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

ACM Reference Format:
David Johnson, Dustin Maas, and Jacobus Van Der Merwe. 2022. NexRAN: Closed-loop RAN slicing in POWDER - A top-to-bottom open-source open-RAN use case. In *The 15th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterization (WiNTECH '21), January 31–February 4, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3477086.3480842

WiNTECH '21, January 31–February 4, 2022, New Orleans, LA, USA

ACM ISBN 978-1-4503-8703-3/22/01…$15.00
https://doi.org/10.1145/3477086.3480842

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 elements (using the standard 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. To our knowledge our work on NexRAN represents the first top-to-bottom open source realization of an O-RAN xApp.

We make the following specific contributions:

- We implement an O-RAN E2 agent in the open source srsRAN code base to allow srsRAN to be used as the underlying mobile functionality in an O-RAN environment.
- We develop RAN slicing in srsRAN and expose that to the O-RAN environment through a (custom) service model.
- We develop a RAN slicing xApp that performs closed-loop RAN slicing control to serve as a top-to-bottom example of controlling RAN functionality in an open RAN ecosystem.
- We evaluate our implementation on the POWDER platform, illustrating the platform's utility in performing open RAN related research.
- We open source all our efforts, including a POWDER profile that enables our experiments to be repeated, to serve as a building block for open RAN related research.

2 BACKGROUND: O-RAN

The O-RAN architecture describes a model for RAN resource control, managed at the upper level by orchestration and automation components (e.g., policy, configuration, and the non-realtime RAN Intelligent Controller (RIC)). These components control and communicate with the near-realtime RIC via the *A1* interface. The near-realtime RIC provides management of and connectivity to RAN nodes (e.g., eNodeB/gNodeB, RU/DU, etc). Its core set of services is extensible by custom third-party xApps, which are instantiated as cloud services and have low-latency connectivity to RAN nodes. xApps communicate with the RIC and its managed RAN nodes via the E2 interface.

The E2 Application Protocol (E2AP) [24] defines several messaging procedures for RAN nodes and xApps. xApps can subscribe to events occurring at RAN nodes by specifying per-subscription triggers (conditions that match events). Subscription requests contain a list of service actions to be run when triggered, and when triggered, those actions may report data or provide notice of the start of a RAN procedure (e.g., X2 handover) as an indication message. xApps may also send asynchronous control request messages to RAN nodes, providing configuration or requesting services. Control messages may also resume or modify a RAN procedure that had been halted when an indication was sent.

General E2AP message procedures are extensible via the *service model* abstraction: service models (often abbreviated E2SM) [25] expose RAN semantics and controls to xApps. For instance, the Key Performance Measurements (KPM) service model [26] supports subscriptions that provide periodic metrics reports. Most E2AP message procedures contain an opaque field that carries service model-specific content. xApps and RAN nodes that support the same service model can then exchange notifications and control messages.

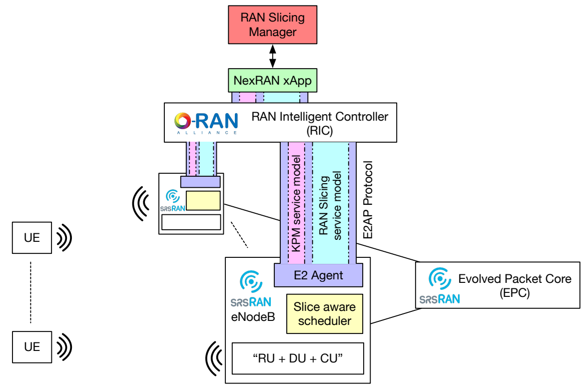

Figure 1: NexRAN Open RAN open source RAN slicing

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 NodeBs; 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

Proportion A:B	Subframe Allocation									
3:2	X	B	B	A	A	X	A	B	B	A
2:1	X	B	A	A	B	X	A	A	B	A
1:1	X	B	A	B	A	X	B	A	B	A
Subframe Number	0	1	2	3	4	5	6	7	8	9

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

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 (IMSIs) used to identify subscribers to the core network. In addition, IMSIs 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 IMSIs, 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 IMSIs, 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 IMSIs 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 IMSIs 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 PDCP 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.

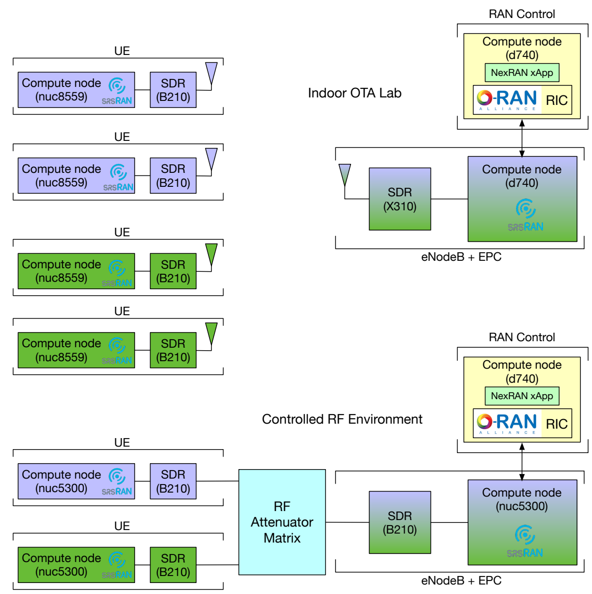

Figure 3: Experimental setup

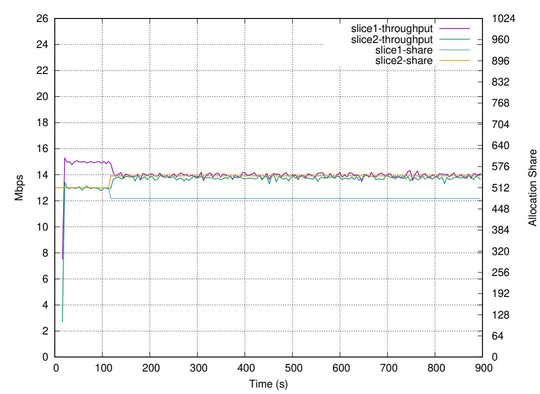

Figure 4: NexRAN balanced slice throughput policy - Indoor OTA Lab

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

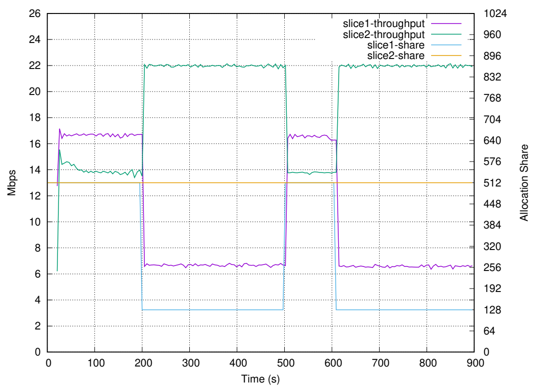

Figure 5: NexRAN slice throttling policy - Indoor OTA Lab

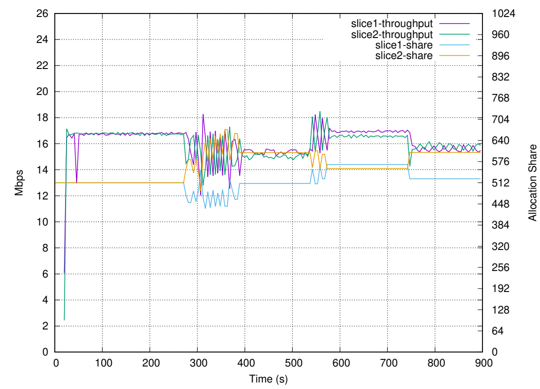

Figure 6: NexRAN balanced slice throughput policy - Controlled RF Environment

0 to 1024. The ratio of these per-slice allocations determines the share of radio resources allocated to a slice. E.g., if two slices get the same allocation, e.g., 512:512, or 768:768, then the slices will each receive half of the available resources. If one slice gets double the allocation of another slice, e.g., 512:256, then that slice would receive double the amount of resources.) For this experiment two UEs are associated with each of two slices. The experiment starts out without the balanced slice throughput policy in place. As shown in Figure 4, each slices starts off with an equal share of the available RAN resources. At approximately 120 seconds into the run the balanced throughput policy is activated. At this point NexRAN adjusts the resource allocations to satisfy the balanced throughput objective. The results show that the balanced throughput objective 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 throughput, while the y-axis on the right shows the per-slice allocation of resources. In the POWDER controlled RF environment the attenuation 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 resource allocation for `slice 2` (to balance its throughput with that of `slice 1`). When the attenuation is reset to 0 dB the resource allocation 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 seconds 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 E2AP 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 E2AP 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 northbound 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 a new control request sent to the NodeB. Either choice has equivalent error-handling mechanisms and service model-opaque descriptor fields to describe the configuration. However, our design for NexRAN's auto-adapative slicing mechanism was intended to support frequent sub-second scheduling parameter changes. To model this configuration as a subscription with a POLICY service action would have required both a *SubscriptionDeleteRequest* and a *SubscriptionRequest* procedure 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 01.00 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 01.00 indexed per-UE byte counters by QCI, and xApp authors may wish to write policies over otherwise-indexed metrics—NexRAN being one example). The O-RAN Alliance is actively developing additional service model specifications, and some of those may have been reusable in place of some of the custom NexRAN service model components.

The reference RIC system provided by the O-RAN Software Community is an excellent proving ground, but we ran into some rough edges in the implementation. For instance, internally, the reference RIC uses the RIC message router (RMR) [27] to pass E2AP messages between endpoints (e.g. RIC microservices and xApps), and there is at least some conflation between RMR and E2AP messages. If an xApp wants to fire multiple subscriptions in parallel, and match subscription response messages with the corresponding request; it must set the RMR transaction id (xid) header field; and the subscription response then carries this transaction id; so the match can be made (as of the cherry release [28]). Otherwise, there is no way to match; the reference RIC changes the requestor_id and instance_id bits in the original subscription request before passing it to the RAN node, so that it can aggregate subscriptions; but then does not "demultiplex" the replies back to the xApp. Based on our reading of the E2AP specification, such matching should be possible solely based on the identifiers in the subscription request.

6 NEXRAN OPEN SOURCE AVAILABILITY

Towards our goal of enabling open RAN related research through our efforts, all software associated with NexRAN are publicly available:

(i) O-RAN RAN slicing POWDER profile [5], a POWDER specification that specifies the hardware and software resources needed to automatically instantiate the setup described in this paper,
(ii) srsRAN with O-RAN E2 and slice aware scheduler [8], i.e., a fork of the srsRAN code base with our enhancements,
(iii) NexRAN xApp [4], the xApp that interacts with both the KPM and RAN slicing service models,
(iv) POWDER fork of e2 core repo with minor bugfixes [6],
(v) POWDER fork of kpimon xApp with bugfixes [7].

7 RELATED WORK

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 basefrom 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 transfered 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 quality-of-service (QoS) and quality-of-experience (QoE) and radio resource allocation based on the flight path of unmanned areal vehicles (UAVs). The most developed use case release by the O-RAN Alliance to date involves traffic steering which, like NexRAN makes use of monitoring via the KPM service model, but the details of RAN control (e.g. handover to steer traffic) are still under development. Further, as we have stated earlier, the O-RAN software typically does not include realizations of the underlying RAN functionality, limiting the usefulness of the use case realizations for practitioners.

Finally, in terms of realistic RAN environments for evaluating open RAN related research, we have made use of the controlled RF environment and indoor over-the-air lab available in the POWDER platform [32] and have made NexRAN available as a POWDER profile [5]. Many wireless testbeds exist, some that make use of commercial (black box) equipment [9, 10] and othere that, similar to POWDER, provides SDR equipment [16, 21, 29]. To our knowledge, however, POWDER is unique in providing both the hardware and the software building blocks to enable open RAN related research.

8 CONCLUSION

In this paper we presented our work on NexRAN, a closed-loop RAN slicing use case developed using the srsRAN and O-RAN Alliance source code bases. To our knowledge, NexRAN is the first open-source top-to-bottom O-RAN use case. We evaluated NexRAN on the POWDER mobile and wireless platform. Our primary goal is to enable open RAN related research; to that end, we released all NexRAN software, including a profile to replicate the work described in this paper on the POWDER platform, as open source.

ACKNOWLEDGMENTS

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

[1] 5G; Study on new radio access technology (3GPP TR 38.912 version 14.1.0 Release 14). 3GPP Technical Report.

[2] FlexRAN in Mosaic5G. http://mosaic-5g.io/flexran/.

[3] FCC Seeks Comment on Open Radio Access Networks. https://www.fcc.gov/document/fcc-seeks-comment-open-radio-access-networks-0, March 2021.

[4] NexRAN xApp. https://gitlab.flux.utah.edu/powderrenewpublic/nexran/, 2021.

[5] O-RAN RAN slicing. https://www.powderwireless.net/p/PowderProfiles/O-RAN, 2021.

[6] POWDER fork of e2 core repo with minor bugfixes. https://gitlab.flux.utah.edu/powderrenewpublic/e2, 2021.

[7] POWDER fork of kpimon xApp with bugfixes. https://gitlab.flux.utah.edu/powderrenewpublic/ric-scp-kpimon, 2021.

[8] srsRAN with O-RAN E2. https://gitlab.flux.utah.edu/powderrenewpublic/srslte-ric/, 2021.

[9] 5G-VINNI Consortium. 5G-VINNI: 5G verticals innovation infrastructure, 2020.

[10] 5GENESIS. 5GENESIS: 5th generation end-to-end network, experimentation, system integration, and showcasing, 2020.

[11] G. Bhanage, R. Daya, I. Seskar, and D. Raychaudhuri. VNTS: a virtual network traffic shaper for air time fairness in 802:16e slices. In *IEEE ICC - Wireless and Mobile Networking Symposium*, 2010.

[12] J. Breen, E. Eide, E. Lewis, D. Reading, A. Buffmire, M. Hibler, D. Maas, R. Ricci, J. Duerig, D. Johnson, A. Orange, D. Schurig, K. Dutt, S. K. Kasera, N. Patwari, L. B. Stoller, J. Van der Merwe, K. Webb, and G. Wong. Powder: Platform for Open Wireless Data-driven Experimental Research. In *ACM WiNTECH proceedings*, September 2020.

[13] S. E. Elayoubi, S. B. Jemaa, Z. Altman, and A. Galindo-Serrano. 5g ran slicing for verticals: Enablers and challenges. *IEEE Communications Magazine*, 57(1):28–34, 2019.

[14] X. Foukas, M. K. Marina, and K. Kontovasilis. Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, page 127–140, New York, NY, USA, 2017. Association for Computing Machinery.

[15] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis. Flexran: A flexible and programmable platform for software-defined radio access networks. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 427–441, New York, NY, USA, 2016. ACM.

[16] IRIS. IRIS - the software defined radio (SDR) testbed, 2020.

[17] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan. Cellslice: Cellular wireless resource slicing for active ran sharing. In *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–10, 2013.

[18] A. Ksentini and N. Nikaein. Toward enforcing network slicing on ran: Flexibility and resources abstraction. *IEEE Communications Magazine*, 55(6):102–108, 2017.

[19] S. Marek. xRAN, Open vRAN, and OpenRAN: What's the Difference? https://www.sdxcentral.com/articles/news/xran-open-vran-and-openran-whats-the-difference/2018/04/, April 2018.

[20] N. Nikaein, C.-Y. Chang, and K. Alexandris. Mosaic5g: Agile and flexible service platforms for 5g research. *SIGCOMM Comput. Commun. Rev.*, 48(3):29–34, September 2018.

[21] NITlab. NITOS facility, 2020.

[22] O-RAN Alliance. O-RAN software community, 2020.

[23] O-RAN Alliance. O-RAN Use Cases and Deployment Scenarios - Towards Open and Smart RAN. https://www.o-ran.org/resources, February 2020.

[24] O-RAN Alliance. O-RAN Working Group 3: Near-Real-time RAN Intelligent Controller - E2 Application Protocol (E2AP). ORAN-WG3.E2AP-KPM-v01.00, 2020.

[25] O-RAN Alliance. O-RAN Working Group 3: Near-Real-time RAN Intelligent Controller - E2 Service Model (E2SM). ORAN-WG3.E2SM-v01.00, 2020.

[26] O-RAN Alliance. O-RAN Working Group 3: Near-Real-time RAN Intelligent Controller - E2 Service Model (E2SM). ORAN-WG3.E2SM-KPM-v01.00, 2020.

[27] O-RAN Project. Ric message router – rmr. https://docs.o-ran-sc.org/projects/o-ran-sc-ric-plt-lib-rmr/en/latest/, 2019.

[28] O-RAN Software Community. Cherry release (dec 2020). https://wiki.o-ran-sc.org/pages/viewpage.action?pageId=20876303, December 2020.

[29] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejski, M. Sherman, Z. Kostic, X. Gu, H. Krishnaswamy, S. Maheshwari, P. Skrimponis, and C. Gutterman. Challenge: COSMOS: A city-scale programmable testbed for experimentation with advanced wireless. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom)*, September 2020.

[30] P. H. A. Rezende and E. R. M. Madeira. An adaptive network slicing for lte radio access networks. In *2018 Wireless Days (WD)*, pages 68–73, 2018.

[31] Software Radio Systems. srsRAN is a 4G/5G software radio suite developed by SRS. https://github.com/srsran/srsran.

[32] The POWDER Team. Powder (the Platform for Open Wireless Data-driven Experimental Research). https://www.powderwireless.net, 2018.

[33] P. L. Vo, M. N. H. Nguyen, T. A. Le, and N. H. Tran. Slicing the edge: Resource allocation for ran network slicing. *IEEE Wireless Communications Letters*, 7(6):970–973, 2018.