Radio Access Network Slicing System

Robert Schmidt, Navid Nikaein
Communication Systems Department, EURECOM, France
Email: {robert.schmidt, navid.nikaein}@eurecom.fr

Word count: 9.000

Abstract

Network slicing is one of the key enablers to provide the required flexibility for the envisioned service-oriented 5G. We introduce a descriptor, a triple consisting of resources, processing and state, as a means to describe base stations (BS) and the embedded slices alike through a unifying description. Second, we propose a RAN slicing system, composed of the RAN runtime execution environment and accompanying controller based on this descriptor. This includes design and performance details of the employed system. Finally, we elaborate on aspects of RAN slicing such as the radio resources, the processing chain of a slice, and its state.

CONTENTS

I. INTRODUCTION

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 deployments), or be shared across all types of resources including radio spectrum and network functions (e.g., all network layers of the protocol stack). A slice might also be customized for a subset of user plane (UP) and control plane (CP) processing with an access to a portion of virtualized radio resources, enabling access to these resources by multiple tenants. Different levels of isolation and sharing across domain-specific resources should be supported. These domains could be on the network level (e.g., radio access network, core network, transport network), on the employed radio access technology (RAT), or administrative (e.g., multiple operators), and shareable resources might include computing, storage, network, hardware, radio, spectrum, network functions, applications, etc.

Radio access network (RAN) slicing is a challenge, as different levels of isolation and sharing, and slice owner customization with respect to UP and CP should be supported within the already complex RAN. Furthermore, radio resources are scarce, and therefore the resource utilization has to be very high. To this end, 3rd Generation Partnership Project (3GPP) mentions RAN slicing realization principles in (3GPP 2017a; 3GPP 2017b) including slicing with RAN awareness, quality of service (QoS) support, resource isolation, service-level agreement (SLA) enforcement and others.

RAN slicing has attracted much attention from researchers and several works study the RAN slicing vision. RadioVisor (Gudipati et al. 2014) can isolate control channel messages, elementary resources such as CPU and radio resource to provide customized services for each slice. A fully isolated solution such as FLARE (Nakao et al. 2017) exposes different virtual base stations (BSs) representing different slices; however, there is no benefit in terms of radio resource allocation multiplexing and network function sharing. The Hap-SliceR radio resource slicing framework (Aijaz 2018) considers resource utilization and slice utility requirements; however, its main focus is on the resource customization for haptic communications. Marabissi et al. (2017) propose a RAN slicing architecture among infrastructure and spectrum for both public safety and commercial services via resource virtualization and dedicated control functions. Rost et al. (2017) separate the radio resource scheduling of a BS 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. Ksentini et al. (2017) propose a RAN slicing architecture allowing radio resource management (RRM) policies to be enforced at the level of physical resource blocks (PRBs) through providing virtual resource blocks (vRBs) by a novel resource visor toward each slice; however, it neither considers functional isolation nor resource customization/abstraction per slice request. Sallent et al. (2017) compare different approaches to split radio resources in terms of the resource granularity and the degrees of isolation and customization; nonetheless, the resource multiplexing capability among slices is not considered. The Orion solution (Foukas et al. 2017) introduces the BS hypervisor to simultaneously isolate slice-specific control logics and share the radio resources. Moreover, the underlying PRBs are grouped into vRBs to be provided only to the corresponding slice. Such work exploits the prerequisites of function isolation and resource virtualization, while it does not consider customization of CP/UP functions in both monolithic and disaggregated RANs. Ferrus et al. (2018) base the proposed RAN slicing framework on service descriptions to flexibly share RAN functions over different network layers; however, it only considers physical resource partitioning without resource virtualization and multiplexing. Finally, the RAN runtime slicing system (Chang et al. 2018b) provides a flexible execution environment to run multiple customized slice instances with the required levels of isolation while sharing the underlying RAN modules and infrastructure in both monolithic and customized manner. The network virtualization substrate (NVS) (Kokku et al. 2012) can efficiently utilize available radio resources among multiple slices. The NetShare approach (Mahindra et al. 2013) extends the NVS approach by applying a central gateway-level component to ensure resource isolation and to optimize resource distribution for each entity. Kokku et al. (2013) propose the CellSlice architecture as a gateway-level solution that can indirectly impact individual BS scheduling decision for slice-specific resource virtualization. He et al. (2015) provide the AppRAN as the application-oriented framework that defines a serial of abstract applications with distinct QoS guarantees. The works of Zaki et al. (2011) provide functional isolation in terms of customized and dedicated CP functionalities for each mobile virtual operator. Nikaein et al. (2015) propose a slice-based "network store" architecture as a platform to facilitate the dynamic network slicing based on the virtual network functions (VNFs) on top of the underlying infrastructures.

Table I summaries the literature on network slicing and compares the works with respect to the radio resource sharing and the multiplexing of CP/UP functionality.

The software-defined networking (SDN) and network function virtualization (NFV) concepts, applied to the RAN, lead to the concept of software-defined RAN (SD-RAN), enabling programmability of the RAN. The RAN is under the control of a controller while maintaining a relative autonomy to avoid a single point of failure. However, a higher entity can override control decisions. Furthermore, the SD-RAN concept shall feature a hierarchical design to enable real-time RAN control operations. FlexRAN (Foukas et al. 2016) is a prime example for an implementation of such a controller featuring both real-time (e.g., remote scheduling) and non-real-time control commands (e.g., user events such as connect and disconnect or handover). More recently, bodies such as ORAN[1] started to work on standardizing the interface between such a controller and the RAN, denoted as E2[2], and distinguishing between controllers working in near-real-time and non-real-time mode. Note that a network of hierarchical controllers allows to integrate such RAN controller with other domains, as formulated for instance within the EU-funded project COHERENT (Kostopoulos et al. 2016).

Furthermore, each separated network should be enabled to flexibly centralize its RAN processing at the centralized/edge cloud to compose the base station functionalities among disaggregated RAN entities, i.e., centralized unit (CU), distributed unit (DU), remote unit (RU), according to the functional splits as documented by the 3GPP (3GPP 2017a). This concept, also called the "disaggregated RAN" and facilitated through SD-RAN and NFV, promises to allow higher deployment flexibility, scalability, upgradability and sustainability by re-using underlying physical infrastructures and the RAN service chaining notion (Tzanakaki et al. 2017), while maintaining centralization for increased base station coordination. However, this leads to multiple functional entities composing a single BS with increased complexity for management, monitoring and control.

The remainder of this chapter is organized as follows. In Section II, we review the concept of a slice descriptor as an extension of a general BS description. We introduce this notion in the RAN and accompanying controller through the RAN runtime (Chang et al. 2018b) and FlexVRAN (Schmidt et al. 2019a) with support for slicing and disaggregated deployments

[1]See https://www.o-ran.org/, retrieved May 29, 2019.

[2]At the time of this writing, there is no final specification yet.

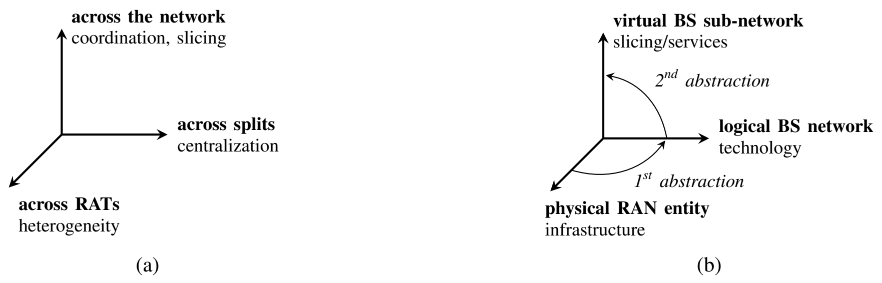


Figure 1. The 5G network can be deconstructed into various dimensions: (a) a RAT-split-network view, generalizing into (b) the physical, the logical, and the virtual dimensions. Source: Schmidt et al. (2019a)

along the three dimensions. In Section III, we present the design of the framework, including performance results. More details about the implementation of the slice descriptor parts can be found in Section IV. Finally, Section V concludes.

II. RAN Slicing System Concept

Several key technologies are widely considered toward 5G, e.g., millimeter-wave and massive multiple-input multiple-output (Shafi et al. 2017), necessitating new RATs. Based on these observations, we identify three dimensions that the future 5G RAN will have to serve, depicted in Fig. 1a:

1) across RATs, in order to cover sub-6 GHz frequencies or millimeter-wave,
2) across splits, i.e., splits allow to achieve a varying degree of centralization, and
3) across the network to allow coordination between densely deployed BSs and different services through slices in order to multiplex them onto the same RAN.

These three dimensions can be generalized into a physical, logical and virtual dimensions. As shown in Fig. 1b, the physical infrastructure using functional splits and various RATs can be abstracted to compose a logical base station (lBS) spanning the full protocol stack, e.g., 4G eNB or 5G gNB. Such an lBS can be seen as an enabler for effective management and control of the network by reducing network complexity of heterogeneous deployments. This lBS is further abstracted to provide a virtualized slice-specific sub-network for network slices, while still maintaining user plane programmability. Such a virtual sub-network is formed from a set of virtual base stations (vBSs) to reveal slice-specific resources and states serving the slice requirements. In summary, this two-level abstraction can not only unify the disaggregated RAN control for network operators but also customize the service deployment for slice owners.

To cope with these challenges and enable slicing in the RAN, we introduce the descriptors to analyze and describe the slicing requirements. Furthermore, we propose a framework consisting of the RAN runtime (Chang et al. 2018b) and FlexVRAN (Schmidt et al. 2019a). The RAN runtime is in charge of providing the slice execution environment for multiple slice instances with the possibility to share the RAN and its resources while guaranteeing slice isolation. It is connected to FlexVRAN that abstracts the underlying RAN from a disaggregated infrastructure into a logically distributed one for the infrastructure owner and provide a customized view on the slice owner's slice network (vRAN) while providing SD-RAN capabilities.

A. Descriptors

A BS is formally represented by a descriptor that defines its capabilities in terms of resources, processing, and state. Note that in the case of a disaggregated BS, such BS descriptor might be "split" among entities.

This concept can be extended to slices. In this case, we consider a slice as a virtual BS, and equally define a slice descriptor that defines the slice service requirements in terms of resources, custom processing (if at all), specific configuration, etc. This is equally motivated to have a similar descriptor across the two previously introduced abstractions. However, the slice descriptor encapsulates the SLA that should be met for such slice instead of merely describing the capabilities.

Resources describe radio spectrum resources which can be divided into bands, carriers, and physical resource blocks for a BS. Regarding a slice, this might include a requested throughput, a share of resources, or other possible resource types that might be requested and can be mapped to the radio resources.

Processing defines a set of functional blocks to perform CP/UP operations, separated through functional splits and described through capabilities. Such capabilities describe the currently active processing entities, but also passive processing capabilities, i.e. what other splits might be possible. Through the processing, a slice might customize its own processing within the RAN.

State is the status of the BS in terms of CP/UP processing and the associated configuration that are built up during the lifetime of the BS or the slice. For example, this includes user equipment (UE) radio resource control (RRC) states, and a slice might customize.

With the help of these descriptors, it is possible to describe and enforce different slice requirements such as

- Performance guarantees
- Isolation
- Customization
- Sharing

To account for the heterogeneity of the underlying system, stemming from possible multi-vendor usecase-driven deployments, the descriptor is represented as a unified data model. An example for such a data model is the network resource model defined by 3GPP (3GPP 2018a).

Finally, to facilitate the selection and association of a UE to a slice, the configuration within the state would also include an ID for every UE within a slice. This is also considered in 3GPP (3GPP 2018b) by means of the Network Slice Selection Assistance Information (NSSAI).

B. RAN Runtime

We propose a RAN runtime slicing system (Chang et al. 2018b) that provides a flexible execution environment to run multiple virtualized RAN instances, i.e., slices, with the requested levels of isolation and sharing of the underlying RAN modules and resources. It enables slice owners to (a) create and manage slices, (b) perform customized control logics (e.g., handover decision) and/or customized CP/UP processing (e.g., packet data convergence protocol (PDCP) and 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 CP/UP state (e.g., user identity) that are revealed by the RAN runtime by interacting with the slice descriptor. The isolation and customization properties provided by the RAN runtime is in favor of the slice owners allowing them to control the slice compositions and the behavior of the underlying RAN module as per service requirements, while the sharing is in favor of the infrastructure provider that enables the efficient and dynamic multiplexing among multiple tenants over resources, processing, and state 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 the RU, the DU and the CU (3GPP 2017a).

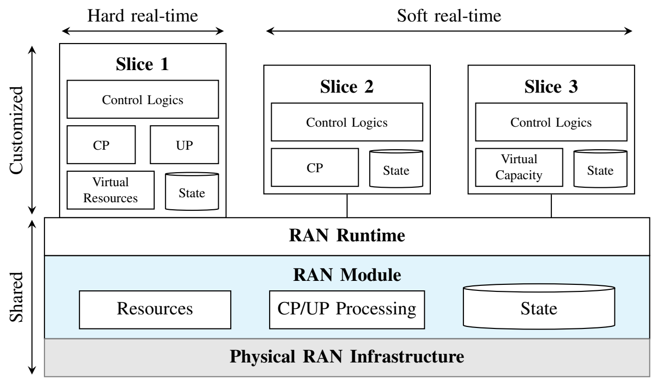


Figure 2. RAN runtime high-level view. Source: Adapted from Chang et al. (2018b)

The 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 application programming interfaces (APIs) to enable the bidirectional interactions between each slice and underlay RAN module in order to monitor or control the CP/UP processing, resources, and states while retaining the isolation among slices.

As explained in Section II-A, the slice descriptor defines slice service requirements. The descriptor might be supplied through a controller (see Section II-C) during the creation or update of a slice in the case of simple slice embedding without necessary processing pipeline modifications for processing customization. The descriptor 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 trade-off between the isolation and the sharing, the states of CP and UP processing are maintained in a database[3] allowing to

[3]This is regardless of whether the network function is stateful or stateless (Matias et al. 2015; Kablan et al. 2017).

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 through the descriptor, the network functions are virtually turned into stateless processing entities which allow 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. A technology-dependent list of such separation can be found in Table II. Note that the CP processing is separated in terms of the functionalities. For instance, the master information block (MIB) and system information block (SIB) are broadcasted commonly to all users with in a cell and are categorized into the BS-common processing, while the random access procedure 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 QoS and the RAN runtime will provide a feasible policy towards the underlying RAN module.

In summary, 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.

C. *FlexVRAN*

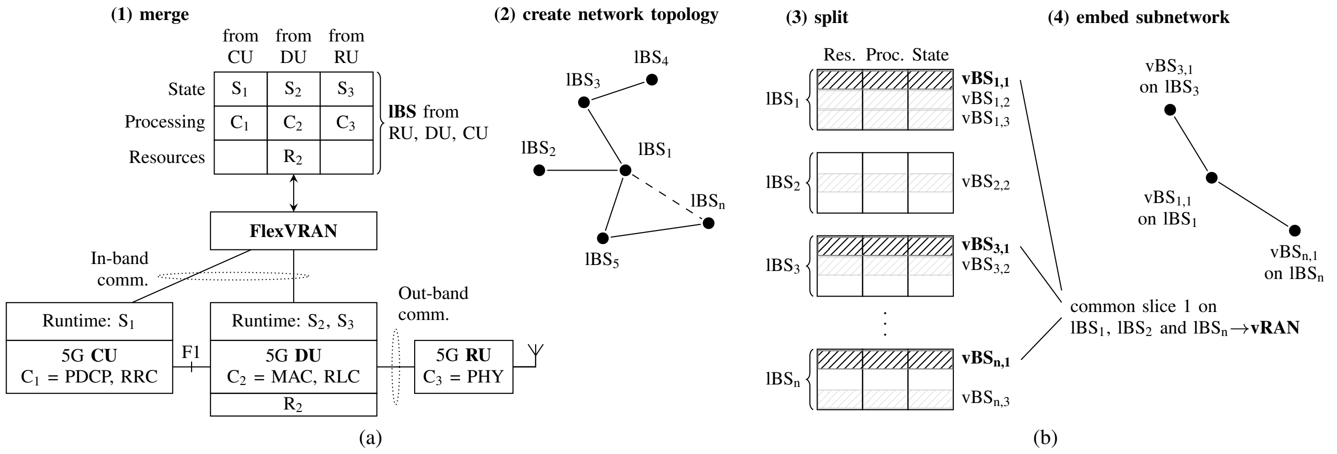


Figure 3. The operations of FlexVRAN to provide the (a) first-level abstraction with operations (1) and (2), and (b) second-level abstractions with operations (3) and (4). Note that the lBS triple in the second-level abstraction only represents customized state, processing, and resources, while common BS functionality remains in the lBS descriptor. A missing topological relation between lBSs (dashed line) is present in the virtual sub-network. Source: Adapted from Schmidt et al. (2019a)

We introduced the notion of a two-level abstraction to cope with the challenges of a sliced, disaggregated RAN. Fig. 3 shows how to achieve the two-level abstraction from physical deployment over logical to virtual base stations in order to form a virtual sub-network that is revealed to a slice. The FlexVRAN controller interacts with a number of *split-aware* RAN runtimes, which act as a local execution environment on top of each monolithic/disaggregated RAN entity. The underlying heterogeneous physical RAN entities host a number of RAN physical network functions (PNFs)/VNFs for CP/UP processing[4]. They are annotated with (a part of) the descriptor, which is exposed by the RAN runtime to the FlexVRAN controller. Due to RAN disaggregation coupled with heterogeneous deployments and technologies, it might however not be possible or feasible to provide a complete descriptor of one BS. Hence, for the first-level abstraction from physical to logical BS, the FlexVRAN controller performs the operation of (1) merging descriptor of multiple RAN entities into one lBS and (2) creating a view of the network topology and annotating the lBS with the complete descriptor, as shown in Fig. 3a. By matching capabilities of different RAN entities such that they complete the capabilities needed for an operational BS, the lBS is created as a unified representation of different deployments with a common data model. After this, the overall network topology is shown to represent the network cell structure in its spatial distribution.

The granularity of information in the descriptor remains the same through merging. In fact, the first abstraction retains information granularity but consolidates the heterogeneity of the physical infrastructure (e.g., through functional splits) into a suitable data model (though certain operations like a reconfiguration of splits might need to expose information about splits,

[4]Some passive RAN entities do not possess the local RAN runtime due to their limited processing capabilities and therefore rely on the in-band control through the remote RAN runtime on top of other entities. For instance, as can be seen in Fig. 3a, the RU relies on in-band control through the DU. To this end, the operating functionalities and the relation toward other RAN entities for these RUs are maintained explicitly by the connecting DU. Also, since the RU is not activated without the DU, we assume the DU to possess the radio resources.

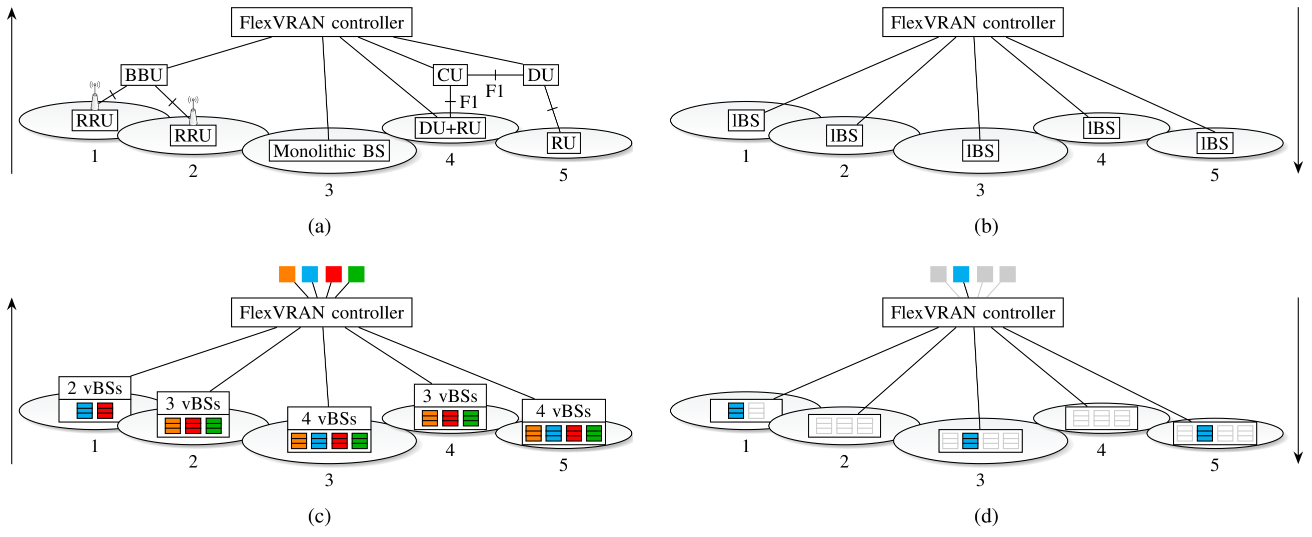


Figure 4. First- and second-level abstractions: (a) underlying RAN entities provided by the RAN runtime to the FlexVRAN controller, (b) the FlexVRAN controller exposes a number of lBSs, (c) each lBS hosts a number of vBSs belonging to different tenants, (d) one slice owner applies its control logic toward its viewed vBSs. The colors represent different slice owners. Source: Adapted from Schmidt et al. (2019a)

subject to access control). Due to this direct relation of the physical and logical representations, the infrastructure owner is able to perform a direct mapping between those two. At this stage, it is possible for the infrastructure owner to apply specific network operator control logic and operate FlexVRAN as an SD-RAN controller.

The FlexVRAN controller performs a second-level abstraction from lBS to vBS, creating a virtual sub-network specific to each slice owner. As shown in Fig. 3b, the controller performs the operation of (1) splitting (slice-wise customizable) lBSs into vBSs and (2) embedding them into the logical network topology in order to reveal a customized view of the virtual network (vRAN) to the slice owner. Splitting refers to storing slice-specific state and configuration, customizing functionality to tailor to the slice requirements, and possibly reserving resources exclusively for such a slice in the descriptor of a vBS. This includes slice-specific control logic, to be enforced by the slice owner, described in the descriptor of a vBS, which differs from the descriptor of an lBS. Shared processing and state remain available read-only for all slices, e.g., monitoring, as long as there is no need for customization. Embedding on the other hand maps the vBSs within the topology of lBSs while decoupling it from the actual geographical position. This withholds network topology information from slice owners[5] and allows easier management of resource conflicts among descriptor. Thus, a slice owner might see a network of vBSs which do not expose the actual geographical position. For instance, vBSs might be moved from one lBS to the next following the user mobility pattern.

Through the capabilities, a slice owner is able to detect which processing functions exist and might customize slice-specific functionality, e.g., hand-over control logics. However, the information granularity, e.g., for specific system parameters, is adjusted depending on the specific SLA. For isolation purposes, the mapping of vBSs to lBSs (and hence to the physical infrastructure) or other slices is not possible from the slice owner's view. By revealing the embedded virtual sub-network, a slice can retain its service requirements via controlling its sub-network and the associated users.

Finally, one important task is to resolve conflicts occurring across different levels of abstraction. In general, the conflicts are resolved on the level that "creates" the corresponding level of abstraction.

- The slice owner can handle the slice-specific conflicts in its viewed vBSs.
- FlexVRAN is responsible for resolving the conflicts to compose an lBS, e.g., the proper chaining of RAN entities to compose available lBSs, and between slices when customization attempts are made or new slices are to be instantiated.
- The RAN entity resolves conflicts among the underlying CP/UP processing, states and resources.

Nonetheless, there might be some conflicts that are not detectable by a given level; thus, the underlying level may reject or amend the control decisions. For instance, when the slice owner aims to configure the allocated rate of its vBS beyond the capability of the corresponding lBS, the FlexVRAN controller can override this control decision in the corresponding vBS configuration.

D. Slice Creation

To illustrate the creation of a slice, we consider an example network as shown in Fig. 4 where the first- and second-level abstractions are performed. It shows five different cells that are formed from multiple BSs with a varying degree of centralization.

[5]A slice owner might be an untrusted third party, or physical user location needs to be protected while verticals need to identify who uses their service.

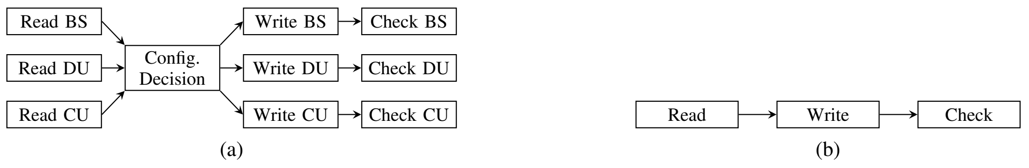


Figure 5. Reduced control complexity of slice creation: (a) Without FlexVRAN (b) With FlexVRAN. Source: Adapted from Schmidt et al. (2019a)

These BSs are handled through the RAN runtime which provides the flexible execution environment to virtualize resources, customize processing composition and the necessary adaptability for monolithic and disaggregated deployments.

1) *Physical Infrastructure*: as shown in Fig. 4a, the example network is heterogeneous. Cells 1 and 2 are deployed through a Cloud-RAN (C-RAN) setup. Cell 3 uses a monolithic BS, while cells 4 and 5 are deployed through a disaggregated BS, consisting of a CU, a DU, and an RU.
2) *Logical Base Station*: As for the first-level abstraction, it merges the information of RAN entities into lBS descriptors and provides the topological information. Fig. 4b shows the simplified RAN as presented to the network operator. Note that information granularity is retained, and e.g., split information is available. On the other hand, this allows to configure a BS on medium access control (MAC) and RRC layers, located on the DU and CU, respectively, without necessarily considering the used split. Such reduction in complexity is shown in Fig. 5: in case of the creation of a slice, not all involved entities (in this example a monolithic BS and a BS consisting of a CU and DU) need to be programmed independently (Fig. 5a) but only the single interface of FlexVRAN is sufficient (Fig. 5b).
3) *Virtual Base Station from a network perspective*: The second-level abstraction further includes the RAN slicing notion by splitting lBS into vBS and embedding a virtualized and customized sub-network into the logical network topology for each slice owner. Fig. 4c first shows the slices embedded into the RAN runtimes and controlled through FlexVRAN. Note that this figure shows the vBSs embedded into lBSs: in the case of cell 4, two split-aware RAN runtimes embed 3 vBSs, whereas only one RAN runtime is needed in the case of cell 3 with 4 vBSs.
4) *Virtual Base Station from a slice-owner perspective*: Finally, at the north-bound interface of FlexVRAN, only the vBSs belonging to a certain slice owner are exposed (Fig. 4d). Note that embedded slices might be moved from one lBS to another and the exact location might be hidden from a slice owner.

Fig. 6 shows the process of abstracting physical RAN entities into lBSs and from there into lBS from a descriptor point-of view as it might happen for cells 4 or 5 in Fig. 4. Both RAN runtimes of DU and CU provide information about the underlying RAN entities, which is merged into a common lBS. This information includes the radio resources (20 MHz), capabilities of the underlying RAN entities, and its state in the form of configuration and statistics. Then, three slices are embedded: a default slice, working as a "placeholder" slice, one slice for video services and a slice targeting the massive Machine-Type Communications (mMTC) use case. The video slice requests a throughput of 15 Mbps (resources), and a list of associated users in the form of their IMSIs (state configuration), without any further customization (processing). The mMTC slice requests a fixed amount of resources in the uplink and might customize the processing for its use case.

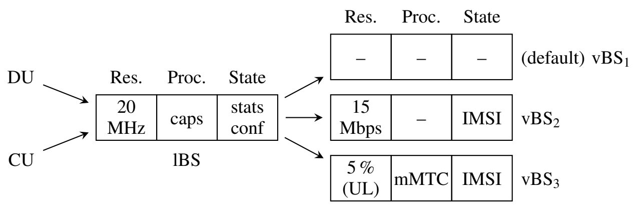


Figure 6. A descriptor is merged from DU and CU into one lBS, then split into two vBSs (one default, one with reserved rate and specified user-list). Source: Adapted from Schmidt et al. (2019a)

III. Design and Performance of the RAN Slicing System

After the introduction of the notion of slice descriptors and how they are implemented in the RAN runtime and corresponding FlexVRAN controller, we elaborate on their design and show the implementation viability.

A. RAN runtime

Fig. 7 illustrates the building blocks of the RAN runtime, consisting of slice data and various CP and UP functions to provide the RAN runtime services. The RAN runtime exposes APIs in the south-bound towards the underlying RAN module in order to allow individual slices to control the RAN module. It is then possible to take the control of its service by requesting virtualized/physical resources, applying its control decisions, and accessing the virtualized state information. The north-bound endpoint towards slices is assured through FlexVRAN as described in the following section.

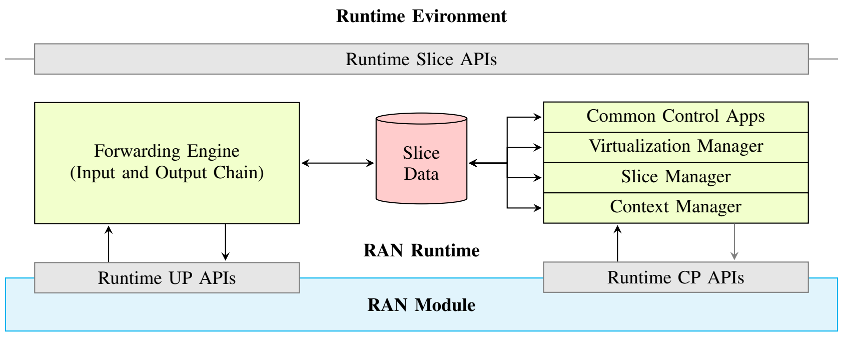


Figure 7. Architecture of the RAN runtime. The RAN runtime environment is provided by FlexVRAN. Source: Adapted from Chang et al. (2018b)

1) Slice Data: Slice data 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, and state.

The slice context describes the basic information and prerequisites to instantiate a slice service and manage corresponding users. 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, module life-cycle management primitives such as start, configure and stop service, and resources. 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 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 the general case, states only include the user-specific functions in RRC CONNECTED and the RRC INACTIVE-CONNECTED (Kim et al. 2017) 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.

2) RAN runtime services: Further, the RAN runtime provides services that can be provisioned for each slice as shown in Fig. 7. To utilize these RAN runtime services, each slice is registered and identified with its identity among possibly disaggregated RAN entities.

The *context manager* manages both slice context and module context by performing create, read, update, delete (CRUD) operations on the slice data. To create a slice context, the context manager first performs the slice admission control based on the provided slice descriptor (see Section II-A) that defines the required processing, resources, and state (as agreed between the slice owner and the infrastructure provider). Upon slice admission control, the module context is used by the context manager to register the requested resources and/or performance at the virtualization manager and slice-specific life-cycle primitives at the slice manager. The former enables the resource partitioning and abstraction to be performed among multiple slices (see Section IV-A), while the latter allows custom CP/UP processing to be applied on the input/output data streams (see Section IV-B). 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 deployments. However, the maximum number of slices that can be deployed depends on multiple factors:

1) the overhead of the RAN runtime,
2) the available resource in terms of compute, memory and link,
3) the requested SLA and resources by each slice,
4) the amount of over-provisioned resources, and
5) the workload of each slice.

The *slice manager* entity is responsible for managing the life-cycle of a slice when instructed by the slice owner via FlexVRAN. 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 end-to-end (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 operational expenditure (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.

The 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 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 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.

The *virtualization manager* is in charge of providing the required level of isolation and sharing to each slice. It partitions resources and states based on the slice and module contexts, abstracts the physical resources and states to/from the virtualized ones, and reveals virtual views to a slice that is customized and decoupled from the exact physical resources and states. A detailed description on the resources aspect of virtualization is in Section IV-A. We omit state partitioning as it can be realized through some well-known approaches such as database partitioning and control access.

The *common control applications* provide 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, control logics of customized RRM applications of two slices will leverage the inter-slice conflict resolution and control logics accommodation to provide their slice-specific control logics through FlexVRAN. 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.

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. The forwarding engine is described in Section IV-B.

B. *FlexVRAN*

The combination of RAN runtime and FlexVRAN follows a hierarchical design suitable to realize real-time operation, and it is composed of a centralized controller that steers the RAN runtime, one or more for each BS in the case of RAN disaggregation. The runtime might act as a local controller with a limited network view, handling control delegated by the controller, or in coordination with other runtimes and the controller.

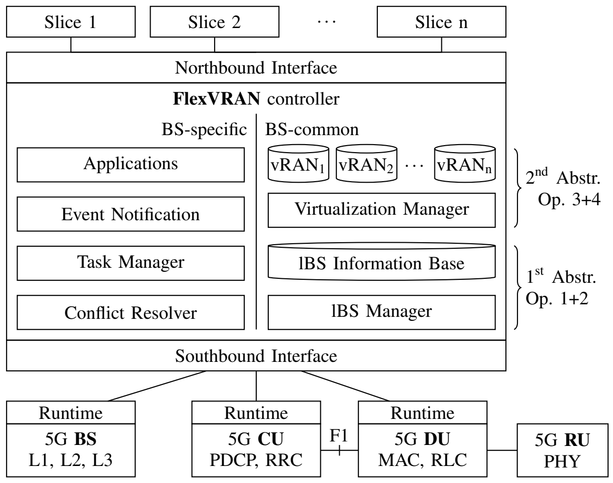


Figure 8. Components and interfaces of FlexVRAN. Source: Adapted from Schmidt et al. (2019a)

Therefore, FlexVRAN is connected southbound to one or more RAN runtimes as shown in Fig. 8 exchanging protocol messages through an asynchronous interface. Since the FlexVRAN controller has no a priori knowledge about the connected RAN runtimes, RAN runtime information (identity and capability) as well as slice information is exchanged in between. Through the two-level abstraction as described in Section II-C, FlexVRAN gives slices a limited view of their vBS, exposing appropriate RAN runtime APIs at the northbound to allow slice owners to register and consume the RAN runtime services, manage their service in coordination with the RAN runtime and a possible service orchestrator, and customize the CP/UP processing as described previously and as agreed in the SLA. Through this, a slice runs in a separate process, either local or remote, and the interface provides a communication channel between a slice and its vRAN. Hence, each slice can be executed in isolation from each other either at the host or guest level leveraging well-known OS and virtualization technologies, such as containers or virtual machines.

FlexVRAN has five main services for managing BS-specific or common information: (a) lBS manager, (b) virtualization manager, (c) task manager, (d) event notification service, and (e) conflict resolver.

First, the lBS manager is responsible to compose the lBS via interacting with the underlying RAN runtime instances through the south-bound interface by performing the operations of the first abstraction as described. For instance, it can merge the underlying RAN entities' capabilities to form an lBS comprising the baseband and protocol processing ranging from physical layer (PHY) up to RRC for both monolithic and disaggregated deployments. Note that the controller will withhold partial information of a BS and wait for the corresponding RAN runtime to connect. lBS data is then stored in the lBS information base and can be update on-the-fly.

The second-level abstraction is done by the virtualization manager, which abstracts the vRANs as virtual sub-networks based on slice owner's request. The northbound interface of the controller can restrict a slice's access to its vRAN, therefore isolating slices from each other through access control. Different levels of virtualization is supported by managing a slice context in the corresponding vRAN, such as the slice SLA and customization configuration.

There are multiple common services provided by the FlexVRAN controller. In the first place, the task manager handles messages from the underlying RAN runtime instances. The event notification service provides updated information to managers and applications at the controller with the required level of granularity. It can happen in a periodic process with a larger window T than the underlying operating period, or on per event basis. For instance, when one physical infrastructure is deactivated, the corresponding life-cycle event will be notified toward the corresponding slice lBSs.

The conflict resolver provides a shared control logic for multiple slices. It aims to admit the customized control logic from different slice-specific control applications, resolve their conflicts, and enforce a feasible policy to underlying lBSs. For instance, two different slices may wish to increase requested resources which exceeds overall BS resources. To this end, the conflict resolver can either apply a policy-based method to decide which control application to be executed or use a learning-based method to generate a predictive model from the historical lBS state for decision making.

C. Results

The feasibility of RAN runtime and FlexVRAN has been assessed through a prototype based on the OpenAirInferface (Nikaein et al. 2014) and FlexRAN (Foukas et al. 2016) platforms. Fig. 9 shows the CPU and memory usage overhead at the RAN runtime for monolithic and disaggregated deployment scenarios. These measurements of the RAN runtime overhead can justify the deployability of the RAN runtime together with the underlying RAN entities (i.e., BS, CU, or DU), while the FlexVRAN controller could be deployed on another physically separated infrastructure component. Moreover, we evaluate this overhead in two different scenarios: (1) zero connected UE (i.e., no user traffic), and (2) one connected UE saturating DL throughput with a video stream.

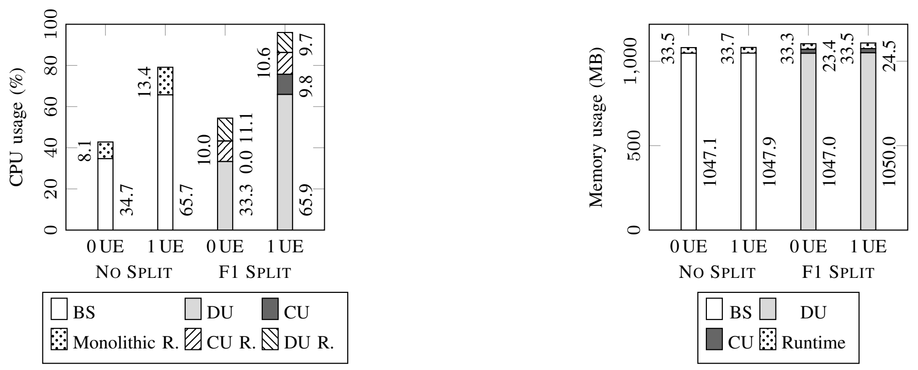


Figure 9. Comparison of OAI and FlexVRAN-enabled OAI. Note that the sum of memory usage of RAN runtimes is given for disaggregated deployments. Source: Schmidt et al. (2019a)

We see that both monolithic (BS) and disaggregated (DU, CU) deployments have similar CPU usage. As a side note, we can observe that the extra processing for F1 interface of the disaggregated deployment is negligible. However, the two RAN runtimes of the disaggregated deployment (CU runtime and DU runtime) will both interact with the FlexVRAN controller, equally contributing to the CPU usage, whereas only one (BS) runtime is needed in the monolithic case.

As for the memory, we can first observe the overall memory usage of the disaggregated deployment (DU and CU) being slightly higher than the monolithic one (BS). The reason behind this is the extra memory overhead for the two RAN entities. In general, the RAN runtime introduces only a small memory overhead confirming its deployability. The number of connected UEs has negligible impact on the memory usage.

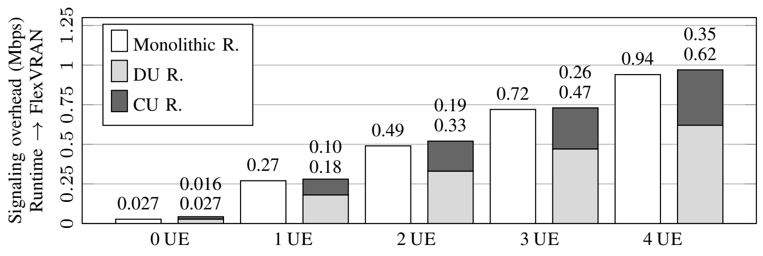


Figure 10. Signaling overhead from RAN runtime to FlexVRAN controller. Source: Schmidt et al. (2019a)

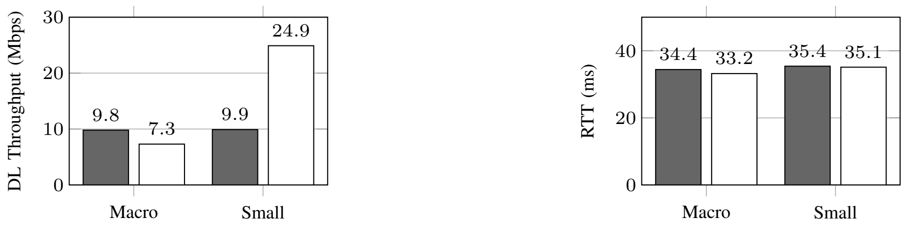


Figure 11. Comparison of a monolithic macro cell using 5 MHz and an F1-split small cell using 10 MHz. The dark bars show the performance of the UE in the video slice, the white ones show the UE using the default slice. Source: Schmidt et al. (2019a)

Furthermore, we investigate the signaling overhead for exchanging the control information between RAN runtime and FlexVRAN controller in Fig. 10, sending statistics every 1 ms. Such signaling overhead is crucial when deploying the control framework over disaggregated infrastructure with a capacity-limited fronthaul/midhaul network. We can see that the generated overhead mainly depends on the number of UEs in order to exchange user information periodically. However, the signaling overhead remains far lower than user plane traffic which might attain Gbps. For disaggregated deployments, the additional overhead is negligible.

Finally, we consider a RAN network composed of two BSs in direct neighborhood using different operating frequencies in the same band. One BSs is a monolithic deployment as macro cell with a single runtime and 5 MHz bandwidth and the other is a small cell using the F1 functional split with each a runtime for DU and CU, operating on 10 MHz bandwidth. Both deployments serve one UE. All three runtimes are connected to the FlexVRAN controller merging three RAN runtimes into two lBSs (cf. operation 1, Fig. 3a) with associated descriptor.

The slice owner aims to provide a video service with reserved resources for a specific event via requesting 10 Mbps of maximum guaranteed bit rate, irrespective of the BS (available resources, deployment type, etc.) and associate its UEs to the corresponding vBS using their IMSIs.

The controller calculates the needed resource share, compares it with existing slices in both lBSs (each with a default slice that can be shrunk), and sends a command to the RAN runtimes to split the lBSs (cf. operation 3, Fig. 3b). The resulting vBS's descriptors contains configuration of associated UEs in the state, with no customized processing and dedicated resources reserved for the requested bit rate. As discussed in Section III-A, an instantiation of slices in the runtime is subject to conflict management that might override a decision. If the instantiation succeeds, the controller embeds the sub-network into the previously built topology using two inter-connected vBSs to reflect that both vBSs serve instantiated video services in the same vRAN (cf. operation 4).

We measure UE performance based on the above experiment setup for throughput and the latency using ping with 20 kB packets and an inter-packet interval of 200 ms. The results shown in Fig. 11 indicate that the slice owner's requested throughput (dark bars) is almost achieved for both UEs for both deployment types. Deviations from the requested bit rate are mainly due to wireless channel variations as well as the fact that our current implementation rounds a bit rate to resource block groups,

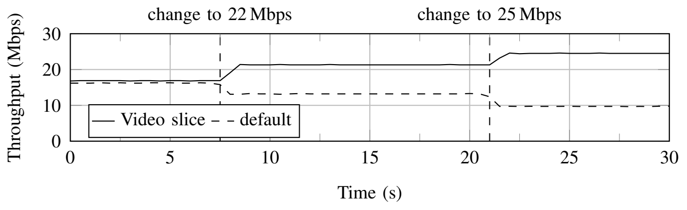


Figure 12. Network throughput after successive writes to the slice descriptor of the corresponding vBS. Source: Schmidt et al. (2019a)

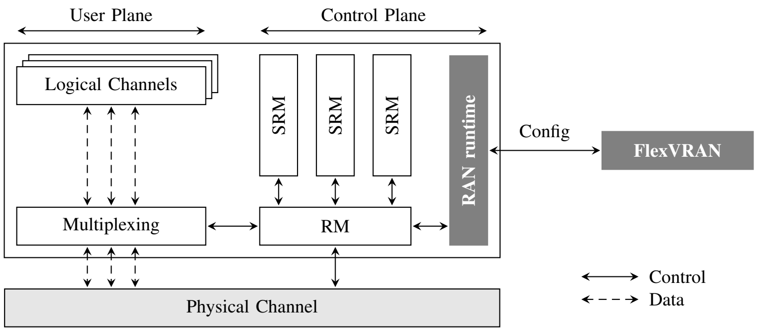


Figure 13. The two-level scheduler, enforcing slice resource usage. Source: Adapted from Ksentini et al. (2017)

leading to "scheduled" bit rates that deviate from the requested bit rate. In both cases, similar delay performance are attained. Note the small additional delay overhead due to the additional hop between CU and DU. We highlight the fact that even with heterogeneous deployments, the proposed control framework can not only use one simple API call to instantiate the requested slice but also facilitate slice owners to customize and control their own virtualized sub-networks.

Finally, we show how a slice owner can control its slice through the vBS descriptor. As shown in Fig. 12, the video slice owner requests a bit rate of 18 Mbps which is later increased to 22 Mbps, then 25 Mbps. Indeed, the requested bit rate is attained shortly after reconfiguration which confirms the applicability and feasibility of the proposed FlexVRAN control framework.

IV. Slice Enforcement

So far, we discussed the slice descriptor as a means to reflect the slicing concept within the RAN, presented the representation of the descriptor within both the RAN runtime and FlexVRAN, and presented their design along with results. In this final section, we investigate some techniques to implement the descriptors, i.e., the realization of slicing with respect to resources, processing, and state.

A. Resources

In the following, we consider radio resources, as they are scarce and hence constitute the main bottleneck of the RAN.

Perfect isolation of slices with respect to radio resources can be guaranteed through a strict dedication of a part of the spectrum, e.g. the solution provided in the works of Nakao et al. (2017) using FLARE. Hence, slices are perfectly separated, but slice elasticity and scalability is reduced. Also, the multiplexing gain, i.e., opportunistically using resources from other slices if they are lightly loaded, is limited, since the dedicated resource model does not allow a slice owner to easily modify the amount of resources.

To overcome this limitation, we consider a two-level MAC scheduler (Ksentini et al. 2017): the first level of scheduling is done by a slice-specific scheduler handling intra-slice traffic, while the second level is done by a common scheduler that maps the first level scheduling propositions to physical radio resources allocation. Through this approach, resources can be dynamically shared between slices, allowing a flexible multiplexing of slice resource requests onto the available resources.

1) Two-level MAC scheduling: Fig. 13 depicts the architecture of the evolved Node B (eNodeB) for two-level radio resource slicing. Logical channels are used to model the UEs's data queue and their mapping to data bearers, which is multiplexed onto the radio resources as decided by the scheduler. The MAC operation is partitioned into two levels: the slice resource manager (SRM) performs the first level by ensuring intra-slice traffic scheduling, while the resource mapper (RM) assigns PRBs to UEs according to the mapping provided by each active SRM. The main difference is thus related to the abstraction (i.e., virtualization) of the PRBs through an abstraction layer in the form of the RM. The SRM is in charge of scheduling resources for UEs belonging to its slice.

Each SRM may use a different scheduler. The RM will expose a slice's user information to each SRM, which will be used by the SRM to schedule UEs over the vRB. The RM is in charge of accommodating the vRBs to PRBs according to the share of resources that should be allowed to each slice. The share of resources can be expressed in terms of percentage of PRBs or the bandwidth to be allocated to a slice.

The proposed two-level scheduling is preferred over joint scheduling (all in one pass), as the latter is very complex and requires multi-dimensional scheduling. Indeed, this type of scheduling algorithms formulates a multi-objective function that should satisfy heterogeneous slice requirements (e.g., latency and bandwidth), where the optimal solution is usually NP-hard. Clearly, using different SRM will ensure: (i) more scalability, as the SRM is created when the slice is instantiated; (ii) dynamic resources management, since the assigned resources can be adapted to the workload demand (service elasticity and scalability).

Due to the eNodeB programmability, the scheduler algorithm can be updated on-the-fly during the slice lifetime. Indeed, a slice owner may decide to change the scheduling algorithm and the allocated resource share, if they seem not efficient or not appropriate for the applications on top of the slice.

a) SRM function: The SRM's main function consists in scheduling the UEs of a slice by assigning vRBs. The vRBs are virtual and do not have any link to the available PRB. Nevertheless, they share the same size in order to ease the alignment of the vRB/UE mapping to PRB/UE. It is worth noting that the size depends on the common physical channel characteristics (i.e., bandwidth, numerology, etc.). Depending on the slice type, the SRM functions and the needed inputs (from the RM) will be different. Usually, the SRM uses the following data: (i) the amount of data per logical channel, in Uplink (UL) and Downlink (DL), including statistical information; (ii) the channel quality of each UE, and (iii) optionally QoS information such as latency requirements

For the three usage scenarios identified by the International Telecommunications Union (2015), different schedulers might be employed:

- For enhanced Mobile BroadBand (eMMB), the popular proportional fair algorithm might be used, weighing fairness and performance to achieve high resource usage.
- For Ultra Reliable, Low-Latency Communications (URLLC), a delay-based scheduler would try to minimize latency while ensuring service reliability through robust modulation schemes at the expense of high efficiency.
- mMTC traffic is more focused on the UL. If the UL traffic is periodical, a scheduler would use a prefixed scheduling pattern (e.g., semi-persistent scheduling (SPS)) to preconfigure scheduling grants in UEs. If the traffic is event-driven, the type of channel access (e.g., regular, random or contention access) would influence the choice of a scheduler.

To demonstrate the effects of customization the SRM scheduler algorithms, we perform the experiment of two slices with different schedulers: Slice 1 uses a proportional fair scheduler allocating resources proportionally to the current achievable throughput and the user's past average throughput. On the other hand, slice 2 employs a blind equal throughput scheduler that strives for achieving the same average throughput, regardless of channel quality. Fig. 14 shows that slice 2 achieves a lower aggregated throughput compared to slice 1 (4.6 Mbps instead of 5.1 Mbps), which confirms the customization property of the proposed framework and its potential impact of the resulted per-user and per-slice performance.

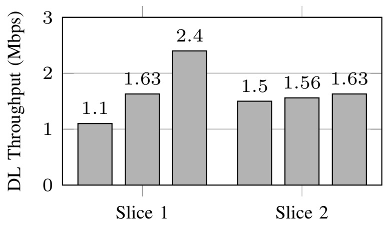
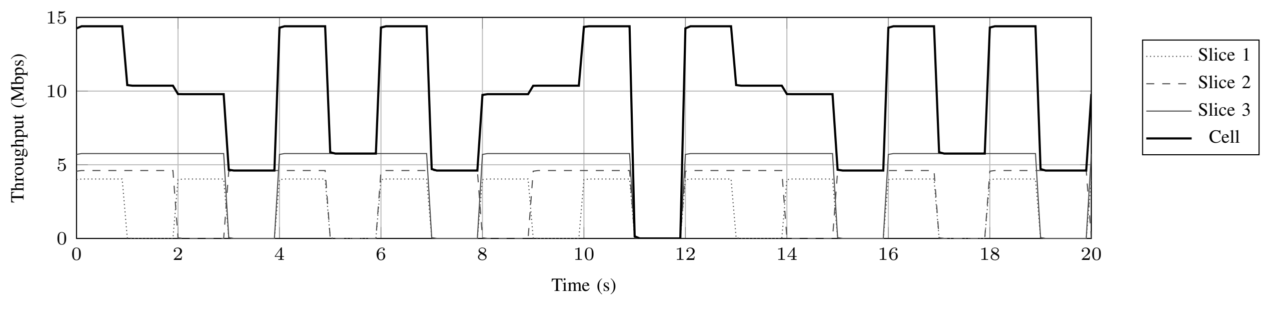


Figure 14. Customization through slicing: slice 1 operates with a proportional fair scheduler, slice 2 applies a blind equal throughput scheduler. Source: Schmidt et al. (2019b)

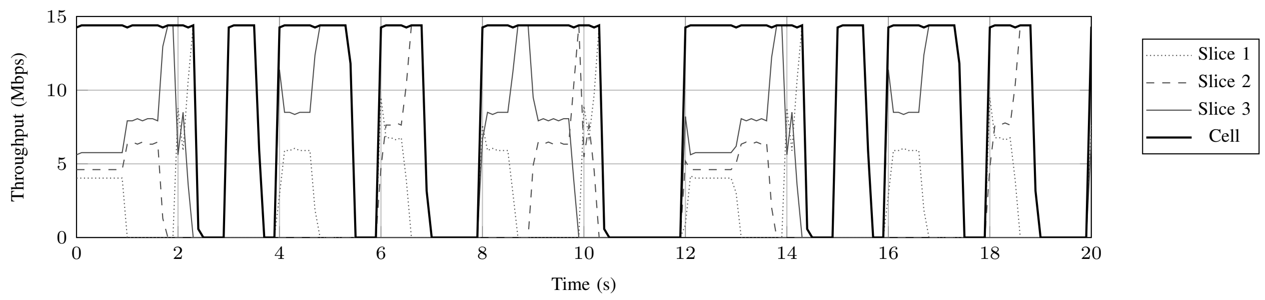
b) RM function: The RM scheduling process begins by sequentially treating each SRM mapping. Two cases may happen when mapping the vRB/UE to PRB/UE. Firstly, if the scheduled PRB for a slice reaches the resource share threshold, then the RM will move to the second slice, and the current slice state will be marked as "paused". Secondly, if the resource threshold is not reached and no UEs is waiting to be scheduled, the remaining PRB will be kept for other slices (to maximize the multiplexing gain), and the current slice state will be marked as "ok". Indeed, the RM will do a second iteration over the slice marked as "paused" to allocate them the remaining PRB that are not used by the slices treated in the first round. Here, a second level of priority among slices could be applied. For instance, URLLC slices will be favored over eMMB as the former requires low latency communication. The process will end when all the slices are marked as "ok", no more PRB are available, or the second iteration is finished in order to ensure that the SRM and RM scheduling can be done during a transmission time interval (TTI) interval, i.e., 1ms. Indeed, the two-level scheduler's complexity is kept low, as the SRM can be run in parallel, and the RM complexity is proportional to the number of slices.

By having a loop on the resource assignment at the RM level, the proposed algorithm will maximize the resources usage, compared to static scheduling. Allowing the slice owner to manage resources for a slice through a resource share will ensure that the scheduling of resources is not fix throughout the slice life cycle and can be adapted when needed. Indeed, this would happen if a quality degradation is monitored at the application level, or the slice operator needs more resources to accommodate more UEs (i.e., case of an event). The RM is also allowed to preempt resources for an emergency slice. In this condition, the SLA indicates that the SRM corresponds to a high priority slice, hence the RM will schedule first all UEs of this slice according to the SRM output, without considering other resource constraints (i.e., no limit on the used resources).

We demonstrate the flexibility through elastic resource flexibility. In this experiment, we consider two cases with (i) static slicing with three slices and a reservation of 28 %, 32 % and 40 % of resources against (ii) three dynamically sized slices that dynamically share their resources and a reservation of 4.032 Mbps (corresponds to 20 %), 4.608 Mbps (32 %) and 5.76 Mbps (40 %). Each slice has one user that periodically downloads a large file. As can be seen in Fig. 15a, resources are wasted when not all users are active, and the cell's throughput is below its maximum of 14.4 Mbps. In case two (Fig. 15b), resources can be shared between slices. This leads to increased throughput for active slices, and the cell efficiency is improved when users active; put differently, the cell is loaded for a shorter time and often becomes completely unloaded.



(a)



(b)

Figure 15. Exemplary resource utilization for three slices. (a) Static resource reservation. (b) Dynamic resource reservation. Source: Schmidt et al. (2019b)

c) Integration with the RAN runtime: The RAN runtime is in control of the RM and communicates to the remote FlexVRAN controller. A slice owner can request a (re)configuration, in real-time, of its resources. FlexVRAN translates these requests to configuration messages, communicated to the RAN runtime. For instance, the scheduling policy of a specific slice as well as the scheduler algorithm to be used by the SRM can be changed. The RAN runtime will configure the SRM and the RM.

2) Radio Resource Virtualization: To support the effective resource enforcement and as alluded in the previous section, resources can be *abstracted*. This serves two purposes: (i) isolation of resources by presenting a virtual view of the resources that are decoupled from the exact physical locations, and (ii) increase the multiplexing gain by adjusting physical allocation types when sharing unused resources. Hence, radio resources can be abstracted to isolate different SRMs from each other using various abstraction types. Furthermore, abstraction allows to increase the resource efficiency through effective multiplexing.

Hence, resource virtualization is one key concept to provide the required level of isolation and sharing that is customized for each slice. It partitions radio resources based on the context and requirements of each slice, abstracts physical resources to/from the virtualized ones, and exposes a virtual view to a slice that is customized and decoupled form the exact physical resources (Chang et al. 2018b). To this end, two steps are necessary to realize such radio resource virtualization.

a) Radio Resource Abstraction: As mentioned in the beginning, there are two main purposes for the abstraction of radio resources. Firstly, the resource isolation by presenting a virtual view of the radio resources that is decoupled from the physical resources, and thus preventing other slices to access or even infer the resources allocated to others (benefits the slice owner). Secondly, it allows resource multiplexing by adjusting the allocation type and scheduling policy, and thus increasing the resource utilization efficiency (benefits the infrastructure provider).

We consider a number of resource types to serve the differing slice resource requirements and granularity and take LTE as a baseline. Table IV identifies four types of resource allocation: in the DL, (1) Type 0 allocation is based on the resource block group (RBG) as the minimum resource granularity that comprises multiple resource blocks (RBs), (2) Type 1 categorizes RBGs into several subsets and only allocates RBs within the same subset, and (3) Type 2 allocates contiguous vRBs that can be physically contiguous (localized vRB) or non-contiguous (distributed vRB). In the UL, there are two resource allocation types: (1) Type 0 allocates PRBs contiguously, and (2) Type 1 allocates non-contiguous RBGs that are distributed in two clusters. The proposed virtual resource block groups (vRBGs) and virtual transport block size (vTBS) form a superset of legacy LTE resource allocation types and provide substantial flexibility for both intra-slice resource allocation and inter-slice resource partitioning,

Fig. 16 provides an illustrative example for a 3 MHz LTE system with 15 PRBs and a physical resource block group (PRBG) granularity of 2 PRBs (8 PRBGs with the last PRBG). These PRBGs are partitioned by the RM for each slice based on the number of required resources and the resource granularity stated. Then, they are virtualized into vRBGs or vTBSs according to the abstraction type. For instance, fixed position resources are requested by slice 2; hence, no virtualization is performed and it is passed PRBGs. In contrast, slice 4 requests a capacity value and its PRBGs are abstracted into vTBS with the corresponding capacity. On the other hand, PRBGs of slice 1 and 3 are virtualized into vRBGs via abstracting the exact frequency/time locations and dimensions. Further, these vRBGs are pooled together to maintain their relative frequency dependencies without revealing their absolute physical frequency locations. The slices (in the form of the SRM) can then perform the scheduling; in the example, slices 1 and 3 do not schedule all vRBGs, whereas slice 4 wishes to schedule more vTBSs.

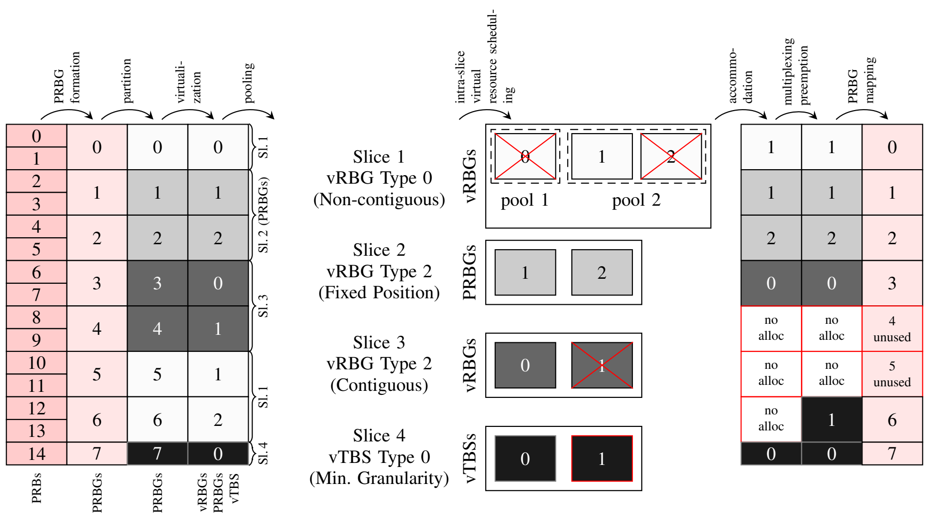


Figure 16. Example for resource abstraction. Note that slices 1 and 3 do not use all assigned resources, and slice 4 wishes to schedule more than initially assigned. Source: Adapted from Chang et al. (2018b)

b) Resource Accommodation and Multiplexing: After the radio resource partitioning, virtualization, and the scheduling within each slice, the scheduling decisions will be accommodated into PRBs as also shown in Fig. 16. The unallocated region can be utilized to satisfy some other slices that request more resources, e.g., vTBS2 of slice 4, as already explained. Moreover, the preemption scheme can also be applied by removing the inter-slice scheduling results of other low-priority slices to boost the performance of a high-priority slice (not depicted). Finally, PRBGs can be mapped and the corresponding control information is formed. Note that this is used to indicate the user about the positions of allocated PRBs together with other physical layer information for successful transportation.

B. Processing

The processing part of the slice descriptor describes if (any) CP/UP customization should be employed for the considered slice. In the following, we consider the main part that allows customization: we consider the forwarding engine as a means to customize CP/UP behavior.

The forwarding engine manages the input and output streams of CP and UP, or simply data streams, across shared and/or customized processing. Fig. 17 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), MAC, and PHY[6]. Input flows of the RAN module for each slice are forwarded either to the customized (i.e., slice 1 and 2) or the shared (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 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 (Bosshart et al. 2013).

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 (3GPP 2017a). 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. 18 shows the input/output forwarding path between CU, DU, and RU to compose a distributed UP processing chain using 3GPP function split option 2 (3GPP 2017a) between CU and DU and option 6 between DU and RU. The input and output endpoints

[6]Further function decomposition within layers is possible, like splitting the PHY into high-PHY and low-PHY.

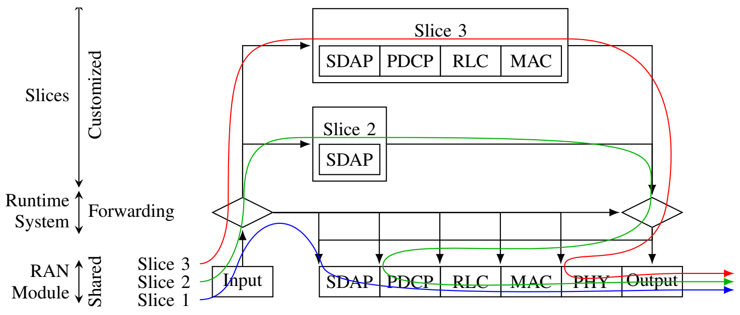


Figure 17. Forwarding engine and UP processing chain. Source: Adapted from Chang et al. (2018b)

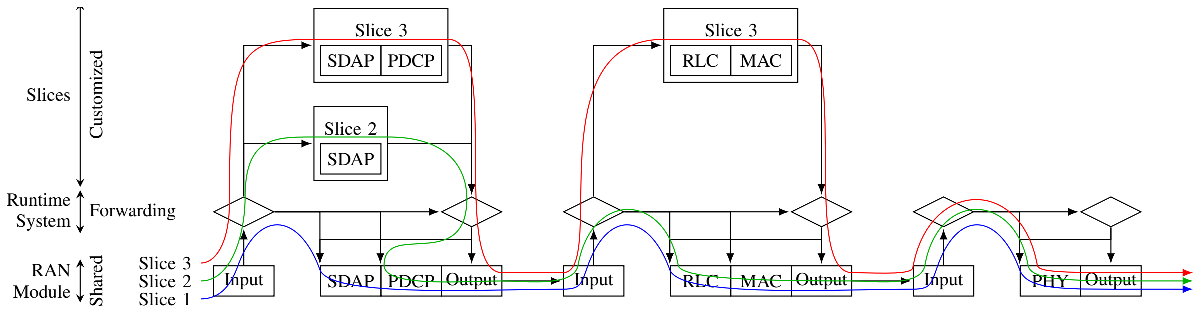


Figure 18. UP forwarding path in three-tier disaggregated RAN (CU, DU, RU). Source: Adapted from Chang et al. (2018b)

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[7].

C. State

The state part of the slice descriptor contains configuration and statistics of the different logical layers of the BS. In order to ensure isolation, the state is divided into BS-common and slice-specific state as already listed in Table II on Page 20. Table V summarizes the main UP state information that shall be maintained and shared in the slice data. This state information is also transferred to FlexVRAN to represent the disaggregated RAN entities as on 1BS. This state information is furthermore available to custom control logics that can be configured on a per-slice basis, as described in Section II-B.

Furthermore, a network graph database might be used to operate on the network information (Chang et al. 2018a). It provides the graph-based primitives (e.g., split or merge) to efficiently model, traverse, and correlate more complex and dynamic relations between densely deployed RAN nodes. Moreover, it can naturally support multi-tenancy through graph partitioning into sub-graphs for multiple substrates (e.g., multidomain or multiservice). Hence, each application can perform its graph-based operations (e.g., shortest path) that take node relationships in time series into account in its abstracted network view.

Comparatively, the network graph shown in Figure 19 depicts three slices in a disaggregated RAN deployment. The disaggregated RUs 1 to 3, DUs 1 and 2, and CU are virtualized for three different services to serve five users (u_1 to u_5). For instance, RU_1 is virtualized for slice 1 and slice 2 as $RU_{1,1}$ and $RU_{1,2}$, respectively. The edges between the vertices represent their relations and be used for different applications. For example, the edges between a user and a virtualized RU represent the measured channel quality or traffic metrics. Such a sub graph can be utilized as input for handover or traffic-steering applications. Additionally, the edges between disaggregated RAN entities, as in $CU_{1,1} \leftrightarrow DU_{1,1} \leftrightarrow RU_{1,1}$, capture their slice-wise association and the applied functional splits for multiservice chaining and placement application.

Furthermore, the edges between virtualized instances within the same physical RAN node (solid double lines) show their relations in terms of sharing (e.g., multiplexing) and priority (e.g., preemption) that can be used for resource management, while the edges between different physical entities (dashed double line) depict the policy of cooperation (e.g., spectrum sharing) or constraint (e.g., exclusive muting), which are useful for dynamic radio spectrum management applications. In summary, the graph database can naturally represent complex relations and be partitioned or combined for control applications among multiple substrates.

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

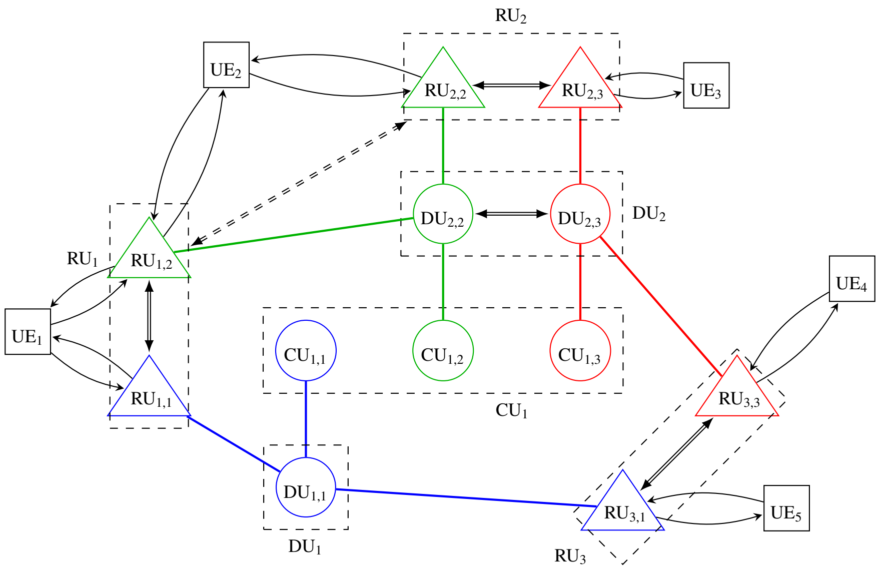


Figure 19. An example of a network graph for a disaggregated RAN deployment. Source: Adapted from Chang et al. (2018a)

V. Conclusion

Network slicing is one of the key enablers of 5G. This chapter introduced the reader to RAN slicing. We started with the notion of descriptors which is defined as a triple of resources, processing, and state as a unifying description of slicing across the physical, logical and virtual levels. In order to serve various flavors of slices, flexibility and efficiency shall be achieved simultaneously across the slice descriptor. To this end, the proposed RAN runtime and FlexVRAN slicing framework aim to flexibly manage the slice life cycle as well as support various slice requirements with respect to its SLA in terms of (1) the 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 RAN and stake holders such as the infrastructure or slice owner. Furthermore, we explained the design of components of RAN runtime and FlexVRAN. This included the description of interfaces for management and orchestration purposes and key results showing the feasibility and performance of the slicing approach in the context of the RAN. Finally, we detailed on implementation details of the descriptor triple. Here, we elaborated on the topics of two-level MAC scheduling, radio resource virtualization, processing chains to allow slice customization and the isolation of slice state.

References

3GPP (Mar. 2017a). *Study on new radio access technology: Radio access architecture and interfaces*. TR 38.801 V14.0.0.

– (Mar. 2017b). *Study on new radio access technology: Radio Interface Protocol Aspects*. TR 38.804 V14.0.0.

– (Oct. 2018a). *5G Network Resource Model (NRM)*. TS 28.541 V15.0.1.

– (July 2018b). *NR; Overall description;* TS 38.300, V15.3.0.

Aijaz, A. (Sept. 2018). Hap-SliceR: A Radio Resource Slicing Framework for 5G Networks With Haptic Communications. *IEEE Systems Journal* 12.3, 2285–2296.

Bosshart, P., Gibb, G., Kim, H.-S., et al. (Aug. 2013). Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. *SIGCOMM Comput. Commun. Rev.* 43.4, 99–110.

Chang, C. and Nikaein, N. (Dec. 2018a). Closing in on 5G Control Apps: Enabling Multiservice Programmability in a Disaggregated Radio Access Network. *IEEE Vehicular Technology Magazine* 13.4, 80–93.

– (2018b). RAN Runtime Slicing System for Flexible and Dynamic Service Execution Environment. *IEEE Access* 6, 34018–34042.

Ferrus, R., Sallent, O., Perez-Romero, J., et al. (2018). On 5G Radio Access Network Slicing: Radio Interface Protocol Features and Configuration. *IEEE Communications Magazine* PP.99, 2–10.

Foukas, X., Marina, M., and Kontovasilis, K. (June 2017). Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture. *The 23rd Annual International Conference on Mobile Computing and Networking (MobiCom '17)*. Snowbird, Utah, USA.

Foukas, X., Nikaein, N., Kassem, M. M., et al. (Dec. 2016). FlexRAN: A flexible and programmable platform for software-defined radio access networks. *CONEXT 2016, 12th International on Conference on Emerging Networking Experiments and Technologies*. Irvine, California, USA.

Gudipati, A., Li, L. E., and Katti, S. (2014). RadioVisor: A Slicing Plane for Radio Access Networks. *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14. Chicago, Illinois, USA: ACM, 237–238.

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

International Telecommunications Union (2015). *IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond. Recommendation ITU-R M.2083-0*. Tech. rep. ITU-R.

Kablan, M., Alsudais, A., Keller, E., et al. (2017). Stateless Network Functions: Breaking the Tight Coupling of State and Processing. *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 97–112.

Kim, J., Kim, D., and Choi, S. (2017). 3GPP SA2 architecture and functions for 5G mobile communication system. *ICT Express* 3.1, 1–8.

Kokku, R., Mahindra, R., Zhang, H., et al. (Oct. 2012). NVS: A Substrate for Virtualizing Wireless Resources in Cellular Networks. *IEEE/ACM Transactions on Networking* 20.5, 1333–1346.

– (Jan. 2013). CellSlice: Cellular wireless resource slicing for active RAN sharing. *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*. Bangalore, India, 1–10.

Kostopoulos, A., Agapiou, G., Junquan, D., et al. (July 2016). *Deliverable D2.2: System Architecture and Abstractions for Mobile Networks, v1.0*. Ed. by F.-C. Kuo.

Ksentini, A. and Nikaein, N. (2017). Toward Enforcing Network Slicing on RAN: Flexibility and Resources Abstraction. *IEEE Communications Magazine* 55.6, 102–108.

Mahindra, R., Khojastepour, M. A., Honghai Zhang, et al. (Oct. 2013). Radio Access Network sharing in cellular networks. *2013 21st IEEE International Conference on Network Protocols (ICNP)*. Göttingen, Germany, 1–10.

Marabissi, D. and Fantacci, R. (2017). Heterogeneous Public Safety Network Architecture Based on RAN Slicing. *IEEE Access* 5, 24668–24677.

Matias, J., Garay, J., Toledo, N., et al. (Apr. 2015). Toward an SDN-enabled NFV architecture. *IEEE Communications Magazine* 53.4, 187–193.

Nakao, A., Du, P., Kiriha, Y., et al. (2017). End-to-end Network Slicing for 5G Mobile Networks. *Journal of Information Processing* 25, 153–163.

Nikaein, N., Marina, M. K., Manickam, S., et al. (Oct. 2014). OpenAirInterface: A Flexible Platform for 5G Research. *SIGCOMM Comput. Commun. Rev.* 44.5, 33–38.

Nikaein, N., Schiller, E., Favraud, R., et al. (2015). Network Store: Exploring Slicing in Future 5G Networks. *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture*. MobiArch '15. Paris, France: ACM, 8–13.

Rost, P., Mannweiler, C., Michalopoulos, D. S., et al. (May 2017). Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks. *IEEE Communications Magazine* 55.5, 72–79.

Sallent, O., Perez-Romero, J., Ferrus, R., et al. (Oct. 2017). On Radio Access Network Slicing from a Radio Resource Management Perspective. *IEEE Wireless Communications* 24.5, 166–174.

Schmidt, R., Chang, C.-Y., and Nikaein, N. (May 2019a). FlexVRAN: A flexible controller for virtualized RAN over heterogeneous deployments. *ICC 2019, 53rd IEEE International Conference on Communications*. Shanghai, China.

– (2019b). Slice Scheduling with QoS-Guarantee Towards 5G. *IEEE Global Communications Conference (Globecom) 2019*. Waikoloa, Hawaii, USA.

Shafi, M., Molisch, A. F., Smith, P. J., et al. (June 2017). 5G: A Tutorial Overview of Standards, Trials, Challenges, Deployment, and Practice. *IEEE Journal on Selected Areas in Communications* 35.6, 1201–1221.

Tzanakaki, A., Anastasopoulos, M. P., and Simeonidou, D. (May 2017). Optical networking interconnecting disaggregated compute resources: An enabler of the 5G vision. *2017 International Conference on Optical Network Design and Modeling (ONDM)*. Budapest, Hungary, 1–6.

Zaki, Y., Zhao, L., Goerg, C., et al. (Aug. 2011). LTE mobile network virtualization. *Mobile Networks and Applications* 16.4, 424–432.

LIST OF TABLES

Table I
State of the Art of RAN slicing

Authors (year)	Solution level	Radio resources	CP function	UP function
Nikaein et al. (2015)	Network level	–	Dedicated	Dedicated
Kokku et al. (2012)	BS level	Physical or virtualized resource sharing	–	–
Mahindra et al. (2013)	Gateway/BS levels	Physical or virtualized resource sharing	–	–
Kokku et al. (2013)	Gateway level	Virtualized resource sharing	–	–
He et al. (2015)	Gateway level	App-oriented virtualized resource sharing	–	–
Aijaz (2018)	Gateway level	Learning-based virtualized resource sharing	–	–
Zaki et al. (2011)	BS level	Physical resource sharing	Dedicated	Dedicated
Foukas et al. (2016)	BS level	Physical or virtualized resource sharing	Shared	Shared
Gudipati et al. (2014)	BS level	Physical 3D resource sharing	Dedicated	Dedicated till programmable radio
Nakao et al. (2017)	BS level	Dedicated spectrum allocation	Dedicated	Dedicated
Marabissi et al. (2017)	BS level	Virtualized resource sharing	Dedicated	Dedicated
Sallent et al. (2017)	BS level	Physical resource sharing	Split into tenant-specific and common	Shared
Rost et al. (2017)	BS level	Physical resource sharing	Split into cell- and user-specific	Dedicated till real-time RLC
Ksentini et al. (2017)	BS level	Flexible resource sharing	Dedicated	Shared
Foukas et al. (2017)	BS level	Virtualized resource sharing	Split into cell- and user-specific	Dedicated till PHY
Ferrus et al. (2018)	BS level	Physical resource sharing	Dedicated	Dedicated or shared till PHY
Chang et al. (2018b)	BS level	Physical or virtualized resource sharing	Split into cell- and user-/slice-specific	Support different levels of isolation and sharing

Table II
BS-common and user-specific functions in the case of LTE

Process	BS-common functions	User-specific functions
Location tracking and paging	Tracking area update, core network (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	–

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 radio bearers

Table IV
Mapping between resource abstraction type and allocation type in LTE

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

Table V
UP NETWORK FUNCTIONS AND THE DECOUPLED STATES

Layer	**Network function**	**Network state**
PHY	Radio frequency processing	Carrier frequency, Spectrum bandwidth
	(Inverse) Discrete Fourier Transform	Point of DFT, Output indexes
	Multi-antenna processing	Transmission mode, beamforming matrix
	(De-)Modulation	Modulation order, reference symbol information
	Bit-rate processing	Information about coding, scrambling, rate matching, and cycle redundancy check
MAC	Hybrid automated repeated request process	HARQ index, user identity, redundancy version
	(De-)Multiplexing	(De-)Multiplexed 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 and service data units
	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	QoS flow identity, QoS profile, mapping policy
	Marking QoS flow identity	QoS flow identity

List of Figures