A Distributed Real-Time Programmable Scheduler Architecture For Cellular Networks

Zhouyou Gu
The University of Sydney
zhouyou.gu@sydney.edu.au

Wibowo Hardjawana
The University of Sydney
wibowo.hardjawana@sydney.edu.au

Wenhao Zhang
The University of Sydney
wenhao.zhang@sydney.edu.au

Dawei Tan
The University of Sydney
dawei.tan@sydney.edu.au

Branka Vucetic
The University of Sydney
branka.vucetic@sydney.edu.au

Simon Lumb
Telstra Corporation Ltd.
simon.lumb@team.telstra.com

David McKechnie
Telstra Corporation Ltd.
david.mckechnie@team.telstra.com

Todd Essery
Telstra Corporation Ltd.
todd.essery@team.telstra.com

ABSTRACT

Two types of cellular network scheduler architectures have been proposed in the open literature: 1) a distributed scheduler with non-programmable logic for network operators due to tight control by the evolved NodeB (eNodeB) vendor at the edge base station (BS), and 2) a centralized scheduler with only non-real-time scheduling logic that is programmable. We propose a new distributed real-time programmable scheduler architecture. We refer to a real-time scheduler as one with logic running every transmission time interval (TTI). The scheduling logic is written independently of the underlying eNodeB software and executed in real-time at the edge BS with the help of a scheduler agent. The proposed architecture is validated in an over-the-air environment with commercial long-term evolution (LTE) devices and 3rd Generation Partnership Project (3GPP) standards-compliant setup.

CCS CONCEPTS

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

KEYWORDS

Real-time, Programmable scheduler, Cellular Network

1 INTRODUCTION

Various traffic classes in current and future cellular networks, like enhanced Mobile Broadband (eMBB), massive Machine Type Communications (mMTC) and ultra Reliable Low Latency Communications (uRLLC), have flexible requirements in terms of bandwidth, latency and reliability that vary over time [10]. As a consequence, cellular network operators will need to have a high degree of programmability in the radio resource-scheduling logic for the current and future base station (BS), referred to as evolved NodeB (eNodeB) or next generation NodeB (gNodeB) in long-term evolution (LTE) and 5G networks [3], to program the scheduler according to the changes in traffic classes and network conditions.

The current implementation of the eNodeB scheduler can be classified into two categories. The first category is a distributed non-programmable scheduler architecture normally used in commercial cellular networks, shown in Fig. 1a. Whilst there are controls available to network operators to influence the behaviour of the scheduling logic to implement quality of service (QoS) by changing its parameters (such as priorities, minimum bit-rates, packet delay budgets etc.), the underlying real-time scheduling logic is tightly integrated into the rest of the eNodeB software at the edge BS. This limits the flexibility of a network operator to implement or procure customized logic to meet their requirements. This integrated architecture provides the benefits of 1) an uninterrupted operation of eNodeB even when the controller fails; 2) real-time scheduling every transmission time interval (TTI). The second category is a recently proposed centralized programmable scheduler architecture, referred to as FlexRAN [6, 7, 12], shown in Fig. 1b. Here the scheduler is first separated from eNodeB. The information exchange between eNodeB and the scheduler is using an interface based on Femto Forum Scheduler Application Programming Interface (FAPI) [5]. FAPI has been considered as a standard for interfacing a scheduler with any eNodeB. The scheduler is then moved from eNodeB at the edge BS to the controller, allowing operators to change the logic of the scheduler as they like. There are two issues with this approach; 1) eNodeB failure occurs when the controller fails; and 2) latency exceeding one TTI between the scheduler and eNodeB, prevents the scheduling logic from running at every TTI which we define as real-time scheduling. To prevent these issues, other centralized approaches such as the current industry initiatives Open RAN Alliance (O-RAN) [13] and SoftRAN [9] have only to date considered the non-real-time features of the scheduler to be programmable at the controller. No architecture in the commercial or open literature allows operators to program the real-time radio resource scheduler.

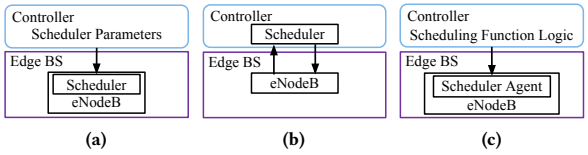

Figure 1: (a) Distributed non-programmable; (b) centralized programmable; and (c) distributed programmable scheduler architectures.

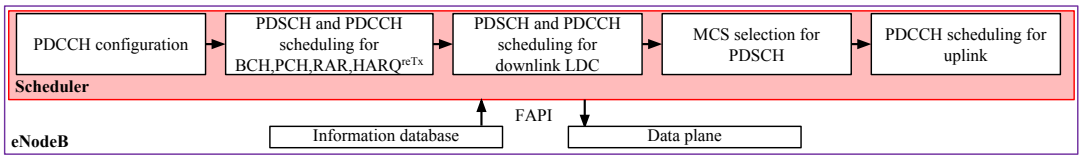

Figure 2: The scheduler architecture in srsENB.

In this paper, we propose a distributed real-time programmable scheduler architecture that will enable radio resource scheduling programmability in the physical downlink shared channel (PDSCH) and physical downlink control channel (PDCCH) for downlink logic data channels (LDCs), shown in Fig. 1c. Note that PDSCH and PDCCH are the physical radio resources arranged in order of time and frequency that transmit downlink data and control information to user equipment (UE) in LTE networks. An open-source eNodeB software from Software Radio Systems (SRS), referred to as srsENB [8], is used to implement the proposed architecture. The main contributions of the paper are as follows. Firstly, we believe this is the first architecture that allows network operators to rewrite scheduling logic at the controller and that uses a scheduler agent at the edge BS to execute that logic in real-time every TTI. To date, the most advanced centralized programmable architecture [7], as shown in our experiments, struggles to work in real-time due to a communication latency between the scheduler and eNodeB that exceeds multiple TTIs. Secondly, the agent incorporates new parameters on radio resource frame configurations when interfacing the scheduler to the eNodeB. These parameters have not been used in any scheduler interface standards such as FAPI [5]. In the current FAPI implementation, the scheduler needs to calculate these parameters, resulting in more information exchange between scheduler and eNodeB and a complex scheduler design. Inclusion of these parameters into FAPI are recommended. Thirdly, we develop a prototype for the above architecture and provide examples how to create scheduling functions to control traffic priority and to slice radio resources for the downlink LDC at the controller in over-the-air environments with commercial LTE devices and an LTE standards-compliant setup, referred to as an experimental LTE network. The prototype is built on top of the open-source eNodeB and evolved packet core (EPC) software from [8], referred to as srsENB and srsEPC, and software-defined radio (SDR) hardware from National Instruments [15]. Experimental results show the benefit of our distributed real-time programmable radio resource scheduler for operators by being able to customize real-time scheduling strategies through programmable logic in cellular networks. Note that the purpose of these experiments is not to benchmark the performance of our implemented algorithms against other implementations but rather to demonstrate that custom-made strategies can be implemented in the proposed architecture.

2 SCHEDULER ARCHITECTURE IN SRSENB

In this section, we first explain the scheduler architecture of srsENB [8] that will be used to develop our proposed distributed programmable scheduler architecture in Section 3. The schematic is shown in Fig. 2. srsENB has the following configurations: 1) resource allocation type 0 in which multiple resource blocks (RBs), each consisting of multiple resource elements (REs), are allocated as resource block groups (RBGs) and the total number of RBGs depends on the operating bandwidth; 2) a single-input-single-output (SISO) transmission mode; 3) each UE has one downlink LDC. The scheduler takes FAPI-compliance inputs from the information database in srsENB. The database stores the information and measurements reported by UEs, EPC and from srsENB itself. These range from a radio network temporary identifier (RNTI) (identifies UEs), QoS class identifier (QCI) to classify the downlink LDC set up for the UE, operating bandwidth, channel quality indicator (CQI) to indicate the achievable spectral efficiency, and the amount of data queued in each downlink LDC every TTI.

At the beginning of every TTI, the scheduler sets the value of the control format indicator (CFI) that specifies the format of PDCCH in the PDCCH configuration function block. Based on the configured CFI, the PDCCH configuration function block returns the candidate list of possible radio resources to be used to transmit downlink control information (DCI) to the UEs. These resources in PDCCH and the candidate list are referred to as control channel element (CCE) and PDCCH candidates (PCs). The PDCCH configuration function block also returns the number of resources left for the downlink data. In the next block, the scheduler in srsENB allocates the resources in PDSCH and PDCCH for the purpose of transmitting multiple downlink data types and their DCIs in PDCCH according to the following data type sequences: broadcast control channel (BCH), paging control channel (PCH), random access response (RAR) and hybrid automatic repeat request retransmission ($\text{HARQ}^{\text{reTx}}$).

UEs with data to transmit in their downlink LDCs to be scheduled are allocated with PDSCH resources in ascending order of RNTIs in this TTI. The process is repeated in the next TTI in a round-robin fashion, until all UEs have been scheduled. The modulation and coding schemes (MCSs) for the downlink data in PDSCH are then selected based on the spectrum efficiency indicated by reported CQI [8]. Finally in the last scheduler block, srsENB allocates the resources in PDCCH for transmitting control information that indicates the setup for uplink transmission by UEs. Note the limitation in srsENB [8] when the control information for the uplink is allocated last. If there are not enough PDCCH resources reserved for the uplink, there is a risk of uplink starvation. Uplink starvation happens when all available PDCCH resources are fully used for downlink transmissions. As a consequence, the control information to schedule uplink tranmissions for UEs cannot be transmitted by srsENB. Once the scheduling is completed for each TTI, the scheduler sends the scheduling allocation for all data types to the data plane with a format that complies with FAPI, which then maps the data into the radio frame and transmits the frame over the air.

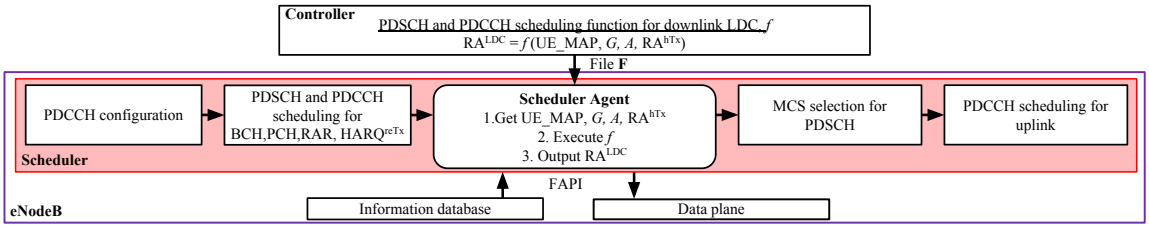

Figure 3: Distributed programmable scheduler architecture in srsENB.

3 DISTRIBUTED PROGRAMMABLE ARCHITECTURE

In this section, we modify srsENB in order to develop a distributed programmable scheduler architecture in srsENB. The developed architecture in srsENB is shown in Fig. 3. This architecture separates the function block of PDSCH and PDCCH scheduling for the downlink LDC described in Section II and keeps the remaining srsENB blocks intact. Note that this architecture can be extended to separate other scheduling functions in srsENB. There are two components in the proposed architecture; 1) the separated function denoted as f that is written at the controller and distributed as a file **F** to srsENB at the edge BS; and 2) a scheduler agent that abstracts input/output parameters for f and executes the logic of f for every TTI at the edge BS. The agent is inserted between PDSCH and PDCCH scheduling (for BCH, PCH, RAR and $\text{HARQ}^{\text{reTx}}$) and MCS selection (for PDSCH) in Fig. 3. These two components will be described in detail in the following sections. We define the downlink data types to be transmitted (described in Section II) as a list $D = \{\text{BCH}, \text{PCH}, \text{RAR}, \text{HARQ}^{\text{reTx}}, \text{LDC}\}$.

3.1 Scheduler agent in proposed architecture

The scheduler agent creates a list of information about UE $1, \ldots, N$, $\text{UE\_MAP} = \{\text{UE}_1, \ldots, \text{UE}_i, \ldots, \text{UE}_N\}$, obtained from the information database and the PDCCH configuration function block. UE_i, in UE_MAP, is a data structure with the information fields $\{\text{RNTI}_i, \text{QCI}_i, \text{CQI}_i, l_i, \text{PCs}_i\}$. RNTI_i is the RNTI of UE_i. CQI_i is the indicated achievable spectrum efficiency of UE_i. l_i is the amount of data queuing in the downlink LDC of UE_i. QCI_i is QoS information attached to the downlink LDC of UE_i, assigned by EPC. PCs_i is a list that contains P_i PDCCH candidates of UE_i. $\text{PCs}_i = \{\text{PC}_{i,1}, \ldots, \text{PC}_{i,P_i}\}$ where $\text{PC}_{i,p}$ is the CCE configuration p in PCs_i. Each $\text{PC}_{i,p}$ is denoted as $\text{PC}_{i,p} = \{\text{AL}_{i,p}, \text{I\_CCE}_{i,p}\}$. $\text{AL}_{i,p}$ is the aggregation level of $\text{PC}_{i,p}$. It denotes the number of consecutive CCEs used in $\text{PC}_{i,p}$, calculated as $2^{(\text{AL}_{i,p}-1)}$, where $\text{AL}_{i,p} \in \{1, \ldots, 4\}$ as defined in 3rd Generation Partnership Project (3GPP) [2]. $\text{I\_CCE}_{i,p}$ is the index of the first CCE of $\text{PC}_{i,p}$. The agent gets RNTI_i, QCI_i, CQI_i, and l_i of each UE_i from the information database, and PCs_i from PDCCH configuration function block. The agent abstracts the total number of RBGs and the numbers of REs in each RBG as G and $A = \{\text{RE}_1, \ldots, \text{RE}_G\}$. RE_n is the number of REs in RBG n calculated in PDCCH configuration.

We now explain how the scheduler allocates radio resources for multiple types of downlink data, denoted by D. The agent first gets PDCCH and PDSCH resource allocations for UEs that will receive transmissions of data type $t \in D$, $t \neq$ LDC, PDSCH and PDCCH scheduling for BCH, PCH, RAR, $\text{HARQ}^{\text{reTx}}$ block of srsENB. This information is collected as a list by the agent defined as $\text{RA}^{\text{hTx}} = \{\text{UE}_i^{\text{hTx}}, \ldots, \text{UE}_{N^{\text{hTx}}}^{\text{hTx}}\}$, where $\text{UE}_i^{\text{hTx}} = \{\text{RNTI}_i^{\text{hTx}}, X_i^{\text{hTx}}, \text{PC}_i^{\text{hTx}}\}$ stores the information on $\text{RNTI}_i^{\text{hTx}}$,the RBG bit-mask, X_i^{hTx}, and possible CCE configurations, PC_i^{hTx}, of UE i in the list. $X_i^{\text{hTx}} = \{x_{i,1}^{\text{hTx}}, \ldots, x_{i,G}^{\text{hTx}}\}$ is the allocated RBG bit-mask of UE i in the list. $x_{i,n}^{\text{hTx}} = 1$ if RBG n is allocated to UE i in RA^{hTx} list, else $x_{i,n}^{\text{hTx}} = 0$. $\text{PC}_i^{\text{hTx}} = \{\text{AL}_i^{\text{hTx}}, \text{I\_CCE}_i^{\text{hTx}}\}$ indicates the aggregation level and the index of the first CCE of UE i in RA^{hTx}. RA^{hTx} is then used as inputs for the separated function f for calculating the remaining PDSCH and PDCCH radio resources for the downlink LDC. f outputs the allocated resources in a format of $\text{RA}^{\text{LDC}} = \{\text{UE}_i^{\text{LDC}}, \ldots, \text{UE}_{N^{\text{LDC}}}^{\text{LDC}}\}$ where $\text{UE}_i^{\text{LDC}} = \{\text{RNTI}_i^{\text{LDC}}, X_i^{\text{LDC}}, \text{PC}_i^{\text{LDC}}\}$. $\text{RNTI}_i^{\text{LDC}}$ is the RNTI of UE i in RA^{LDC}. $X_i^{\text{LDC}} = \{x_{i,1}^{\text{LDC}}, \ldots, x_{i,G}^{\text{LDC}}\}$ is the allocated RBG bit-mask used to transmit the downlink LDC of UE i. $x_{i,n}^{\text{LDC}}$ is a binary number indicating the allocation of RBG n to UE i in RA^{LDC}. If RBG n is allocated, $x_{i,n}^{\text{LDC}} = 1$; otherwise, $x_{i,n}^{\text{LDC}} = 0$. $\text{PC}_i^{\text{LDC}} = \{\text{AL}_i^{\text{LDC}}, \text{I\_CCE}_i^{\text{LDC}}\}$ are the aggregation level and the index of the first CCE of UE i in RA^{LDC}. The list of scheduling parameters used by the agent is shown in Table 1. Note that Table 1 can be extended easily to include additional parameters for 5G New Radio (NR) since it has the same fundamental frame and data plane structure as LTE [1, 2].

We now define the 3GPP constraints on resource allocation. The first one is that the same UE cannot be scheduled for both the higher-priority data and the transmission of the downlink LDC in one TTI. Mathematically this can be written as

$$\text{RNTI}_i^{\text{LDC}} \neq \text{RNTI}_j^{\text{hTx}} \tag{1}$$

where $\text{RNTI}_i^{\text{LDC}} \in \{\text{RNTI}_1, \ldots, \text{RNTI}_N\}$ for $i = 1, \ldots, N^{\text{LDC}}$, $j = 1, \ldots, N^{\text{hTx}}$. N^{hTx} and N^{LDC} denote the number of UEs in RA^{hTx} and RA^{LDC} , respectively. The second constraint is that a RBG is available to downlink LDC only when it has not been allocated to high priority data types. By denoting the availability of RBG n for downlink LDCs with a binary;

$$\text{RBG}_n = \begin{cases} 0, & \text{if } \exists\, x_{i,n}^{\text{hTx}} = 1,\ i = 1, \ldots, N^{\text{hTx}}\,, \\ 1, & \text{otherwise}\,. \end{cases} \tag{2}$$

The third constraints is that the available RBG cannot be allocated to multiple UEs in RA^{LDC},

$$\sum_{i=1}^{N^{\text{LDC}}} x_{i,n}^{\text{LDC}} \leq \text{RBG}_n\,. \tag{3}$$

Table 1: Scheduling Parameter Abstraction for Agent

Inputs	
Name	**Description**
UE_MAP	The list of UE information.
UE_i	The information of UE i in UE_MAP.
RNTI_i	The RNTI of UE_i.
QCI_i	The QCI of UE_i.
CQI_i	The spectrum efficiency of UE_i.
l_i	The amount of data queued of UE_i.
PCs_i	The possible CCE configurations of UE_i.
$\text{PC}_{i,p}$	The CCE configuration p in PCs_i.
$\text{AL}_{i,p}$	The aggregation level of $\text{PC}_{i,p}$.
I_CCE_i	the index of the first CCE of $\text{PC}_{i,p}$.
G	The total number of RBGs in the bandwidth.
A	The list of the number of REs per RBG.
RE_n	The number of REs of RBG n in A.
RA^{hTx}	The resource allocation for high-priority data
UE_i^{hTx}	The resource allocation for UE i in RA^{hTx}
$\text{RNTI}_i^{\text{hTx}}$	The RNTI of UE_i^{hTx} in RA^{hTx}.
X_i^{hTx}	The allocated RBG bit-mask of UE_i^{hTx}in RA^{hTx}.
$x_{i,n}^{\text{hTx}}$	The binary indicator for RBG n of X_i^{hTx}.
PC_i^{hTx}	The CCE configuration of UE_i^{hTx} in RA^{hTx}.
AL_i^{hTx}	The aggregation level of PC_i^{hTx}.
$\text{I\_CCE}_i^{\text{hTx}}$	the index of the first CCE of PC_i^{hTx}.
Outputs	
Name	**Description**
RA^{LDC}	The resource allocation for LDC
UE_i^{LDC}	The resource allocation for UE i in RA^{LDC}
$\text{RNTI}_i^{\text{LDC}}$	The RNTI of UE_i^{LDC}in RA^{LDC}.
X_i^{LDC}	The allocated RBG bit-mask of UE_i^{LDC}in RA^{LDC}.
$x_{i,n}^{\text{LDC}}$	The binary indicator for RBG n of X_i^{LDC}.
PC_i^{LDC}	The CCE configuration of UE_i^{LDC}in RA^{LDC}.
AL_i^{LDC}	The aggregation level of PC_i^{LDC}.
$\text{I\_CCE}_i^{\text{LDC}}$	the index of the first CCE of PC_i^{LDC}.

Lastly, one CCE cannot be used by multiple UEs or data types,

$$\{\text{I\_CCE}_i^{t_1}, \ldots, \text{I\_CCE}_i^{t_1} + 2^{\text{AL}_i^{t_1}} - 1\} \bigcap \{\text{I\_CCE}_j^{t_2}, \ldots, \text{I\_CCE}_j^{t_2} + 2^{\text{AL}_j^{t_2}} - 1\} = \emptyset \,, (t_1, i) \neq (t_2, j) \,, \tag{4}$$

where $t_1, t_2 \in \{\text{hTx}, \text{LDC}\}$, $i = 1, \ldots, N^{t_1}$ and $j = 1, \ldots, N^{t_2}$.

3.2 Separated scheduling function

The separated scheduling function, f, is written at the controller and integrated by the scheduler agent as $\text{RA}^{\text{LDC}} = f(\text{UE\_MAP}, G, A, \text{RA}^{\text{hTx}})$. UE_MAP, G, A and RA^{hTx} are the inputs of f and RA^{LDC} is the output of f. f consists of four cascaded functions, f_{in}, f_{pdsch}, f_{pdcch} and f_{out}, all of which are stored in one file, **F**, shown in Fig. 4 and explained below. f_{in}, shown in Algorithm 1, is first run with UE_MAP, G, and RA^{hTx} as inputs and UE_MAP, g and RA^{LDC} as outputs. f_{in} first excludes UEs that has been scheduled by srsENB for other downlink higher-priority data types according to (1), as shown in lines 1-8. This is done by creating RA^{LDC} that contains all UEs in UE_MAP. We remove UEs whose RNTI is listed in RA^{hTx} from RA^{LDC} and UE_MAP. f_{in} then estimates the availability of RBGs for LDC by using (2) and creates the bit-mask to indicate the available RBGs as $g = \{\text{RBG}_1, \ldots, \text{RBG}_G\}$ in lines 9-10.

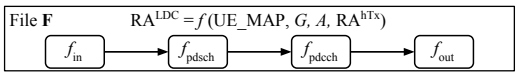

Figure 4: The separated scheduling function, f, in File F

Algorithm 1 The logic of f_{in}.

Input UE_MAP, G, RA^{hTx}; **Output** UE_MAP, g, RA^{LDC}.
1: create $\text{RA}^{\text{LDC}} = \emptyset$.
2: **for** i from 1 to N **do**
3: add UE_i^{LDC} with $\text{RNTI}_i^{\text{LDC}} = \text{RNTI}_i$ into RA^{LDC}.
4: **end for**
5: **for** i from 1 to N^{hTx} **do**
6: remove UE_j^{LDC} with $\text{RNTI}_j^{\text{LDC}} = \text{RNTI}_i^{\text{hTx}}$ from RA^{LDC}.
7: remove UE_j with $\text{RNTI}_j = \text{RNTI}_i^{\text{hTx}}$ from UE_MAP.
8: **end for**
9: **for** n from 1 to G **do** calculate RBG_n by using (2).
10: create $g = \{\text{RBG}_1, \ldots, \text{RBG}_G\}$.

Algorithm 2 The logic of f_{out}.

Input RA^{LDC}; **Output** RA^{LDC}.
1: **for** i from 1 to N^{LDC} **do**
2: **if** $\sum_{n=1}^{G} x_{i,n}^{\text{LDC}} = 0$ **then** remove UE_i^{LDC} from RA^{LDC}.
3: **end for**

The functions f_{pdsch} and f_{pdcch} are then used to run the scheduler logic of PDSCH and PDCCH for the downlink LDC. f_{pdsch} takes UE_MAP, g, A and RA^{LDC} as the inputs and overwrites RA^{LDC} as the output. f_{pdcch} takes UE_MAP, RA^{hTx} and RA^{LDC} as the inputs and overwrites CCE allocations of the UEs in RA^{LDC} as the output. Note that the allocations in f_{pdsch} and f_{pdcch} must satisfy the constraints defined in (3) and (4). We show examples on programming f_{pdsch} and f_{pdcch} according to the operator's need in Section 5. Finally, f_{out} takes RA^{LDC} as the input and returns RA^{LDC} as the output in which UEs with no RBG allocated are excluded, as shown in Algorithm 2, lines 1-3. Note that the programmability of the scheduling logic can be extended to other blocks in Fig. 3 and/or to 5G NR features by identifying relevant scheduling parameters and writing the logic in f.

4 COMPARISON WITH FAPI

The scheduling parameters in Table 1 except PCs_i and RE_n are also defined in the FAPI. Thus to get PCs_i and RE_n, an FAPI agent needs to provide additional information such as cyclic prefixes, radio frame indexes and $\text{HARQ}^{\text{reTx}}$ feedback channels to the separated scheduling function shown in Fig. 3. In our scheme, however, the agent directly abstracts scheduling parameters, PCs_i and RE_n, for the separated function's inputs. This reduces scheduling parameter exchange between the scheduling function and the agent and therefore simplifies its logic design as compared to FAPI-based one.

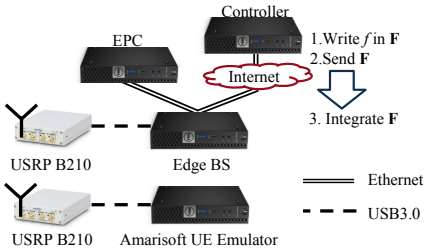

Figure 5: The experimental implementation.

Algorithm 3 f_{pdsch} for traffic prioritization.

Input UE_MAP, g, A, RA^{LDC}; **Output** RA^{LDC}.
1: create $\text{indexes}_1 = \{i \mid \text{QCI}_i = 1\}$ and $\text{indexes}_2 = \{i \mid \text{QCI}_i = 2\}$.
2: set $n = 1$.
3: **for** k from 1 to 2 **do**
4: set i as the element at the start of indexes_k.
5: **while** $n \leq G$ **do**
6: **if** $\text{RBG}_n = 0$ **then** set $n = n + 1$ and **Continue**
7: **if** $l_i > 0$ **then**et $x^{\text{LDC}}_{i,n} = 1$, $l_i = l_i - \text{CQI}_i\text{RE}_n$ and $n = n + 1$.
8: go to the next i in indexes_k
9: **if** all UEs in Class k has been scheduled **then Break**
10: **end while**
11: **end for**

Algorithm 4 f_{pdcch} for traffic prioritization.

Input UE_MAP, RA^{hTx}, RA^{LDC}; **Output** RA^{LDC}.
1: create $\text{indexes}_1 = \{i \mid \text{QCI}_i = 1$ and $\sum_{n=1}^{G} x_{i,n} > 0\}$.
2: create $\text{indexes}_2 = \{i \mid \text{QCI}_i = 2$ and $\sum_{n=1}^{G} x_{i,n} > 0\}$.
3: **for** k from 1 to 2 **do**
4: **for** i in indexes_k **do**
5: **for** p from 1 to P_i **do**
6: **if** $\text{PC}_{i,p}$ satisfies (4) **then** set $\text{PC}_i = \text{PC}_{i,p}$ and **Break**
7: **end for**
8: **if** no $\text{PC}_{i,p}$ in PCs_i satisfies (4) **then** set X^{LDC}_i with all 0.
9: **end for**
10: **end for**

5 PROTOTYPE

We now describe the implementation of the proposed distributed programmable scheduler architecture (DPSA) in the experimental LTE networks. The purpose of experiments in this section is to show that different real-time scheduling algorithms can be achieved in the proposed architecture. We compare scheduling algorithms implemented in the proposed architecture with algorithms implemented in two other systems, a non-QoS aware distributed non-programmable scheduler architecture (DNSA) represented by unmodified srsENB [8] and the centralized programmable scheduler architecture (CPSA) proposed in FlexRAN [7]. The time required for information exchange between the scheduler and srsENB, referred to as communication latency and the downlink throughputs for various also configurations are measured in experiments.

The experiment deployed in our lab is shown in Fig. 5. We create an edge BS by running srsENB on a computer located in our lab. It has a USB 3.0 connection to a Universal Software Radio Peripheral (USRP) B210 RF front-end [15], used to convert I/Q samples generated by srsENB into an RF signal. The open-source EPC [8] is operated on another computer and is connected to the edge BS via an Ethernet link. The controller runs on a computer at a different location, outside our lab. It communicates with the edge BS over the Internet. We use UE emulators [4] to replicate multiple UEs on a computer with a USRP B210. The experimental LTE network runs in Band 7 with an operating bandwidth of 10MHz (e.g., $G = 17$) and CFI = 3. Each UE is assumed to have an application with a fixed 5Mbps best-effort UDP traffic. The packet size is fixed as 1300 bytes. In the controller, the operator writes the codes for f described in Section 3.2 using the C++ programming language and stores it in the file **F**. **F** is then distributed to the edge BS using git protocol [14]. At the edge BS, the scheduler agent integrates **F** into srsENB.

5.1 Traffic prioritization example

Let us assume that we have UEs $1, \ldots, N$ in the experimental network. The operator now decides that the traffic for UEs indexed by odd numbers is more important than that indexed by even numbers, leading to two traffic classes. $\text{QCI}_i = 1$, where i is a set of UEs indexed by odd numbers, is set to indicate higher priority. The rest of the UEs are assigned with $\text{QCI}_i = 2$, where i denotes the UEs indexed with even numbers. We run experiments with the different numbers of UEs, $N = 2, 4, 6, 8, 10$. The operator will need to reprogram f at the controller in order to recognize new traffic classes; it cannot be done in a non-programmable scheduler in today's eNodeB. An example of how to write PDSCH and PDCCH functions, f_{pdsch} and f_{pdcch} at the controller, is shown in Algorithms 3 and 4.

In f_{pdsch}, the indexes of UEs in UE_MAP are separated into two class lists based on their QCI values, as shown in line 1 of Algorithm 3. f_{pdsch} allocates the available RBGs to the UEs in Class 1 that have the higher priority in every TTI. This RBG allocation is done in two steps executed inside the while loop of Algorithm 3 to satisfy (3). We first check whether RBGs can be allocated to a downlink LDC; if not, we move to the next one, as shown in line 6. We then allocate the available RBG by setting its bit-mask to UEs with data to transmit in the downlink LDC, as shown in lines 7-10. The amount of data of UEs is calculated by subtracting the amount of data transmitted on the allocated RBG from the queue size of the UE. The amount of data transmitted on the RBG is calculated by multiplying the spectrum efficiency of the UE with the number of REs on this RBG. If all UEs in Class 1 have been scheduled, f_{pdsch} will move to Class 2 and will repeat the same process, as shown in line 11.

We then program f_{pdcch}. It first classifies the UEs by separating the UEs, scheduled by f_{pdsch} in RA^{LDC}, into Class 1 and 2 based on their QCIs, as shown in lines 1-2 of Algorithm 4. Starting from Class 1 (which has the higher priority), the CCE configuration satisfying (4) will be searched for every UE, as shown in lines 3-5. Once CCE configurations have been found, the UE will be allocated with this configuration, as shown in line 6. Else the transmission of the UE will be dropped by setting its RBG bit-mask as 0, as shown in line 8. The same process is repeated for Class 2.

Fig. 6 compares the average UE throughputs in DPSA and DNSA [8]. When the number of UEs, N, is small, the average UE throughputs in both architectures are 5 Mbps. When N increases, the average UE throughput of Class 1 in DPSA remains at 5 Mbps, while the throughputs of Class 2 in DPSA decrease significantly. This is because the capacity of the network cannot deliver 5 Mbps for every UE and the programmed scheduling functions in DPSA will prioritize UEs in Class 1, resulting in too little resources being allocated

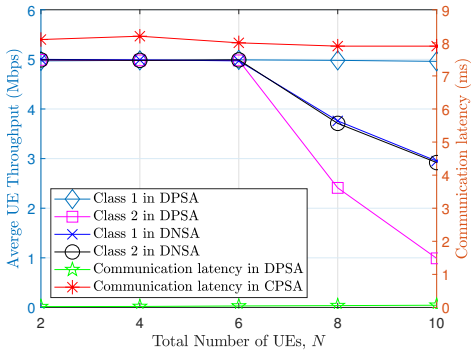

Figure 6: Traffic prioritization example.

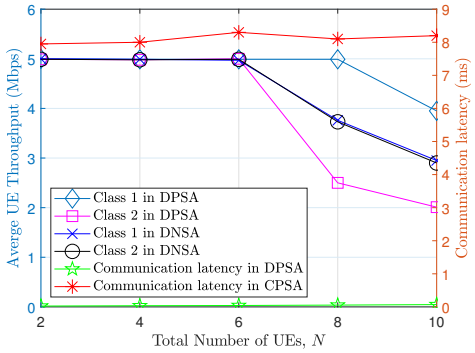

Figure 7: Resource slicing example.

to Class 2. Compared to the DNSA, the high degree of real-time programmability in DPSA results in 60% more throughput to the UEs in Class 1 when N is large. Fig. 6 also compares the communication latency in DPSA with the one in CPSA [7], which are around 8 TTIs and 0.05 TTI (e.g., 8 and 0.05 msec), regardless the number of UEs. For the scheduler to run in real-time, a latency of less than 1 TTI is compulsory, achieved only by using our proposed DPSA.

5.2 Resource slicing example

Let us assume that Class 1 and 2 in the experimental network described before represents two different verticals with high and low data rate requirements. The operator now wants to slice RBGs to satisfy different QoS of the two verticals. Note that Next Generation Mobile Network (NGMN) defines slicing as running multiple vertical networks in parallel in a share resource [11]. This can be done by reserving exclusive RBGs, 2/3, to Class 1 and the less remainder, 1/3, to Class 2. We assume that the proportion of the RBGs reserved for Class k is R_k, and $R_1 = 2/3$ and $R_2 = 1/3$ for Class 1 and 2, respectively. f_{pdsch} is then reprogrammed to slice RBGs for different classes by replacing line 5 of Algorithm 3 with

$$\textbf{while } n \leq G \text{ and } W_k < round(R_k G) \textbf{ do} . \tag{5}$$

The number of RBGs allocated to Class K is denoted as W_k and $round(R_k G)$ is a function to round the maximum allocated RBGs for Class k to the nearest integer. For PDCCH scheduling, we use the same algorithm as shown in Algorithm 4.

Fig. 7 compares the average UE throughputs in DPSA and DNSA [8]. When the number of UEs, N, is small, the average UE throughputs in both architectures are 5 Mbps. When N increases to 10, the average UE throughput of Class 1 drops to 4Mbps. This is because UEs in Class 1 can only use 2/3 of available RBGs. Compared to the DNSA, the high degree of programmability of the DPSA results in 30% more throughput going to the UEs in Class 1 when N is large. By using the proposed DPSA, we can program scheduling function logic as we like. Similar communication latency behaviour as in Section 5.1 are observed in Fig. 7.

6 CONCLUSION

In this paper, we address the challenge of programming the scheduler in a cellular network by proposing a distributed programmable architecture and implementing it in srsENB [8]. The proposed architecture separates the PDSCH and PDCCH scheduling functions for the downlink LDC from the srsENB scheduler. This allows the codes of the separated scheduling function to be written independently of the underlying eNodeB and executed in real-time at the edge BS. The prototyping activities in the experimental LTE network have shown that the proposed architecture provides a high degree of programmability in the scheduling logic and allows the operator to apply real-time customized strategies according to their needs.

ACKNOWLEDGMENT

This project is directly funded by Telstra Corporation Ltd. and supported by Australian Research Council Discovery Early Career Research Award (DE150101704).

REFERENCES

[1] 3GPP. 2017. *Physical layer; General description.* Technical Specification (TS) 38.201. 3GPP. Version 15.0.0.
[2] 3GPP. 2018. *Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.* Technical Specification (TS) 36.213. v15.2.0.
[3] 3GPP. 2019. The 3rd Generation Partnership Project. Retrieved Feb 12, 2019 from http://www.3gpp.org/about-3gpp
[4] Amarisoft. 2019. Amarisoft UE. Retrieved Feb 12, 2019 from https://www.amarisoft.com/
[5] Femto Forum. 2010. *LTE MAC Scheduler Interface Specification.* Technical Report. Version 1.11.
[6] 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*. ACM, 127–140.
[7] 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 *CoNEXT'2016*. ACM, 427–441.
[8] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. 2016. srsLTE: an open-source platform for LTE evolution and experimentation. In *WiNTECH'2016*. ACM, 25–32.
[9] Aditya Gudipati, Daniel Perry, Li Erran Li, and Sachin Katti. 2013. SoftRAN: Software defined radio access network. In *HotSDN'2013*. ACM, 25–30.
[10] Akhil Gupta and Rakesh Kumar Jha. 2015. A survey of 5G network: Architecture and emerging technologies. *IEEE access* 3 (2015), 1206–1232.
[11] NGMN. 2019. Next Generation Mobile Networks. Retrieved Feb 12, 2019 from https://www.ngmn.org/
[12] Navid Nikaein, Chia-Yu Chang, and Konstantinos Alexandris. 2018. Mosaic5G: Agile and flexible service platforms for 5G research. *ACM SIGCOMM Computer Communication Review* 48, 3 (2018), 29–34.
[13] O-RAN. 2019. Open RAN Alliance. Retrieved Feb 12, 2019 from http://https://www.o-ran.org/
[14] Linus Torvalds. 2005. Git. Retrieved Feb 12, 2019 from https://git-scm.com/
[15] USRP. 2011. National Instruments USRP. Retrieved Feb 12, 2019 from https://www.ettus.com