Map (e.g., $\text{MaxUn}\left(Map\right)$ of Fig. 11b is twice as the value of Fig. 11a), and a weight w can balance these two objectives. Such problem can be mapped to the NP-hard two-dimensional knapsack problem, which makes the complexity to find the optimal solution be non-polynomial [60], [61]. Hence, finding the optimal inter-slice resource partition is cost and time prohibitive when the number of slice increases. Prior works provide the heuristic algorithms [62], [63], but they only focus on one special case that considers a single SCS with contiguous granularity. We hereby propose the granularity-based heuristic algorithm that can sequentially partition resources as explained in the following paragraphs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access



(a) Contiguous granularity



(b) Slice-specific granularity

Fig. 11: Examples of radio resource partitioning.

1) Proposed algorithm: The overall proposed algorithm is presented in Alg. 1 that sequentially prioritizes the k-th slices, i.e., s_k, based on the prioritization policy (i.e., $priority$) and then partitions resources according to its granularity, i.e., g_k. As each slice can support more than one SCSs, the remapping operation from the base SCS (SCS_b) to another SCS (scs) is necessary for the number of requested resources (F_{scs}, T_{scs}) and fixed position (FF_{scs}, FT_{scs}) through the scsMap $(\cdot)$ function shown in Alg. 1. Note that the requested data rate R_k can be mapped to the number of requested radio resources using the per-slice channel state information (CSI), i.e., CSI_k[9] as well as the corresponding MCS index. Moreover, the granularity-based partitioning algorithms include the ones for the fixed position (Alg. 2), contiguous (Alg. 3), non-contiguous (Alg. 4) and minimum granularity (Alg. 5). Afterwards, a resource grid remapping through the $RGMap(\cdot)$ function in Alg. 1 aims to map the resource grid from the selected SCS for the k-th slice (i.e., $SCS[k]$) to other SCSs. Finally, all satisfied slices after partitioning are included in the set $\mathcal{S}_p$.

When applying the fixed position algorithm (cf. Alg. 2), the FindFRe $(\cdot)$ function checks the feasibility of the fixed position allocation (i.e., starts from $FF_{scs}[k]$ and $FT_{scs}[k]$ in frequency and time domain respectively) and outputs 1 when feasible (0 otherwise). While the FindRe $(\cdot)$ function is used in the contiguous algorithm (cf. Alg. 3) and it outputs a set of 2-tuples comprising all possible contiguous positions in frequency and time domain, respectively. Specifically, PF set comprises first entry of the 2-tuple set, while PT set includes the second entry. Then, we pick the position with the largest unallocated rectangular using the aforementioned MaxUn $(\cdot)$ function over the resource grid allocation map. In non-contiguous algorithm (cf. Alg. 4), the FindUnRe $(\cdot)$ function outputs all available positions in a set of 2-tuples (i.e., PF set includes the first entries and PT set contains the second entries) without requiring a contiguous portion. Then, we allocate sequentially in time domain following the decreasing order of available resources over the frequency domain using the sorting function sort $(\cdot)$ shown in Alg. 4. Specifically, all possible time indexes (i.e., from 1 to $T_b \cdot \frac{scs}{SCS_b}$) are ranked based on the number of available frequency domain resource (i.e., aF). The minimum granularity algorithm of Alg. 5 also applies the same FindUnRe $(\cdot)$ function to find all available positions and uses InMaxRec $(\cdot)$ to check that these available positions are within the largest rectangular region (output 1 in In) or not (0 otherwise). Finally, all possible positions (i.e., indexed from 1 to $Size$) are sorted in the ascending order based on whether they are in the maximum rectangular or not (i.e., In) for later resource partitioning.

2) Complexity analysis: The overall inter-slice resource partition algorithm of Alg. 1 is composed of four granular-specific ones as shown from Alg. 2 to Alg. 5. In following paragraphs, we firstly analyze the complexity of each granular-specific algorithm and then summarize the overall complexity.

In Alg. 2, the most complex operation is to find the largest rectangular in the resource grid, i.e., MaxUn $(\cdot)$, for all available SCSs, and thus its complexity equals to $\mathcal{O}(|SCS| \cdot (N_b \times T_b))$. In Alg. 3, the complexity is proportional to the number of available SCSs, the size of possible locations (i.e., $|PF|$), and the operation to find the largest rectangular. In the worst case, $|PF|$ equals to the size of resource grid; therefore, the complexity of Alg. 3 is written as $\mathcal{O}\left(|SCS| \cdot (N_b \times T_b)^2\right)$. The complexity of Alg. 4 depends on the operation of finding the largest rectangular as well as the sorting operation. The former complexity is proportional to $(N_b \times T_b)$ as mentioned beforehand, while the latter is proportional to T_b^2 in the worst case as there are T_b elements to be sorted. Thus, its complexity will be $\max\left(\mathcal{O}(|SCS| \cdot (N_b \times T_b)), \mathcal{O}\left(|SCS| \cdot T_b^2\right)\right)$. Furthermore, the most complex operation of Alg. 5 is to sort all available positions, i.e., $|PF|$ elements, and thus the complexity of such algorithm can be written as $\mathcal{O}\left(|SCS| \cdot (N_b \times T_b)^2\right)$.

In summary, the complexity of the overall algorithm in Alg. 1 is proportional to (1) the number of slices, i.e., $|\mathcal{S}|$, (2) the slice prioritization policy, and (3) the highest complexity among the aforementioned four algorithms. Note that only constant time will be spent when a predefined priortization policy is applied (e.g., according to the SLAs)[10]. Thus, the overall complexity is written as $\mathcal{O}\left(|\mathcal{S}| \times |SCS| \times (N_b \times T_b)^2\right)$. Further, as the number of eligible SCS numerologies is limited, e.g., up to 5 allowed SCSs defined by 3GPP in [55], we can further write the overall complexity as $\mathcal{O}\left(|\mathcal{S}| \times (N_b \times T_b)^2\right)$.

[9]It can base on the average CSI among its served users.

[10]Extra complexity is required when adopting dynamic search in the policy.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

Algorithm 1: Inter-slice Resource Partition Algorithm

Input : T_b and N_b are resource grid size in time and frequency
$\mathcal{S}$ is the set of slices
Output: Map is the resource grid allocation map
SCS is the set of applied SCS of each slice
$\mathcal{S}_p$ is the satisfied slice set

begin
  $\mathcal{S}_p = \varnothing$; /* Initialize the satisfied slice set */
  foreach $s_k \in \mathcal{S}$ **do**
    $Sat\,[i] = 0$; /* Initialize satisfaction index of each slice */
    $SCS\,[i] = 0$; /* Initialize select SCS of each slice */
    foreach $scs \in SCS_k$ **do**
      /* Map request resource and fixed position to all SCSs.*/
      $[F_{scs}\,[i]\,, T_{scs}\,[i]]$=scsMap $(N_k, T_k, R_k, CSI_k, scs, SCS_b)$;
      $[FF_{scs}\,[i]\,, FT_{scs}\,[i]]$=scsMap $(FF_i, FT_i, 0, 0, scs, SCS_b)$;
  foreach $scs \in SCS$ **do**
    for $i = 1$ **to** $N_b \cdot scs/SCS_b$ **do**
      for $j = 1$ **to** $T_b \cdot SCS_b/scs$ **do**
        $Map_{scs}\,[i]\,[j]$=0 ; /* Reset resource grid allocation */
  while *isempty* $(\mathcal{S}) ==$ *false* **do**
    s_k = prioritize $(\mathcal{S}, priority)$; /* Get most prioritized slice */
    switch g_k **do**
      case *Fix* **do**
        $[Sat\,[k]\,, SCS\,[k]\,, Map]$ = FPos (s_k, Map) ; (cf. Alg. 2)
      case *Con* **do**
        $[Sat\,[k]\,, SCS\,[k]\,, Map]$ = Con (s_k, Map) ; (cf. Alg. 3)
      case *NonCon* **do**
        $[Sat\,[k]\,, SCS\,[k]\,, Map]$ = NCon (s_k, Map) ;(cf. Alg. 4)
      case *Min* **do**
        $[Sat\,[k]\,, SCS\,[k]\,, Map]$ = Min (s_k, Map) ; (cf. Alg. 5)
    if $Sat\,[k] == 1$ **then**
      Map=RGMap$(Map, SCS\,[k])$;/* Remap grid to all SCSs */
      $\mathcal{S}_p$ = SetUnion $(\mathcal{S}_p, s_k)$; /* Add slice into satisfied set */
    $\mathcal{S}$ = SetDiff $(\mathcal{S}, s_k)$; /* Remove prioritized slice */

Algorithm 2: Fixed Position Resource Partition (FPos)

Input : s_k is target slice
$IMap$ is the input resource grid allocation map
Output: Sat is the slice satisfaction index
SCS is the selected SCS for the target slice
$OMap$ is the output resource allocation map

begin
  $MaxRec = 0$; /* Initialize the maximum unused rectangular */
  $Sat = OptSCS = 0$; /* Initialize satisfaction index and select SCS */
  foreach $scs \in SCS_k$ **do**
    if *FindFRe*$(F_{scs}\,[k]\,, T_{scs}\,[k]\,, scs, IMap_{scs}, FF_{scs}\,[k]\,, FT_{scs}\,[k])$ **then**
      $Sat = 1$; /* Current slice is satisfied*/
      $tMap = IMap_{scs}$;
      for $i = 0$ **to** $F_{scs}\,[k] - 1$ **do**
        for $j = 0$ **to** $T_{scs}\,[k] - 1$ **do**
          $tMap\,[i + FF_{scs}\,[k]]\,[j + FT_{scs}\,[k]] = k$;
      $tRec$ = MaxUn $(tMap)$; /* Find max unused rectangular */
      if $tRec > MaxRec$ **then**
        $SCS = scs$;
        $MaxRec = tRec$;
        $OMap_{scs} = tMap$;

3) Performance evaluation: As mentioned before, the sequential resource partitioning is based on the prioritization policy (i.e., $priority$ in Alg. 1); hence, high priority slices will impact the available positions for low priority ones. In the following, the performance of five different priortization policies are evaluated:

1) *Optimal*: Search all possible permutations to get the best ordering in terms of the objective function in Eq. (1).

Algorithm 3: Contiguous Resource Partition (Con)

Input : s_k is target slice
$IMap$ is the input resource grid allocation map
Output: Sat is the slice satisfaction index
SCS is the selected SCS for the target slice
$OMap$ is the output resource allocation map

begin
  $MaxRec = 0$; /* Initialize the maximum unused rectangular */
  $Sat = SCS = 0$; /* Initialize satisfaction index and select SCS */
  foreach $scs \in SCS_k$ **do**
    /* Find possible positions PF/PT in time/freq domain */
    $[PF, PT]$ = FindRe $(F_{scs}\,[k]\,, T_{scs}\,[k]\,, scs, IMap_{scs})$;
    for $p = 1$ **to** $|PF|$ **do**
      $Sat = 1$; /* Current slice is satisfied*/
      $tMap = IMap_{scs}$;
      for $i = 0$ **to** $F_{scs}\,[k] - 1$ **do**
        for $j = 0$ **to** $T_{scs}\,[k] - 1$ **do**
          $tMap\,[i + PF\,[p]]\,[j + PT\,[p]] = k$;
      $tRec$ = MaxUn $(tMap)$; /* Find max unused rectangular */
      if $tRec > MaxRec$ **then**
        $SCS = scs$;
        $MaxRec = tRec$;
        $OMap_{scs} = tMap$;

Algorithm 4: Non-contiguous Resource Partition (NCon)

Input : s_k is target slice
$IMap$ is the input resource grid allocation map
Output: Sat is the slice satisfaction index
SCS is the selected SCS for the target slice
$OMap$ is the output resource allocation map

begin
  $MaxRec = 0$; /* Initialize the maximum unused rectangular */
  $tIdxCount = 0$; /* Initialize time index counter */
  $Sat = SCS = 0$; /* Initialize satisfaction index and select SCS */
  foreach $scs \in SCS_k$ **do**
    /* Find unused resources position (PF/PT) in $IMap_{scs}$ */
    $[PF, PT]$ = FindUnRe $(IMap_{scs})$;
    for $j = 1$ *to* $T_b \cdot scs/SCS_b$ **do**
      $aF\,[j]$=find $(PT == j)$; /* Count avail resources at time j */
      if $|aF\,[j]| \geq F_{scs}\,[k]$ **then**
        $tCount = tCount + 1$; /* Increase time index counter */
    if $tCount \geq T_{scs}\,[k]$ **then**
      $Sat = 1$; /* Current slice is satisfied*/
      $tMap = IMap_{scs}$;
      /* Sort time indexes base on descending order of aF */
      $Torder$ = sort $\left(1 : T_b \cdot \frac{scs}{SCS_b}, aF, \text{'descend'}\right)$;
      for $j = 1$ *to* $T_{scs}\,[k]$ **do**
        $FIdx = PF\,[aF\,[Torder\,[j]]]$;
        for $i = 1$ **to** $F_{scs}\,[k]$ **do**
          $tMap\,[i + FIdx\,[i]]\,[Torder\,[j]] = k$;
      $tRec$ = MaxUn $(tMap)$; /* Find max unused rectangular */
      if $tRec > MaxRec$ **then**
        $SCS = scs$;
        $MaxRec = tRec$;
        $OMap_{scs} = tMap$;

2) *Random*: Randomize the slice ordering in each allocation window T.
3) *Greedy*: Use the greedy method to prioritize the slice that can generate the largest unallocated rectangular.
4) *Granularity*: Sort slices based on the their granularities in the following order: fixed position, contiguous, non-contiguous, and minimum granularity.
5) *Granular & Greedy*: Use the two-sequential sorting, in which the first sort is based on *granularity* and the second is based on the *greedy* method.

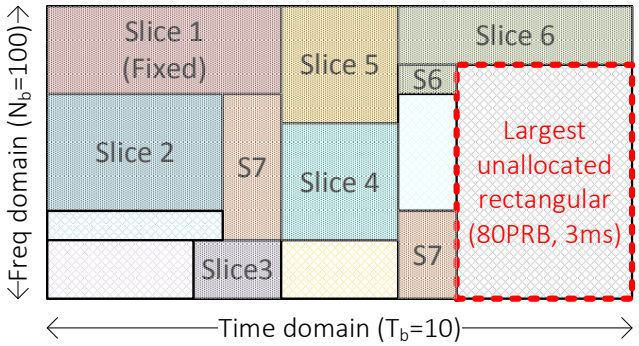
Algorithm 5: Min granularity Resource Partition (Min)

Input : s_k is target slice
$IMap$ is the input resource grid allocation map
Output: Sat is the slice satisfaction index
SCS is the selected SCS for the target slice
$OMap$ is the output resource allocation map

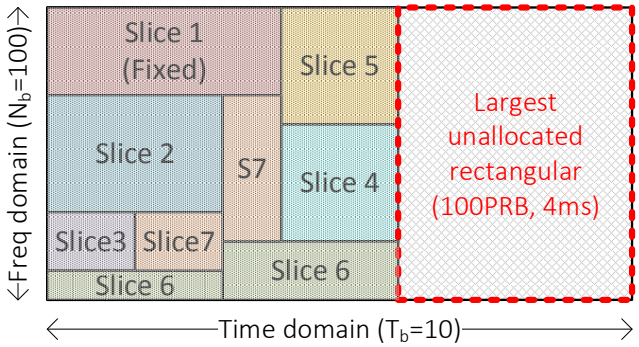
begin
  $MaxRec = 0$; /* Initialize the maximum unused rectangular */
  $Sat = SCS = 0$; /* Initialize satisfaction index and select SCS */
  foreach $scs \in SCS_k$ **do**
    $[PF, PT] =$ FindUnRe $(IMap_{scs})$;
    $Size = |PF|$; /* The size of all available positions */
    $[In] =$ InMaxRec (PF, PT) ;
    /* Sort resource base on whether it is in the largest rectangular*/
    $Order =$ sort $(1 : Size, In,$ 'ascend') ;
    if $|PF| \geq F_{scs}[k] \times T_{scs}[k]$ **then**
      $Sat = 1$; /* Current slice is satisfied*/
      $tMap = IMap_{scs}$;
      for $pos = 1$ **to** $F_{scs}[k] \times T_{scs}[k]$ **do**
        $tMap[PF[Order[pos]]][PT[Order[pos]]] = k$;
      $tRec =$ MaxUn $(tMap)$; /* Find max unused rectangular */
      if $tRec > MaxRec$ **then**
        $SCS = scs$;
        $MaxRec = tRec$;
        $OMap_{scs} = tMap$;

The evaluation results are shown in Fig. 13 with 7 slices. Each slice can serve a number of users and it requests a time-varying uniformly-distributed aggregated resources with $N_k \sim$ Uniform $(1.6, 9)$ MHz and $T_k \sim$ Uniform $(1, 10)$ ms, $\forall s_k \in \mathcal{S}$. Note that the granularities of all seven slices are the same as the ones shown in Fig. 11, and the applicable SCS set for the k-th slice is $SCS_k = \{15, 30, 60\}$ kHz, $\forall s_k \in \mathcal{S}$. As for the RAN infrastructure, the radio bandwidth is 20 MHz with $F_{base} = 15$ kHz and allocation window is 10 ms with $T_{base} = 1$ ms. Fig. 13a then shows the slice satisfaction ratio for all seven slices or for each granularity type (i.e., fixed, contiguous, non-contiguous and minimum). The *optimal* policy reaches the highest satisfaction ratio (82% on average for all 7 slices) but with much higher time complexity (e.g., 1 day for the considered scenario). From the figure, one can observe that the *Granular & Greedy* one (81%) outperforms the others and is very close to the optimal policy as it not only follows the elasticity of resource granularity (i.e., granularity) but also seeks for the largest unallocated region (i.e., greedy) at the meantime.

Moreover, the resource grid utilization ratio over the resource grid allocation map Map is shown in Fig. 13b with three components: (1) the partitioned resources, (2) the largest unallocated rectangular, and (3) other unallocated resource in box plot. Both *random* and *greedy* policies have a larger unallocated rectangular ratio (20% and 23% on average) at the cost of a significantly lower slice satisfaction ratio (73% and 71% on average) shown in Fig. 13a, i.e., more unallocated resources are due to the lower slice satisfaction ratio. Conversely, the percentage of the largest unallocated rectangular is close between the case that uses the optimal policy (12%) and the case that applies the *Granular & Greedy* policy (10%), which confirms the performance of the proposed algorithm. Finally, the *Granular & Greedy* policy takes much less execution time, i.e., polynomial time, to provide such close performance, which justifies its efficiency and applicability.



(a) Intra-slice scheduling results



(b) Compact inter-slice accommodation

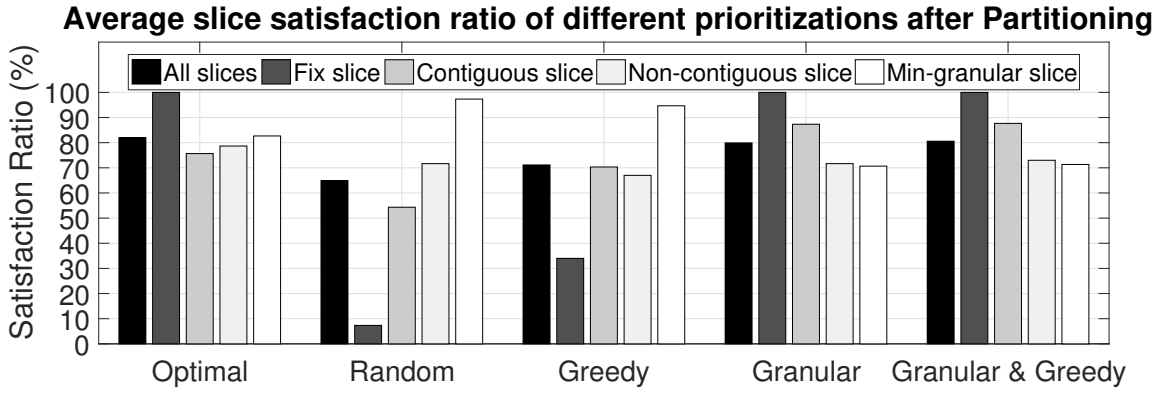
Fig. 12: Examples of inter-slice resource accommodation.

B. Radio resource accommodation

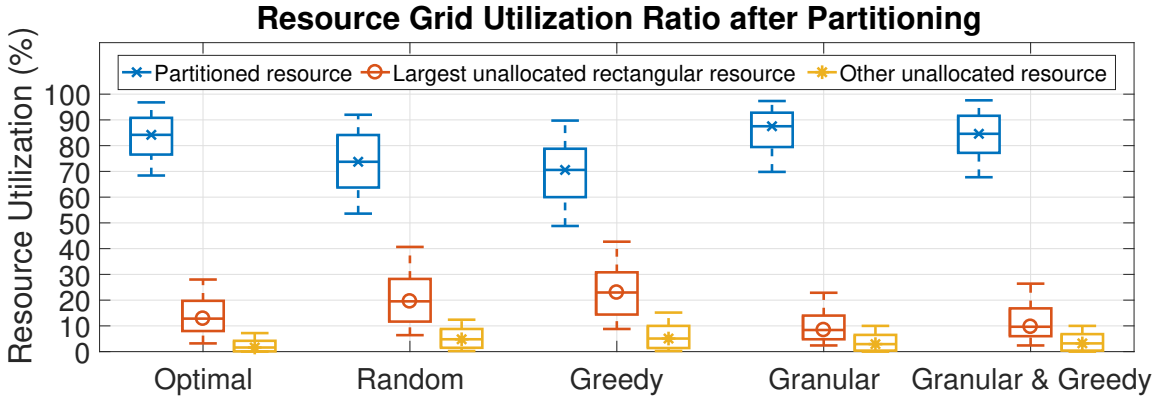
After the inter-slice partitioning and intra-slice scheduling, the RAN runtime can accommodate these scheduling decisions to physical resources and generate the corresponding CI (cf. step f in Fig. 7b). In Fig. 12a, an example is shown based on the outcomes of inter-slice partitioning (cf. Fig. 11b) as well as intra-slice scheduling. The intra-slice scheduling decisions are marked with the gray portions as the scheduled parts, while the transparent portions are the unscheduled resources (i.e., unused by the intra-slice scheduler). However, a larger unallocated rectangular is formed in Fig. 12b via accommodating the per-slice scheduling results in a more compact way. Such compactness relies on the *resource abstraction* mechanism as mentioned in Section IV-C3. Through such scheme, the inter-slice accommodation is not necessary mapped to the same physical partitioned resource except for the slices that request the fixed position granularity, i.e., $g_k =$ Fix.

Like the inter-slice resource partitioning, our objective here contains the two complementary goals in (1). Hence, we can apply almost the same algorithm as in Alg. 1 with following modifications: (1) adopt the SCS selected in the inter-slice resource partitioning, i.e., $SCS[k]$, $\forall s_k \in \mathcal{S}$ outputted from Alg. 1, (2) prohibit other slices to utilize the resources partitioned for the fixed-position granularity slice, and (3) replace the number of requested resource (i.e., T_k and N_k) with the number of scheduled resource (i.e., T_k^a and N_k^a are the number of scheduled resources in time and frequency domain for s_k). The aggregated traffic arrival rate of each slice is assumed to be proportional to the number of requested radio resource (i.e., $N_k \times T_k$) that is further multiplied with a time-varying uniformly-distributed *traffic arrival ratio* p.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

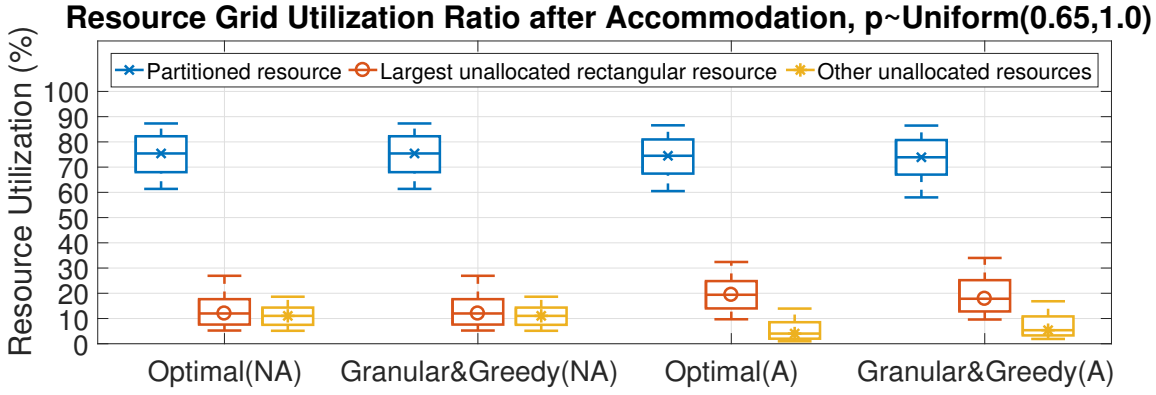


(a) Average slice satisfaction ratio

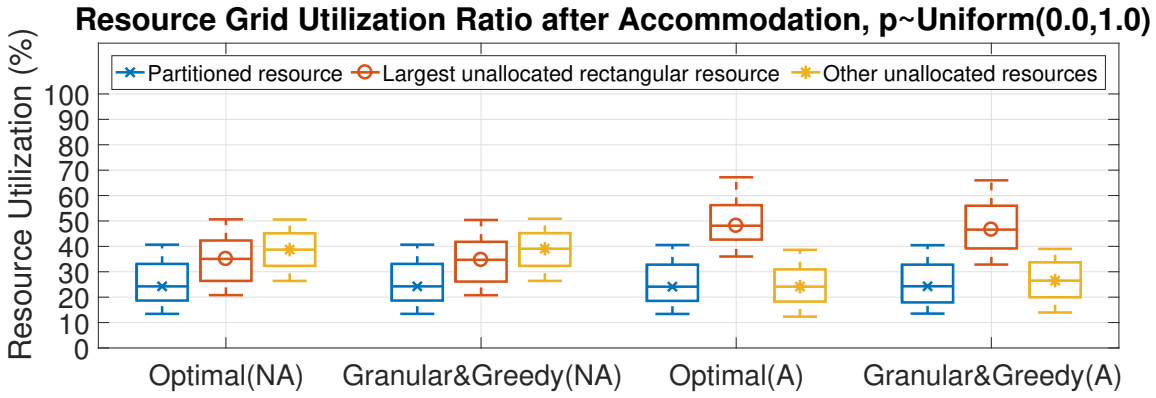


(b) Resource grid occupation ratio

Fig. 13: Performance of different slice priortization policies in resource partitioning.



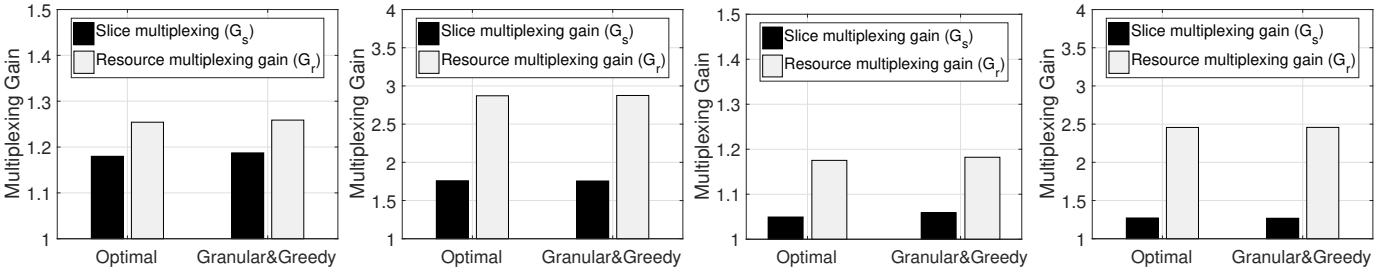
(a) Traffic arrival ratio $p \sim$ Uniform $(0.65, 1.0)$



(b) Traffic arrival ratio $p \sim$ Uniform $(0.0, 1.0)$

Fig. 14: Performance of different slice priortization policies and resource abstraction in resource accommodation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access



(a) Average multiplexing gain with $p \sim$ Uniform $(0.65, 1.0)$
(b) Average multiplexing gain with $p \sim$ Uniform $(0.0, 1.0)$
(c) Worst case multiplexing gain with $p \sim$ Uniform $(0.65, 1.0)$
(d) Worst case multiplexing gain with $p \sim$ Uniform $(0.0, 1.0)$

Fig. 15: Slice and radio resource multiplexing gain among different cases.

We hereby evaluate the performance of two slice priortization policies, i.e., *Optimal*, *Granular & Greedy*, in Fig. 14 considering two cases: (a) no resource abstraction (denoted with NA in Fig. 14), and (b) resource abstraction is applied except for fixed-position slices (denoted with A in Fig. 14). In the former case, all intra-slice scheduling decisions are accommodated within the partitioned resource (e.g., Fig. 12a), while the latter allows more freedom when accommodating (e.g., Fig. 12b). In Fig. 14a, we can see that no abstraction case only shows $\sim 2\%$ increasing in terms of the largest unallocated rectangular when comparing with the *Granular & Greedy* inter-slice partitioning result (cf. Fig. 13b). In contrast, with resource abstraction scheme, the optimal and *Granular & Greedy* priortization policies provides $\sim 9.8\%$ and $\sim 8.2\%$ enhancement, respectively. Such benefit is further boosted when the average traffic arrival ratio p is decreased as shown in Fig. 14b, i.e., p is changed from Uniform $(0.65, 1.0)$ to Uniform $(0.0, 1.0)$. These results show the resource abstraction advantages in terms of defragmenting the overall resource grid. Finally, more slices can be further satisfied and we denote the final set of satisfied slices after accommodation and multiplexing as $\mathcal{S}_a$.

C. Multiplexing gain

To explicitly represent the level of multiplexing benefits, we formulate the statistical multiplexing gain in two aspects: (1) slice and (2) radio resource block. First of all, based on the results shown in the previous two paragraphs, we can see that the number of satisfied slices after the inter-slice partitioning (i.e., $|\mathcal{S}_p|$) is smaller than the number of satisfied slices after the inter-slice accommodation and multiplexing (i.e., $|\mathcal{S}_a|$) via utilizing the unallocated resource. Hence, the statistical multiplexing gain in the slice aspect can be written as

$$G_s = \frac{\text{Number of satisfied slices after accommodation and multiplexing}}{\text{Number of satisfied slices after partitioning}} = E\left[\frac{|\mathcal{S}_a|}{|\mathcal{S}_p|}\right], \tag{2}$$

where $\mathcal{S}_p$ and $\mathcal{S}_a$ are introduced in the previous two paragraphs respectively. Another aspect is to view the multiplexing gain in terms of the radio resource block via dividing the number of utilized resource blocks after accommodation and multiplexing by the number of scheduled resource blocks after the intra-slice resource scheduling:

$$G_r = \frac{\text{Number of utilized resource after accommodation and multiplexing}}{\text{Number of intra-slice scheduled resources}} = E\left[\frac{\sum_{s_k \in \mathcal{S}_a} T_k^a \cdot N_k^a}{\sum_{s_k \in \mathcal{S}_p} T_k^a \cdot N_k^a}\right], \tag{3}$$

where T_k^a and N_k^a are the number of allocated resource blocks. Note that these two multiplexing gain formulations in Eq. (2) and Eq. (3) depend not only on the results of inter-slice resource partitioning and accommodation but also on the characteristics of extra slices to be satisfied, i.e., their traffic patterns and resource granularities.

In Fig. 15, we show the multiplexing gain based on the inter-slice partitioning and accommodation results (i.e., optimal and *Granular & Greedy* in Fig. 14), and utilize the aforementioned traffic pattern, i.e., *traffic arrival ratio* p, as introduced in the previous paragraph for each extra slice. Two resource granularity cases are considered for each extra slice: (1) contiguous granularity, and (2) random granularity between contiguous, non-contiguous and minimum. The former shows that the multiplexing gain in its *worst case* as all extra slices require the contiguous resources, while the latter one shows the *average* multiplexing gain. The average multiplexing gain in Fig. 15a is approximately 1.18 (G_s) and 1.26 (G_r) for both optimal and *Granular & Greedy* policies. When the average traffic arrival ratio is decreased (i.e., p), more unused resources can be multiplexed, and hence the multiplexing gain shown in Fig. 15b are increased to 1.75 (G_s) and 2.87 (G_r). Note that the resource multiplexing gain is more significantly increased than the slice multiplexing gain as there are more unused resources can be multiplexed.

Further, even the multiplexing gains of both traffic arrival ratios are reduced when considering the worst case in Fig. 15c (1.05/1.18 for G_s/G_r) and Fig. 15d (1.27/2.45 for G_s/G_r); however, both priortization policies still show close values. Note that such worst case results provide the lower bounds of multiplexing gain. When comparing the average and the worst cases, the slice multiplexing gain is reduced more obviously than the resource multiplexing gain as extra slices are rejected mostly due to their requested contiguous granularity rather than the lack of radio resources. In summary, the multiplexing gain is represented in both slice and resource block aspects. The former is more related to the resource granularity (e.g., contiguous, non-contiguous, minimum), while the latter one concerns more on the traffic pattern (i.e., N_k, T_k, p).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

VI. Proof of Concepts

To validate the concept of RAN runtime slicing system and explore different use cases, we implemented an LTE-based prototype of RAN runtime following aforementioned design in Section IV. The RAN runtime is developed based on the FlexRAN agent[11] over the OAI platform [15], and each instantiated slice is built on top of the FlexRAN controller[12] with the customized CP, UP, and CL. The main functionalities of the proposed RAN runtime services and CP/UP APIs are implemented and integrated within the agent. Slice selection for each user is done based on the the public land mobile network (PLMN) information, as a part of the unique international mobile subscriber identity (IMSI), in order to allow each user to associate to a slice. Note that as specified by 3GPP in [64], a single-network slice selection assistance information (S-NSSAI) can identify a slice and it comprises the (1) slice/service type (SST) and (2) slice differentiator (SD) to differentiate the slice service. Then, the user can send the NSSAI information that includes up to 8 S-NSSAIs to identify it preference(s) for slice selection.

The current implementation of a slice service descriptor is shown in Listing 1. It can describe a slice by its BS name (i.e., `name`), cell identifier (i.e., `cell_id`), and service types (i.e., `service_types`), where each service is defined by a set of service policies (i.e., `service_policy`) in both downlink and uplink directions that will be applied when a slice is created or updated. Specifically, the service policies are expressed in terms of (1) the number of requested resources (i.e., vRBGs) and performance (i.e., rate and latency), (2) slice isolation requirement, and (3) slice priority as shown in listing 1. In terms of the requested resource abstraction types, currently the vRBG type 0/1 and vTBS type 0 are available and they can utilize the downlink and uplink resource allocation type 0 (see TABLE V). Hence, each slice will be associated with a vRBG pool in each TTI, and the overall resource partitioning is updated in every allocation window T. Note that the slice isolation property (cf. `requested_isolation`) can allow a slice to reserve its resources (i.e., without any multiplexing), whereas the slice priority (cf. `requested_priority`) is used to accommodate the resources and to preempt resources from other slices.

Using the aforementioned slice service descriptor, three slices are created on the top of the a single BS. They communicate with the RAN runtime using the asynchronous communication channels. Each slice embeds the control logics and operates on the virtualized resources and states based on the modified version of FlexRAN controller and its software development kit (SDK). In following, we describe the experiment setup for each use case and present the respective results demonstrating the slice performance tradeoff between isolation/sharing as well as the flexibility in terms of changing the RAN service definition dynamically.

[11]https://gitlab.eurecom.fr/oai/openairinterface5g
[12]https://gitlab.eurecom.fr/flexran/flexran-rtc

Listing 1: Slice service descriptor.

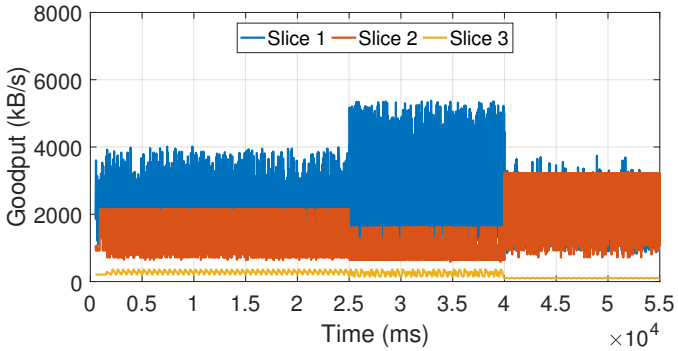
```
enb_slices :
  - name: BS1
    cell_id: val
    service_types: [ST1, ST2, ST3]
  - name: BS2
    ...
service_policy :
    ST1 :
        UL :
            requested_vrbg: val
            requested_rate: val
            requested_latency: val
            requested_priority: val
            requested_isolation:  val
        DL :
            requested_vrbg: val
            requested_rate: val
            requested_latency: val
            requested_priority: val
            requested_isolation:  val
    ST2:
        ...
    ST3:
        ...
```

A. Radio Resource and Control Logic Isolation

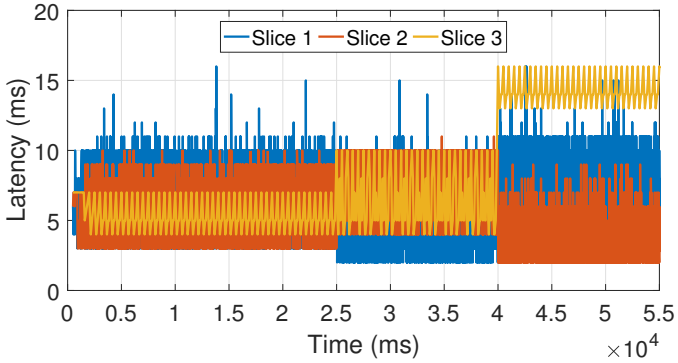
To demonstrate the impact of inter-slice resource partitioning, we deploy three slices with different traffic patterns as follows: slice 1 with a variable bit rate emulating 720p video streaming, slice 2 with compressed variable bit rate emulating a surveillance IP camera with 30 frame per second (FPS), and slice 3 with constant low bit rate emulating periodical sensing data. Each slice serves 5 different users (i.e., user 1 to user 5 belong to slice 1, user 6 to user 10 belong to slice 2, and user 11 to user 15 belong to slice 3), and each user transmits uplink and downlink data on the default radio bearer. Then, three different resource partitioning manners are applied at different time intervals: (a) fair partitioning that allocates 33% of total vRBGs to each slice before time instance $t_1 = 25s$, (b) greedy partitioning between t_1 and $t_2 = 40s$ that allocates 60% of vRBGs to slice 1, and 20% per slice 2 and slice 3, and (c) proportional partitioning after t_2 that allocates 50% of vRBGs to slice1, 40% to slice 2, and 10% for slice 3. Note that our focus in this experiment is on the number of requested resources (i.e., `requested_vrbg`), and thus the impacts of priority and isolation are not taken into account. As for the intra-slice scheduling, we apply a simple fair scheduling among users.

From the results presented in Fig. 16, it can be observed that the slice aggregated good-put and average latency can significantly fluctuate when applying different inter-slice resource partitioning policies. However, any change of the inter-slice partitioning has no impact on the intra-slice scheduling policy as shown in Fig. 17, in which all users are scheduled and the fairness is preserved. The above results confirm the capability of RAN runtime in providing isolation among slices and performance guarantee, matching the challenge listed in Section IV-A. Further, it implies that the inter-slice partitioning and intra-slice scheduling can be decoupled and developed individually for different purposes to meet the requirements of both infrastructure provider and slice owner.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

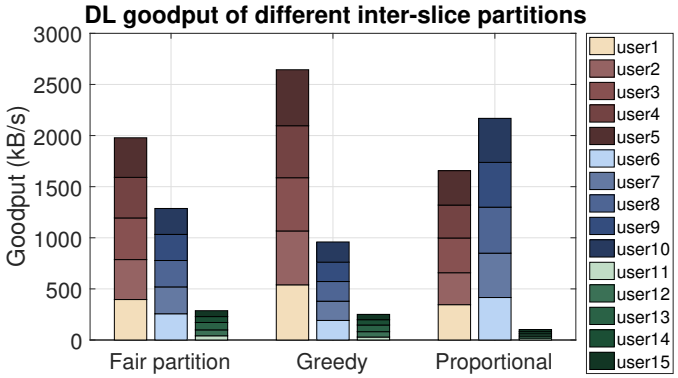


(a) Aggregated good-put

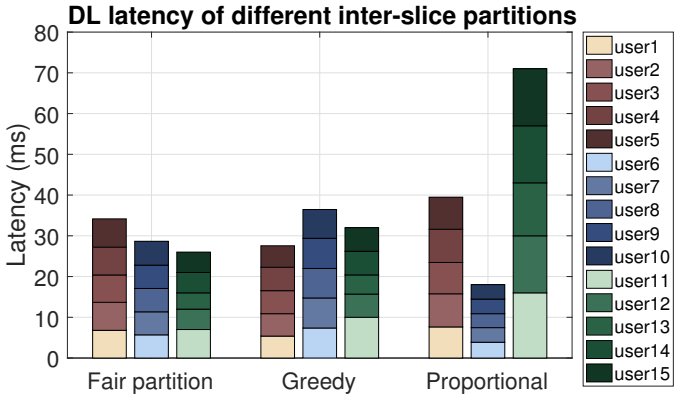


(b) Average latency

Fig. 16: Slice performance of dynamic inter-slice partitioning.



(a) Per-user average Good-put



(b) Per-user average latency

Fig. 17: User performance of dynamic inter-slice partitioning.

B. Radio Resource Preemption and Multiplexing

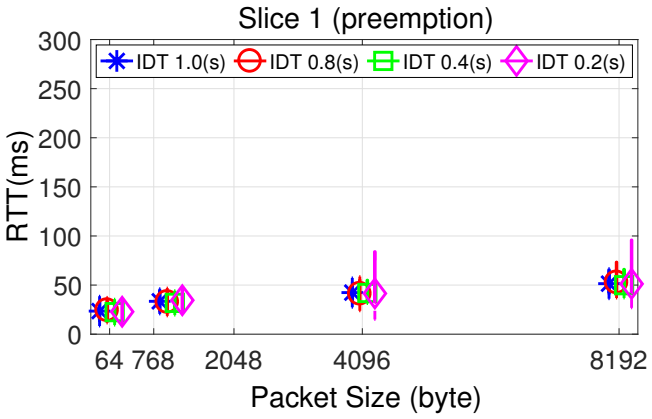
In this experiment, we demonstrate the impacts of resource multiplexing and preemption, i.e., `requested_priority` and `requested_isolation` in Listing 1, on the perceived performance in a scenario with three slices, each hosting one user. Specifically, besides the applied resource abstraction/virtualization scheme, different slice service policies are explored: (a) slice 1 can preempt resources of all other slices when the actual (aggregated) rate exceeds the requested rate, (b) slice 2 can only increase its multiplexing gain by utilizing the unallocated resources, and (c) slice 3 may sustain its requested data rate as it can neither preempt nor multiplex resources but is subject to the preemption from high priority slice (i.e., slice 1).

We firstly show the box plot of measured round trip time (RTT) distribution in Fig. 18 with different packet size (PS) ranging from 64 to 8192 bytes and inter-departure time (IDT) from 0.2 to 1 second. We can observe that the smallest RTT with the lowest variability is achieved for slice 1, as such slice has the ability to preempt resources from others, and hence it can utilize available radio resources to meet its instantaneous traffic dynamics. Slice 2 is able to maintain the average RTT compared with slice 1 with opportunistic improvement when there are some unallocated resources to be multiplexed (cf. Fig. 7b). However, it suffers from delay variability caused by the scheduling delay. Slice 3 experiences the largest average RTT (almost twice as slice 1) with the highest variability, and it represents a typical best effort service. Besides, the relations between the measured RTT and the characteristics of traffic (i.e., PS and IDT) are observed as the following. We can see that there is a positive correlation between RTT and PS for slice 1, as such slice does not experience any scheduling delay due to the resource preemption scheme, and thus the RTT is only proportional to the size of packet. As for slice 3, an extra positive correlation is observed between the IDT and the measured RTT, i.e., the higher IDT has a higher RTT. The reason being that the longer IDT traffic of slice 3 suffers from the scheduling delay as it can neither preempt others' resource nor multiplex unused resources to reduce its RTT. By contrast, for slice 2, there is no straightforward relation between IDT and RTT since it can utilize some unused resource opportunistically.

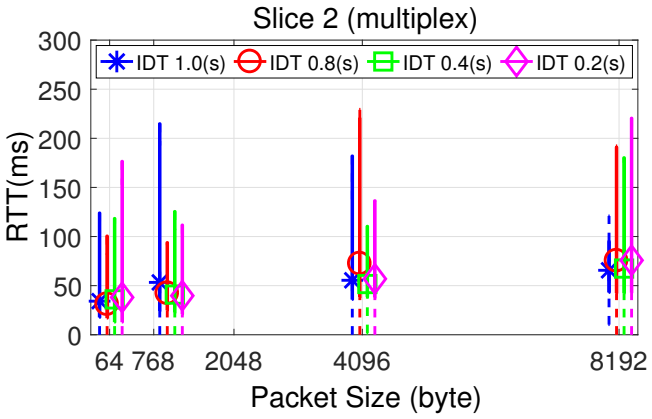
When examining the slice aggregated good-put and delay-jitter in Fig. 19, it can be seen that slice 1 can flexibly adapt its data rate as a function of its workload by preempting the resources from other slices, i.e., from 3 Mbps to 6 Mbps, while slice 2 experiences a data rate drop from its desired 10 Mbps to 8 Mbps. The same trend is observed in the delay jitter measurement, in which slice 1 experiences the minimum jitter as it has the highest priority and slice 3 suffers the largest delay jitter due to its lowest priority.

The above results reveal that the impact of slice policy in terms of the multiplexing and priority when creating a slice. They enable resource reservation and preemption to potentially meet the slice-specific QoS requirements as well as the resource multiplexing to increase the efficiency of resource utilization by sharing the unused resources.

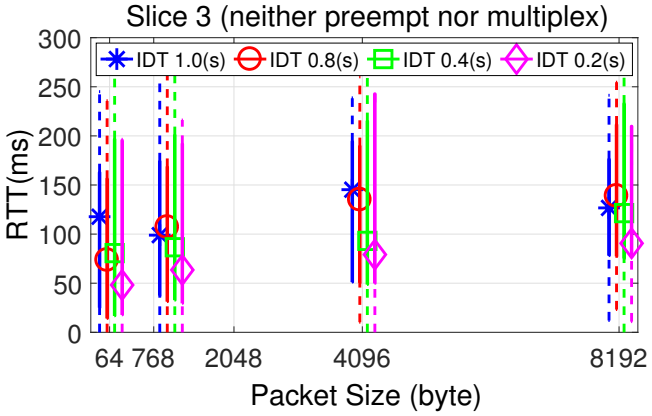
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access



(a) Slice 1



(b) Slice 2



(c) Slice 3

Fig. 18: Impact of preemption and multiplexing on RTT.

C. Network function and state flexibility

We then show the capability of the RAN runtime to change the service definition of the underlying RAN module between monolithic and disaggregated deployments from the infrastructure provider perspective. In particular, we consider three possible RAN deployments at different time instances without instantiating any slice: (a) monolithic RAN deployment at t_1, (b) disaggregated RAN deployment using 3GPP split option 8 [42] at t_2, and (c) using 3GPP split option 7-1 at t_3. Such BS only has a single antenna and it is operated in Frequency-Division Duplexing (FDD) mode with 5 MHz radio bandwidth. Our considered disaggregated RAN deployment uses UDP/IP based Ethernet transportation over the fronthaul interface with one switch between RU and DU to route the traffic. The UP measurement results are shown in Fig. 20 in terms of the good-put, delay jitter and RTT when a 15 Mbps traffic flow is transferred in the DL direction. We can see that these is no good-put drop when changing the functional split. This is because the considered splits (i.e., performing cell processing at the RU) only require 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 the Ethernet packet loss when changing the split as well as the extra time spent for the Ethernet packet transport (i.e., packetization [65] and radio sample compression/decompression) along the fronthaul links and the switch.

We have to mention that although the changes of functional split are mainly reserved for the infrastructure provider to ensure the network service performance in the devised approach, while the update of split can be made possible for a slice owner by appropriately customizing the CP/UP functions at each RAN module. In such a case, the RAN runtime shall make sure that the SLA is maintained when there is any change in the service service descriptor, and transfer the CP/UP states between disaggregated RAN modules (i.e., RU, DU, CU), as listed in TABLE VI.

VII. Conclusions and Future works

In this work, we propose the RAN runtime slicing system that serves as a flexible execution environment to run multiple customized slice instances with the required levels of isolation while sharing the underlying RAN modules and infrastructure. We elaborate on the design of such system and identify the its functionalities in both control and user planes. 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 RAN runtime is introduced to compose the input and output data stream for a flexible processing pipeline composition. We also propose the inter-slice resource partitioning and accommodation approach that can satisfy the requests of different granularities and maintain a significant multiplexing gain with acceptable complexity. Finally, we implement the proposed RAN runtime slicing system over the OAI platform in three use cases that exactly match aforementioned RAN slicing challenges.

In the future, we plan to extend the current work in several directions: (1) extend the resource abstraction approach to support additional performance metrics (e.g., latency, reliability), (2) formulate the QoS satisfaction objective when partitioning/accommodating radio resources, (3) examine the performance impact on the function dedication/sharing on different network layers, and (4) establish a collaboration scheme between multiple RAN runtime instances to enable the large-scale control logics.

Acknowledgement

This work has received funding from the European Union's Horizon 2020 Framework Programme under grant agreement No. 762057 (5G-PICTURE) and No. 761913 (SliceNet).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2847610, IEEE Access

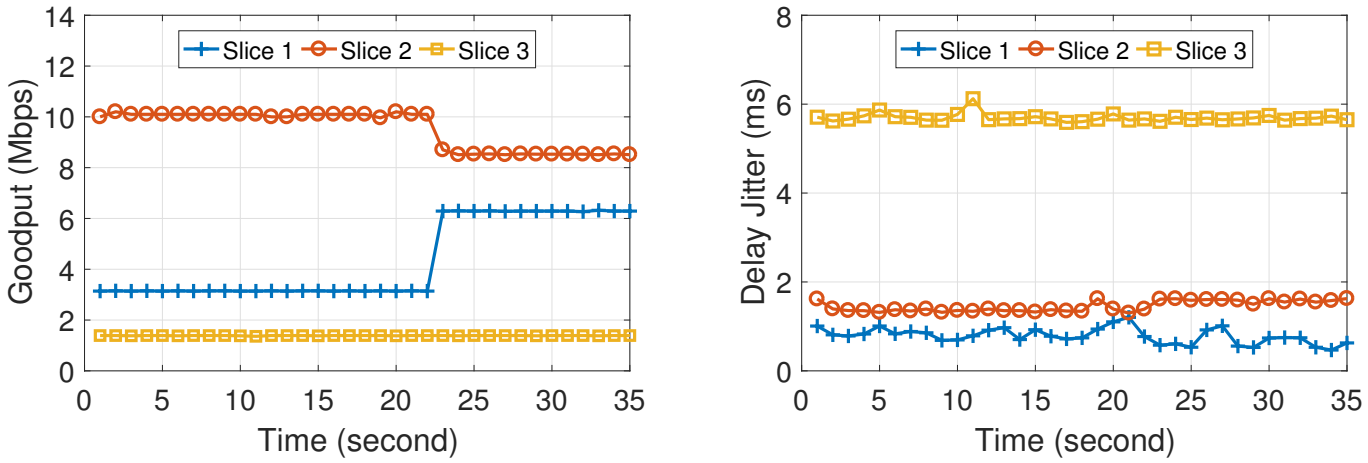


Fig. 19: Impact of preemption and multiplexing on good-put and delay jitter.

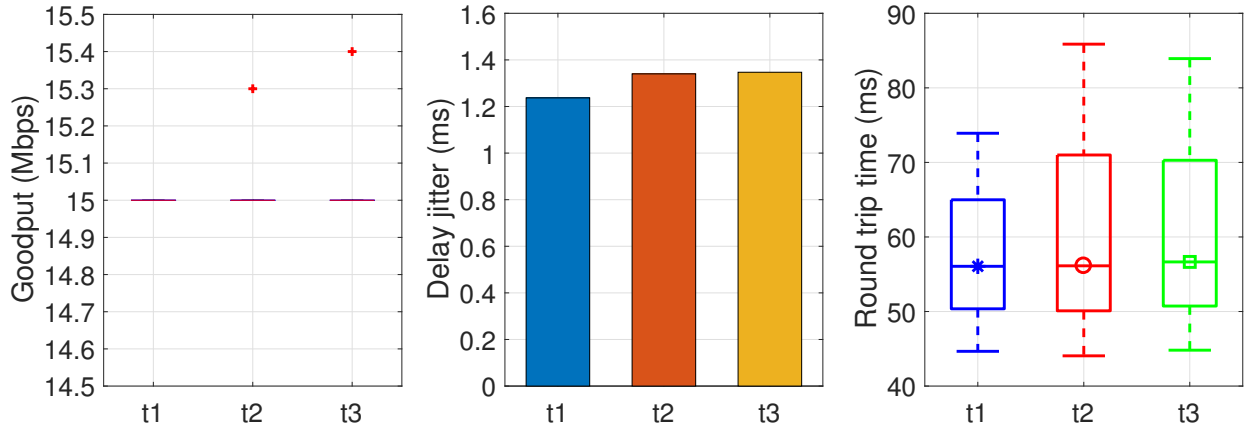


Fig. 20: Flexible RAN deployment impacts on good-put, delay jitter and RTT.

REFERENCES

[1] *TR 28.801 Study on management and orchestration of network slicing for next generation network (Release 15)*, 3GPP, Sep. 2017.

[2] 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.

[3] 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, May 2016.

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

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

[6] NGMN Alliance, "Description of Network Slicing Concept," Tech. Rep., Jan. 2016.

[7] 5G PPP Architecture Working Group, "View on 5G Architecture," *White Paper*, Jul. 2016.

[8] V. G. Nguyen and Y. H. Kim, "Slicing the Next Mobile Packet Core Network," in *2014 11th International Symposium on Wireless Communications Systems (ISWCS)*, Aug. 2014, pp. 901–904.

[9] 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, Mar. 2015.

[10] 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*, ser. SIGCOMM '17. ACM, 2017, pp. 348–361.

[11] 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, Aug. 2017.

[12] X. Foukas, M. Mahesh K., and K. Kontovasilis, "Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '17. ACM, 2017, pp. 127–140.

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

[14] *TR 23.711 Enhancements of Dedicated Core Networks selection mechanism (Release 14)*, 3GPP, Sep. 2016.

[15] 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, Oct. 2014.

[16] 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 *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16. ACM, 2016, pp. 427–441.

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

[18] 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, pp. 134–142, Oct. 2011.

[19] 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, Mar. 2014.

[20] 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*, ser. MobiArch '15. ACM, 2015, pp. 8–13.

[21] 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, pp. 146–154, Aug. 2017.

[22] X. An, R. Trivisonno, H. Einsiedler, D. von Hugo, K. Haensge, X. Huang, Q. Shen, D. Corujo, K. Mahmood, D. Trossen, M. Liebsch, and C.-T. Phan, "End-to-End Architecture Modularisation and Slicing for Next Generation Networks," *CoRR*, vol. abs/1611.00566, 2016. [Online]. Available: http://arxiv.org/abs/1611.00566

[23] 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, Jul. 2016.

[24] 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, Mar. 2016.

[25] 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, Jul. 2016.

[26] 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, Aug. 2017.

[27] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. Leung, "Network Slicing Based 5G and Future Mobile Networks: Mobility, Resource Management, and Challenges," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 138–145, Aug. 2017.

[28] A. Rostami, P. Ohlen, K. Wang, Z. Ghebretensae, B. Skubic, M. Santos, and A. Vidal, "Orchestration of RAN and Transport Networks for 5G: An SDN Approach," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 64–70, Apr. 2017.

[29] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, "NVS: A Substrate for Virtualizing Wireless Resources in Cellular Networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1333–1346, Oct. 2012.

[30] 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, Jul. 2013.

[31] R. Mahindra, M. A. Khojastepour, H. Zhang, and S. Rangarajan, "Radio Access Network Sharing in Cellular Networks," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Oct. 2013, pp. 1–10.

[32] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, "CellSlice: Cellular Wireless Resource Slicing for Active RAN Sharing," in *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*, Jan. 2013, pp. 1–10.

[33] J. He and W. Song, "AppRAN: Application-Oriented Radio Access Network Sharing in Mobile Networks," in *2015 IEEE International Conference on Communications (ICC)*, Jun. 2015, pp. 3788–3794.

[34] A. Aijaz, "Hap − SliceR: A Radio Resource Slicing Framework for 5G Networks With Haptic Communications," *IEEE Systems Journal*, pp. 1–12, Jan. 2017.

[35] 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, Aug. 2011.

[36] C. Liang and F. R. Yu, "Wireless Virtualization for Next Generation Mobile Cellular Networks," *IEEE wireless communications*, vol. 22, no. 1, pp. 61–69, Feb. 2015.

[37] A. Gudipati, L. E. Li, and S. Katti, "RadioVisor: A Slicing Plane for Radio Access Network," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. ACM, 2014, pp. 237–238.

[38] 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, May 2017.

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

[40] 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, Nov. 2016.

[41] K. Katsalis, N. Nikaein, E. Schiller, R. Favraud, and T. I. Braun, "5G Architectural Design Patterns," in *2016 IEEE International Conference on Communications Workshops (ICC)*, May 2016, pp. 32–37.

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

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

[44] M. Yang, Y. Li, D. Jin, L. Su, S. Ma, and L. Zeng, "OpenRAN: A Software-defined Ran Architecture via Virtualization," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 549–550, Aug. 2013.

[45] 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.

[46] 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*, ser. HotSDN '13. ACM, 2013, pp. 25–30.

[47] 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, Dec. 2014.

[48] 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*, ser. HotSDN '12. ACM, 2012, pp. 109–114.

[49] 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*, ser. HotNets-XIII. ACM, 2014, pp. 6:1–6:7.

[50] O. Sallent, J. Perez-Romero, R. Ferrus, and R. Agusti, "On Radio Access Network Slicing from a Radio Resource Management Perspective," *IEEE Wireless Communications*, vol. 24, no. 5, pp. 166–174, Oct. 2017.

[51] 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, Apr. 2015.

[52] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless Network Functions: Breaking the Tight Coupling of State and Processing," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, 2017, pp. 97–112.

[53] J. Kim, D. Kim, and S. Choi, "3GPP SA2 Architecture and Functions for 5G Mobile Communication System," *ICT Express*, vol. 3, no. 1, pp. 1–8, 2017.

[54] A. A. Zaidi, R. Baldemair, H. Tullberg, H. Bjorkegren, L. Sundstrom, J. Medbo, C. Kilinc, and I. Da Silva, "Waveform and Numerology to Support 5G Services and Requirements," *IEEE Communications Magazine*, vol. 54, no. 11, pp. 90–98, Nov. 2016.

[55] *TS 38.211 NR; Physical channels and modulation (Release 15)*, 3GPP, Dec. 2017.

[56] Open vSwitch. [Online]. Available: http://openvswitch.org/

[57] 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," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 99–110, Aug. 2013.

[58] 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 *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7.

[59] O. Arouk, N. Nikaein, and T. Turletti, "Multi-Objective Placement of Virtual Network Function Chains in 5G," in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, Sep. 2017, pp. 1–6.

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

[61] C.-Y. Chang, N. Nikaein, and T. Spyropoulos, "Radio Access Network Resource Slicing for Flexible Service Execution," in *2018 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, May 2018.

[62] J. Van De Belt, H. Ahmadi, and L. E. Doyle, "A Dynamic Embedding Algorithm for Wireless Network Virtualization," in *2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall)*, Sep. 2014, pp. 1–6.

[63] M. Yang, Y. Li, L. Zeng, D. Jin, and L. Su, "Karnaugh-map Like Online Embedding Algorithm of Wireless Virtualization," in *The 15th International Symposium on Wireless Personal Multimedia Communications*, Sep. 2012, pp. 594–598.

[64] *TS 23.501 System Architecture for the 5G System; Stage 2 (Release 15)*, 3GPP, Jul. 2017.

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