# FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks

Xenofon Foukas
The University of Edinburgh
x.foukas@ed.ac.uk

Navid Nikaein
Eurecom
Navid.Nikaein@eurecom.fr

Mohamed M. Kassem
The University of Edinburgh
M.M.M.Kassem@sms.ed.ac.uk

Mahesh K. Marina
The University of Edinburgh
mahesh@ed.ac.uk

Kimon Kontovasilis
NCSR "Demokritos"
kkont@iit.demokritos.gr

## ABSTRACT

Although the radio access network (RAN) part of mobile networks offers a significant opportunity for benefiting from the use of SDN ideas, this opportunity is largely untapped due to the lack of a software-defined RAN (SD-RAN) platform. We fill this void with FlexRAN, a flexible and programmable SD-RAN platform that separates the RAN control and data planes through a new, custom-tailored southbound API. Aided by virtualized control functions and control delegation features, FlexRAN provides a flexible control plane designed with support for real-time RAN control applications, flexibility to realize various degrees of coordination among RAN infrastructure entities, and programmability to adapt control over time and easier evolution to the future following SDN/NFV principles. We implement FlexRAN as an extension to a modified version of the OpenAirInterface LTE platform, with evaluation results indicating the feasibility of using FlexRAN under the stringent time constraints posed by the RAN. To demonstrate the effectiveness of FlexRAN as an SD-RAN platform and highlight its applicability for a diverse set of use cases, we present three network services deployed over FlexRAN focusing on interference management, mobile edge computing and RAN sharing.

## 1. INTRODUCTION

Mobile data traffic has seen a dramatic rise in recent times due to the rapid adoption of smartphones and their support for data-hungry services like video streaming. This trend is expected to continue into the foreseeable future, posing a major challenge for future mobile networks. Besides, the need to support newer communication paradigms like machine-to-machine (M2M) and device-to- device (D2D) makes the problem more challenging as they are characterized by high control signaling-to-data ratios, not the regime for which current mobile networks are designed for.

The aforementioned observations have become driving factors behind the impending evolution of mobile networks to the 5G era. Key 5G requirements include: 1000x increase in peak data rate, support of 100x more devices, sub-millisecond round-trip communication latency and the reduction of energy/cost through the optimized network management [10]. These requirements are necessitating innovation in various dimensions ranging from scaling capacity (via ultra densification, new radio access technologies and spectrum bands) to newer, more flexible mobile network architectures.

Software Defined Networking (SDN) is among the key technologies considered in the context of evolving mobile networks. SDN has gained significant traction over the past decade, mainly in the context of data centers and wired networks. This is brought about by paradigm shifting ideas underlying SDN, which are the separation of the control from the data plane through a well-defined API (e.g., OpenFlow), the consolidation of the control plane and the flexibility introduced to the network through its programmability. These fundamental SDN ideas can contribute towards addressing various challenges faced by current and future mobile networks. Not surprisingly, there has been considerable research interest on software-defined mobile networks in recent years with much of the early work focusing on mobile core given its similarity to wired networks (e.g., [24],[29]).

The radio access network (RAN) part of mobile networks, both the costliest and the most complex part of the mobile network infrastructure, arguably offers even greater opportunities to benefit from SDN ideas. One reason is that strategies and technologies being adopted to improve spectrum efficiency and scale system capacity — cell densification, use of multiple radio access technologies (e.g., LTE and WiFi), use of advanced PHY techniques like Coordinated Multi-point (CoMP), etc. — require a high level of coordination among base stations, which SDN can naturally enable. As another reason, softwarization of control in mobile networks, especially in the RAN, not only allows easier evolution to the future through programmability but also enables a wide range of use cases and novel services (as we will discuss later). At the same time, a software-defined RAN (SD-RAN) design is challenging given the unique nature of wireless resources to be managed and the stringent timing constraints associated with some key RAN control operations (e.g., MAC scheduling). As discussed in Section 2, existing SD-RAN work, although abundant (e.g., [22], [9], [17], [40]), is largely conceptual with no implemented solution that researchers can use as a reference to evaluate their SD-RAN designs and to assess the benefit of new SD-RAN enabled services.

In this paper, we develop FlexRAN that to the best of our knowledge is the first open-source SD-RAN platform, thereby filling the above mentioned void. FlexRAN incorporates an API to separate control and data planes that is tailored for the mobile RAN. FlexRAN controller design and implementation factors in the need to make real-time RAN control applications feasible. Moreover, FlexRAN is designed with flexibility, programmability and ease of deployment in mind. FlexRAN offers a great degree of flexibility to easily and dynamically realize different degrees of coordination among base stations reflecting centralized to fully distributed modes of operation. It offers programmability at two levels, one in the form of RAN control/management applications that can be built over the FlexRAN controller and the other within the controller to be able to update the implementation of any control function on the fly. FlexRAN is also transparent to the end-devices, aiding easier deployment and evolution.

Specifically, we make the following contributions:

- (Sections 3 and 4) Realizable SD-RAN design in the form of FlexRAN. FlexRAN incorporates an API (FlexRAN Agent API) for clean separation of control and data planes in the RAN. Its hierarchical master-agent controller architecture is well suited for real-time RAN control operations while allowing reprogrammability and reconfigurability via its features of virtualized control functions and control delegation following Network Functions Virtualization (NFV) principles.
- (Sections 4 and 5) Implementation of FlexRAN over the OpenAirInterface (OAI) LTE platform [31] is shown to be efficient to the extent the use of FlexRAN is imperceptible to an end user device compared to using vanilla OAI, even when considering the time critical MAC scheduling operations. We also thoroughly characterize the FlexRAN performance behavior under different network conditions, varying number of UEs, and in presence of control delegation.
- (Section 6) We show results from using FlexRAN in a diverse set of use cases relevant to current and future mobile networks, namely interference management, mobile edge computing and RAN sharing. This demonstrates the ease with which new applications and services can be realized with FlexRAN, thereby its effectiveness as an SD-RAN platform. We also discuss additional use cases that FlexRAN can enable.

## 2. RELATED WORK

Software-defined control of mobile networks has received substantial attention from the research community in recent years with both academia and industry recognizing its benefits and proposing ways to integrate SDN principles to operational mobile networks [27]. Owing to the similarity between mobile core and wired networks much of the cellular SDN research till date focuses on the core part of mobile networks. The focus of this body of work [24],[29], [13], [33], [32], [28], [20] is on novel control designs based on SDN and NFV to address key core network issues of traffic and mobility management, and enable mobile networks to scale in the presence of high volumes of traffic.

While the scope of some of the above mentioned works includes the RAN, none of them address radio resource management, a vital aspect of the RAN and the focus of our work. RAN radio resource management is unique in the type of resources managed and the stringent timing constraints associated with some of the key functions (e.g., scheduling). In the last few years, there have been *several high-level conceptual works on SD-RANs* that do consider the radio resource management aspect [22], [9], [17], [40], [39], [8], [12]. SoftRAN [22] is among the earliest of these works. It introduced the idea of a big base station abstraction aimed at turning dense network deployments into sparse ones through the separation of the control and the data plane. In SoftRAN, control functions are *statically* separated into central and distributed ones based on their time criticality and their requirement for a centralized network view (e.g., centralized handovers and distributed downlink scheduling). Later works outline several potential SD-RAN designs, targeting different applications and settings. Similarly to SoftRAN, these works consider the tradeoffs between cost and efficiency for the control of the RAN and accordingly propose designs that follow either a hierarchical approach as in SoftRAN (e.g., [9], [17], [40]), where different layers of controllers are responsible for different operations based on their time criticality, or a fully centralized approach (e.g., [39], [8], [12]) where all the processing (L1/2/3) is performed centrally at a cloud data center.

A common and key limitation of the aforementioned SD-RAN works, which serves as the main motivation of our work, is that none of them have been implemented (and thus do not consider the associated challenges such as real-time control in their designs) nor do they tackle the issue of separating the control and data planes in the RAN in a practical and concrete manner. Moreover, none of these works offer mechanisms to make control in the RAN adaptive and flexible by allowing a dynamic functional split (e.g., centralized to distributed scheduling and vice versa) depending on the deployment scenario and the constraints posed by the underlying network conditions at any given point in time. Finally, not all of the proposed SD-RAN architectures are transparent to the UEs, with some designs proposing the introduction of programmability even at the UE-level (e.g., [40], [15]), raising backward compatibility concerns for legacy devices.

There are also works in the literature that are relevant from a RAN programmability perspective but can be viewed as complementary to our focus on a SD-RAN platform capable of performing radio resource management. For example, RadioVisor [25] deals with the challenge of radio resource virtualization in the RAN, allowing individual SD-RAN controllers to manage their own isolated slices. OpenRadio [14] and PRAN [38] deal with data plane programmability and how new wireless protocols can be implemented on-the-fly programmatically. Finally, the work of Tsagkaris et al. [36] proposes a software-defined framework for self organizing networks (SON) that simplifies the management of SON functions via a controller based on SDN principles.

## 3. FlexRAN OVERVIEW

This section gives a high-level overview of the FlexRAN SD-RAN platform, the key contribution of this paper. We present FlexRAN in the context of LTE for concreteness and to match with its current implementation, using the LTE terminology of eNodeBs and UEs for base stations and mobile devices. It is however important to note that there is nothing LTE-specific that FlexRAN assumes, thus its design is general and equally suitable for future mobile RAN architectures.

Fig. 1 provides a high-level schematic of the FlexRAN platform, which is made up of two main components: **FlexRAN Control Plane** and **FlexRAN Agent API**. The control plane follows a hierarchical design and is in turn composed of a **Master Controller** that is connected to a number of **FlexRAN Agents**, one for each eNodeB. The agents can either act as local controllers with a limited network view and handling control delegated by the master, or in concert with other agents and the master controller. The control and data plane separation is provided by the FlexRAN Agent API which acts as the southbound API (*a la* OpenFlow) with FlexRAN control plane on one side and eNodeB data plane on the other side.

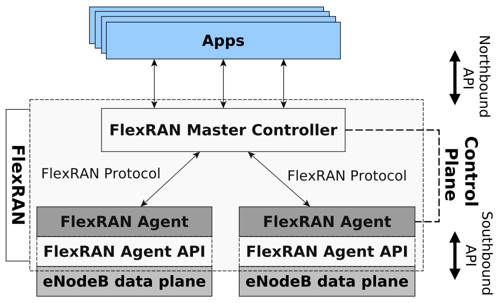


Figure 1: High-level schematic of the FlexRAN platform.

The FlexRAN **Protocol** facilitates communication between the master controller and the agents by allowing a two-way interaction between them. In one direction, the agent sends relevant messages to the master with eNodeB statistics, configurations and events, while in the other direction the master issues control commands that define the operation of the agents. In contrast to typical SDN controllers found in the wired domain, the FlexRAN controller has been designed with support for time critical RAN operations (e.g., MAC scheduling) in mind. Due to this real-time aspect and in order to fully utilize the power of FlexRAN, in the ideal case the communication channel between the agents and the master would be a high-bandwidth and low-latency channel (e.g., optical fiber path). However it should be noted that this is not a hard constraint, as the system provides flexibility to operate in non-ideal networking conditions with a small impact on its performance and capabilities (see Section 4.3.1).

Note that Fig. 1 does not include a UE, reflecting the fact that FlexRAN is transparent to UEs. The FlexRAN agent ensures that any command issued by the master that would affect the operation of a UE will be passed to the eNodeB, which will in turn apply the modification using the RAN technology standard in use (LTE protocol in our implementation). This transparency ensures the evolvability of the system, since new LTE and FlexRAN protocol extensions can be implemented at the RAN without causing backwards compatibility issues for users and without disrupting the use of their devices with constant updates to enable support of new network features. As long as a UE adheres to the LTE standard, it will be fully compatible with the FlexRAN platform.

On top of the master controller lies a northbound API, which allows RAN control and management applications to modify the state of the network infrastructure (eNodeBs and UEs) based on the statistics and events gathered from the eNodeBs in the FlexRAN control plane. Such applications could vary from simple monitoring applications that obtain statistics reporting which can be used by other apps (e.g., mobile edge computing app) to more sophisticated applications that modify the state of the RAN (e.g., MAC scheduler).

## 4. FlexRAN DESIGN & IMPLEMENTATION

### 4.1 Design Challenges

Based on the discussion of Sections 1 and 2 we identified a number of challenges that FlexRAN should resolve in order to be an effective SD-RAN platform:

- Separation of the control from the data plane in a clean and programmable way (Section 4.2).
- Adaptive and flexible RAN control with support for a dynamic control function placement depending on the deployment scenario and the constraints posed by the underlying network conditions (Sections 4.3.1 and 4.3.2).
- Support for the deployment of network applications over the controller, considering critical real-time applications (e.g. a MAC scheduler) (Sections 4.3.3 and 4.4).

In the following subsections, we describe the way that we overcame these challenges through the detailed description of the components that make up the FlexRAN platform (Fig. 1) as well as their implementation details over the OpenAirInterface LTE software platform. We present our design in a bottom-up manner, starting with the FlexRAN Agent API.

### 4.2 FlexRAN Agent API

All the access stratum protocols of LTE (RLC/MAC, PDCP, RRC) can be decomposed into two parts; the control part that makes the decisions for the radio link and the action part that is responsible for applying those decisions. For example, the control part of the MAC makes scheduling decisions (resource block allocation, modulation and coding scheme etc), while the action part applies them. Similarly, part of the logic of the RRC protocol decides on UE handovers, while the actual handover operation requires RRC to perform the corresponding action. Based on this taxonomy, FlexRAN separates the RAN control plane from the data plane by detaching the control logic from the action and consolidating all the control operations in a logically centralized controller, which in FlexRAN comprises of Master Controller and Agents interacting via the FlexRAN Protocol (see Fig. 1). As a result, eNodeBs only handle the data plane to perform

| Call Type | Description | Target | Example outcomes of function calls |
|---|---|---|---|
| Configuration (Synchronous) | Get/Set configurations of target | eNodeB/Cell/UE/ Logical Channel | eNodeB ID, Number of cells, Cell id, UL/DL bandwidth, Number of antenna ports, RNTIs, UE transmission mode |
| Statistics (Asynchronous) | Request/reply statistics reporting. | List of cells/UEs | Transmission queue size, CQI measurements, SINR measurements |
| Commands (Synchronous) | Apply control decisions | Agent Control Modules (See Section 4.3.1) | Scheduling decisions, DRX commands, Handover initiation |
| Event-triggers (Asynchronous) | Notify control plane about changes in the data plane | Master Controller | Initiation of Transmission Time Interval, UE attachment, Random access attempt, Scheduling requests |
| Control Delegation (Synchronous) | Push control functions and modify their behavior at the agent side | Agent Control Modules (See Section 4.3.1) | Swap DL scheduler or mobility manager, Modify threshold of signal quality for handover initiation |

Table 1: Type of function calls in FlexRAN Agent API.

all the action-related functions (e.g., applying scheduling decisions, performing handovers, applying DRX commands, (de)activating component carriers in carrier aggregation).

To control and manage the eNodeB data plane, we introduce the FlexRAN Agent API, which defines a set of functions that constitute the southbound API of FlexRAN and are the primary enabler for software-defined control of the RAN. These functions allow the control plane to interact with the data plane in five ways: (1) to obtain and set configurations like the UL/DL bandwidth of a cell; (2) to request and obtain statistics like transmission queue sizes of UEs and SINR measurements of cells; (3) to issue commands to apply control decisions (e.g., calls for applying MAC scheduling decisions, performing handovers, activating secondary component carriers); (4) to obtain event notifications like UE attachments and random access attempts; and (5) to perform a dynamic placement of control functions to the master controller or the agent (e.g. centralized scheduling at the master controller or local scheduling at the agent-side). These API calls can be invoked either by the master controller through the FlexRAN protocol using the message handler and dispatcher entity residing at the agent side (Figure 2) or directly from the agent if control for some operation has been delegated to it. The API calls are currently defined using the C language. A detailed list of the function call types is shown in Table 1.

Through the FlexRAN Agent API it becomes possible to develop two types of applications: (1) applications related to the control and management of the RAN resources like schedulers, interference and mobility managers etc. (e.g. by controlling resource block allocations, modulation and coding schemes, handovers and DRX cycles) and (2) applications relying on the monitoring of the RAN resources to make more sophisticated decisions (e.g. adaptive video streaming based on channel quality, resource-block allocations for RAN sharing etc) (see Sections 6 and 7.1). It should be noted that FlexRAN does not deal with the control of flows in the wired domain and therefore does not directly support related applications like routing. To enable this, FlexRAN should be coupled with corresponding flow-control solutions, like an OpenFlow-based SDN controller.

## 4.3 FlexRAN Controller Architecture

### *4.3.1 FlexRAN Agent*

In this subsection, we elaborate on the different subcomponents of the FlexRAN Agent architecture shown in Fig. 2.

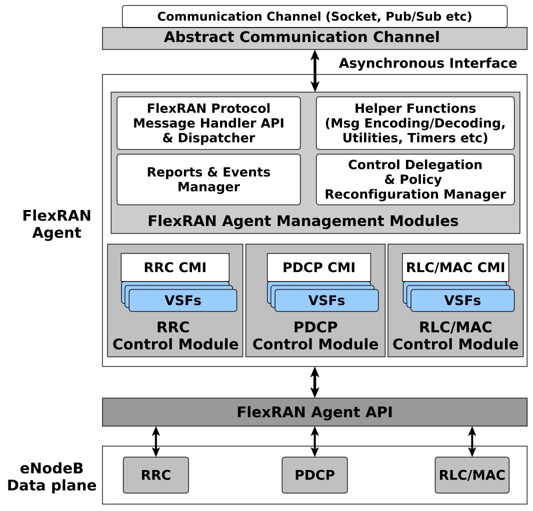

Communication Channel (Socket, Pub/Sub etc)
Abstract Communication Channel
Asynchronous Interface
FlexRAN Protocol Message Handler API & Dispatcher
Helper Functions (Msg Encoding/Decoding, Utilities, Timers etc)
Reports & Events Manager
Control Delegation & Policy Reconfiguration Manager
FlexRAN Agent Management Modules
FlexRAN Agent
RRC CMI
PDCP CMI
RLC/MAC CMI
VSFs
RRC Control Module
PDCP Control Module
RLC/MAC Control Module
FlexRAN Agent API
eNodeB Data plane
RRC
PDCP
RLC/MAC

Figure 2: The architecture of a FlexRAN Agent.

**Virtualized Control Functions.** To allow flexible and programmable control of the RAN infrastructure, the FlexRAN Agent provides a number of eNodeB *Control Modules* as illustrated in Fig. 2. These modules reflect a logical separation of the control operations that an eNodeB in the standard LTE architecture has to perform on the radio side and can be seen as a set of individual control subsystems, each targeting a specific area of control. Since the LTE standard (the technology supported by our current implementation) already provides a precise definition and scope for such control operations through the Access Stratum protocols (RRC, MAC/RLC, PDCP), FlexRAN adopts the same structure for the agent's control modules, with each module providing functionality according to the scope of its corresponding LTE protocol (e.g., MAC/RLC control module for scheduling, RRC control module for the radio resource control).

Each control module is in turn composed of a well-defined set of functions called *Virtual Subsystem Functions (VSFs)*. The VSFs implement the action that needs to be taken by the agent during a corresponding operation. As an example, consider the MAC control module which is broadly responsible for various scheduling operations of the eNodeB. For each of the scheduling operations (e.g., UE specific downlink and uplink scheduling, broadcast scheduling), FlexRAN defines a VSF that designates how the agent should behave for the corresponding operation. For instance, the UE specific downlink scheduling VSF could designate that the agent should forward the scheduling decision sent by the master controller to the data plane or that the agent should make its own decision based on a higher-level policy defined by the master.

The number and type of VSFs that each control module supports is defined through a well-defined *Control Module Interface (CMI)*, which essentially allows the agent to abstract the set of operations of each control module from their corresponding implementations. In this way, the agent reacts to a specific event (e.g., time for downlink scheduling) without having to worry about the underlying implementation of the operation. Through this design decision, the FlexRAN agent becomes very flexible, programmable and extensible since new operations can be introduced simply by extending the CMI, while at the same time the functionality of these operations can easily be redefined in a technology-agnostic manner via the abstract FlexRAN Agent API. As already discussed, a message handler and dispatcher entity residing at the agent side (see Fig. 2) is responsible to receive FlexRAN protocol messages from the FlexRAN master controller and forward them to be handled by the appropriate VSF of the corresponding control module using the FlexRAN Agent API.

**Control Delegation.** FlexRAN allows the master controller to assume control of the underlying infrastructure and orchestrate its operation by making and pushing control decisions at a very fine time granularity (per subframe). While this is a feature that is essential when considering centralized time-critical applications (e.g., coordinated remote scheduling of eNodeBs), such fine grained control is not always desirable nor even possible. For example, FlexRAN enabled small cells may not all be connected to the master via a high-speed backhaul (e.g., optical fiber link), making the exchange of all the required FlexRAN protocol messages (MAC statistics, scheduling decisions, event notifications, etc.) at a subframe granularity a very challenging task. In such cases, it is preferable to let the individual agents make time-critical decisions as per the policy specified by the master.

A naive way to achieve this would be to identify a set of hardcoded policies at the agent for each delegated operation. For example, the agent might offer a local scheduler with two policies (round-robin and proportional fair) to choose from. If the master chooses to delegate scheduling to an agent due to an unsuitable master-agent communication channel (e.g., high latency), it would have to choose among the available hardcoded policies. However, such a mechanism can be very limiting since it does not allow the modification of the agent's behavior at runtime nor its extension with new functionality in the future. FlexRAN avoids this via two complementary mechanisms: **VSF updation** and **policy reconfiguration**.

**VSF updation** This mechanism exploits the control function virtualization feature described above and allows the master to modify the behavior of the VSFs of a control module on-the-fly by "pushing" new code to the agent over the FlexRAN protocol. This code is actually a callback assigned to one of the CMI function calls that corresponds to a specific control module - VSF pair. The callback function is able to access and modify the underlying eNodeB data plane using the FlexRAN agent API discussed in Section 4.2. A VSF update FlexRAN protocol message designates the name of the control module and the VSF that the new code is intended for and contains the actual code in the form of a shared library that has been compiled against the agent architecture. The pushed code is initially stored in a cache memory at the agent-side until the master decides to modify the behavior of the corresponding VSF. The agent cache can store many different implementations for a specific VSF, which the master can swap at runtime. As an example, considering the case of the downlink UE scheduling VSF, the master could push two schedulers to the agent, a local proportional fair scheduler and a stub for a centralized remote scheduler, which it could switch at runtime (e.g., based on the network conditions).

**Policy reconfiguration** This mechanism is complementary to VSF updation since it allows the master to swap the agents' VSFs and reconfigure their parameters at runtime. A policy reconfiguration FlexRAN protocol message indicates the VSFs to be modified using YAML syntax (Fig. 3). At the top level, the control module name is indicated, followed by a sequence of VSFs to be modified, each composed of two (optional) sections; *behavior* and *parameters*. The *behavior* section is used for swapping VSFs, i.e., it is an instruction to the agent to link a specific CMI function call to one of the callbacks stored at its cache through the VSF updation mechanism. The *parameters* section indicates a list of parameters that can be modified for the specific VSF. These parameters act as a public API that the controller can modify and can either refer to a single value or a sequence of values (e.g., an array). The available parameters depend on the VSF implementation (e.g., two scheduler implementations could have different sets of parameters based on their functionality).

```
<Control Module Name>:
  -  <VSF Name1>:
        behavior: <callback name>
        parameters:
           parameter<i>: val
  -  <VSF Name2>:
        behavior: <callback name>
        parameters:
           parameter<j>: [val1, val2, val3]
```

Figure 3: Structure of a policy reconfiguration message.

The control delegation capabilities through the virtualization of control functions offered by FlexRAN follow the NFV principles and indeed bring runtime service function chaining capabilities to the RAN by adding a virtualization layer over the RAN infrastructure and allowing the flexible placement of RAN control functions closer or further away from the base station based on the networking conditions, the available computing resources and the requirements of the operator in terms of performance. However, it should be noted that such capabilities relevant to the RAN are yet to be considered by NFV standards specifications, like that of ETSI NFV.

An important issue when considering control delegation via the above mechanisms is concerned with the security implications that might arise from pushing new VSFs to the agent which modify the behavior of the eNodeB. One way to deal with this is to force the agent to accept only code that is signed from a trusted authority, similarly to how third-party device drivers for an operating system need to be verified before installation. Since FlexRAN targets a critical part of the mobile infrastructure, getting certification from trusted authorities is expected to become a common practice for developers of VSFs; one could also envision an online VSF store similar to mobile app stores. Another measure to limit the effects of unwanted behavior from the VSFs is to allow the control modules of the agent to run in a sandboxed mode, where each VSF will need to ask for permissions to use the various parts of the FlexRAN agent API, similarly to how Android apps request the permission of the user to access different services of their device. In this way, the network operator could quickly identify VSFs that present an unexpected behavior.

**eNodeB Report and Event Management.** One of the responsibilities of the FlexRAN agent is to provide statistics reports and event notifications to the master for the various control modules (e.g., transmission queue sizes and random access attempts for the MAC module, radio bearer statistics and reference signal received power measurements for the RRC module). The master can use the FlexRAN protocol to make asynchronous requests for such reports and notifications and the FlexRAN agent is responsible to register these requests and to notify the master through the Reports & Events Manager (illustrated in Fig. 2) once the results are available.

FlexRAN supports three types of reports: *one-off*, *periodic* and *triggered*. A *one-off* report is sent by the agent to the master only once as a reply to its initial request, while a *periodic* report is sent at fixed intervals that are designated in the initial request sent by the master, using the TTI as a time reference for the length of the interval. A *triggered* report is sent by the agent aperiodically and only when there is a change in the contents of the requested report. Similarly, the master can choose whether or not to be notified for a specific event occurring at the eNodeB by registering for it at the agent using the FlexRAN protocol. These statistics reports offer high level of flexibility in tuning the level of interaction between the agents and the master, thereby the degree of coordination. For example, triggered reports of MAC statistics is required for a remote scheduler deployed at the master. However, if the scheduling application of the controller is intended to be used as a non-real time application at a more coarse-grained timescale as a hypervisor of a local agent-side scheduler, the master can only request periodic or even one-off reports.

**Extending OpenAirInterface with FlexRAN.** We implement the FlexRAN platform over OpenAirInterface (OAI) [31]. OAI to our knowledge is the most complete open-source LTE software implementation. As such, it provides the right base to realize FlexRAN. Recall that the focus of FlexRAN is to achieve software based control of a mobile RAN that is totally decoupled from the data plane. Since we use LTE as the concrete setting for our implementation, it essentially involves using the implementation of FlexRAN as the LTE RAN control plane over the data plane implementation that is retained from the base OAI. To achieve this, we had to bypass the control plane of OAI, which does not clearly separate control and data planes, and then interface it with FlexRAN via the newly defined FlexRAN Agent API. This clean separation offered by FlexRAN simplifies the development of control applications, which now appear as modules completely separated from the data plane (based on OAI in the current implementation); this is not possible with the standard LTE RAN and therefore even with vanilla OAI. Although in our implementation the FlexRAN Agent resides over OAI eNodeB data plane, it is important to note that FlexRAN as a whole (including the FlexRAN Agent) is a separate entity allowing the agent to be realized even on a physically different machine.

All the FlexRAN Agent Management Modules (Fig. 2) and the FlexRAN Agent API were implemented from scratch, creating a basic agent stub that could be enriched with functionality through the implementation of the FlexRAN Control Modules identified earlier in this section. For our prototype the focus was on the RLC/MAC control module, due to the significant challenges that it presents in terms of its stringent time constraints. For this module, the proper CMI was identified and the corresponding scheduling VSFs were implemented. Even though the functionality related to these VSFs was already present in OAI, the involved control and data plane operations were tightly coupled, which was counter to the intention behind FlexRAN. To overcome this, the OAI eNodeB source code was refactored to allow the separation of the eNodeB data plane from the control logic as per the design of FlexRAN and all the required function calls were added to the FlexRAN Agent API to support this separation. For example, function calls were added to the API to obtain data plane MAC/RLC related information like transmission queue sizes of UEs, along with calls for applying scheduling decisions. To better highlight the effort required for this prototype implementation, more than 10000 lines of code were written for the agent management modules and Agent API and more than 6000 lines of original OAI code were refactored to support the control and data plane separation.

The FlexRAN agent API, as well as the CMIs are written in C, while the whole eNodeB/agent implementation currently supports x64 and ARM Linux systems. As a consequence, any VSF written for an OAI FlexRAN agent currently needs to be implemented in C and compiled against this architecture. The agent is multi-threaded with one thread responsible for the management of reports and events, one for dispatching the incoming protocol messages to the control modules and calling the corresponding VSFs, and two for the asynchronous network communication with the master controller.

### *4.3.2 FlexRAN Protocol*

The abstract communication channel is another feature of the FlexRAN agent ( Fig. 2) for the interactions of the agent with the master. The main FlexRAN agent components communicate and exchange protocol messages with the master through an asynchronous interface that abstracts the communication operations. The communication channel implementation can vary (socket-based, pub/sub etc) depending on the master that the agent is interfacing with. It should be noted that the interface between the master and the agent needs to be asynchronous as the agent may need to perform certain operations in specific time intervals (e.g., a scheduling decision per TTI) while protocol messages from the master can arrive in asynchronously (e.g., receiving a request for a measurement report). In the current implementation, TCP is used for the communication of the agents with the master and the exchange of protocol messages. The protocol messages are implemented using Google Protocol Buffers [1], which provides an optimized platform-neutral serialization mechanism and allows the expression of protocol messages in a language-agnostic manner. Detailed specification of the protocol messages are omitted due to space constraints.

### *4.3.3 FlexRAN Master Controller*

The master controller (Fig. 4) constitutes the brain of the FlexRAN control plane as it manages the operation of the FlexRAN agents. In FlexRAN, we employ a custom design for the master instead of using a conventional OpenFlow-based one and this is for two reasons: (1) the nature of control in the RAN is tied to a significant extent to the control of radio resources which cannot be effectively captured by the flow abstraction; (2) the RAN presents a requirement for the support of real-time applications with very quick reaction times, a feature not essential for SDN control in the wired domain.

**Management of network information.** The RAN Information Base (RIB) is a key component that maintains all the statistics and configuration related information about the underlying network entities, i.e. UEs, eNodeBs and FlexRAN agents. The RIB is always loaded in memory for improved performance and is structured as a forest graph. The root node of each tree in this forest is an agent associated with the master, while the nodes of the second level are the cells associated with a specific agent/eNodeB. Finally, the leaves of the trees are the UEs associated to a specific (primary) cell. It should be noted that the current implementation of the RIB does not provide any high-level abstraction for the stored information, revealing raw data to the northbound API.

**Support for real-time applications.** The applications as well as the Events Notification Service of the master consult the RIB to perform any operation, but they do not modify the RIB directly. Instead they issue control commands through the northbound interface, which indirectly affect the RIB state through the modifications performed to the eNodeB data plane and the agent state. These modifications are reflected back to the master through the statistics reports and event notifications sent by the agents. Only the RIB Updater component of the master can update the RIB with the information received from the agents (Fig. 5). This design decision improves the support of real-time applications (e.g., MAC scheduler) that must be non-blocking in order to meet their time constraints (e.g., a TTI for the scheduler). Allowing all components to modify the RIB could give rise to write conflicts, negatively affecting the system performance. Having just a single writer (the RIB Updater) and multiple readers helps avoid this problem. The update frequency of the RIB depends on the way the FlexRAN agent reporting and event notification mechanism is configured by the master.

Since the FlexRAN controller is designed to support real-time RAN applications, a Task Manager is responsible for handling all the tasks (both applications and core management modules) running on the master. More specifically, it is responsible to start, stop and pause applications; to assign priorities to running services and to allow them to execute based on their time constraints. For example, the Task Manager would assign a very high priority to a centralized MAC scheduler running on the master, whereas a non time-critical monitoring application would get a lower priority. To support real-time control, the Task Manager is implemented as a non-preemptive thread running in an infinite loop and operating in cycles of length equal to a TTI, where each cycle is composed of two slots — one for the execution of the RIB Updater (e.g., 20% of the TTI) and the other for the execution of the applications as well as the Event Notification Service threads (e.g., 80% of the TTI). This guarantees the mutual exclusion of the reads and writes in the RIB and allows the applications to operate in a non-blocking mode while accessing the RIB, ensuring the real-time operation of the controller.

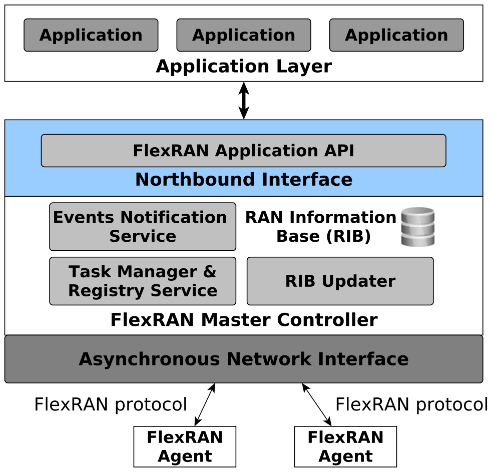


Figure 4: Components of the FlexRAN master controller and its interface to the application layer.

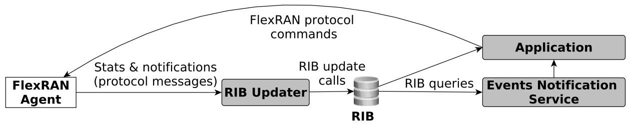

FlexRAN protocol commands
Application
Stats & notifications (protocol messages)
RIB update calls
FlexRAN Agent
RIB Updater
RIB queries
Events Notification Service
RIB

Figure 5: Flow of information for updating the RIB.

**Master controller implementation.** The master controller is implemented from scratch using C++ and currently supports x64 Linux systems (kernel >= 3.14 for support of real-time applications). The master can operate in a non real-time mode, supporting only applications that are not time-critical, with the advantage of being more lightweight. The Task Manager in the non-real time mode does not enforce a strict duration of the cycle as tasks are not scheduled with a real-time priority and thus could take longer than a TTI.

## 4.4 Northbound API and Applications

RAN control and management applications in the application layer communicate with the master through the northbound interface (Fig. 4), which allows the applications to monitor the infrastructure through the information obtained from the RIB and apply their control decisions through the agent control modules. The applications run as threads and use the FlexRAN Application API (currently in C++) to register with the Registry Service of the master, access the RIB and send control messages to the agents. Applications can be broadly divided into two categories: (*periodic* or *event-based*). Periodic applications employ a periodic execution pattern (e.g,, a periodic scheduler) whereas the execution of event-based applications is triggered by specific events (e.g., a mobility manager that expects changes in the received signal strength of a UE to react). The Events Notifications Service of the master controller notifies the applications (mainly of the event-based type) about any changes that might have occurred on the agent side. Some applications could fall into both categories and it is ultimately the application developer who would choose the appropriate execution pattern.

## 5. SYSTEM EVALUATION

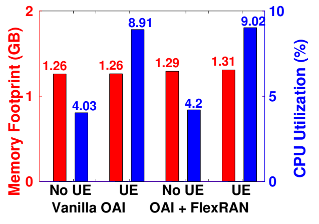
In this section we evaluate the engineering decisions and the design choices behind FlexRAN. For the experiments, a FlexRAN master controller was deployed, in which one or more agent-enabled eNodeBs (depending on the experiment) were connected through dedicated Gigabit Ethernet connections. Each eNodeB was also connected to a machine acting as the EPC, running the corresponding EPC software implementation (openair-cn [2]). All the test machines were equipped with quad-core Xeon CPUs at 3.4GHz and 16GB of RAM. Depending on the experiment, the testbed was used either in emulation mode (emulated PHY layer and emulated UEs) or with a real RF front-end (Ettus B210 USRP) and a COTS UE (Nexus 5 smartphone). All the experiments were conducted with the same eNodeB configuration, namely FDD with transmission mode 1 and 10MHz bandwidth in band 5.

### 5.1 Comparison to Vanilla OAI

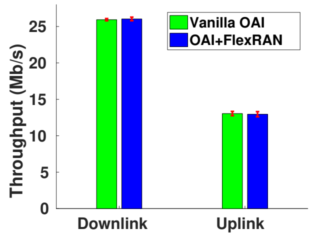
We begin by investigating the overhead introduced to an eNodeB by the addition of the FlexRAN agent in terms of memory and CPU utilization (Fig. 6a) comparing a vanilla OAI eNodeB and a FlexRAN-capable eNodeB in two cases; one with the system in idle state and one with a COTS UE connected and performing a speedtest. As we can observe, there is a very slight increase in the memory footprint and the CPU utilization in the FlexRAN case, due to the threads used for the operation of the agent and the protocol messages exchanged with the FlexRAN master controller. Despite the aforementioned overhead incurred from the FlexRAN agent, the communication of the eNodeB with the UE is fully transparent, with the UE experiencing the same service quality in its connection as with the vanilla OAI (Fig. 6b).

### 5.2 Scalability

Next, we evaluate how FlexRAN scales as the number of eNodeBs and UEs increases. For these experiments we used OAI in emulation mode with the PHY layer abstracted in order to perform tests with a large number of UEs. It is noted that this choice has a minimum effect in the obtained results,



(a) Normalized CPU utilization (8-processors) and memory footprint with(out ) UE



(b) Downlink and Uplink throughput comparison

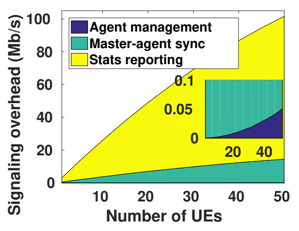
Figure 6: Comparison of Vanilla OAI to FlexRAN .

since the focus of our evaluation was in operations occurring above the PHY which were unaffected by the emulation.

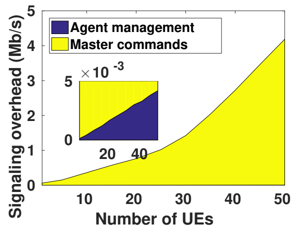
#### 5.2.1 *Controller-agent signaling overhead*

One very important element regarding the scalability of FlexRAN is the network overhead incurred by the FlexRAN protocol, especially when support for real-time applications is required. For this, we measured the signaling overhead between the agent and the master in the demanding case of deploying a centralized scheduling application and for a varying number of UEs. To study the system's scalability, we tested a scenario with the worst case configuration signaling-wise, where statistics reports were sent from the agent to the master every TTI, the centralized scheduler undertook all scheduling decisions at a TTI granularity and the master was fully synchronized with the agent at a TTI level using the appropriate FlexRAN protocol synchronization messages. During the experiment uniform downlink UDP traffic was generated for all the UEs, in order to force the centralized scheduler to frequently send scheduling decisions to the agent.

The agent-to-master network overhead for 50 UEs can reach 100 Mb/s (Fig. 7a). The main source of this overhead are the periodic statistics reports (buffer status reports, CQI measurements etc.) followed by the master-agent synchronization messages, with the overhead of the agent management related messages being negligible. One important thing to notice is that the agent-to-master signaling overhead increases sublinearly with the number of connected UEs due to the aggregation of relevant information in the FlexRAN protocol messages (e.g. list of UE status reports) and their optimized serialization by the Google Protocol Buffers library.



(a) Agent-to-master



(b) Master-to-agent

Figure 7: Signaling overhead for the communication between the master and the agent using the FlexRAN protocol.

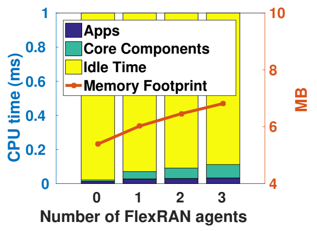


Figure 8: Utilization of master TTI cycle and memory footprint (16 UEs/eNodeB).

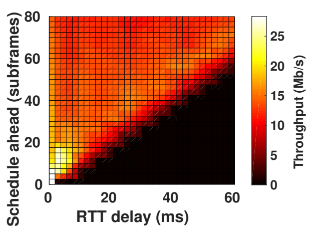


Figure 9: Effect of latency and scheduling ahead time on downlink throughput.

In the case of the master-to-agent signaling (Fig. 7b), the overhead is much lower compared to the previous case (less than 4Mb/s) and comes almost entirely from the scheduling decisions sent by the centralized scheduler. In contrast to the previous case, this overhead is increasing with a higher rate as the number of UEs goes up. The reason is that the larger the number of UEs, the less resources are available for scheduling each UE and therefore more TTIs will be required for scheduling, leading to an increase in the overall number of scheduling decisions sent by the controller to the agent.

The aforementioned results show that FlexRAN is suitable even in demanding scenarios like multi-cell scheduling on a per-TTI basis, where depending on the deployment (macro or small-cell) the agent can be connected to the master either through a dedicated high-bandwidth channel (e.g. optical fibers) or through a lower bandwidth channel (e.g. a VDSL connection). In practice, the controller will apply policies or delegate the scheduling to the agent, which will significantly reduce the network overhead. This overhead could be further reduced through agent configuration changes. For example, by setting the periodicity of the MAC reports to 2 TTIs, this overhead could be reduced to almost half without any significant impact in the system's performance. Other methods that could be considered to reduce this overhead could be compression algorithms for the protocol messages or event-triggered instead of periodic message transmissions.

### 5.2.2 *Master controller resources*

Using the previous setup, we measured (Fig. 8) the requirements of the master in computing resources and memory for a varying number of connected agents (16 UEs/eNodeB). As already discussed, the master operates in TTI cycles, where part of each cycle is allocated to the execution of applications (80% in this experiment) and the rest to the execution of the core components of the master. As we can observe, the operation of the master is lightweight, with only a small fraction of the total cycle being actually utilized. The execution time of the core components increases as we add more agents, mainly due to the increase in the updates that need to be performed by the RIB updater. The memory footprint of the master is also very small and its increase is mainly related to the increase of the RIB size.

## 5.3 Control channel latency impact

One very important aspect for real-time control in FlexRAN is the impact of the control channel latency between the master and the agent. To test its effect we used a COTS UE scheduled in the downlink by a centralized scheduling application running at the master. The scheduler was implemented so that it could be parametrized to make scheduling decisions $n$ subframes ahead of time, meaning that the scheduler would observe the MAC state (transmission queue sizes of UEs, signal quality etc) at subframe $x$ and would issue a scheduling decision that should be applied by the data plane in subframe $x + n$. The scheduling decision will be valid and can be applied by the agent only if its designated time is greater than the latency in the master-agent control channel.

Based on this setup we modified the schedule ahead parameter of the application and the latency in the control channel using the *netem* tool [3] and measured the UE downlink throughput for various configurations (Fig. 9). As we can observe, the lower triangular region of the figure depicts a throughput of 0, indicating that for these configurations the UE was unable to complete network attachment. This is due to the one-way delay of the control channel being greater than the schedule ahead time, meaning that scheduling decisions always miss their deadline. Moreover, the control channel delay affects the synchronization between the master and the agent, since the agent subframe reported to the master is always outdated by an offset equal to half the RTT delay. Since the scheduler relies to this outdated value to make a scheduling decision, the scheduling ahead time should take it into account. Assuming a symmetrical RTT delay, the schedule ahead time should be set to a value that is at least equal to it; half to make up for the outdated subframe value reported by the agent and half to ensure that the scheduling decision will be valid at the time it arrives to the agent.

In the upper triangular region we have all the cases in which the controller application successfully manages to schedule the UE. In this case the application is able to schedule the UE even for a control channel with a very high latency, as long as the schedule ahead parameter is configured properly. As the RTT delay and the schedule ahead time increases, the throughput gradually drops. One reason for this is that higher RTT delays make the information stored in the RIB (e.g. CQI measurements) more outdated, leading to wrong scheduling decisions (e.g. due to a bad modulation and coding scheme choice) that could affect the throughput. Moreover, for increased values of the schedule ahead parameter, the scheduler needs to make predictions further into the future, while making assumptions about the outcome of previous transmissions for which it has not yet received any feedback. Depending on the use case and on the scheduling performance requirements, one could choose to either use approximation methods like scheduling ahead of time to mitigate the effect of latency or to delegate control to the agents for the time critical functions that are affected by this latency as long as coordinated operation among the eNodeBs is not a hard constraint.

## 5.4 Control delegation performance

Since control delegation is one of the most important features of FlexRAN we evaluated the mechanisms of VSF updation and policy reconfiguration in terms of efficiency and service continuity. For this experiment we used the same setup as in Section 5.3, starting with a centralized scheduler running as an application at the master. At the same time, we built an equivalent (in terms of functionality) local scheduler, as a VSF for the MAC control module of the agent, using the FlexRAN agent API. The experimental scenario involved the pushing of this code to the agent using the FlexRAN protocol and the dynamic switching between the local and the remote scheduler through the policy reconfiguration mechanism.

Using this setup, we tested the downlink throughput of the attached UE, while swapping the local and the remote schedulers with various frequencies down to the TTI level (1ms), and observed the same application performance of 25 Mb/s (figure not included for brevity). The code is pushed to the agent-side only once and is stored in its local cache meaning that no additional overhead is incurred for the control delegation. Moreover, the absolute VSF load time required to swap between the local and remote scheduler is very small (~103ns) and therefore does not disrupt service continuity as it only forms an insignificant fraction of the overall TTI.

## 6. FlexRAN USE CASES

Till date, the lack of an implemented SD-RAN platform meant that there was no way to actually study SD-RAN benefits and use cases. To demonstrate the usefulness of FlexRAN towards this end, we now present diverse use cases that SD-RANs in general and FlexRAN in particular can enable.

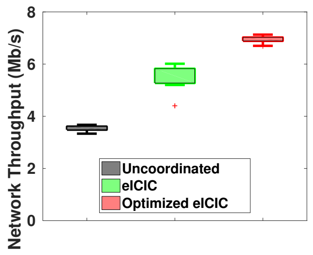
### 6.1 Interference Management

One of the ways to cope with the requirement for higher throughput in the RAN is the creation of Heterogeneous Networks (HetNets) composed of multiple small cells within the area of a macro cell. However, such dense network deployments are more susceptible to interference. One proposed solution is the enhanced Inter-Cell Interference Coordination (eICIC) [21] that introduces the concept of Almost-Blank Subframes (ABS) during which the macro cells are muted to allow small-cells to transmit user traffic. Even though eICIC is one effective way to manage interference, the semi-static configuration of ABSs can lead to an underutilization of the radio resources when small cells are idle. To remedy this, we consider an optimized eICIC mechanism that allows the macro-cell to exploit periods of inactivity of the small-cells to transmit to UEs even during ABSs as long as no small-cell is transmitting at the same time. Such a mechanism requires a high-level of coordination among cells which cannot be easily achieved using the traditional X2 interface [18] [11]. To deal with this, we build a downlink UE scheduling application over FlexRAN that exploits its consolidated control plane to implement this optimized eICIC mechanism.

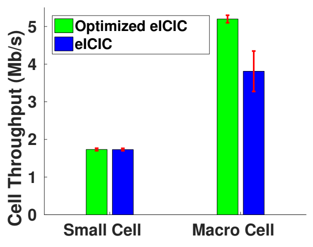
Specifically, we implemented a centralized scheduler application on top of the master and two different types of local agent-side downlink schedulers, one for the macro-cell and one for the small-cells of a region. During a non-ABS, the macro-cell eNodeB performs the scheduling using its agent-side scheduler, while the agent-side schedulers of the small-cells remain inactive exactly as in a normal eICIC case. However, during an ABS the centralized scheduler at the master performs a coordinated UE scheduling of all cells and decides whether the macro or the small cells should be scheduled, always giving priority to the small-cells. The scheduling decisions are then pushed to the agents using the FlexRAN protocol and are applied by the agent-side schedulers that during an ABS act as stubs of the centralized scheduler.

Two agent-enabled OAI eNodeBs were used, one acting as a macro-cell and one as the small-cell, both running in emulation mode over the same physical machine. The reason we resorted to emulation over the same machine is that eICIC requires a microsecond-level synchronization of eNodeBs, which was not supported by our hardware. Moreover, due to a limitation of the current OAI implementation, the association of UEs over different eNodeBs is not supported in emulation mode when the PHY is abstracted. This forced us to use the more computationally intensive full-PHY emulation mode of OAI (that involves convolution of the real PHY signal with an emulated-channel in real time) and to limit the number of emulated UEs used for this experiment to 4.

One UE was associated with the small-cell and three were associated with the macro-cell. Downlink UDP traffic was generated uniformly for all the UEs and the overall network throughput was measured (Fig. 10a) in three cases: (i) an uncoordinated case, where each eNodeB performed scheduling independently, (ii) a simple eICIC use case with 4 ABSs and (iii) the optimized eICIC use case with the same ABS configuration as case (ii). The optimized eICIC use case almost doubled the overall network throughput over the uncoordinated scenario and had an improvement of about 22% over the simple eICIC scenario. The reason for the difference between the two eICIC use cases is that while the small-cell throughput remains the same (Fig. 10b), the throughput of the macro-cell increases under optimized eICIC because the ABSs not used by the small-cell are assigned by the centralized scheduling application to the macro-cell.



(a) Downlink network throughput in uncoordinated, simple and optimized eICIC.



(b) Overall macro and small cell downlink throughput for simple and optimized eICIC.

Figure 10: Throughput benefits of optimized eICIC.

### 6.2 Mobile Edge Computing

For our second use case, we consider FlexRAN as a deployment platform for Mobile Edge Computing (MEC) applications. MEC allows developers and content providers to deploy their services over the network edge. This presents many benefits including ultra-low latency, high bandwidth and real-time access to radio network information which can be used by applications for optimization purposes [23]. Such applications are expected to be deployed in a centralized manner and therefore doing this using the conventional LTE architecture becomes a challenging task. Moreover, their deployment assumes the existence of a programmable network, which is naturally enabled by FlexRAN.

In this context, we show how the consolidated control plane and the real-time network information provided by FlexRAN can be beneficial for services such as video streaming. To show this, we used the DASH [34] streaming service and studied the effects of fluctuating a mobile's signal strength to the video streaming bitrate adaptation performed by the DASH reference client [4]. The idea of the experiment is that the channel quality reported by the UEs to the eNodeB through a Channel Quality Indicator (CQI) value (in the range [0,15]) indicates the modulation and coding scheme that the scheduler should use for the data transmissions of a UE and therefore has a direct impact in its highest achievable throughput. Knowing this can allow the streaming service to make smarter decisions regarding the optimal bitrate compared to a case when only transport layer information is available.

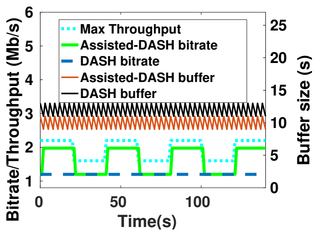
In order to obtain reproducible results, we emulated the fluctuations of the channel quality between the eNodeB and the UE and measured the maximum achievable TCP throughput of a COTS UE for various fixed CQI values. Moreover, for the same CQI values, we used the reference DASH client and the available test videos that it offered, to measure the maximum sustainable bitrate of a video stream (i.e. a bitrate that would never lead to buffer freezes) for the offered bitrate levels. The measurement results (Table 2) indicate that the TCP throughput needs to be greater (even double) than the video bitrate in order to always maintain a high quality; this is consistent with related observations in the literature [37].

| CQI | TCP Throughput (Mb/s) | Max sustainable bitrate (Mb/s) |
|---|---|---|
| 2 | 1.63 | 1.4 |
| 3 | 2.2 | 2 |
| 4 | 3.3 | 2.9 |
| 10 | 15 | 7.3 |

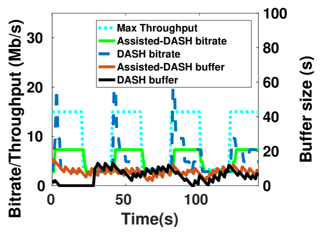
Table 2: Measurements of max TCP throughput and max sustainable bitrate of video stream for various CQI levels.

We used FlexRAN to implement a simple MEC application that uses the RIB to obtain real-time information about the CQI values of the attached UEs. The application computes an exponential moving average of the UE CQI and maps it to the optimal video bitrate based on the measurements of Table 2. The bitrate is then forwarded through an out-of-band channel to a modified version of the DASH reference client where it is used to adapt the video stream's quality. To simplify things, we performed the experiment in an ideal setting, where the channel quality fluctuation was the only cause for a change in the bitrate (a single UE attached to the network with the DASH video being the only source of traffic).

For our experiment, we considered two cases. In the first, we used a video [5] with three bitrates (1.2, 2 and 4 Mbps) and we introduced a small variation in the CQI value (from 3 to 2 and vice versa). In the second, we used a 4K video [6] with six available bitrates (2.9, 4.9, 7.3, 9.6, 14.6 and 19.6 Mbps) in which the CQI value was changed drastically (from 10 to 4 and vice versa). In both cases, we measured the bitrate selected by the default and the FlexRAN-assisted players in comparison to the maximum achievable TCP throughput, as well as their buffer sizes (Fig. 11). In the first case (Fig. 11a), the default player always kept the bitrate at the lowest value (1.2 Mb/s), even in times when the available throughput increased by 40%, meaning that the change in channel quality did not become apparent to the transport layer. On the other hand, the FlexRAN-assisted player exploited the information obtained by the RAN and managed to better adapt to the changing network conditions. Therefore, even though neither player experienced buffer freezes, the default player underutilized the available resources. In the second case (Fig. 11b), the default player aggressively attempts to increase the bitrate when the CQI increases, setting it to 19.6 Mb/s, even though the maximum achievable throughput is 15 Mb/s. This quickly results in TCP congestion, leading the player to significantly lower the bitrate (lower than the max sustainable bitrate) in order to adapt. At the same time this leads to frequent and sometimes long buffer freezes (e.g. sec 5-30). On the other hand, the MEC application running over FlexRAN can quickly identify the maximum sustainable bitrate (7.3 Mb/s) given the CQI measurements observed at the RIB and does not follow the same aggressive behavior as the default player, leading to a more stable and overall higher video quality.



(a) Low throughput variability



(b) High throughput variability

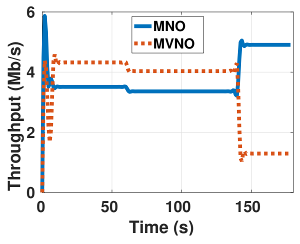
Figure 11: Rate adaptation of DASH vs FlexRAN assisted DASH and corresponding buffer sizes.

## 6.3 RAN Sharing & Virtualization

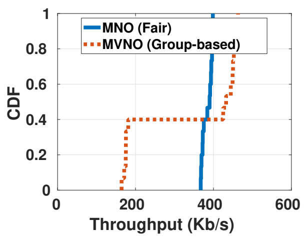
A side effect from the densification of cells is the increase of the infrastructural CAPEX and OPEX. This leads to the creation of new business models, where multiple Mobile Network Operators (MNOs) share the same *passive* infrastructure such as masts and backhaul links in order to save costs. On top of that, a second level of *active* sharing can happen, where MNOs share the network equipment as well as provide wholesale access to Mobile Virtual Network Operators (MVNOs), allowing them to provide voice and data services using part of the available resources [19]. However, this can pose a significant challenge for the management of the RAN, since the requirements of operators in terms of the radio resources and the applied policies of scheduling and mobility management can constantly change based on the needs of their subscribers and the underlying setting. The control and management of such operations can be greatly simplified through the introduction of programmability in the RAN.

Based on this, we used FlexRAN as an enabler of active RAN sharing and on-demand resource allocation over an LTE network. More specifically, exploiting the virtualization capabilities of FlexRAN , we implemented a downlink UE scheduler for the agent-side that supports the dynamic introduction of new MVNOs to the RAN and the on-demand modification of the scheduling policy per operator. An application running at the master exploits the policy reconfiguration mechanism of FlexRAN to modify the parameters of the agent-side scheduler (scheduling policy and number of resource blocks per MVNO). To test this and in order to be able to support a large number of UEs, we used a single agent-enabled OAI eNodeB in emulation mode with the PHY abstraction enabled. We configured the agent-based scheduler to support one MVNO that shared the available resources with the RAN's MNO.

For our first experiment, each operator was assigned 5 UEs for which uniform UDP downlink traffic was generated, while the percentage of radio resource blocks allocated to each operator was dynamically adjusted based on their requirements. The results in terms of the total throughput per operator are illustrated in Fig. 12a. Initially, the MNO was allocated 70% of the radio resources and the MVNO was allocated the remaining 30%. At 10s, the master controller application sent a policy reconfiguration message, setting the available resources of the MNO to 40% and of the MVNO to 60%, simulating a brief requirement for additional resources for the MVNO. Then, at 140s, the application sent a second policy reconfiguration message that re-adjusted the radio resources so that 80% were allocated to the MNO.



(a) Dynamic allocation of resources



(b) CDF of UE throughput based on scheduling type

Figure 12: Policy reconfiguration for MVNO management

As a second experiment, we implemented an agent-side scheduler supporting a fair scheduling policy and a group-based policy of premium and secondary users, with 70% of the resources allocated to the premium users and the rest to the secondary. One MNO and one MVNO was employed, where the MNO was assigned the fair policy and the MVNO the group-based one. Each operator was allocated half of the available radio resources and was assigned 15 UEs. In he group-based MVNO case, nine UEs belonged to the premium group and the remaining six acted as secondary users. We generated uniform UDP downlink traffic for all the UEs and measured the throughput of each UE per operator. The results are illustrated in the CDFs of Fig. 12b. In the case of the MNO, all the UEs had a throughput of about 380Kb/s due to their fair scheduling policy. On the other hand, in the case of the MVNO, UEs assigned to the premium group had a throughout of about 450Kb/s, while the UEs of the secondary group achieved a throughput of less than 200Kb/s.

## 7. DISCUSSION

### 7.1 Other Example Use Cases

Here we make a brief discussion on a non-exhaustive list of further use cases that were currently not implementable due to the limitations posed either by the OAI platform underlying FlexRAN or by the current LTE specifications.

**Mobility Management.** In current practice, handover decisions mainly rely on the signal strength of mobile devices. However, the centralized network view offered by FlexRAN could enable more sophisticated mobility management mechanisms that consider additional factors, e.g., the load of cells or use-driven resource requirements of mobile devices, leading to an optimization of the device-network associations.
**Network Slicing.** Future 5G networks are envisioned to support a wide range of vertical segments with a diverse set of performance and service requirements through the Slicing of the physical network into multiple isolated logical networks [30] [7]. Among the key features required to achieve network slicing is the existence of an end-to-end management and orchestration framework (e.g. [35]) and the means to virtualize the network resources. FlexRAN can be seen as a key enabler of virtualization in the RAN, providing a simple and flexible platform for the dynamic control and allocation of radio resources based on the needs of the deployed services.
**Device Centric Networking.** Another interesting application of FlexRAN could be in the domain of device-centric networking [16]. In this, the association of mobile devices is decoupled from the cell, e.g. by using different cells for control and data traffic, simplifying the adoption of new communication paradigms like D2D. While realizing such a paradigm would also require changes in the physical layer, its deployment could greatly be simplified through the centralized control of FlexRAN, as it inherently involves the coordinated control of multiple network nodes, e.g. control traffic through one cell and data traffic through another.
**Spectrum Sharing.** Shared spectrum access is seen as a promising solution that allows operators to cope with the high increase of mobile data traffic. One spectrum sharing mechanism that is of particular interest to MNOs is Licensed Shared Access (LSA) [26]. It enables incumbents not concerned with civilian wireless and mobile data communications to authorize other users (e.g. the MNOs) to access all or part of the spectrum allocated to them for designated periods of time and in designated places based on some agreement. An LSA controller dynamically manages the access to the shared spectrum based on these agreements. Such an operation could easily be implemented as an application on top of FlexRAN.

### 7.2 Adaptability Beyond LTE

As already discussed in Section 3, FlexRAN was presented in the context of LTE for concreteness and to match its current implementation. However, the design and the mechanisms supported by FlexRAN are not LTE-specific and are therefore equally suitable for future mobile RAN architectures. More specifically, the mechanisms of virtualized control functions and control delegation are technology-agnostic and use API calls and protocol messages that are completely decoupled from the underlying technology. The main difference that these mechanisms would present in the context of another technology is that the number and type of the control modules and VSFs on the agent side would change to reflect the capabilities and needs of the new technology (e.g. no PDCP module for WiFi). However, the mechanisms of policy reconfiguration and VSF updation make no assumption about the exact type and structure of the control modules at the agent side, allowing the controller to designate it on the fly based on the underlying implementation (see Fig. 3).

Apart from the technology agnostic part of the agent API, FlexRAN also requires a number of technology specific API calls. For example, LTE requires scheduling commands which are not applicable and therefore are not required in the WiFi domain. This means that for FlexRAN to be fully compliant with other technologies, the FlexRAN Agent API needs to be extended with additional function calls specifically tailored for the domain of interest, closely resembling the idea of device drivers found in operating systems. The more extended the API is for a specific technology, the more capabilities are offered for the control of the underlying radio access technology (RAT). It should also be noted that the FlexRAN protocol has been structured in such a way that it could be easily extended to support new messages that are technology specific without affecting the functionality of the existing ones.

Another important issue is the extensibility of FlexRAN in future 5G fronthaul architectures where base stations are expected to adopt a distributed design, with Remote Radio Units (RRUs) decoupled from the Baseband Unit (BBU) and connected through a fronthaul link (e.g. optical fibers) [18]. In such an architecture, the functional split between the RRU and the BBU is expected to play a significant role, depending on the capacity and the latency of the fronthaul link. The design of FlexRAN could be adopted in such an architecture, where the data plane and the agents could be placed on the RRU side and the control plane could be consolidated at the master residing at the BBU. Based on the quality of the fronthaul link, the control delegation features of FlexRAN could be exploited to perform a dynamic and adaptive functional split of the control operations while retaing the realtime deadlines. For example, in case of a high latency fronthaul link, the master could delegate control of the time critical PHY and MAC operations to the RRU, while in case of a low latency link all operations could be residing at the BBU side.

## 7.3 Issues for Future Work

Here we outline several extensions and aspects for future work on FlexRAN. Firstly, a conflict resolution mechanism ensuring state consistency becomes valuable when third-party network applications need to be supported. For example, such a mechanism should prohibit the deployment of multiple applications that may simultaneously issue scheduling decisions for the same resource blocks, effectively leading to conflicts. Secondly, a significant issue is related to control delegation, where the VSF code pushed to the agents by the controller needs to be compiled against the processor architecture of the target agent using the programming language of the FlexRAN agent API (currently in C). Introducing a high-level domain-specific language that would make the development of VSFs technology-agnostic would greatly simplify this process, especially in cases where the underlying infrastructure is heterogeneous in terms of the agent implementation. Another issue that needs to be further considered is related to the security concerns of the VSF updation mechanism, which could cause complications for the deployment of third-party applications (see Sec. 4.3.1). Furthermore, as already discussed, FlexRAN does not currently employ any high-level abstractions in the northbound API and instead reveals raw information. Therefore, introducing abstractions for control and for RIB access could greatly simplify the development of control and management applications. Finally, improving the scalability of FlexRAN for wide area settings by introducing another layer of control and broadening its scope to go beyond the control and management of the radio resources in the cellular RAN by considering other domains like the core network and multi-RAT settings would provide a more holistic SDN solution for future mobile networks.

## 8. CONCLUSIONS

In this paper we have presented FlexRAN, a flexible and programmable SD-RAN platform. FlexRAN enables the separation of the control from the data plane through a custom-tailored southbound API, while providing inherent support for real-time RAN applications. FlexRAN offers significant benefits to the RAN, including the flexibility to dynamically modify the degree of coordination among base stations to realize both distributed and centralized modes of operation and the programmability to adapt control over time turning the RAN into an evolvable network. All these while being transparent to end-devices, simplifying its deployment and promoting innovation both for the industrial and the academic research community. The implementation of FlexRAN over the OpenAirInterface LTE platform and our evaluation results confirm the feasibility of its deployment even when considering time critical operations like MAC scheduling. The effectiveness of FlexRAN as an SD-RAN platform was highlighted through a diverse set of use cases in the context of current 4G and future 5G networks, focusing on interference management, mobile edge computing and RAN sharing.

## Acknowledgments

We thank our shepherd Wenjun Hu and the anonymous reviewers for their helpful suggestions on improving the paper. This work was partly funded by the European FP7 FLEX project (612050) and H2020 COHERENT project (671639).

## 9. REFERENCES

[1] https://developers.google.com/protocol-buffers/.

[2] https://gitlab.eurecom.fr/oai/openair-cn.

[3] http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[4] http://dashif.org/reference/players/javascript/v2.1.1/samples/dash-if-reference-player/index.html.

[5] http://dash.edgesuite.net/dash264/TestCases/2a/qualcomm/1/MultiResMPEG2.mpd.

[6] http://dash.edgesuite.net/akamai/streamroot/050714/Spring_4Ktest.mpd.

[7] 5G PPP Architecture Working Group. View on 5G Architecture, 2016.

[8] I. F. Akyildiz et al. SoftAir: A software defined networking architecture for 5G wireless systems. *Computer Networks*, 85:1–18, 2015.

[9] H. Ali-Ahmad et al. CROWD: an SDN approach for DenseNets. In *Second European Workshop on Software Defined Networks (EWSDN)*, pages 25–31. IEEE, 2013.

[10] J. G. Andrews et al. What will 5G be? *IEEE Journal on Selected Areas in Communications*, 32(6):1065–1082, 2014.

[11] A. Apostolaras et al. Evolved User Equipment for Collaborative Wireless Backhauling in Next Generation Cellular Networks. In *12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 408–416. IEEE, 2015.

[12] M. Arslan et al. Software-Defined Networking in Cellular Radio Access Networks: Potential and Challenges. *Communications Magazine, IEEE*, 53(1):150–156, 2015.

[13] A. Banerjee et al. Scaling the LTE Control-Plane for Future Mobile Access. In *Proceedings of the 11th ACM CoNEXT*. ACM, 2015.

[14] M. Bansal et al. OpenRadio: A Programmable Wireless Dataplane. In *Proceedings of the 1st workshop on Hot topics in Software Defined Networks*, pages 109–114. ACM, 2012.

[15] C. Bernardos et al. An Architecture for Software Defined Wireless Networking. *Wireless Communications, IEEE*, 21(3):52–61, 2014.

[16] F. Boccardi et al. Five Disruptive Technology Directions for 5G. *Communications Magazine, IEEE*, 52(2):74–80, 2014.

[17] T. Chen et al. SoftMobile: Control Evolution for Future Heterogeneous Mobile Networks. *Wireless Communications, IEEE*, 21(6):70–78, 2014.

[18] I. Chih-Lin et al. Recent Progress on C-RAN Centralization and Cloudification. *Access, IEEE*, 2:1030–1039, 2014.

[19] X. Costa-Pérez et al. Radio Access Network Virtualization for Future Mobile Carrier Networks. *Communications Magazine, IEEE*, 51(7):27–35, 2013.

[20] S. Costanzo et al. OpeNB: A framework for Virtualizing Base Stations in LTE Networks. In *IEEE International Conference on Communications (ICC)*, pages 3148–3153. IEEE, 2014.

[21] S. Deb et al. Algorithms for Enhanced Inter Cell Interference Coordination (eICIC) in LTE HetNets. *IEEE/ACM Transactions on Networking (TON)*, 22(1):137–150, 2014.

[22] A. Gudipati et al. SoftRAN: Software Defined Radio Access Network. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 25–30. ACM, 2013.

[23] Y. C. Hu et al. Mobile Edge Computing - A Key Technology Towards 5G. *ETSI White Paper*, 11, 2015.

[24] X. Jin et al. SoftCell: Scalable and Flexible Cellular Core Network Architecture. In *Proceedings of the 9th ACM CoNEXT*, pages 163–174. ACM, 2013.

[25] S. Katti and L. E. Li. RadioVisor: A Slicing Plane for Radio Access Networks. In *Open Networking Summit 2014 (ONS 2014)*, 2014.

[26] J. Khun-Jush et al. Licensed shared access as complementary approach to meet spectrum demands: Benefits for next generation cellular systems. In *ETSI Workshop on reconfigurable radio systems*, 2012.

[27] L. E. Li et al. Toward Software-Defined Cellular Networks. In *European Workshop on Software Defined Networking (EWSDN)*, pages 7–12. IEEE, 2012.

[28] T. Mahmoodi and S. Seetharaman. Traffic Jam: Handling the Increasing Volume of Mobile Data Traffic. *Vehicular Technology Magazine, IEEE*, 9(3):56–62, 2014.

[29] M. Moradi et al. SoftMoW: Recursive and Reconfigurable Cellular WAN Architecture. In *Proceedings of the 10th ACM CoNEXT*, pages 377–390. ACM, 2014.

[30] NGMN Alliance. 5G White Paper, 2015.

[31] N. Nikaein et al. OpenAirInterface: A flexible platform for 5G research. *ACM SIGCOMM CCR*, 44(5):33–38, 2014.

[32] K. Pentikousis et al. MobileFlow: Toward Software-Defined Mobile Networks. *Communications Magazine, IEEE*, 51(7):44–53, 2013.

[33] Z. A. Qazi et al. KLEIN: A Minimally Disruptive Design for an Elastic Cellular Core. In *Proceedings of the 2nd ACM SIGCOMM Symposium on Software Defined Networking Research*. ACM, 2016.

[34] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia*, pages 62–67, 2011.

[35] A. Syed and J. Van der Merwe. Proteus: A network service control platform for service evolution in a

mobile software defined infrastructure. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pages 257–270. ACM, 2016.

[36] K. Tsagkaris et al. SON Coordination in a Unified Management Framework. In *77th IEEE Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2013.

[37] B. Wang et al. Multimedia Streaming via TCP: An Analytic Performance Study. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 4(2):16, 2008.

[38] W. Wu et al. PRAN: Programmable Radio Access Networks. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 6. ACM, 2014.

[39] M. Yang et al. OpenRAN: A Software-defined RAN Architecture Via Virtualization. *ACM SIGCOMM Computer Communication Review*, 43(4):549–550, 2013.

[40] V. Yazıcı et al. A new control plane for 5G network architecture with a case study on unified handoff, mobility, and routing management. *Communications Magazine, IEEE*, 52(11):76–85, 2014.