FlexRIC: An SDK for Next-Generation SD-RANs

Robert Schmidt, Mikel Irazabal, Navid Nikaein
EURECOM
Sophia-Antipolis, France
{robert.schmidt,mikel.irazabal,navid.nikaein}@eurecom.fr

ABSTRACT

Unlike previous mobile networks, 5G New Radio (5G-NR) provides unprecedented flexibility in the radio access network (RAN) to support diverse use cases in a multi-tenant environment. In this context, the need for programmability and control through software-defined radio access networking (SD-RAN) is well established. While the underlying RAN is designed to be ultra flexible and lean, existing SD-RAN controllers are either not flexible to address all use cases or use a one-size-fits-all approach.

In this paper, we present FlexRIC, a flexible and efficient software development kit (SDK) that enables to build specialized service-oriented controllers. FlexRIC has a modular architecture with minimal footprint and is designed with extensibility in mind.

We validate the SDK building concrete implementations of two specialized controllers for state-of-the-art 5G use cases: (1) a recursive RAN controller that virtualizes the network to allow multiple tenants to concurrently control and operate their services in a shared infrastructure over the heterogeneous landscape of 5G networks, and (2) an SD-RAN controller providing programmability for multi-radio access technology (RAT) RAN slicing, and flow-based traffic control targeting low-latency communications. The results reveal that FlexRIC reduces the round-trip time by two while incurring 83 % less CPU compared with O-RAN's reference implementation, and uses 10x less CPU and one third of the memory when compared to FlexRAN. Such performance is required to unleash the potential of emerging 5G use cases.

CCS CONCEPTS

• **Networks** → **Mobile networks**; **Programmable networks**.

KEYWORDS

5G, SD-RAN, SDK, RIC, Virtualization, Slicing

ACM Reference Format:
Robert Schmidt, Mikel Irazabal, Navid Nikaein. 2021. FlexRIC: An SDK for Next-Generation SD-RANs. In *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), December 7–10, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3485983.3494870

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CoNEXT '21, December 7–10, 2021, Virtual Event, Germany
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9098-9/21/12…$15.00
https://doi.org/10.1145/3485983.3494870

1 INTRODUCTION

5G shifts the current paradigm beyond new radio and spectrum in three major areas: (i) from monolithic to disaggregated architecture, (ii) from proprietary protocols and closed systems to commodity hardware and standardized interfaces, and (iii) from a communication-oriented to service-oriented networking. To this end, flexibility, forward compatibility, and ultra-lean design are fundamental design principles of 5G that allow to not only satisfy diverse services, deployments and requirements, but also to ensure sustainability and future evolution to support a wide range of use cases, many of which are not yet defined or cannot be foreseen.

In general, what is needed from networking equipment, such as a base station, is simplicity, reliability and high-performance. Towards this end, the whitebox approach suggests a new model, where advanced networking with different or evolving use case-specific requirements is just general-purpose computing on commoditized equipment with possibility to remotely monitor, control and program network entities, known as software-defined networking (SDN). While the need for SDN in 5G and beyond is well-established, there is still a lack of proper software-defined radio access networking (SD-RAN) design and a prototype implementation that adheres to the fundamental design principles of 5G.

FlexRAN [1] was the first attempt to realize a real-time SD-RAN platform for research purposes. It applies the principles of SDN (i.e., decoupling of control plane and user plane) by defining a south-bound control protocol, which connects the radio access network (RAN) to a centralized controller. However, its protocol is tightly coupled with the underlying radio access technology (RAT), and its architectural design was not designed to support 5G's required modularity for protocol and service extensions in a multi-RAT, multi-vendor environment. Further, FlexRAN adds overhead by requiring applications to poll for new messages (e.g., statistics updates). Additionally, FlexRAN does not scale as the number of UEs increases: for instance, we measured a x10 times more CPU usage and x3 times memory utilization in considered scenarios (see Section 5.3).

O-RAN[1] is a recent operator-led attempt to standardize SD-RAN by specifying the control protocol between the RAN and a RAN Intelligent Controller (RIC), termed E2 [2]. E2 exposes RAN internals to external applications (designated as "xApps") to perform different tasks, e.g., radio resource management, through a set of "service models" (SM). SMs expose functionality that can be monitored or controlled, e.g., key performance indicators. (Table 1 summarizes the E2 terminology used in this paper; Appendix A provides a primer about E2.) For the purpose of validating the design, O-RAN provides a reference implementation of the RIC (throughout this document, we refer to its "Cherry" release). Following a micro-service architecture, O-RAN proposes to implement each controller

[1] https://www.o-ran.org/

Table 1: Terminology used in this paper.

Abbrev.	Meaning
E2	O-RAN-defined interface RIC/RAN elements
E2AP	Application Protocol for E2 management and SM encapsulation
E2SM, SM	Service Model/Specification for information exchange
iApp	Internal application for controller specialization
RAN	Radio Access Network
RAN function	Controllable functionality within E2 Node
RIC	RAN Intelligent Controller
SD-RAN	Software-Defined Radio Access Networking
SDK	Software Development Kit
xApp	External application, communicating with the RIC

platform component (e.g. E2 termination, databases, monitoring and logging frameworks) and xApps in separate containers, orchestrated by a Kubernetes cluster. Such decision establishes architectural boundaries to the development of future RICs and xApps. It couples the RIC with a containerized technology, contrary to an open technologically agnostic design. Furthermore, it imposes an isolation between processes that may not suit all use cases. Using C++ terminology, it violates the "zero-overhead principle" [3] as it imposes overhead adding non required features, proposing a "one-size-fits-all" design. In fact, due to the O-RAN RIC implementation, transmitting and receiving an Ethernet Maximum Transmission Unit (MTU, i.e., 1500 B) within an unloaded modern workstation (i.e., local host) lasts approximately 1 ms (see Section 5.4), limiting its usage in various scenarios found in low latency services. Additionally, O-RAN's reference implementation consumes large amount of resources (e.g., 160 GB of storage are recommended[2]), requires knowledge of Docker and Kubernetes, and various programming languages.

A number of challenges arise when designing an SD-RAN that is (1) RAT-agnostic and vendor-independent, (2) ultra-lean, flexible and forward-compatible towards heterogeneous use case requirements, and that allows to (3) specialize the SD-RAN infrastructure towards these use cases. To this end, we propose FlexRIC, a novel event-driven software development kit (SDK) to build a range of SD-RAN controllers, each specialized for (i) a set of use cases, such as traffic control or slice control, (ii) network entities including 5G centralized unit (CU) or distributed unit (DU), (iii) radio access technologies like 4G and 5G, and (iv) software/equipment vendors such OpenAirInterface or SRSRAN. By combining the FlexRIC SDK, consisting of an agent and server library, with application-specific logic and communication interfaces, specialized controllers can be composed to build a multi-service disaggregated SD-RAN infrastructure to meet the requirement of one or multiple use-cases. FlexRIC unleashes the potential of SD-RAN in the envisioned 5G use cases [4], where (i) the radio spectrum, (ii) the traffic flows, and (iii) user associations and handovers can be controlled, coordinated, and optimized through xApps. In terms of interoperability, FlexRIC is O-RAN compatible by means of E2AP control protocol, and has an internal representation to adapt the E2 protocol to diverse encoding/decoding schemes and transport protocol. Finally, FlexRIC is also relevant for industrial and private 5G networking-related use-cases, where a one-size-fits-all SD-RAN approach, like the O-RAN RIC, is too complex and/or its overhead could become a bottleneck or even prohibitive.

In summary, this paper makes the following contributions:

- Present a flexible, forward compatible, ultra-lean SDK design, in the form of FlexRIC, that enables to build a range of SD-RAN controllers and to compose a recursive or multi-service SD-RAN infrastructure tailored to particular use cases.
- Introduce a novel set of extendable and composable SMs, including monitoring, slicing and traffic control, to enable a higher level of abstraction needed for emerging use cases (i.e., flow-level traffic control and radio resource slicing).
- Provide concrete implementations of specialized FlexRIC controllers: (a) service-specific SD-RAN controllers, and (b) a recursive controller, centered around 5G use cases, together with a detailed experimental evaluation of the various aspects of the SDK that highlight its scalability, adaptability, and performance compared to state-of-the-art solutions.

2 RELATED WORK

SD-RAN. A number of SD-RAN controllers brought the separation of control and user plane with the associated programmability. FlexRAN [1] pioneered the first practical open-source implementation of an SD-RAN controller on top of an LTE cellular network. FlexRAN provides a separation of the RAN control and data plane through a custom protocol with a focus on real-time RAN control applications. Unfortunately, the south-bound protocol is tailored towards specific control operations, limiting its extensibility, and it lacks modularity, which limited the contributions that it received during the last years. Furthermore, it incurs unnecessary overhead by requiring applications to poll for updates such as RAN statistics instead of notifying them via a publish/subscribe pattern. Another implementation of an SD-RAN controller, the EmPOWER platform [5], focuses on management of RAN via policies without support of (real-time) RAN control, limiting its deployment use cases. Moreover, similarly to FlexRAN, it uses a non-standardized custom south-bound interface.

More recently, the industry initiative O-RAN was formed to manage the increasingly complex cellular network [6]. As part of this effort, an interface between base stations and controllers (E2) [2] was standardized. Furthermore, O-RAN developed a reference SD-RAN platform, the O-RAN RIC, to validate their architecture. O-RAN's RIC is based on a micro-service architecture (deployed over Docker and orchestrated through Kubernetes), and its design requires the use of a non-negligible amount of resources, since a default deployment requires platform functions in 15 containers (using storage and RAM), and, for instance, E2AP messages need to be decoded both at the "E2 termination" and xApp (see Section 5.4). Furthermore, O-RAN also decided to couple their RIC design to specific implementations (e.g., Redis or Prometheus), limiting its forward compatibility, and applications need to poll these databases to discover new agents, bearing overhead. Such architectural design and implementation decision clearly violates the ultra-lean

[2]Source: Getting started guide of Cherry release, https://wiki.o-ran-sc.org/display/GS/Near+Realtime+RIC+Installation

design of 5G to minimize the platform footprints. In essence, there exists a considerable amount of use-cases found in low-latency scenarios, for which O-RAN design will not be able to provide the required constraints, making it inflexible. Similarly, the OpenNetworkingFoundation recently announced that they also developed and integrated an O-RAN compliant E2 agent and RIC [7]. Unfortunately, the implementation is closed, and thus, we could not investigate further.

Network Virtualization. FlowVisor [8] is a proxy that enables network virtualization. It uses OpenFlow [9] as the abstraction layer to virtualize the network resources and slice the network to allow multiple SDN controllers to concurrently control their (virtual) network. It has been used for slicing the RAN with OpenRoads [10, 11], an SDN controller for Wi-Fi/WiMAX networks, but does not provide the necessary primitives to virtualize RANs, since there is no support for managing the scarce radio resources. The theoretic work of RadioVisor [12] proposes an abstraction to virtualize the network. It defines slices in the form of resource sub-grids to isolate operators from each other, which can then sub-slice their network according to subscriber attributes. Finally, full slice virtualization in the RAN has been explored through Orion [13]. Orion uses a hypervisor on a base station to isolate and abstract services and multiplexes the radio resources of each service onto the common resources. However, Orion's virtualization is limited to radio resource scheduling, and does not provide support for connection, mobility management, or flow control. Additionally, since it relies on a single hypervisor at the base station, its design limits its deployment in disaggregated base stations and the feasibility of interference management due to the absence of coordination between hypervisors.

In summary and to the best of the authors' knowledge, there is no platform that unifies previous approaches to SD-RAN and virtualization. In this paper, we propose the event-driven FlexRIC SDK to enable SD-RAN controllers that can be flexibly specialized towards diverse requirements, following the 5G principles of flexibility and forward-compatibility. FlexRIC is designed to be ultra-lean, consuming the resources demanded by the service, without adding extra features when they are not required. Additionally, by adopting the E2 protocol structure, it remains fully compliant to industry efforts, and is vendor-independent. Its modular SDK structure allows a smooth composition of service-specific controllers, including radio resource scheduling for multiple services.

3 FLEXRIC SDK OVERVIEW

FlexRIC is a SDK, consisting of a server library and an agent library with two optional extensions: controller-internal applications (iApps) and communication interfaces. The objective of the SDK is to facilitate the realization of specialized SD-RAN controllers to target specific use cases, while being simple to use. FlexRIC does not support extra features endogenously following the *zero-overhead principle*. In its simplest form, the SDK can be used to implement an SD-RAN controller using an E2-compatible protocol, as it is shown in Fig. 1.

The agent library is the basis to extend a base station with the agent functionalities. It provides an API to implement custom RAN functions, i.e., RAN functionality that can be monitored and/or

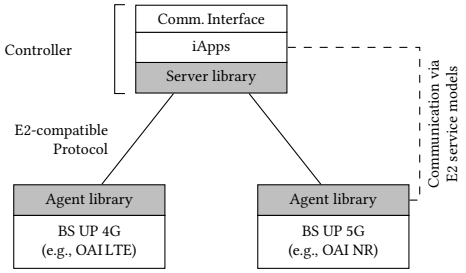


Figure 1: The FlexRIC SDK consists of an agent and a server library. The agent library provides integration of a base station to a controller, which is specialized to a particular use case using iApps.

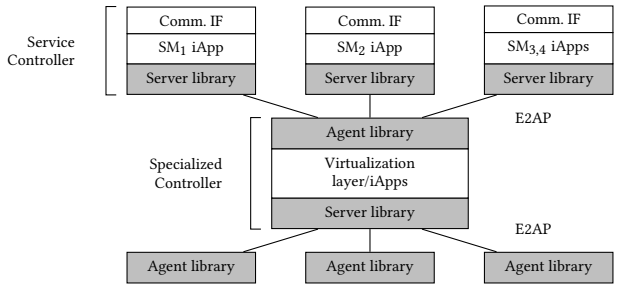


Figure 2: Using the FlexRIC SDK, different specialized controllers can be built, like a "recursive" controller.

controlled by applications, and comes with a bundle of pre-defined RAN functions that implement a set of SMs that can be included.

A controller is built through the server library, iApps, and optionally a communication interface. The server library manages agent connections, and multiplexes messages between iApps and the agents. Through the iApps, it is possible to modularly build specialized controllers: based on the considered use-case, iApps can implement SMs themselves, or expose information to external applications (xApps) via a northbound communication interface. In the latter case, xApps can control the RAN while being functionally isolated from the controller, which is the favored method of the reference SD-RAN controller implemented by O-RAN, but bears a certain overhead. Finally, it is even possible to recursively expose an agent interface at the northbound by reusing the agent library, as shown in Fig. 2. Such a controller employs a virtualization layer to delegate control to multiple (per-slice) controllers, each operating on a different set of SMs. Such recursive property of FlexRIC allows not only to compose various specialized controllers into a multi-service SD-RAN infrastructure but also abstract out the heterogeneity of the 4G/5G deployment topology. We will explore some possible controllers in Section 6.

The FlexRIC SDK provides an abstraction of the E2 interface via an internal representation of E2 messages, and users of the SDK do not have to be concerned with the actual encoding and transport of the messages. It is therefore straight-forward to integrate FlexRIC with pre-existing E2-compliant SD-RAN infrastructure. However, the standard mandates an encapsulation of ASN.1-encoded data

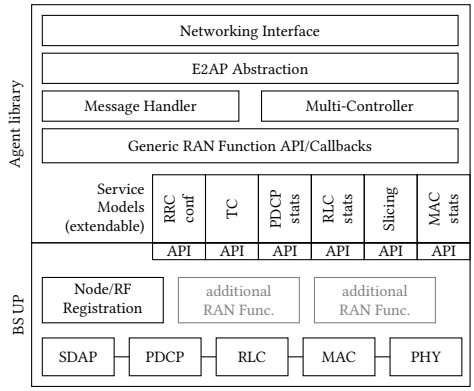


Figure 3: The FlexRIC agent architecture and its integration with a user plane implementation.

inside of ASN.1 and its transport over the SCTP protocol, which may be inefficient in certain cases, for instance when the message size becomes large (case for monitoring information). Therefore, the SDK also supports to change both the encoding scheme and transport protocol allowing the integration of vendor-specific possibly more efficient E2 protocol for low-overhead, real-time control between a controller and an agent.

4 FLEXRIC SDK ARCHITECTURE

In the following, we elaborate on the FlexRIC architecture, with a particular focus on the following design challenges:

(1) A RAT-agnostic and vendor-independent SD-RAN design and its integration of any base station implementation (Section 4.1),
(2) An SD-RAN design according to 5G principles, i.e., (1) ultra-lean for low-latency or resource-restricted use cases, avoiding unnecessary overhead, and with (2) flexibility and forward-compatibility towards novel use cases (Sections 4.1-4.3),
(3) A low-coupled design that allows the specialization of SD-RAN controllers (Section 4.2) towards particular use case requirements, such as traffic control, slicing, and recursive slicing through network virtualization (as presented in Section 6).

4.1 FlexRIC Agent

The main design principle of the agent library relies on its easy integration into various base station implementations and deployment scenarios interfacing with an E2 controller, while at the same time providing additional functionality for handling multiple controllers in a multi-service environment, which are detailed in the following.

4.1.1 Agent Architecture. The architecture of an agent using the FlexRIC SDK is shown in Fig. 3. It consists of the agent library, and a base station.

The agent library provides the necessary support to connect to a controller, such as O-RAN RIC. It consists of a networking interface, the E2AP abstraction, a message handler, and a generic RAN function API. The E2AP abstraction provides an intermediate representation for the E2 protocol, relieving RAN functions from E2 protocol specificities (see Section 4.3). Further, the agent provides the generic RAN function API to implement RAN functions with custom SM-specific logic. This API defines callbacks for E2AP messages, i.e., (i) subscription requests (for new information subscription), (ii) subscription delete request (removal of subscriptions), and (iii) control messages (to trigger SM-specific actions), which need to be implemented by RAN functions. Finally, we pre-defined SMs for the considered use cases, namely slicing control and traffic control (c.f. Section 6). The SMs are tailored towards specific RAN sublayers (for statistics) or RAN control endpoints (for control) to easily integrate the agent library in disaggregated base stations, and expose a simplified API, facilitating the integration with various base station implementations.

The base station provides basic node information, such as the public land mobile network (PLMN), and registers RAN functions according to the underlying node's capability: unlike what is shown in Fig. 3, not all RAN layers are present in every node for disaggregated base stations, and FlexRIC natively supports such disaggregation through the selection of appropriate RAN functions. The base station uses the interface of pre-defined RAN functions to expose data and handle control messages, or it might define additional RAN functions using the generic RAN function API.

The agent library is not tied to any existing cellular user plane implementation or RAT and is therefore inherently vendor-neutral and RAT-neutral, making it a reusable component for multi-vendor and multi-RAT scenarios.

4.1.2 Multi-Controller Support at the Agent Library. The agent library provides means to connect to multiple controllers. This becomes useful in a multi-service context, e.g., for "recursive" controllers that virtualize RAN control (Section 6.2) or base station hypervisors like Orion [13], which permits shared control and exposes information to multiple controllers while also providing isolation between them.

The FlexRIC agent library assists the handling of multiple service controllers through (1) the management of additional controllers (setup, teardown, providing controller origin to RAN functions for message handling), and (2) a UE-to-controller association.

The UE-to-controller association is used to indicate the UEs that are to be exposed to each controller. An active E2 subscription addresses all (or an indicated subset) of UEs. However, a given RAN function does not know a priori which UEs are to be exposed to a particular controller. Therefore, when handling messages, the agent is able to look up and reveal the UEs that belong to the corresponding controllers.

The agent library associates every UE to the first controller, and provides no means for an automatic association of UEs to additional controllers, e.g., through 5G RRC slice identifiers (S-NSSAI). Rather, this has to be triggered through a controller based on information through SMs, as the agent might not always be capable to determine an association. Consider the example of a split base station with CU and DU, and separate controllers that are concerned with the DU exposing a remote scheduling SM, as shown in Fig. 4. When a new UE arrives and should be connected to the separate controller, its association depends on the selected PLMN of the UE, which is decoded in the CU. The agent of the DU therefore cannot infer such an association and needs to be assisted from the CU controller

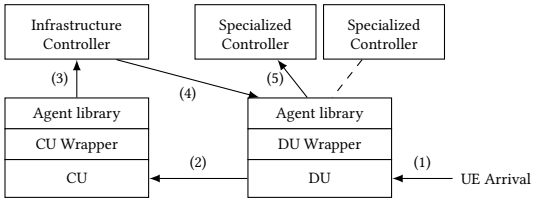


Figure 4: For disaggregated base stations, a controller needs to configure the UE-to-controller association in all agents (step 4) to ensure that a connecting UE (step 1) is exposed to the specialized controller (step 5).

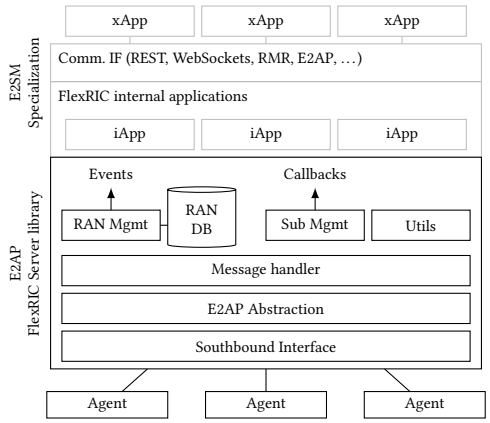


Figure 5: The FlexRIC server library provides communication means and an E2AP abstraction. Specialized controllers are implemented through internal applications (iApps). They provide services to external applications (xApps) through custom, controller-specific northbound interfaces.

about the arrival of this new UE, configuring the agent to expose the corresponding UE at the specialized controller.

Note that service-level agreements (SLA) are *not* part of multi-controller management. In fact, SLAs are tied to specific SMs, as the underlying resources vary (e.g., resource blocks in MAC or queues in RLC/SDAP/PDCP). Therefore, it is the SM, implemented by the RAN function, to perform sufficient admission control upon subscriptions of the controllers, and ensure that the requested operations are conflict-free.

4.2 FlexRIC Controller

FlexRIC permits to smoothly create specialized, service-specific, E2-compatible controllers. As shown in Fig. 5, it consists of (1) the FlexRIC server library, providing means to communicate with agents using an E2AP abstraction, and (2) a controller specialization.

4.2.1 Controller Specialization. A controller specialization implements SD-RAN-related functionality, and is realized through internal applications (iApps), a communication interface, and external applications (xApps). The iApps implement specific controller behavior, either directly through SMs within the iApps themselves, or by providing platform services that can be leveraged by xApps. For this purpose, a controller specialization typically exposes a northbound communication interface using a custom protocol, such as a simple REST interface (e.g., FlexRAN [1]), the RMR library (e.g., O-RAN RIC), a message broker (e.g. Redis), or E2AP itself to interact with other specialized controllers. In this regard, we describe service-specific (slicing, traffic control) and network virtualization controller designs in Section 6.

4.2.2 FlexRIC Server Library. The FlexRIC server library's objective is to multiplex agent connections and dispatch E2AP messages. As the agent library, the server library abstracts the E2AP communication. The server library is designed as an event-driven/callback-driven system, following the ultra-lean design principle to impose minimal overhead. Thus, it invokes iApps only when there are new messages, unlike systems like FlexRAN that use polling.

The RAN management functionality handles connection-related events such as an agent connection. An application that subscribed for new agent connections uses the included information to send a subscription if it encounters suitable RAN functions. Further, the RAN management functionality stores information in the RAN database (RAN DB), allowing to query information about the composition of the RAN network. For this purpose, the RAN management also handles disaggregated deployments by merging agents that belong to the same base station (e.g., CU agent and DU agent) into the same RAN entity, facilitating base station control across agents, and provides events to signal when a complete RAN is formed from disaggregated entities.

The subscription management's task is to (i) keep track of existing subscriptions and (ii) deliver arriving subscription-related messages to the corresponding iApps. When an iApp requests a new subscription directly or on-behalf of an xApp, it provides a set of callbacks that are called to inform the iApp (and subsequently the xApp) about the subscription outcome, and to dispatch the corresponding indication messages. When a message arrives, the subscription management simply selects the iApp for which the message is sent and forwards it through the provided callback.

It has to be pointed out that the server library itself does *not* implement any SM, and does not request any information from the agent by itself. Instead, iApps have to trigger SM-related communication (generally on behalf of xApps), and the server library provides the platform to multiplex messages between agents and iApps.

4.3 E2 Protocol Abstraction

We identified four orthogonal abstractions in O-RAN's E2 specification.

(1) The transport protocol, defined by O-RAN to be SCTP.
(2) The encoding/decoding algorithms at the procedures defined by O-RAN to be ASN.1 PER.
(3) The encoding/decoding algorithms at the SMs defined by O-RAN to be ASN.1 PER.
(4) The semantic of E2AP.

For handling the transport protocol, a wrapper is created to abstract the communication interface allowing to easily switch between different transport protocols. For the encoding/decoding

of E2AP, we modeled an intermediate representation, representing E2AP procedures without loss of information and independent of any particular encoding/decoding algorithms, and freeing iApps/xApps from these encoding tasks. To this end, we implemented the most common 20 out of 26 E2AP messages using the O-RAN standard ASN.1 and 12 out of 26 messages using Google Flatbuffers (FB) [14]. This increases the flexibility of the SDK: if the bandwidth is scarce, ASN.1 presents better compression rates, while in scenarios where the CPU represents the bottleneck, FB is preferred (see Section 5.2). Thanks to the intermediate representation, FlexRIC is open for adding new encoding/decoding algorithms, without having to modify its source code, ensuring forward-compatibility. Similarly, for the SMs, we allow custom encoding/decoding algorithms aiming to support future changes.

Finally, we did not create an abstraction around the semantic of E2AP, as this would imply to completely redesign the protocol (e.g., making it stateless instead of the current stateful implementation that is achieved, for example through the setup request procedure) [15].

4.4 Implementation of the SDK

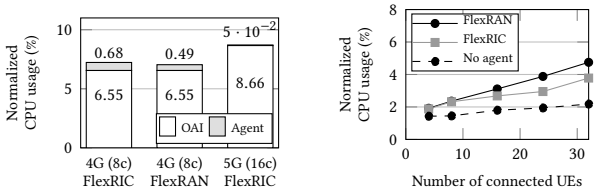
The implementation of the SDK provides minimum, yet sufficient mechanisms to implement a service-specific controller. The agent and server libraries (SDK), including the E2 abstraction, are roughly 10K lines of C11 code (we use generics to achieve compile time polymorphism), not using any external dependencies except for Flatbuffers and/or ASN and dedicating more than 4K lines to the E2AP message encoding/decoding algorithms. We chose C as it is the *de facto lingua franca* in computer science, and therefore, an interface for another language can be easily implemented (e.g., through SWIG).

A controller might need to handle indication messages from many agents. While the current implementation is single-threaded, a multi-thread extension is conceivable. The interface between the FlexRIC server library and iApps uses an event-based/callback-based system to pass E2 messages. For scenarios where message processing time is an issue, and given that the handling of indication messages in the server library is stateless, it is possible to pass messages to different threads, facilitated by the event-based system, and handle messages in multiple threads for better scalability. We remark that POSIX sockets are thread-safe, and sending messages from multiple threads is also feasible.

Since the RIC seems to become an essential part of the 5G ecosystem, we intend to release FlexRIC as an open-source SD-RAN software platform for research and development under appropriate license to boost its adoption by both academia and industry. Moreover, since the RIC without SMs is of limited use, we also plan to release the provided monitoring (i.e., MAC, RLC, PDCP, and RRC) and control SMs (i.e., slice control) going beyond O-RAN definitions, as well as the OAI 4G and 5G prototype integration and the controller specializations of Section 6, as open-source.

5 PERFORMANCE EVALUATION

In this section, we rigorously evaluate the performance of the FlexRIC SDK, starting from the agent library via the E2 abstraction to the server library, and finally compare to the O-RAN RIC. Depending on the experiment, we use either a test agent, or a FlexRIC agent on top of OpenAirInterface [16] (OAI) (tag 2021.w20). We deactivate the CPU sleep states on all machines to ensure consistent measurements.



(a) Radio deployment, LTE/NR. **(b) L2 simulator for LTE.**

Figure 6: Normalized CPU usage of FlexRIC and FlexRAN. Note that the LTE cell has 8 cores, the NR cell 16.

5.1 Overhead in the User Plane

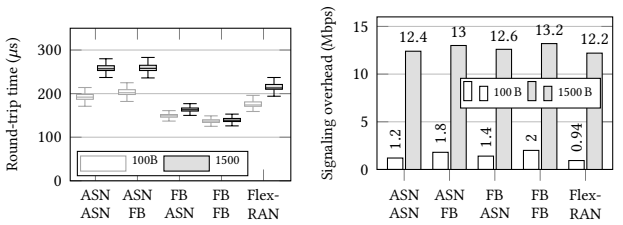
We measure the CPU usage introduced by the FlexRIC agent library and compare it to FlexRAN [1] (LTE only). To this end, we export statistics measurement using the built-in statistics of FlexRAN and the integrated statistics SMs of FlexRIC at 1 ms frequency; in both cases, we enable all statistics for MAC, RLC, and PDCP (excluding HARQ), covering approximately the same data (PDCP/RLC packet and byte counters, MAC statistics such as CQI and used resource blocks, etc.). Note that both use different encoding schemes (i.e., Protobuf [17] for FlexRAN, FB for FlexRIC); our purpose is to verify that FlexRIC incurs comparable overhead as FlexRAN, which was designed for real-time control.

We measure the CPU usage over both LTE and NR (non-standalone mode, NSA) base stations. We used a RF setup with Ettus B210 radios for both cells. The LTE cell runs on an Intel Core i7 with 8 cores @ 3.2 GHz, uses a bandwidth of 5 MHz (25 RBs) and serves 3 UEs at MCS 28. The NR cell runs on an Intel Xeon Gold 6208U with 16 cores @ 2.9 GHz, has a bandwidth of 20 MHz (106 RBs) and serves 3 UEs at MCS 20. Fig. 6a shows that both FlexRIC and FlexRAN incur a small overhead. The relative overhead decreases when deploying FlexRIC over NR, due to a more demanding physical layer.

To analyze the overhead for more UEs, we used the "L2 simulator" of OAI, an emulation mode without the physical layer. Fig. 6b shows that FlexRIC performs slightly better than FlexRAN especially for more UEs, and thus, FlexRIC is valid for the same scenarios where FlexRAN has been validated (e.g., real-time MAC scheduling). The results confirm the scalability of the FlexRIC agent library, achieving slightly better performance for many UEs (up to 1 % less CPU load for 32 UEs) due to more efficient encoding of indication messages through Flatbuffers.

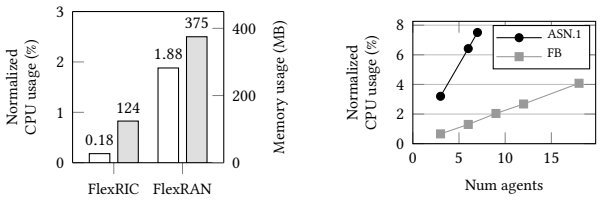
5.2 Impact of E2AP/E2SM Encoding

In the following, we evaluate the impact of two encoding schemes, FB and ASN.1, for E2AP and the E2 service model (E2SM), on round-trip time (RTT) and signaling rate. E2 enforces a double encoding of messages: a first encoding pass is done for the "inner" E2SM, and a second for the "outer" E2AP. We modified the "Hello World" SM



(a) Round-trip time. (b) Signaling overhead.

Figure 7: Comparison of E2AP/E2SM encoding schemes using E2SM-HW ping.



(a) Comparison to FlexRAN. (b) Overhead for many UEs.

Figure 8: CPU usage at the controller.

(HW-E2SM) provided by O-RAN to perform a ping by sending a control message to the RAN function, to which the agent responds with an indication message. To also study the impact of the E2SM encoding, we translated the SM 1:1 from ASN.1 to FB. Further, we modified FlexRAN to measure the RTT and signaling rate. FlexRAN does not oblige a double encoding as described above.

To measure the RTT, the iApp pings the agent every second for small (100 B) and medium (1500 B) message payloads. The results in Fig. 7a indicate that switching from all-ASN.1 to all-FB encoding reduces the average RTT by roughly 25 % and 66 % for small and medium payloads, due to the encoding overhead of ASN.1. Using an ASN.1/FB (E2AP/E2SM) encoding even increases the round-trip time, since the larger FB E2SM message (for each FB message, we observe 30-40B overhead) needs to be encoded again by ASN.1 for E2AP. Thus, where ultra-low latency is required, FB might be preferable, especially for larger payloads, but networking delay (our Ethernet-based campus network has RTTs below 1 ms) would make this difference much less pronounced. The results of FlexRAN indicate the arrival of ping replies *at its networking queues*. We observe that its absolute RTT and relative RTT increase is between that of FB and ASN.1 cases, most likely due to the Protobuf encoding, but always slower than the FB E2AP encoding of FlexRIC. Furthermore, due to FlexRAN's design, an application has to poll for the results every ms; thus, in FlexRAN, the RTT from an application's point of view *is always one ms*, which we omit for readability.

To put the RTT improvements into perspective, we measured the generated message signaling rate for a high ping rate (one packet every 1 ms, which is 4G's transmission time interval). The results in Fig. 7b indicate that switching from an ASN.1/ASN.1 to an FB/FB encoding increases the signaling rate by 67 % for small payloads due to the overhead of FB; for large payloads, however, the signaling rate overhead is almost negligible. Comparing the "mixed" encodings, we see that the FB/ASN.1 encoding signaling overhead only slightly increases compared to ASN.1/ASN.1, whereas the ASN.1/FB combination does not decrease signaling load, rendering this combination useless. In comparison, FlexRAN has the smallest signaling rate, since it does not enforce a double encoding. This advantage almost vanishes for large payloads.

In conclusion, FB encoding is useful for larger payloads and/or frequent message exchange under the assumption that the wired capacity is not the bottleneck, at least for E2AP. On the other hand, if the wired capacity is the bottleneck or if infrequent, small messages are forwarded, ASN.1's processing overhead in terms of CPU is balanced by a lower signaling overhead. In general, it might be preferable to use FB in E2AP, as it only slightly increases signaling overhead while keeping round-trip times low.

5.3 Scalability of the Controller

Next, we evaluate the scalability of the FlexRIC server library in terms of CPU utilization and memory usage and compare it to FlexRAN [1]. The FlexRIC controller consists of the server library and a statistics iApp that saves incoming messages to an in-memory data structure, similar to FlexRAN. We only consider the agent-to-controller direction, which typically carries more traffic than in the opposite direction, even for high-traffic scenarios such as remote scheduling [1]. Both controllers are on an Intel Core-i7 machine with 12 cores at 3.2 GHz. As can be seen in Fig. 8a, FlexRIC incurs only one tenth of the CPU usage of FlexRAN, due to using FB instead of Protobuf. Also, FlexRIC organizes its internal data structure more efficiently, leading to reduced memory consumption.

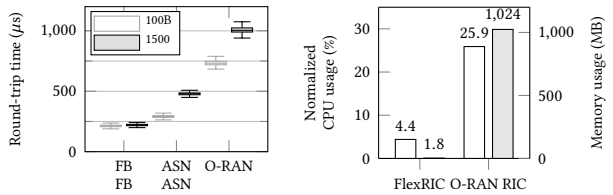
To further understand the limitations of FlexRIC, we test using dummy test agents (not connected to any base station) that export the same statistics (in FB) as from a real base station (statistics for MAC excluding HARQ, and RLC and PDCP), each agent emulating a connection of 32 UEs with a unique default bearer. Fig. 8b shows the CPU utilization of FlexRIC over varying number of agents when using a FB or ASN.1 encoding for E2AP. It is apparent that FB has around 4 times lower CPU usage than ASN.1. Since FB's design avoids an explicit decoding step, reading directly from "raw" bytes, the subscription management in the server library can look up the corresponding subscription much faster, resulting in less CPU usage, directly translating to serving many more agents. This suggests that ASN.1 encoding on E2AP can become a limiting factor in terms of CPU, whereas FB is rather limited by the network, as for 18 agents, the signaling reaches almost 700 Mbps. When reducing the message frequency to 10 ms, the FlexRIC SDK was furthermore able to handle around 100 agents (not depicted in the figure), confirming FlexRIC's scalability.

5.4 Comparison to O-RAN RIC

The reference O-RAN RIC architecture relies on micro-services, containerized through Docker and orchestrated by Kubernetes. Correspondingly, each micro-service's image uses disk space. Table 2 lists the image sizes of a dockerized FlexRIC for a RTT test (as in Section 5.2) and a statistics use case (as in Section 5.3), and the O-RAN RIC platform (15 platform components) with corresponding

Table 2: Docker image sizes.

Component	Size (MB)
FlexRIC + HW-E2SM	76
FlexRIC + Stats E2SMs (FB)	94
O-RAN RIC (platform)	2469
HW xApp	170
Stats xApp	166



(a) Round-trip times for two hops. **(b) CPU and memory usage.**

Figure 9: Comparison of O-RAN RIC and (dockerized) FlexRIC.

O-RAN xApps. It is apparent that due to containerization of all platform components individually, the O-RAN RIC requires much more storage, going contrary the 5G design principle of ultra-leanness, forcing to need resources even for scenarios where the advantages offered by Docker and Kubernetes are not needed.

The O-RAN architecture impacts the latency, since it imposes two hops for messages (from xApp via "E2 termination" to agent). In the following experiment, we measured the RTT with two distinct payloads (i.e., 100 B and 1500 B) as in Section 5.2, between (i) a FlexRIC agent and a FlexRIC controller utilizing the E2SM-HW, and (ii) a FlexRIC agent and O-RAN RIC. In FlexRIC, we use a relaying controller to emulate two hops, which, unlike O-RAN RIC, is not imposed by FlexRIC but added to carry out a fair comparison. As it can be observed from Fig. 9a, O-RAN RIC increases the RTT by at least three times for small and two times for medium load compared to FlexRIC.

In a second experiment, we consider a monitoring use case similar to Section 5.3, in which 10 dummy agents export MAC statistics (excluding HARQ) for 32 UEs using E2AP indication messages every ms. We measure CPU usage and memory footprint as reported by *docker stats*, shown in Fig. 9b. (For the O-RAN case, we added the CPU and memory usage of the platform components and xApp.) The design of O-RAN RIC imposes that indication messages are decoded twice, once in the "E2 termination", and the xApp. The CPU usage of FlexRIC is 83 % lower than O-RAN RIC, the O-RAN xApp alone using as much CPU as FlexRIC, and the rest being used by the "E2 termination". We attribute this to an inefficient implementation (we used the default "E2 termination" image, version 5.4.8). Also, the O-RAN RIC uses much more RAM, which is due to the high number of platform functions running in separate containers, which are also partially written in higher-level languages, such as Go, as opposed to FlexRIC's C. These findings confirm that O-RAN RIC imposes additional overhead by design for communication between xApps and agents, even if not required, which might become a bottleneck.

6 CONTROLLER SPECIALIZATIONS

FlexRIC's modular design permits to easily compose specialized controllers on top of the server library through the SMs, iApps and xApps. Moreover, its design opens up new opportunities for the development of state-of-the-art research controllers (e.g., recursive or multi-RAT controllers).

Some tradeoffs that need to be evaluated when designing a new controller can be summarized in the following:

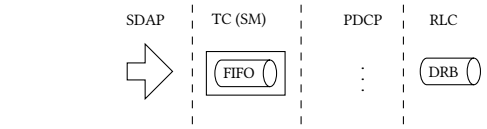
(1) Transport. FlexRIC's internal representation of the E2AP messages, permits different encodings for the messages. Additionally, new encoding schemes could be easily aggregated in the future if needed, following the forward compatibility principle of 5G.
(2) SMs. The controller exposes or modifies the RAN through the SMs. Therefore, suitable SMs that forward/receive data to/from the controller need to be present at the RAN. However, adding SMs that are not needed by the controller just consumes resources, which violates the ultra-lean design principle.
(3) iApps/communication interface/xApps. SMs are either handled by iApps directly or over the communication interface through xApps. Use-case requirements, e.g., low-latency or security, might require or preclude xApps. In the case of xApps, an iApp submits the RAN information to the xApps and may receive commands from them. Thus, flexibly selecting a correct mechanism to communicate with the xApp may be crucial in some scenarios.
(4) Isolation. FlexRIC, iApps and xApps are software programs. Therefore, if security is a concern, techniques applied to software programs can be used (e.g., virtual machines, containers) with their associated tradeoffs.

6.1 Service-Oriented SD-RAN Controllers

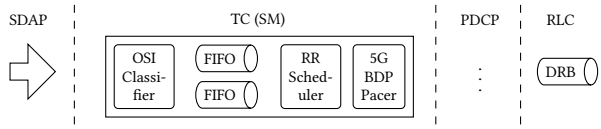
On the one hand, the heterogeneous scenarios for which 5G is designed, hinders to identify the features that a controller should implement, while on the other hand, 5G claims to provide forward compatibility. Using the FlexRIC SDK, we customize controllers tailored specifically to application needs, achieving the flexibility which 5G necessitates.

Following, we present different service-oriented SD-RAN controllers that are prototyped on top of FlexRIC to validate its design and that represent state-of-the-art 5G scenarios.

6.1.1 Flow-based Traffic Controller. Since latency requirements have increased in 5G, controlling the traffic is of utmost importance in a packet-switched network, where TSN support through Packet Delay Budget is standardized [18]. 5G inherits the benefits, as well as the drawbacks, of previous packet-switched cellular networks generations. One of its most important drawbacks is the phenomenon known as bufferbloat [19]. The bufferbloat specifically occurs at the cellular network since i) the bottleneck is commonly located at the radio link; ii) the RLC sublayer is provided with large buffers to absorb the brusque changes that the radio channel may suffer



(a) TC SM in transparent mode.



(b) TC SM with an OSI classifier, two FIFO queues, a round-robin scheduler and a 5G-BDP pacer.

Figure 10: Sublayers with buffers in 5G downlink path.

and avoid starving it; and iii) most of contemporary traffic is transported through a lossy based congestion control algorithm (i.e., TCP Cubic), where the algorithm cannot differentiate between the propagation time and the large sojourn time that packets experience in a bloated buffer.

To avoid the bufferbloat, standardization efforts propose to segregate into different DRBs the traffic of flows prone to generate the bufferbloat and flows that do not cause it [20], similar of how slicing is designed for 5G where a new PDU Session is generated per slice, and thus, the traffic flows are segregated in different DRBs. In a similar spirit, we designed a traffic control SM (TC SM) to abstract the configuration of multiple flows within the RAN, similarly to how OpenFlow [9] abstracts flows in a switch. As shown in Fig. 10, the TC SM uses a classifier to segregate packets into flows, schedules the queues, and limits the rates through a pacer. In this manner, we enhanced current 3GPP specifications, similarly to the architecture explored at [21]. An xApp subscribes to the RLC and TC statistics through their monitoring SMs. We used Redis as a message broker used by an iApp to forward messages to the xApp. Similarly, we used a REST interface for submitting control messages. The components of this specialization are summarized in Table 3.

To illustrate a simple, yet complete and realistic example [22], and show how a traffic control xApp can improve the latency, we generated two flows in downlink. One emulating a one minute G.711 VoIP conversation through UDP data frames of 172 bytes with an interval of 20 ms using *irtt* [23], resulting in a bandwidth consumption of 64 Kbps, and a second flow emulating a bufferbloat-prone flow using *iperf3*. We start the *irtt* flow 5 seconds before the *iperf3* flow. However, instead of segregating the flows in DRB queues as suggested in [22], we created a second queue at the TC SM, as shown in Fig 10b. We used the 5G Quectel 500Q-GL COTS as the UE and OAI (tag 2021.w20) with an agent and validated the proposed traffic control SM.

The example can be summarized as follows: Once the xApp notices that the sojourn time of the packets belonging to the low-latency flow increase beyond a limit, it decides to perform three actions. As its first action, it generates a second FIFO queue. Next, it creates a 5-tuple filter (i.e., source and destination addresses and ports, as well as, protocol) to segregate the low-latency flow packets from the rest. Following, it loads a 5G-BDP pacer [19], to backlog the packets into the queues located at the TC, and thus, avoid bloating the RLC DRB buffer. Lastly, the scheduler is a simple Round Robin, that pulls packets from active queues.

Table 3: TC controller specialization.

Comp.	Used
xApp	Custom program with *libhiredis* and *libcurl* [25]
Comm. IF	Redis message broker, REST (POST)
iApps	RLC, TC stats forwarder (Redis)
	TC SM manager (REST command relay)
Support	Server library

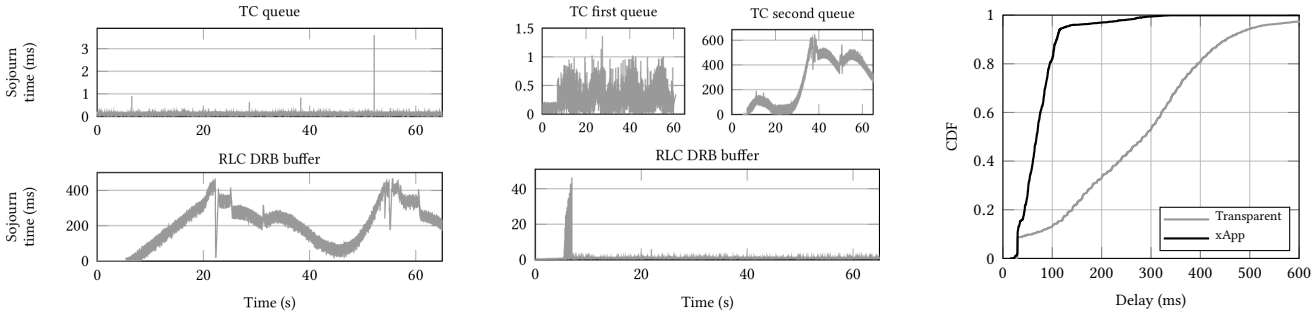
On one hand, Fig. 11a shows the bufferbloat effect in vanilla cellular network systems. TCP Cubic bloats the last buffer before the bottleneck link, which in this case is the RLC DRB buffer. Therefore, if flows with low latency requirement share the data path with a greedy flow, they will suffer large sojourn times. On the other hand, Fig. 11b shows how segregating the traffic and using a pacer, limits the bufferbloat problem to flows that share a queue with the greedy flow. The 5G-BDP pacer maintains the DRB buffer uncongested and backlogs the packets into the TC SM. It tries to submit just enough packets to the DRB not to starve it, without bloating it. In this manner, almost full RB utilization can be achieved, while maintaining uncongested the DRB buffer even when the radio channel link varies under realistic traffic conditions [21]. Moreover, generating a second queue and segregating the traffic at the TC avoids experiencing large sojourn times to the VoIP flow's packets. Lastly, Fig. 11c shows the clear advantage of intelligently using TC through an xApp. The RTT of the VoIP flow when is previously segregated is in the order of four times faster. However, Fig. 11c also shows that even though the largest part of the delay is associated with the downlink path, there exists other buffers that may need to be investigated further, as the RTT when no *iperf3* traffic is added varies between 20 and 40 ms.

We implemented the queues, the classifier, the scheduler and the pacer as shared objects to enable loading them online. In this way, the behavior of the traffic flows within our SM can be dynamically changed according to traffic and network conditions. The objects could even be downloaded them from a trusted network store [24].

Some examples of 5G scenarios that will need traffic control capabilities to fulfill their latency requirements are URLLC (e.g., V2X) or eMBB (cloud AR) services.

6.1.2 RAT-Unaware Slicing Controller. Network slicing is considered to be the enabler to multiplex heterogeneous services onto a common network, where services are assigned to slices in the network. On the one hand, the scarce radio resources of the RAN have to be shared between multiple services, and isolation is necessary if quality-of-service (QoS) is of concern. On the other hand, unused resources should be shared among services to use the available spectrum efficiently.

To facilitate the configuration and management of slices in the RAN, we designed a slicing control SM (SC SM) that abstracts the slice configuration. The SM assumes a distinction between a slice scheduler and a UE scheduler, as shown in Fig. 12. Upon the MAC scheduling phase, first the slice scheduler distributes resources



(a) Sojourn time in transparent mode. **(b) Sojourn time in xApp case.** **(c) CDF of the RTT of VoIP flows.**

Figure 11: Sojourn times for TC transparent mode and with two queues and CDF of VoIP flows for both cases.

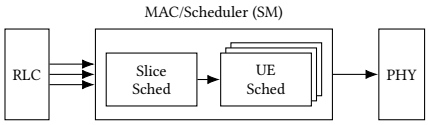
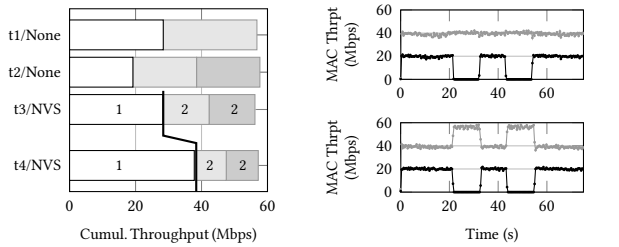


Figure 12: The slice scheduler attributes resources to user schedulers, which schedule UEs in their slice.

Table 4: Slicing controller specialization.

Comp.	Used
xApp	Command line: *curl* [25]
Comm. IF	REST (GET/POST)
iApps	Internal DB for RAN stats (cf. FlexRAN RIB [1])
	SC SM manager (REST command relay)
Support	Server library



(a) Isolation (UEs are marked corresponding slice). **(b) Static resource attribution vs. resource sharing.**

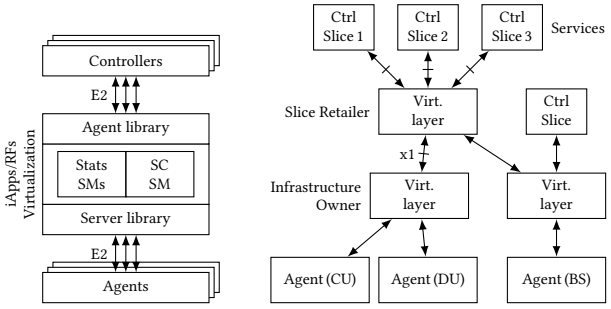
Figure 13: Resource slicing isolates services, but resources might be shared for efficiency.

among slices, and for each selected slice, the corresponding UE scheduler distributes resources among the UEs. The SM allows to configure the slice algorithm (setting the slice scheduler) and a list of slices with algorithm-specific parameters (selecting the user scheduler and configuring its available resources). The xApp is oblivious of the RAT, as the SM is designed in a RAT-independent way, and supports any slice algorithm implemented in the user plane. Further, through RRC UE notifications, the xApp discovers the UE-to-service association through the selected PLMN identification or slice information (S-NSSAI [18]) provided in the attach procedure, configures new slices if necessary, and associates the UE to its slice. The SC SM iApp at the controller exposes the configuration using an HTTP REST north-bound interface, making the SM available to xApps. The components of this specialization are summarized in Table 4.

In this example, we use OAI (tag 2021.w20) and Google Pixel 5 phones over a 106 RB (20 MHz) NR carrier in band 78; the modulation-and-coding scheme is fixed to 20 for all UEs. The UEs receive constant downlink traffic generated using *iperf* such that the radio resources of the cell are exhausted at all times. A proportional fair scheduler is used that equally distributes resources between UEs in the absence of slicing. We employ the NVS algorithm [26].

In the first experiment, we demonstrate the isolation property of slicing in 5G (Fig. 13a). The objective is to ensure that the UE in slice 1 (white) gets 50 % of the resources, or around 30 Mbps. At time instance 1, no slicing is active; however, since only two UEs are present, they equally share the resources. Upon a new connection of a UE, at time instance 2, the resources are equally shared between all three UEs, violating the requirement that the white UE receives 50% of resources. Therefore, at time instance 3, the xApp deploys slices with equal resource share and associates the white UE to slice 1 and the rest to slice 2, which ensures half of the resources for the white UE. Similarly, at time 4, the user plane enforces the xApp's request of 66 % resources for slice 1.

A fixed allocation of resources can lead to underutilization of the already scarce radio resources in the RAN. Therefore, it is advantageous to flexibly distribute or share resources between slices if UEs do not consume the resources reserved for their slice. In the second experiment, we connect two UEs, associated with two slices with 66 % (gray) and 34 % (black) of resources. In the upper graph of Fig. 13a, we configured slices to not share any resources. This leads to wasted resources as the gray slice does not use the additional resources while the black slice is inactive. On the other hand, if resource sharing is permitted, the gray slice increases throughput by 50 %, which shows how the NVS algorithm exploits the idea of offering isolation while efficiently using the radio spectrum.



(a) Architecture with SM-specific virtualization for slicing.

(b) Recursive control allows abstracting topologies and build hierarchies of controllers.

Figure 14: Control of the RAN can be shared between multiple controllers through a virtualization layer.

5G scenarios that will benefit from this SM are those related to guaranteed SLA/QoS. Examples are industry 4.0, or any use case that involves multiple services being multiplexed onto a common structure, e.g., IoT or emergency services over public networks.

6.2 Recursive Slicing

As a consequence of increased densification for base stations, capital and operational expenditure for cellular networks are further increasing. RAN sharing is envisioned as a way to reduce costs. O-RAN assumes that operators might host the virtual network functions (VNF) of base stations of a visiting operator and enable remote access ("remote" E2) to control those VNFs [27]. While this allows a reduction of costs in infrastructure, there is no potential for multiplexing gains when using dedicated base station VNFs. On the other hand, since the operators are competitors, the sharing operator does not want to give full access to prevent malicious access or erroneous configurations.

We show how to share the VNFs between operators, enabling full potential for multiplexing gains in the radio spectrum, and give both operators the possibility to configure slices individually within their (virtual) network. For this purpose, we propose a controller specialization that forwards control to multiple northbound controllers. However, it is of paramount importance to *avoid* any conflicts resulting from control decisions of different controllers.

We design such a controller specialization, as shown in Fig. 14a, by reusing the agent library as the northbound communication interface. It recursively exposes the E2 interface to multiple guest controllers, using the agent library's multi-controller functionality (see Section 4.1.2). Multiple iApps, fulfilling the role of RAN functions towards the agent library, implement a virtualization layer. Since the sliceable resources within a base station are highly heterogeneous, comprising flow control for multiple bearers (e.g., TC SM), radio connection control, and resource scheduling (e.g., SC SM), such virtualization layer is SM-specific. We focus on slicing the scheduling resources, using the SC SM abstraction presented previously, and partition the MAC statistics. The components of this specialization are summarized in Table 5.

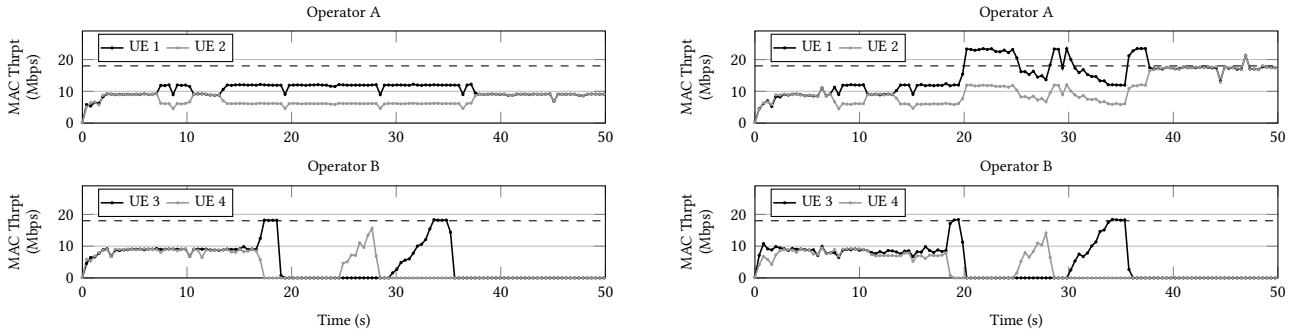
Table 5: Virtualization layer specialization.

Comp.	Used
xApp	Slicing controller (cf. Section 6.1.2)
Comm. IF	Agent library
iApps	MAC stats (partitioning)
	SC SM virtualization
Support	Server library

The virtualization layer abstracts both the assigned radio resources and slice IDs from their physical representation (southbound) to a virtual representation (north-bound). The service level agreement (SLA) indicates the resource share of physical resources that are assigned to an operator. The iApp virtualizes the resources of NVS [26] by scaling the physical resource share by the inverse of the SLA to a virtual resource share of 100 % (see Appendix B for a detailed description). This scheme guarantees that no controller can exceed its assigned resources, effectively avoiding any conflicts. Since admission control of the virtual slices ensures that the total resource share does not exceed 100 %, the physical resource share cannot be higher than the guaranteed SLA, and NVS ensures that every slice receives its configured amount of resources. Further, as the north-bound controllers can freely chose their slice IDs, conflicts between slice IDs are avoided by mapping (virtual) IDs in the range 0-9 into (physical) IDs in disjoint intervals for each operator. Finally, the MAC statistics SM is sliced by only revealing UEs to a controller which are among the respective operator's subscribers.

We compare the cases of two dedicated infrastructures of operators A and B, and compare it to a shared infrastructure. In the second case, the virtualization controller connects the slicing controllers of the operators to the infrastructure. The virtualization controller abstracts the resources and slices the user groups and exposes relevant information to the RAN slicing controllers of operators A and B. Both operators have a resource share of 50 % of the total base station resources. We use OAI 4G/LTE (tag 2021.w20) with four UEs (two per operator). The dedicated infrastructures are two dedicated eNBs with each 25 RBs (5 MHz), and the shared infrastructure a single eNB with 50 RB (10 MHz), all in band 7. Note that in this case, we use the same slicing controllers over a 4G infrastructure instead of 5G, highlighting the multi-RAT capability of FlexRIC and its SC SM.

Fig. 15 shows the network throughput as perceived by the two slicing controllers (the dashed lines mark the maximum throughput of a single dedicated eNB). Initially, both controllers have no slice configured, and since the resource share is the same (50 %), all UEs have the same throughput. At around 8 and 11 s, operator A configures two sub-slices (66 %, 33 %). This has no impact on operator B, proving the isolation capabilities of the system. Once one of operator B's UEs has no throughput, the other UE in the same virtual network benefits from more resources. However, when operator B has no traffic, the resources are equally shared between the two sub-slices of operator A in the shared infrastructure case, proving the sharing capabilities. On the other hand, if the infrastructures are dedicated, much of the resources are lost. If one operator is unloaded, the multiplexing gain can attain 100 %.



(a) Dedicated infrastructures (two eNBs). **(b) Shared infrastructure (one eNB).**

Figure 15: Comparison of dedicated and shared infrastructure of two operators with (in total) four UEs. The controllers can freely reconfigure slices. In the sharing case (b), isolation and sharing properties are preserved.

To allow controllers to not only configure sub-slices within their service, but exert direct control on the UE allocation, the virtualization layer could be extended with the abstraction already described in Orion [13]. However, unlike Orion, this virtualization layer is recursive in that virtualization layers might be stacked recursively, similar to FlowVisor [8].

Apart from virtualizing the control plane, a recursive controller also allows to abstract the network topology. As shown in Fig. 14b, a disaggregated base station deployment, consisting of CU and DU, might simply be exposed as one base station entity. Similarly, more complicated deployments, such as one CU and multiple DU, might be exposed as multiple base stations, and control commands, such as handover for mobility load balancing, are translated into the appropriate control commands for the deployment in consideration.

The slicing concept defined by 3GPP imposes the use of a new PDU session per slice, which compromises a new PDCP, RLC and possibly SDAP entity per slice. Therefore, creating a slice within a slice is a logical concept rather than physical one, as for example, no RLC entity can be recursively generated inside an RLC entity. However, the logical concept of recursively assigning resources remains a valid a useful abstraction, applicable to many use cases.

Some examples of possible 5G scenarios include RAN sharing, or a network operator that leases resources to service providers while allowing certain level of configuration and/or monitoring access.

6.3 Controller for Hosting O-RAN xApps

While FlexRIC employs an E2-compatible south-bound (allowing the agent to interface with any E2-compatible controller), the FlexRIC SDK does *not* provide a fully O-RAN-compatible non-realtime controller [28]. Instead, it focuses on the core functionality of handling E2 nodes, multiplexing the messages between iApps and RAN functions, and allowing interoperability on the E2AP level between a FlexRIC controller and an E2 node, following the 5G principles of flexibility, forward compatibility, and ultra-lean design.

A number of services are required to host xApps [28]: (1) a messaging infrastructure that allows xApps to send messages between xApps and the controller; (2) subscription management, e.g., merging identical subscriptions; (3) xApp management to deploy xApps; (4) a database for xApps to write and read information gathered through SMs; and (5) additional services such as security, logging, and fault management. A FlexRIC controller specialization targeting O-RAN xApps needs to implement these services as (SM-independent) iApps to push SM functionality into xApps.

Having a simple-to-use E2 controller, as opposed to cluster-based implementations such as O-RAN RIC or ONF SD-RAN, is of high importance. First, such a controller would facilitate research using standard O-RAN xApps from commercial deployments, while using significantly less resources. Second, it would reduce xApp development and debugging lifecycle, which does not need to communicate with multiple services. The corresponding FlexRIC controller specialization could be used as a simple-to-use O-RAN RIC replacement, hosting xApps that implement standard O-RAN use cases [27].

7 CONCLUSION

In this paper, we presented FlexRIC, an SDK that serves as the pillar to build specialized, multi-service SD-RAN controllers. Using the abstractions provided by the SDK, it is possible to compose customized controllers that smoothly adapt to the envisioned state-of-the-art 5G scenarios. Additionally, we experimented with SMs for flow-level traffic control (i.e., TC SM) and resource-level slicing control (SC SM) and validated them to build service oriented controllers that reduce the latency in modern cellular networks. Further, we built an SD-RAN virtualization controller that multiplexes virtual RANs of multiple tenants, such as operators, onto a shared infrastructure while isolating their networks. Such virtualization control exposes a virtualized view of the RAN and enables them to independently control their network, even over disaggregated base stations. Lastly, FlexRIC follows a "zero-overhead principle", resulting in better performance than real-time SD-RAN controllers, without consuming resources imposed by unused features, contrary to O-RAN. We plan to open-source FlexRIC to the community under appropriate license to boost its adoption by both academia and industry. As future work we plan to deploy further SMs that enhance FlexRIC capabilities (e.g., AI/ML SMs), port the existing SMs to other base stations (i.e., SRSRAN), and develop an interface for FlexRIC's server for different languages to facilitate its wider adoption.

ACKNOWLEDGMENTS

The authors would like to thank the shepherd Domenico Giustiniano and the anonymous reviewers for their helpful feedback on improving the manuscript. This work received funding from the European Commission's Horizon 2020 Framework Programme under Grant No. 857201 (5G-VICTORI), No. 957317 (Affordable5G), and No. 101017226 (6G-BRAINS).

REFERENCES

[1] Xenofon Foukas, Navid Nikaein, Mohamed M. Kassem, Mahesh K. Marina, and Kimon Kontovasilis. 2016. 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). Association for Computing Machinery, Irvine, California, USA, 427–441. DOI: https://doi.org/10.1145/2999572.2999599.

[2] 2020. Architecture & E2 General Aspects and Principles. Technical Specification O-RAN.WG3.E2GAP-v01.01. O-RAN Working Group 3.

[3] Bjarne Stroustrup. 1994. *The Design and Evolution of C++*. Addison Wesley.

[4] 2015. IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond. Recommendation ITU-R M.2083-0. Technical report. ITU-R, (September 2015).

[5] Estefanía Coronado, Shah Nawaz Khan, and Roberto Riggio. 2019. 5G-EmPOWER: A Software-Defined Networking Platform for 5G Radio Access Networks. *IEEE Transactions on Network and Service Management*, 16, 2, 715–728. DOI: https://doi.org/10.1109/tnsm.2019.2908675.

[6] Chih-Lin I and Sachin Katti (eds.) 2018. O-RAN: Towards an Open and Smart RAN. Whitepaper. O-RAN Alliance, (October 2018). https://www.o-ran.org/s/O-RAN-WP-FInal-181017.pdf.

[7] Open Network Foundation. 2021. Open Air Interface virtual summer workshop. (June 25, 2021). https://www.openairinterface.org/docs/workshop/2021-06-SUMMER-VIRTUAL-WORKSHOP/PDF/09_SHAD_ANSARI_MTS-OPEN_NETWORKING_FOUNDATION_O-RAN_E2AP_IMPLEMENTATION_IN_OAI.pdf.

[8] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. 2009. FlowVisor: A Network Virtualization Layer. Technical report. http://OpenFlowSwitch.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf.

[9] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38, 2, (March 2008), 69–74. DOI: https://doi.org/10.1145/1355734.1355746.

[10] Kok-Kiong Yap, Masayoshi Kobayashi, Rob Sherwood, Te-Yuan Huang, Michael Chan, Nikhil Handigol, and Nick McKeown. 2010. OpenRoads: Empowering Research in Mobile Networks. *SIGCOMM Comput. Commun. Rev.*, 40, 1, (January 2010), 125–126. DOI: https://doi.org/10.1145/1672308.1672331.

[11] Kok-Kiong Yap, Rob Sherwood, Masayoshi Kobayashi, Te-Yuan Huang, Michael Chan, Nikhil Handigol, Nick McKeown, and Guru Parulkar. 2010. Blueprint for Introducing Innovation into Wireless Mobile Networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures* (VISA 10). Association for Computing Machinery, New Delhi, India, 25–32. DOI: https://doi.org/10.1145/1851399.1851404.

[12] Aditya Gudipati, Li Erran Li, and Sachin Katti. 2014. RadioVisor: A Slicing Plane for Radio Access Networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (HotSDN 14). ACM, Chicago, Illinois, USA, 237–238. DOI: http://doi.org/10.1145/2620728.2620782.

[13] Xenofon Foukas, Mahesh K. Marina, and Kimon Kontovasilis. 2017. 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). ACM, Snowbird, Utah, USA, 127–140. DOI: http://doi.org/10.1145/3117811.3117831.

[14] Google. 2021. FlatBuffers white paper. (May 14, 2021). https://google.github.io/flatbuffers/flatbuffers_white_paper.html.

[15] 2020. E2 Application Protocol (E2AP). Technical Specification O-RAN.WG3.E2AP-v01.01. O-RAN Working Group 3.

[16] Navid Nikaein, Mahesh K. Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, and Christian Bonnet. 2014. OpenAirInterface: A Flexible Platform for 5G Research. *SIGCOMM Comput. Commun. Rev.*, 44, 5, (October 2014), 33–38. DOI: http://doi.org/10.1145/2677046.2677053.

[17] 2021. Protocol Buffers. https://developers.google.com/protocol-buffers/.

[18] 3GPP. 2021. System architecture for the 5G System (5GS). TS 23.501 V15.12.0. 3GPP, (January 2021).

[19] 2021. Bufferbloat. (June 21, 2021). https://www.bufferbloat.net/projects/.

[20] 2021. Use of L4S in 5GS. Change Request S2-2103904. 3GPP, (May 18, 2021). https://www.3gpp.org/ftp/tsg_sa/WG2_Arch/TSGS2_145E_Electronic_2021-05/Docs/S2-2103904.zip.

[21] Mikel Irazabal, Elena Lopez-Aguilera, Ilker Demirkol, Robert Schmidt, and Navid Nikaein. 2021. Preventing RLC Buffer Sojourn Delays in 5G. *IEEE Access*, 9, 39466–39488. DOI: https://doi.org/10.1109/ACCESS.2021.3063769.

[22] 2021. Enabling time-critical applications over 5G with rate adaptation. Technical report. Ericsson, (May 2021). https://www.ericsson.com/en/reports-and-papers/white-papers/enabling- time-critical-applications-over-5g-with-rate-adaptation.

[23] Pete Heist. 2021. irtt (Isochronous Round-Trip Tester). (June 21, 2021). https://github.com/heistp/irtt.

[24] Navid Nikaein, Eryk Schiller, Romain Favraud, Kostas Katsalis, Donatos Stavropoulos, Islam Alyafawi, Zhongliang Zhao, Torsten Braun, and Thanasis Korakis. 2015. Network Store: Exploring Slicing in Future 5G Networks. In *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture* (MobiArch 15). ACM, Paris, France, 8–13. DOI: http://doi.org/10.1145/2795381.2795390.

[25] curl developers. 2021. curl: command line tool and library for transferring data with URLs. (October 20, 2021). https://curl.se/.

[26] Ravi Kokku, Rajesh Mahindra, Honghai Zhang, and Sampath Rangarajan. 2012. NVS: A Substrate for Virtualizing Wireless Resources in Cellular Networks. *IEEE/ACM Transactions on Networking*, 20, 5, (October 2012), 1333–1346. DOI: https://doi.org/10.1109/TNET.2011.2179063.

[27] Chai Li and Arda Akman (eds.) 2020. O-RAN Use Cases and Deployment Scenarios – Towards Open and Smart RAN. Whitepaper. O-RAN Alliance, (February 2020).

[28] 2020. Near-Real-time RAN Intelligent Controller; Near-RT RIC Architecture. Technical Specification O-RAN.WG3.RICARCH-v01.01. O-RAN Working Group 3.

[29] 2020. E2 Service Model (E2SM), KPM. Technical Specification ORAN-WG3.E2SM-KPM-v01.00.00. O-RAN Working Group 3.

[30] 2020. E2 Service Model (E2SM), RAN Function Network Interface (NI). Technical Specification ORAN-WG3.E2SM-NI-v01.00.00. O-RAN Working Group 3.

A E2 PRIMER

To allow FlexRIC to remain compatible with the industry consortium (O-RAN) effort to create standardized SD-RAN controller interfaces, it adopts the E2 interface [2]. Due to E2's novelty and FlexRIC's heavy usage of it, we provide a primer for the interested reader in the following section.

A.1 E2 Overview

The interface for interconnecting RAN elements and a RAN controller is the E2 interface. The RAN elements are called "E2 nodes" and can refer to an eNB, gNB, or just a part thereof, e.g., CU or DU, while the RAN controller is called the "Near-realtime RAN intelligent controller" (RIC). The E2 interface has been defined by O-RAN in an attempt to provide standardized interface, enabling interoperability among diverse multi-vendor implementations [2, Section 5.1]

- Send pre-defined information on pre-defined trigger events from the RAN to the RIC (*reports*),
- Send *control* messages and *policies* from RIC to RAN, and
- Enable the RIC to process procedures at the RAN's place (*insert*).

In short, the objectives of E2 are "exposure of selected E2 Node data" (configuration, statistics, measurements, …) and "enabl[ing] the Near-RT RIC to control selected functions" [2, Section 5.2] of the RAN (such as triggering procedures, installing policies, RAN configuration, …).

The E2 interface serves to connect xApps to RAN functions, as shown in Fig. 16. A 1 to N relationship between the RIC and the E2 Nodes is foreseen. Each E2 node has an agent, managing the E2 connection, and a number of *RAN functions*. RAN functions refer to functionality within an E2 Node (e.g., triggering a handover). The RIC is a controller that provides a terminating point for all

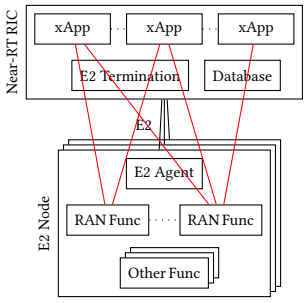


Figure 16: E2 enables RIC applications ("xApps") to control the RAN through RAN functions. [2]

E2 Nodes, hosts external applications (i.e., xApps), and additional services (e.g., database functionality).

As can be seen in the figure, there exists a many-to-many mapping between xApps and RAN functions. This link allows xApps to control the RAN functions. The interaction between the RIC and xApps on one side and the E2 node and RAN functions on the other is enabled through (i) the E2 application protocol (E2AP) [15] and (ii) E2 service models (E2SM).

A.2 E2AP

"The E2AP supports the functions of E2 interface by signaling procedures" [15] that allow RIC and E2 nodes to communicate. The protocol messages are divided into the classes:

E2 Global Procedures are messages for connection management between a controller and the agent.

E2 Functional Procedures are messages specifically destined to RAN functions within the E2 agent for functional message exchange.

The functional procedures can be further divided into:

Subscription Allow xApps to subscribe to event triggers in RAN functions.

Indication Allow RAN functions to forward information to the RIC, and

Control Allow xApps to execute a procedure within a RAN function.

A.3 E2SM

Until now, the *actual* interaction between an xApp and a RAN function has not been described. This is the role of E2SMs, which are specifications in their own right. Each model E2SM provide the "service" that a RAN function can fulfill. There are four basic actions/services named *report*, *insert*, *control*, and *policy*, each of which is transported through the E2AP procedures.

Reports contain information that are sent from the E2 nodes to the RIC/xApps. As it can be seen in Fig. 17a, an xApp sends an E2AP subscription for requesting a report. The E2SM specifies the types of reports ("action definition") that can be sent and the trigger events that lead to them. Once the trigger is detected, the RAN function sends an E2SM report that is transported through an E2AP indication message.

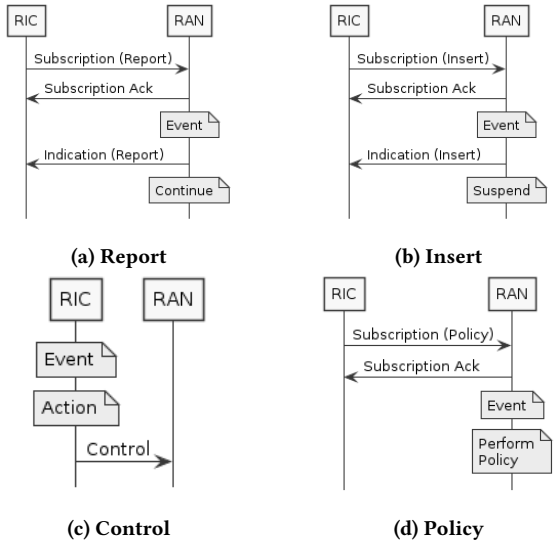


Figure 17: The four basic "services" that are specified in E2SMs to define the interaction between an xApp and a RAN function.

Inserts are messages sent from E2 nodes to inform an xApp of an event. As it is shown in Fig. 17b, an xApp first subscribes for an insert message via an E2AP subscription. Once the trigger has been detected, the RAN function sends an E2SM insert message that is transported via an E2AP indication.

Control messages are sent from the RIC/xApp to the RAN function to execute an operation, as highlighted in Fig. 17c.

Policies are predefined operations that the RAN function should execute upon a trigger as shown in Fig. 17d.

A.4 Standardized E2SMs

At the time of writing this paper, two E2SMs have been standardized:

(1) Performance metrics (E2SM-KPM) [29] defines various report types on periodic timer expires.
(2) Network interface (E2SM-NI) [30] allows interface manipulation, supporting interfaces such as X2, S1, etc. It defines (i) report messages for message exchange on interface, (ii) insert messages, copying the occurred interface message to the xApp, (iii) control messages to inject interface messages into interfaces, or (iv) policies to help the RAN function to decide how to proceed with a message on a certain interface.

B VIRTUALIZING NVS

NVS defines (1) capacity slices with a share of resources c_s, and (2) rate slices with a reserved rate r_s^{rsv} over a reference rate r_s^{ref}. The reference rate is an indicator for the minimum rate such slice needs to achieve in scheduled slots to prevent excess utilization of resources under bad channel conditions. NVS shows that both slice types are equivalent and that each slice can be guaranteed its reserved resources under the assumption that the total resource

share does not exceed the base station resources for slices s:

$$\sum_s c_s + \sum_s \frac{r_s^{\text{rsv}}}{r_s^{\text{ref}}} \leq 1 \ .$$

We reuse the NVS notation to show how multiple controllers can independently create their (sub-)slices within their virtual network with a resource share q (service level agreement). We assume a set of slices $\mathcal{S}$ in the network, with $\sum_{s \in \mathcal{S}} q_s \leq 1$. We have for virtual slice s a physical resource representation:

$$\sum_i c_{s,i}^{\text{phys}} + \sum_i \frac{r_{s,i}^{\text{rsv,phys}}}{r_{s,i}^{\text{ref,phys}}} \leq q_s$$

which is exposed as a virtualized resource representation

$$\sum_i c_{s,i}^{\text{virt}} + \sum_i \frac{r_{s,i}^{\text{rsv,phys}}}{r_{s,i}^{\text{ref,virt}}} \leq 1 \ ,$$

where each (sub-)slice's virtual capacity c^{virt} is scaled by the factor $\frac{1}{q_s}$. Note that a slice's virtual rate is represented as the physical rate; the scale operation is expressed by a scale of the physical reference rate down by q_s. As an example, consider a base station with a throughput of 100 Mbps that is shared equally (50 Mbps) by two operators. If one operator creates a slice with a 5 Mbps slice over reference 50 Mbps (10% resources), it is mapped back into the real resources as a 5 Mbps slice and reference rate of 100 Mbps (a 5% share, corresponding to the SLA).

This scheme guarantees that no controller can exceed its assigned resources, avoiding any conflicts. Since admission control of the virtual slices ensures that the total resource share does not exceed 100%, the physical resource share cannot be higher than the SLA.