NexRAN: Closed-loop RAN slicing in POWDER - A top-to-bottom open-source open-RAN use case

David Johnson  
johnsond@cs.utah.edu  
University of Utah  
Salt Lake City, Utah, USA

Dustin Maas  
dmaas@cs.utah.edu  
University of Utah  
Salt Lake City, Utah, USA

Jacobus Van Der Merwe  
kobus@cs.utah.edu  
University of Utah  
Salt Lake City, Utah, USA

ABSTRACT

Much like earlier "network softwarization" efforts, the Open RAN concept is poised to have a transformative impact on the manner in which radio access networks (RANs) are realized and operated. The inherent complexity of the RAN ecosystem and the fact that it is rapidly evolving makes Open RAN a rich area of research into use cases, system realization, security, and more. This same complexity, however, hampers research efforts. Specifically, there is a lack of end-to-end open source software and fully-developed use cases associated with the Open RAN ecosystem. Further, to truly advance the state of the art will require use cases to be explored in realistic wireless environments. This paper describes our efforts to address these shortcomings by realizing NexRAN, a top-to-bottom, open-source Open RAN use case in the POWDER mobile and wireless research platform. Specifically, NexRAN allows closed-loop control of a RAN slicing realization in an O-RAN ecosystem. RAN slicing is implemented in the srsRAN open source mobility stack and is exposed through a custom service model to the NexRAN xApp, which executes on a RAN intelligent controller (RIC) from the O-RAN Alliance. The NexRAN xApp realizes policy driven closed-loop control of RAN slices by reading the current state of RAN elements (using the O-RAN key performance measurements (KPM) service model) and controlling slice behavior via the custom slicing service model. We demonstrate and evaluate NexRAN in the POWDER platform and have open sourced all aspects of our realization to enable research into this domain.

CCS CONCEPTS

• Networks → Mobile networks; Network architectures; Network experimentation.

KEYWORDS

open RAN, RAN slicing, closed-loop RAN control, wireless testbed

1 INTRODUCTION

The "softwarization" of network functionality—software-defined networking, network function virtualization, network programmability, network virtualization—that has fundamentally changed networking over the last decade is now also being applied to mobile networks in general and the radio access network (RAN) in particular. Specifically, the "Open RAN" concept has evolved from early research prototypes [15] to consortia with broad industry participation [22] and has also attracted interest from regulators [3]. The inherent complexity of the RAN ecosystem, coupled with the fact that RAN functionality by itself is rapidly evolving, suggests Open RAN as a broad emerging research area, with open issues in applicable use cases, spectrum management, systems realization, security, and more.

This same inherent complexity, however, hampers research efforts in this area. First, there is a lack of open source frameworks to bootstrap research efforts in this space. It is true that, for example, the O-RAN Alliance provides open source software via the O-RAN Software Community [22]. These code bases provide the O-RAN "control stack" but do not themselves provide the necessary O-RAN support for existing RAN implementations (e.g., an O-RAN-enabled eNodeB/gNodeB). Second, and related, the example use cases that are currently available within these open source frameworks are still under development (e.g., traffic steering, admission control, etc), or are limited in functionality (e.g., RAN metrics collection). As a result, it is difficult for practitioners to develop an understanding of the full end-to-end functionality enabled by an open RAN approach. Third, while there is general agreement about the potential of an open RAN approach, developing use cases that could truly advance the state of the art requires exploration and testing in realistic wireless environments to explore and validate the feasibility of the open RAN architecture and the applications it enables.

This paper describes our efforts to address these shortcomings by realizing NexRAN, a top-to-bottom open-source Open RAN use case in the POWDER mobile and wireless research platform [12]. Specifically, NexRAN allows closed-loop control of a RAN slicing realization in an O-RAN ecosystem. RAN slicing is implemented in the srsRAN open source mobility stack and is exposed through a custom service model to the NexRAN xApp, which executes on a RAN intelligent controller (RIC) from the O-RAN Alliance. Our RAN slicing implementation realizes a form of slicing where different slices share the same frequency band, UEs can be explicitly
associated with slices, and a slice-aware scheduler in the base station implements the RAN resources associated with each slice. The NexRAN xApp realizes policy driven closed-loop control of RAN slices by reading the current state of RAN nodes (e.g., eNodeBs and gNodeBs). The E2 agent implements the core E2 Application Protocol (E2AP), has access to the internal RAN components in the eNodeB’s stack to monitor and modify RAN parameters, and supports E2 service models to export RAN metrics and controls to xApps. NexRAN exposes this functionality, via a RESTful API, to a RAN slicing manager. The slice manager can create slices, bind/unbind them to multiple eNodeBs, bind/unbind UEs to those slices, and dynamically modify slice resource allocations. We describe our design and implementation in the following sections using a top-down approach.

3 NexRAN DESIGN AND IMPLEMENTATION

Figure 1 provides an overview of the NexRAN Open RAN framework we have realized by combining software from the O-RAN Software Community and srsRAN. Specifically, we added a slice-aware scheduler and an O-RAN E2 agent to srsRAN, and a custom xApp to control slicing. As shown in Figure 1, E2 is a north-bound interface that connects the RIC with underlying radio equipment, such as eNodeBs and gNodeBs. The E2 agent implements the core E2 Application Protocol (E2AP), has access to the internal RAN components in the eNodeB’s stack to monitor and modify RAN parameters, and supports E2 service models to export RAN metrics and controls to xApps. NexRAN exposes this functionality, via a RESTful API, to a RAN slicing manager. The slice manager can create slices, bind/unbind them to multiple eNodeBs, bind/unbind UEs to those slices, and dynamically modify slice resource allocations. We describe our design and implementation in the following sections using a top-down approach.

3.1 xApp and Northbound API

We developed a custom NexRAN xApp in C++, using some of the xApp and RIC message router (RMR) framework libraries provided by the O-RAN Software Community. The xApp implements the NexRAN RAN slicing service model, and consumes an extended KPM service model to obtain metrics for use in auto-adaptive RAN slicing (further described in Section 3.5).

The NexRAN xApp provides a northbound, RESTful interface for administrative control and monitoring. It defines several primary objects: NodeB, Slice, UE, each of which may be created, updated, and deleted. When a NodeB is created in the xApp, the xApp attempts to subscribe to the NodeB’s events via the E2 protocol. A Slice contains a scheduling policy definition. Administrators may bind Slices to NodeB’s; this tells the NodeB’s scheduler that the slice and its associated UEs should be scheduled according to the slice’s policy. Finally, administrators create UE objects to inform NexRAN of particular, known IMSIs that may connect to a NodeB. UEs may be bound to a single Slice at a time; this binding tells the scheduler that the UE should be scheduled in accordance with its parent Slice’s...
We extended srsRAN with an E2 agent that implements the core (A, B) and three different proportional shares. X represents the special subframes that prioritize unidentified UEs.

<table>
<thead>
<tr>
<th>Proportion A:B</th>
<th>Subframe Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:2</td>
<td>X B B A A X A B B A</td>
</tr>
<tr>
<td>2:1</td>
<td>X B A A B X A A B A</td>
</tr>
<tr>
<td>1:1</td>
<td>X B A B A X B A B A</td>
</tr>
</tbody>
</table>

Figure 2: NexRAN subframe allocation examples for two slices (A, B) and three different proportional shares. X represents special subframes that prioritize unidentified UEs.

We add a single class that is responsible for (1) reacting to messages from the E2 agent and updating the slicing configuration; (2) generating the subframe allocation for the current slicing configuration; (3) reacting to interfaces from lower layers in the stack and tracking which temporary UE identities belong to each slice; and (4) providing a list of the RNTIs that should be prioritized in each subframe.

Policy. UEs may be unbound from Slices, and Slices unbound from NodeBs, at any time.

### 3.2 RAN slicing service model

The NexRAN service model maps the northbound API onto common E2 abstractions and messages—the xApp sends E2 messages to NodeBs in response to northbound API invocations. Most northbound API create or update operations map to E2 Control messages. For instance, Slice create, update, and delete map to SliceConfig and SliceDelete control messages, and binding or unbinding a UE to and from a Slice map to SliceUeBindRequest and SliceUeUnbindRequest messages. Per-slice proportional allocations can be modified via the SliceConfig message. The service model also provides periodic E2 Indication messages in response to subscriptions from xApps; the indication contains a list of bound slices and UEs active at that NodeB.

### 3.3 E2 Agent

We extended srsRAN with an E2 agent that implements the core E2AP protocol and provides abstractions for further extension to new E2 service models. The E2 agent implements two service models: the NexRAN RAN slicing service model, and an extension of the KPM service model. Our KPM extensions are additive in nature, to index metrics by slice in addition to UEs, and are therefore backwards-compatible.

When the srsRAN eNodeB initializes, the E2 agent connects to the O-RAN RIC and runs the E2 Setup procedure to advertise its list of supported service models to the RIC and its xApps. srsRAN is multithreaded, and it and the E2 agent dedicate threads to specific tasks. Service models may also spawn their own threads as needed to implement asynchronous notifications, e.g., those that periodically report metrics or events in response to xApp subscriptions.

Our E2 agent provides an implementation of the standard O-RAN key performance measurements (KPM) service model [26] to provide metrics. As shown in Figure 1, we have also implemented a custom 3GPP-like service model to expose our RAN slicing implementation as a set of abstractions and controls to xApps executing on the RIC.

### 3.4 Slice scheduler

The slice scheduler at the eNodeB implements a subframe-based proportional slicing method for data on the physical downlink shared channel (PDSCH) using the slice definitions described by the NexRAN service model, and provided by the slice manager via the xApp. With the exception of a periodic special subframe included to guarantee that UEs which have yet to be identified and associated with a slice are able to attach to the network, each subframe gives priority to a single slice. By default, if the slice with priority in a given subframe doesn’t consume all of the available resources, UEs from other slices may be scheduled after those from the priority slice.

Slices are scheduled in a round-robin fashion, each receiving one or more consecutive subframes per round according to their allocation share. Figure 2 shows allocations for a two-slice scenario using a few example shares, which can be described as the ratio A:B of subframes allocated to each slice per round. The columns marked X represent the periodic special subframes. A scheduling round is complete when the proportional allocation defined by the slice manager has been satisfied.

In general, NodeBs do not keep track of the international mobile subscriber identities (IMISIs) used to identify subscribers to the core network. In addition, IMISIs are transmitted over the air as rarely as possible in order to protect subscriber identities; temporary mobile subscriber identities (TMSIs) are used instead. At the same time, the NexRAN xApp identifies the UEs that belong to each slice using IMISIs, and is completely unaware of the TMSIs and radio network temporary identifiers (RNTIs) that might be assigned by the core network and eNodeB, respectively. As such, it keeps the eNodeB informed about the slices and associated IMISIs, and it is left to the eNodeB to determine the identity (IMSI and TMSI) of each UE, map that identity to its RNTI, and then update the TMSIs and RNTIs if they change over time. In order to accomplish this, we (1) use a fresh instance of the srsRAN EPC, so that UEs are forced to send their IMISIs the first time they attach; and (2) modify the srsRAN eNodeB to decode several non-access stratum (NAS) messages in transit between the UEs and the EPC in order to capture the IMISIs transmitted in the initial attach procedure, and then capture and update their corresponding TMSIs over time.

We add a single class that is responsible for (1) reacting to messages from the E2 agent and updating the slicing configuration; (2) generating the subframe allocation for the current slicing configuration; (3) reacting to interfaces from lower layers in the stack and tracking which temporary UE identities belong to each slice; and (4) providing a list of the RNTIs that should be prioritized in each subframe to the scheduler.

The scheduler is work-conserving by default, meaning that UEs belonging to the priority slice are scheduled first, followed by UEs belonging to other slices, and finally by UEs not associated with any slice, as long as there are remaining resource blocks. In special subframes, unidentified UEs are scheduled first, followed by the UEs belonging slices, followed by UEs not associated with any slice. UEs in each category are scheduled round-robin within the subframe.

We note that RAN slicing at the granularity of subframes is not a novel idea; the authors of [30] use a similar approach at the eNodeB to evaluate their slice optimization strategy in a simulated network. Our contribution in this regard is an open source subframe-based slice scheduler that enables the top-to-bottom RAN slicing use case.
3.5 Policy-driven dynamic slice scheduling
The NexRAN xApp allows administrators to configure the proportional allocation scheduler on a per-slice basis, and provides allocation policy extensions through which the xApp can dynamically modify slice resource allocations. We have implemented two such extensions: balanced slice throughput and slice throttling. These extensions monitor per-UE and per-slice throughput and other metrics, via our extended KPM service model, and modify per-slice proportional allocations according to policy and load.

The balanced slice throughput extension attempts to drive slices to the same overall throughput, as measured by the KPM service model at the PDPC layer. This mechanism sums the total bandwidth used by all auto-equalized slices in each new KPM report, checks if any slices have diverged from an equal distribution, and if so, computes new share values (proportions) for each slice. This mechanism is only invoked if at least 30% of the reporting NodeB’s available PRBs were used, so that low-throughput slices are not unfairly starved.

The slice throttling extension attempts to prevent slices from consuming too much bandwidth in a given time period. It accepts several parameters: throttle_period, throttle_threshold, and throttled_share. When throttle_threshold throughput is exceeded within any throttle_period window, the slice’s share is set to throttled_share for a duration of throttle_period, and when the period ends, throttling is removed. The policy maintains its threshold counters during throttling, and per-period throughput is not reset at the end of a throttle_period.

4 EVALUATION
We evaluated NexRAN on the POWDER platform. Our evaluation specifically focuses on illustrating the top-to-bottom closed-loop nature of our implementation. As described earlier, the NexRAN xApp “reads” the RAN related measurements using the extended KPM service model and “writes” (controls) the RAN slices via the NexRAN RAN slicing service model. Specifically, the NexRAN xApp realizes the two specific RAN control policies described in Section 3.5, i.e., a policy that balances the per-slice throughput between slices and a policy to throttle the throughput of a slice if its aggregate throughput over a specific time period exceeds a certain threshold.

4.1 Experimental setup
We performed an evaluation of NexRAN using the POWDER indoor over-the-air (OTA) lab and the POWDER controlled RF environment. Figure 3 shows the two setups and the equipment involved in each configuration. Each UE was realized using a small-form-factor compute node (Intel NUC 8559 or 5300), an NI B210 SDR, and srsRAN (release 20.10.1). The eNodeBs were realized by combining a compute node (Dell R740 or NUC 5300), an NI X310 or B210 SDR, and a modified version of srsRAN 20.10.1 that includes our RAN slicing implementation. The compute node also executed the evolved packet core (EPC) network. This “mobile infrastructure” is controlled by the RIC and NexRAN xApp executing on another compute node (Dell R740). (For our evaluation the mobile infrastructure and the RAN control setup are realized as two separate experiments that are interconnected via a shared network (VLAN) connection. This POWDER capability to interconnect two separate experiments is convenient when the experiment profiles are complex (as is the case here), and/or can be used independently or combined with other profiles.) As shown in Figure 3 the indoor OTA lab configuration involves one eNodeB and four UEs, while the controlled RF environment setup has two UEs and one eNodeB.

4.2 Evaluation Results
Figure 4 shows a time series of NexRAN in an indoor OTA lab setup and realizing the balanced slice throughput policy. The y-axis on the left shows the aggregate per-slice throughput. The y-axis on the right shows the per-slice allocation of resources. (Our implementation represents slice allocations as an integer from the range
The y-axis on the left again shows the aggregate per-slice throughput between radios can be programmatically controlled. For

of resources. In the POWDER controlled RF environment the at-

pattern repeats in the rest of the run.

to be monitored throughout and the same throttled/unthrottled

throughput than slice 2 share of the resources, with

slice 1 since the competing slice (slice 2)

remains unchanged, slice 2

is subject to a slice throttling policy, while slice 1

again shows the aggregate per-slice throughput, while the y-axis on the left

is maintained throughout the experiment.

Figure 5 shows a time series of NexRAN in an indoor OTA lab

setup and realizing the slice throttling policy. The y-axis on the left

again shows the aggregate per-slice throughput, while the y-axis on

the right shows the per-slice allocation of resources. In this example

slice 1 is subject to a slice throttling policy, while slice 2 is not.

During the initial part of the experiment, i.e., up to approximately

200 seconds, both slices get half of the available resources, and

the xApp start collecting usage data associated with slice 1. At

200 seconds slice 1 exceeds its threshold and the throttling policy

reduces its resource allocation dramatically. During this period,

i.e., from 200 seconds to 500 seconds, while its resource allocation

remains unchanged, slice 2 achieves much higher throughput

since the competing slice (slice 1) is throttled. The throttling

period ends at 500 seconds, and both slices return to an equal

share of the resources, with slice 1 again achieving a higher

throughput than slice 2. The throughput of slice 1 continues to

be monitored throughout and the same throttled/unthrottled

pattern repeats in the rest of the run.

Figure 6 shows a time series of NexRAN in the controlled RF

environment and realizing the balanced slice throughput policy.

The y-axis on the left again shows the aggregate per-slice through-

put, while the y-axis on the right shows the per-slice allocation

of resources. In the POWDER controlled RF environment the atten-
nuation between radios can be programmatically controlled. For

this experimental run the attenuation of the path between the UE

associated with slice 2 and the eNodeB is modified in a sequence

of 0 dB, 20 dB, 0 dB, 20 dB etc. When the attenuation is increased

to 20 dB, the NexRAN balanced throughput policy increases the re-

source allocation for slice 2 (to balance its throughput with that of

slice 1). When the attenuation is reset to 0 dB the resource alloca-

tion of both slices are adjusted to approximate parity. For example,

in Figure 6 the experiments start with 0 dB attenuation for all RF

paths and the slices have a balanced throughput of approximately

17 Mbps. At approximately 280 seconds the attenuation is increased

and after a period of adjustment, at approximately 380 seconds,

the throughput between the slices are roughly balanced again at a

(reduced) rate of approximately 15 Mbps. At approximately 530 sec-

onds the attenuation is reset to 0 dB and after some adjustment the

throughput for both slices returns to approximately 17 Mbps, etc.

5 EXPERIENCES AND DESIGN CHOICES

In this section, we describe some of our O-RAN design choices and

experiences.

Mapping an API to E2AP procedures. As described in Section 2, the

eE2AP protocol provides several procedure styles relevant to

xApps: subscriptions (which when triggered, fire REPORT, INSERT,
or POLICY service actions); asynchronous control requests from

xApp to RAN node; control requests that resume or modify an

ongoing RAN procedure; and notifications (indication messages).

The core eE2AP is not intended to be the point of extension for

xApp designers (that is the role of the service model abstraction), so

we had two choices when mapping the NexRAN RESTful JSON north-

bound API to the E2 interface. First, we could model the NodeB/Slice

and Slice/UE binding configuration as subscription requests with

POLICY actions—and with service-model-defined, opaque policy
descriptions containing the binding information, and use this policy

as the configuration input to the slicing scheduler. Second (which

we chose), we could model each binding configuration change as

Figure 5: NexRAN slice throttling policy - Indoor OTA Lab

Figure 6: NexRAN balanced slice throughput policy - Controlled RF Environment
Looking at the configuration of a subscription for each policy change, despite the fact that only a small parameter subset may have changed. Mapping the RESTful API to asynchronous control request messages to the RAN nodes is a better match for the NexRAN service.

Defining a service model. Although the E2AP protocol is defined in a 3GPP-ish, asn.1 style, and the service models defined by the O-RAN Alliance are similarly defined, service model definitions are opaque to E2AP. Therefore, if integrating a pre-existing system that has a RESTful JSON-based API, it is valid to define the service model in JSON. In the NexRAN system, we found it most natural and convenient to expose a RESTful JSON northbound API—but we opted to map this API to an asn.1 service model definition. Currently, xApps must implement asn.1-based E2AP subscription and control messages directly, and other useful service models (e.g., KPM and others currently under development) are asn.1-based, so conformity is sensible, although perhaps a bit more painful than other options. Additionally, the E2AP INSERT subscription service action is designed to interrupt an existing LTE/5G procedure flow, transmit the relevant initiating message to the subscribed xApp, and halt the procedure while waiting for the xApp to possibly modify the procedural flow or response; all of these procedures are asn.1-based.

Rapidly-evolving landscape. The O-RAN RIC specifications and reference software are still under active development. For instance, the E2AP Setup procedure initiator changed from the RIC to the RAN prior to the 0.01 release. The KPM service model specification, while already useful, can (and surely will) support a wider variety of useful metrics, and additional indexings of them (e.g., KPM and others currently under development) are asn.1-based, so conformity is sensible, although perhaps a bit more painful than other options. Additionally, the E2AP INSERT subscription service action is designed to interrupt an existing LTE/5G procedure flow, transmit the relevant initiating message to the subscribed xApp, and halt the procedure while waiting for the xApp to possibly modify the procedural flow or response; all of these procedures are asn.1-based.

The primary contributions of our work involve using RAN slicing to realize a closed-loop, open RAN use case implemented as a top-to-bottom open-source artifact and evaluating it in a realistic RAN environment. As such our work is related to previous RAN slicing efforts, to other open RAN environments and use cases, research platforms that enable realistic RAN evaluations (and of course the open source stacks from srsRAN [31] and the O-RAN Alliance [22] that enabled our work).

RAN slicing is a fairly well studied topic, including relatively recent efforts exploring plausible slicing implementation options for 5G RAN [13], the Orion LTE based RAN slicing implementation focused on per-slice performance guarantees [14], analytical treatment of RAN slicing resources [33], as well as earlier efforts that realized RAN slicing without modifying basestation schedulers [17] and in RAN slicing in a WiMAX environment [11]. Beyond these research efforts RAN slicing is also included in the 5G new radio efforts being undertaken by 3GPP [1]. Of these efforts, NexRAN slicing is most related to the Orion effort which also focused on a systems implementation of RAN slicing using an open source mobility stack.

In terms of open RAN environments our work specifically utilizes the open source ecosystem provided by the O-RAN Alliance [22]. The O-RAN Alliance is a world-wide community that draws its large and growing members base from mobile operators, vendors as well as research and academic institutions. O-RAN was also influenced by efforts associated with the xRAN Forum, Cisco’s Open vRAN and the Telecom Infra Project (TIP) OpenRAN [19]. Earlier, mostly academic, efforts to create open and programmable RAN environments include a programmable RAN slicing architecture [18] and the FlexRAN work [15] that was one of the earliest efforts to explore clean software-defined RAN abstractions. FlexRAN was also adopted into the Mosaic-5G effort[2], which has recently been transferred to fit into the OpenAirInterface Software Alliance.

In terms of academic research, the FlexRAN implementation [2, 15] is a popular platform to enable use case development [20]. The O-RAN Alliance has also published a white paper describing a variety of use cases [23]. Some of these anticipated use cases are quite sophisticated, e.g., AI-enabled optimization of massive MIMO
We thank the anonymous WiNTECH ’21 reviewers for their comments on this paper. We performed our experiments on machines provided by the POWDER testbed [12]. This material is based upon work supported by the National Science Foundation under Grant Number 1827940.

REFERENCES

[27] IRIS. IRIS - the software defined radio (SDR) testbed. 2020.
[31] FlexRAN in Mosaic5G. http://mosaic-5g.io/flexran/.