CLEI electronic journal, Volume 25, Number 2, Paper 2, May 2022

An Open Source Multi-Slice Cell Capacity Framework

Gabriela Pereyra, Lucas Inglés, Claudina Rattaro and Pablo Belzarena
Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay, 11300,
pereyra.gab@gmail.com; {lucasi,crattaro,belza}@fing.edu.uy

Abstract

5G is the new 3GPP technology designed to solve a wide range of requirements. On the one hand, it must be able to support high bit rates and ultra-low latency services, and on the other hand, it should be able to connect a massive amount of devices with loose bandwidth and delay requirements. Network Slicing is a key paradigm in 5G, and future 6G networks will inherit it for the concurrent provisioning of diverse quality of service. As scheduling is always a delicate vendor topic and there are few free and complete simulation tools to support all 5G features, in this paper, we present Py5cheSim. This is a flexible and open-source simulator based on Python and specially oriented to simulate cell capacity in 3GPP 5G networks and beyond. To the best of our knowledge, Py5cheSim is the first simulator that supports Network Slicing at the Radio Access Network level. It offers an environment that allows the development of new scheduling algorithms in a researcher-friendly way without the need of detailed knowledge of the core of the tool. The present work describes its design and implementation choices, the validation process, the results and different use cases.
Keywords: 5G, Network Slicing, simulator, scheduling, framework.

1 Introduction

The requirements of mobile services have changed over the years together with cellular technologies. In response to the increase of mobile broadband services and the new use cases expected to take place in the market for the following years, the fifth generation of mobile communications (5G) is already becoming a reality by the new 3GPP (3rd Generation Partnership Project) technology. The services supported by 5G fall under three categories formulated by ITU-R, i.e., enhanced Mobile BroadBand (eMBB), massive Machine-Type Communications (mMTC), and Ultra-Reliable Low-Latency Communications (URLLC). These heterogeneous services have distinctive and extreme requirements for network performance [1]. First, eMBB traffic is a direct extension of the 4G broadband service that focuses on a higher data rate (20 and 10 Gbits/s downlink and uplink peak data rates, respectively), with a large payload and prolonged internet connectivity based applications. Second, mMTC focuses on uplink communications of massive low rate devices (connection density about $1,000,000$ devices per km2). Finally, URLLC services target mission-critical communications such as autonomous vehicles, tactile internet, or remote surgery. Their main requirements are ultra-high reliability with a packet error rate (PER) around 10^{-5} and low latency (1 ms).

Such diversity in terms of service requirements demands a high degree of flexibility in radio interface design. As LTE (Long Term Evolution, 4G) technology was initially designed with Mobile Broadband (MBB) services evolution in mind, it does not provide enough flexibility to optimally multiplex the different types of services envisioned by 5G. This problem is due to the fact that there is no a unique radio interface configuration that suits all the different service requirements. Taking that into consideration, 5G networks are being designed to support multiple radio interface configurations and mechanisms to multiplex these services with different configurations in the same available spectrum. This concept is known as Network Slicing [2, 3, 4, 5], and is a 5G key feature that needs to be supported end to end in the network (Radio Access, Transport and Core Network). As a result, 5G Radio Access Networks (RAN) will add the different services resource allocation problem to the users' resource allocation traditional one (inter-slice and intra-slice resource allocation problems, respectively).

In terms of RAN Physical layer design, Subcarrier Spacing (SCS) in 5G was proposed to be variable to adapt to different requirements [6] [7]. High SCS, or high numerology as it has been called, will be configured for services with low delay requirements or high data rates. As a way to increase available resources, 5G adds new parts of the spectrum: the millimeter Waves. The standard provides frequency bands above 6

GHz for NR (New Radio) TDD (Time Division Duplex) use. In addition, the standard proposes analog beamforming to improve coverage and introduce high subcarrier space with low TTI (Transmission Time Interval) for lower delays. Another key design concept is the compatibility with legacy systems, i.e., 4G. Consequently, the SCS basis should be the same as in LTE, 15kHz. 5G should also support LTE-Advanced features like Carrier Aggregation (CA) and Multiple Input Multiple Output (MIMO).

There is a vast bibliography on 5G and is composed of research articles, vendor white papers, and web site presentations [2, 3, 8]. However, given that scheduling is always a delicate vendor topic, few free tools simulate cell capacity in 3GPP networks. Existing tools represent only a selected set of features presented in the 5G standard. In fact, no currently known simulator has the specific features and flexibility needed to implement and evaluate a complete cell capacity analysis. Even more, no one implements Network Slicing at the RAN level. In short, in the simulator market, either network simulators or system-level simulators are used. The former often implement layer by layer most of the procedures described in the 3GPP standard, making simulations hard to configure and implying high processor loads. The latter type of simulator often present a high degree of simplification to cover a wide range of cells with affordable resource use.

We have developed a new simulator in the Python platform in order to tackle these problems, called Py5cheSim (Python 5G Scheduler Simulator). Its general design goal is to keep it as simple and flexible as possible for scheduler implementation. In addition, the purpose of Py5cheSim is to build a specific tool for simulating cell capacity in a 5G network for Frequency Division Duplex (FDD) and Time Division Duplex (TDD) operation, including different types of schedulers for the various slices that 5G Networks can handle. Py5cheSim allows us to analyze inter and intra-slice scheduling. There is also no need to implement layer by layer all the procedures defined in the standard. This new simulator is lighter, quicker and more straightforward than many of the existing free tools. Last but not least, as Python offers a vast choice of libraries for Artificial Intelligence (AI) development, Py5cheSim allows to implement easily AI-based algorithms.

To summarize, the main contributions of Py5cheSim are listed below:

1. it is being developed as a free, open-source software project under the GNU license;
2. it is the first free simulator that supports Network Slicing at the Radio Access Network level;
3. it includes a framework that allows the design of new resource allocation algorithms without understanding in detail how the simulator works; and
4. its intermediate position between network and system level simulators achieves very good performance for large scale network simulations.

Some preliminary ideas of Py5cheSim were published in our previous articles [9, 10]. In [10] we presented Py5cheSim version 1.0. We are now introducing a new version that incorporates an associated library and a graphical user interface. Said library allows the end-user to implement (intra-slice and inter-slice) scheduling algorithms efficiently and straightforwardly, without the need to study the operation of the entire simulator. Compared with [10] we include more validated scenarios, and we also present different use cases showing the potential of the simulator and its library.

The rest of the paper is structured as follows. We start in Section 2 giving a complete overview of the related works in 5G simulation tools. In Section 3 we briefly describe Py5cheSim characteristics and its architecture. In Section 4 we present Py5cheLiSA which is the library for scheduling algorithms and in Section 5 we illustrate its potential and versatility showing some usage examples. Then, in Section 6 we present the validation results and some examples of usage in realistic 5G scenarios considering Network Slicing scheduling. Part of the validation test consists of comparing the performance of Py5cheSim with a reference simulation tool. Finally, Section 7 discusses our roadmap and plans and concludes the work.

2 Related Work

Different research groups have developed simulators targeting 5G network characteristics in recent years, being 5G-LENA and Vienna the most popular ones. Other examples, considering only software that is openly available for academic purposes, are 5G-K-Sim, Simu5G, SyntheticNET, and OpenAirInterface.

5G-LENA [11] is a GPLv2 simulator designed as a pluggable module to ns-3. It is strongly based on lte-LENA and mmWaves modules [12, 13]. Ns-3 is a network simulator, very rich in terms of technologies supported. The 5G-LENA module in particular provides several parameter configuration options, making it an excellent choice for different scenario simulations. These modules present some disadvantages: the high degree of complexity and processing capacity needed to configure and run a simulation. The former is not a problem if one is a C++ developer and has experience with ns-3, but the second is unavoidable because

of the simulator's nature. These modules implement layer by layer most of the procedures described by the 3GPP standard, so a simple ten-minute simulation with high bandwidth and several users can take hours in a standard PC. Additionally, although 5G-LENA supports many NR features, the current version of this module (NRv1.2 available since Jun 2021) does not implement mini-slots and network slicing scheduling. It is important to mention that in part of our simulator validation process, we use 5G-LENA as the basis (in particular in everything related to intra-slice scheduler module: MCS (Modulation and Coding Scheme), BER (Bit Error Rate), SINR (Signal-to-Interference-plus-Noise Ratio), TBS (Transport Block Size), and throughput calculation or generation).

Concerning system-level simulators, one piece of software that stands out is the Vienna Simulator [14, 15]. This MATLAB tool, which is available for download under an academic use license, permits link-level and system-level simulations. It is based on its predecessor Vienna LTE simulator. The latest version of Vienna was released in 2021. This version does not support key NR features like mini-slot scheduling, network slicing, mmWave propagation models, and 256-QAM modulation. Also, the simulator does not include the possibility to perform an uplink simulation and use non-full buffer traffic model.

Simu5G [16, 17], based on OMNeT++ framework written in C++, is also categorized as a system-level simulator. Simu5G simulates the data plane of the 5G RAN (rel. 16) and core network. It supports many interesting features that are not present in others (e.g. FDD and TDD modes, dual connectivity, carrier aggregation, different numerologies). However, according to its lastest version 1.2.0, it does not simulate all possible features in relation to resource allocation like network slicing or mini-slot, being essential for URLLC traffic. Unlike Vienna, which is well tailored for the evaluation of lower-layer procedures, including signal-processing techniques, Simu5G, as well as Py5cheSim, is a discrete-event, application-level simulator.

5G-K-Sim [18, 19] is a complete C++ tool that includes link-level, system-level, and network-level simulations. Its network version has a SDN/NFV module, which is an essential function for the network slicing technology. However, it does not support end-to-end network slicing capabilities (in particular, RAN-slicing). 5G-K-Sim was developed at the earliest stages of the 5G standardization process and is non-fully standard compliant.

SyntheticNET [20] is a Python simulator that is focused on modeling a realistic handover process, including a realistic urban mobility module. The authors of SyntheticNET said that a free version of SyntheticNET would be available soon for academic purposes. It supports different numerologies and mmWaves, but the authors have not yet specified what other 5G features it supports.

Finally, it is important to mention OpenAirInterface [21, 22]. It consists of open-source software running on general-purpose processors. This development supports many of the NR specifications. Its main limitation is its lack of ability to scale simulations up to large networks, but it represents an interesting tool to validate new proposals in real testbeds.

3 Py5cheSim Design: characteristics and architecture

In this section, we present a brief description of our simulator. First, we introduce the simulator characteristics and features, and at the end of the section, we explain the simulator architecture. In the next section we will present its associated library oriented to the development of scheduling algorithms.

3.1 Main characteristics

The general design goal for Py5cheSim was to keep it as simple as possible, aiming to maintain the biggest freedom degree possible when it comes to scheduler implementation. Python was used to develop the simulator as it is considered to be a powerful and versatile language, provided with tons of packages developed for specific purposes, from discrete event simulation to machine learning tools. The tool used to implement discrete event simulation was SimPy [23].

Figure 1 shows some of the main concepts involved in a simple simulation. One cell may serve one or more UE (User Equipment) groups. Each UE group has a number of UEs with a defined traffic profile. Traffic profile is described by the Packet Flow parameters. Packet Flow is transported through the air by Bearers. As Py5cheSim focuses on the radio part of the network, only radio Bearers are considered. Each UE connected to the cell has a radio link with a quality given by the UE SINR. Packets are processed through a simplification of the radio interface protocol stack and transmitted over the air through Transport Blocks using resources from the time-frequency grid, as can be seen on the diagram.

Py5cheSim implements RAN Slicing as a core feature using a two-level scheduler composed of an Intra Slice Scheduler and an Inter Slice Scheduler. The first one is oriented to solve resource allocation between different UEs of the same Slice, and the second to allocate resources between the different Slices. Each Slice has a set of requirements and a configuration. Configuration is set automatically depending on Slice requirements in terms of delay, band, the number of UEs to serve, traffic profile, UE capabilities, and

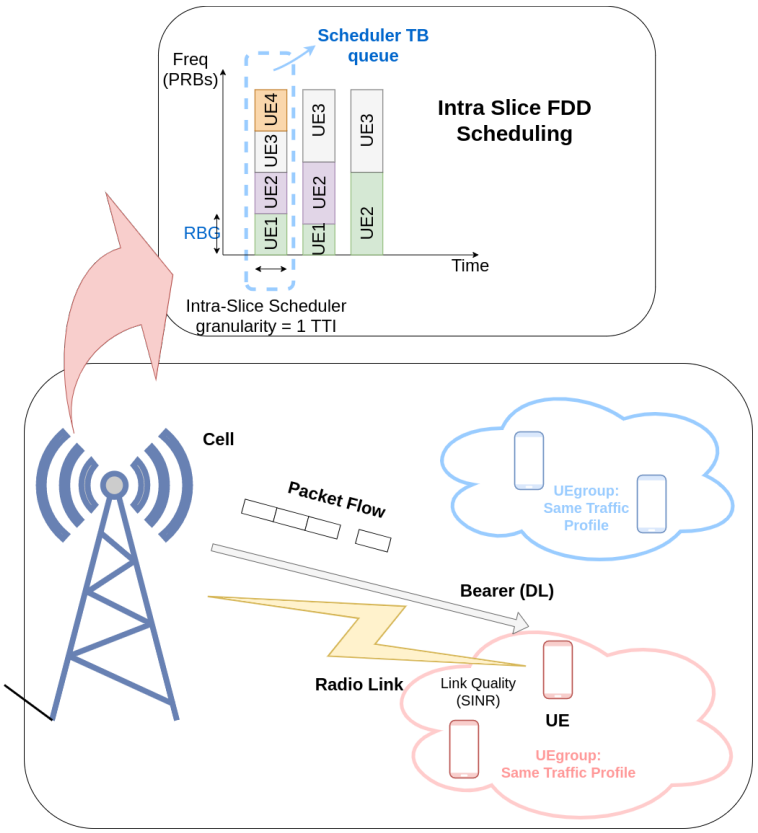


Figure 1: Py5cheSim main concepts general diagram.

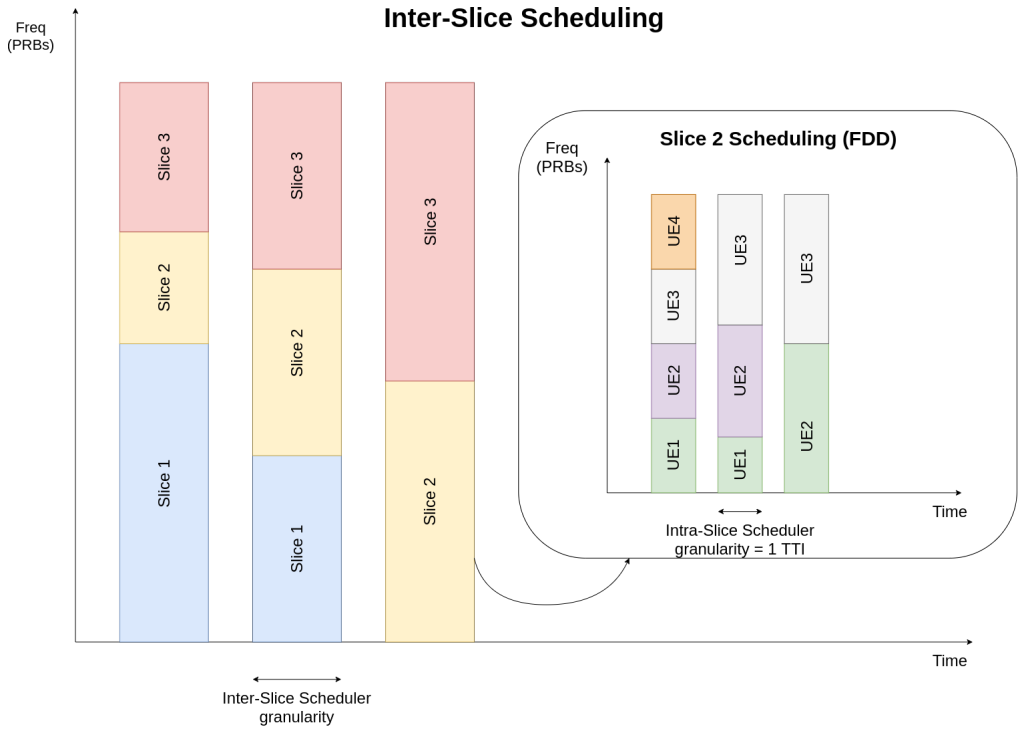


Figure 2: Inter and Intra Slice schedulers' basic implementation scheme.

availability. For each Slice, numerology/SCS/TTI, duplexing mode, scheduler algorithm to use, signaling load, and allocated PRBs (Physical Resource Blocks) are set at the initialization of the simulation. Slice allocated PRBs can change according to Inter Slice scheduler decision with a custom granularity. Figure 2 shows the basic idea behind this RAN Slicing implementation.

Py5cheSim supports multiple numerologies, FDD and TDD frame (depending on the cell band set for the

simulation), uplink and downlink bearers, and a basic implementation of Carrier Aggregation and Single-User/Multi-User MIMO functionalities. Transport Block Size (TBS) calculation, which depends on both the number of allocated PRBs and the MCS, is based on 3GPP Technical Specifications [24, 25]. At the moment, MCS allocation is based purely on UE's SINR. An SINR-MCS table was generated from 5G-LENA for a wide range of SINRs being an input of Py5cheSim. However, different MCS allocation algorithms could be implemented overwriting the *setMod* method (see IntraSliceSche in Figure 4).

Py5cheSim also supports different traffic profiles' configuration by groups of UEs that can emulate the different 5G services (eMBB, URLLC and mMTC). The traffic profile is set in terms of average packet size (in bytes) and inter-arrival times (in ms). The implemented traffic model was based on the one considered in [26].

3.2 Architecture

The Simulator is built on the modules of Figures 3 and 4. UE, Cell, IntraSliceSch, InterSliceSch and Slice are the simulation core.

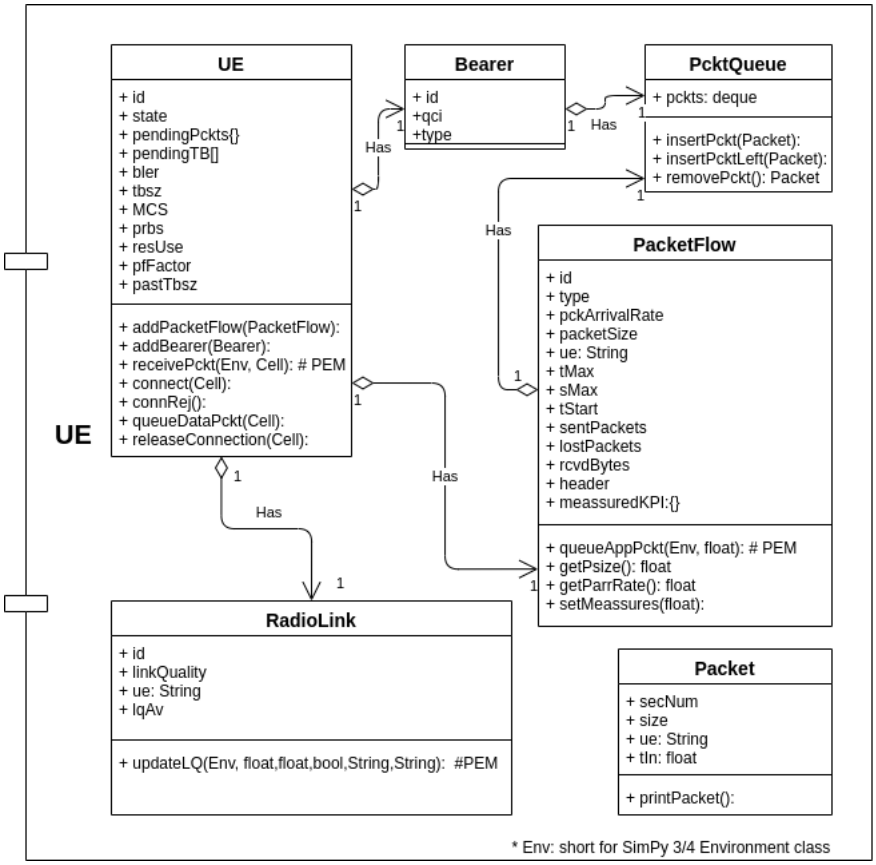


Figure 3: UE module class diagram. Define UE parameters and traffic. In particular PacketFlow class is responsible for creating different traffic flows.

IntraSliceSch and Cell have implemented the basic schedulers for one and several slices, respectively. The default scheduling algorithm in these classes is Round Robin. Other schedulers must be defined as classes inherited from the Base Scheduler's ones defined in the former modules overwriting the *resAlloc* method. In Figure 4 three scheduling algorithms for inter-slice scheduling are shown as examples: Round Robin by default, Proportional Fair and a modified version of Round Robin. In terms of intra-slice scheduler, for FDD simulations resource allocation for different UEs here is done in terms of PRB, then the next traditional scheduling algorithms are actually supported: Round Robin and Proportional Fair. For TDD simulations resource allocation is done in a TTI granularity along the entire band. Only Round Robin TDD scheduler is supported at the moment. Other algorithms can be added as new classes inherited from TDD Scheduler class. In next sections, as the associated library is presented, we will show some AI-scheduling algorithms which are also supported in the new version of Py5cheSim.

Two classes have been developed to support inter Slice Scheduling: InterSliceSch and Slice (the yellow ones in Figure 4). The first one implements the inter-slice scheduler, as it dynamically allocates band PRBs

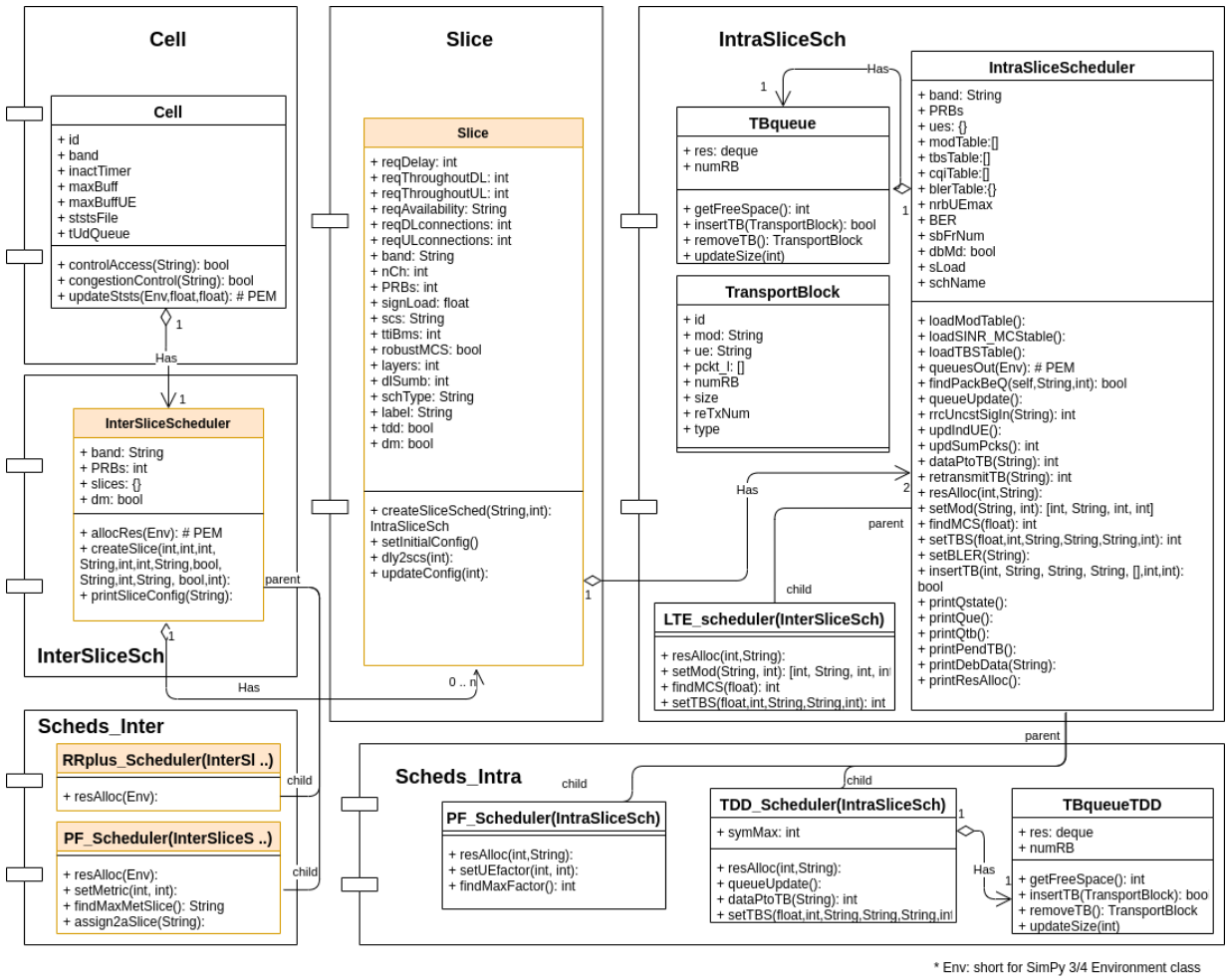


Figure 4: Cell, Slice, and Schedulers modules class diagrams. Cell module defines cell configuration, statistics management, and base interslice scheduler configuration. On the other hand, IntraSliceSch is the base of intraslice scheduler configuration. Sched_inter and Sched_intra have other inter or intraslice schedulers configuration, respectively.

between the different configured Slices. The second one manages Slices requirements and translates to Slice configuration (each Slice is associated with an instance of intra-slice Scheduler). The simulator allows to configure the time granularity for the inter-slice scheduling decision by setting the *granularity* attribute in the interSliceScheduler class.

The simulator's basic operation can be seen in Figure 5. The application generates a packet flow through the *queueAppPckt* method. Each packet is stored in an application queue, the one that appears first in Figure 5. Then, when the UE reaches the connected state in the cell, the DRB (Data Radio Bearer) is established and its packets go to the bearer queue through the *receivePckt* method. Afterwards, the scheduler assigns resources for all the active bearers and takes packets from there to make TB (Transport Blocks) with an appropriate MCS according to the current UE SINR at that moment and puts them in the Scheduler queue through the *queueUpdate* method. Finally the scheduler takes the TB from the queue at each TTI and sends them through the air interface. The TB are successfully received with a probability of 1-BLER (Block Error Rate). Please note that BLER can be set overwriting the *setBLER* method.

Naturally, 5G networks have been progressively deployed and will coexist for a relatively long time with the existing 4G (LTE/LTE-Advanced) infrastructure. To favor the above transition, we have implemented the LTE_scheduler class, which inherits from IntraSliceSch (see Figure 4). In this simulation environment LTE traffic can be served by a LTE slice in a 5G cell.

More details of the tool design can be found in the master's thesis [27].

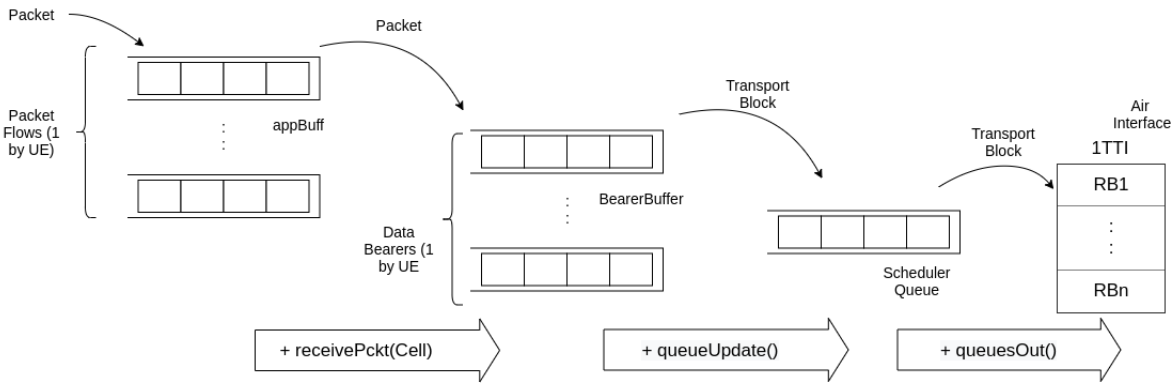


Figure 5: Intra Slice Scheduler Queues operative.

4 Py5cheLiSA, Library for Scheduling Algorithms

With the emergence of 5G, as we mentioned in Section 2, a large amount of software has been developed to simulate the behavior of these networks. Considering the number of features that the network must handle, these simulators become very complex when incorporating new custom developments. In particular, users must study the extensive software code to develop schedulers. In addition to the complexity of the assignment algorithm, understanding the code is challenging, as it is often unclear and far-fetched. One of the main objectives of Py5cheSim is to make it accessible to the user regardless of their programming knowledge.

To achieve that, we have developed a library (Py5cheLiSA) that allows the end-user to implement algorithms efficiently and straightforwardly, without the need to study the operation of the entire simulator. Our library is an open-source project for the Py5cheSim and presents a valuable set of tools for implementing intra-slice and inter-slice schedulers. In terms of metric calculations (essential in scheduling algorithms), we introduce a novel approach that addresses any complex assignment algorithm from a matrix analysis. Furthermore, the library includes a module that facilitates the development of schedulers based on AI. To make it user-friendly, we have documented all the library projects with pydoctor.

Following, we will present Py5cheLiSA, which can be divided into two interconnected parts, one aimed at facilitating the development of intra-slice algorithms and the other for inter-slice ones.

4.1 Py5cheLiSA intra-slice

It consists of three modules: Scheduler Utils, UE statistics, and AI Schedulers Utils. They are designed for complementary use, "a module, a functionality", according to the user's needs. In Figure 6 we show an overview of this library.

4.1.1 Lib Utils, allocation by user metrics

Scheduler Utils' top priorities are to be easy, versatile, and efficient. As the core of the library itself, it contains the main tools for developing resource allocation algorithms, which are implemented through direct interaction with the Py5cheSim main classes presented in the previous section.

Intra-slice resource allocation is carried out by observing the current state of the devices, which depends on a previously defined metric. We can consider the scheduler as a system that has as input the set of measures for each user, and as output a grid with the allocation of resources for each user. In traditional scheduling algorithms, PRBs allocation is needed to identify the UE with the highest metric (which can depend on the PRB to be allocated).

Based on the work of our students [28], considering that matrix manipulation provides a powerful and compact way for the end-user to solve problems with ease, we have included in this module of the library a novel approach that allows calculating the metrics of each of the UE devices. The authors of [28] presented the matrix-based algorithm and its implementation designed for the 5G Vienna simulator in Matlab. In our implementation, the backbone of this matrix system is the Python class framework_UEfactor (see Figure 6), which contains the semantic description of the schedulers' metrics calculation. Moreover, it constitutes the end-users' starting point for developing their own algorithm.

Inside the backbone class, we define a "layer" attribute, a key component for the design of schedulers' metric, as it contains the description of operations performed by the algorithm. A single layer can perform a set of matrix operations from its coefficients. If we use multiple layers consecutively, we can achieve any given

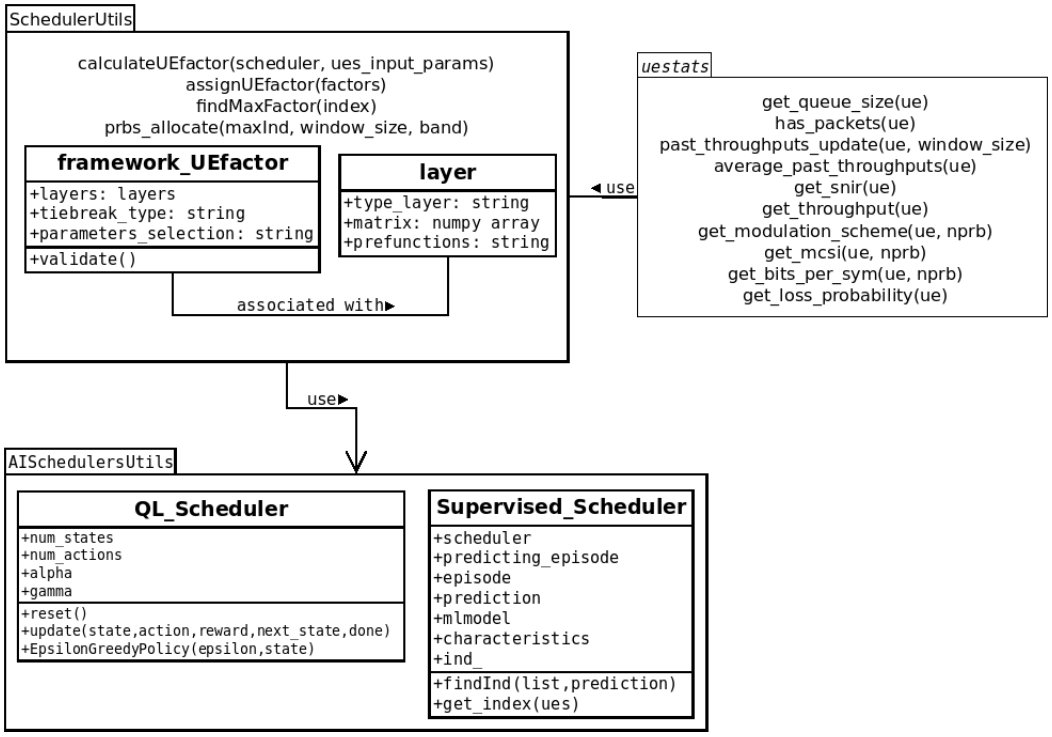


Figure 6: Py5cheLiSA intra-slice architecture.

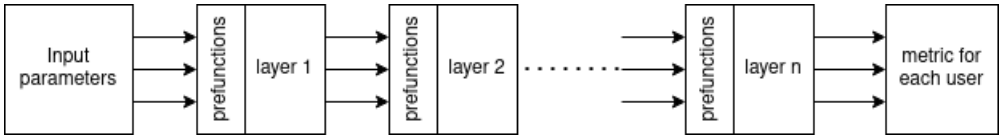


Figure 7: Framework_UEfactor idea.

metric (see Figure 7 for details). Each layer has three input fields: the type of layer, the coefficient matrix, and the pre-functions to be performed. Matrix coefficients' values will depend on the type of operation the layer will do, which is linked to the type of layer.

Formally, we have that given a matrix C of the form

$$C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1m} \\ c_{21} & c_{22} & \dots & c_{2m} \\ \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nm} \end{bmatrix}$$

we can define two types of layers depending on the metric calculation:

- Linear combination of inputs (type S): The output Y will be defined as

  $$Y = CX$$

  or

- Product of inputs raised to exponents (type P): In this case, the output will also be an array, where the y_i component will have the form

  $$y_i = \prod_{j=0}^{n} x_j^{c_{i,j}}.$$

Even though we can obtain almost any metric that we propose using this method (working as a cascade of layers as is shown in Figure 7). We may be interested in performing functions that are not contemplated in the layer types, such as trigonometric, exponential, and logarithmic functions. For these particular cases,

we allow the pre-functions field. Through it, the specific function is applied to the layer input x_i, before performing the matrix operation. In Section 5, we will show an example of this metric calculation procedure applied to different scheduling algorithms.

4.1.2 Getting the statistics

As we have mentioned earlier, the allocation of resources is based on the system's current state. Taking this into account, we have developed the Statistics module (UE statistics), which provides the end-user with tools to obtain measurements of the final wireless devices. It is helpful to access the UEs information for the assignment and later analysis of algorithm performance. This module has potential since it allows the user to get information of different statistics without knowing in detail the core of the Py5cheSim. The standard use of this module is in conjunction with the Utils module, as shown in Figure 8. Moreover, as seen in the image, the former obtains the measurements directly from the simulator and exchanges them in order to be user by the main module of the library. The latter will also write a statistic that contains the description of the metric for each user at each TTI.

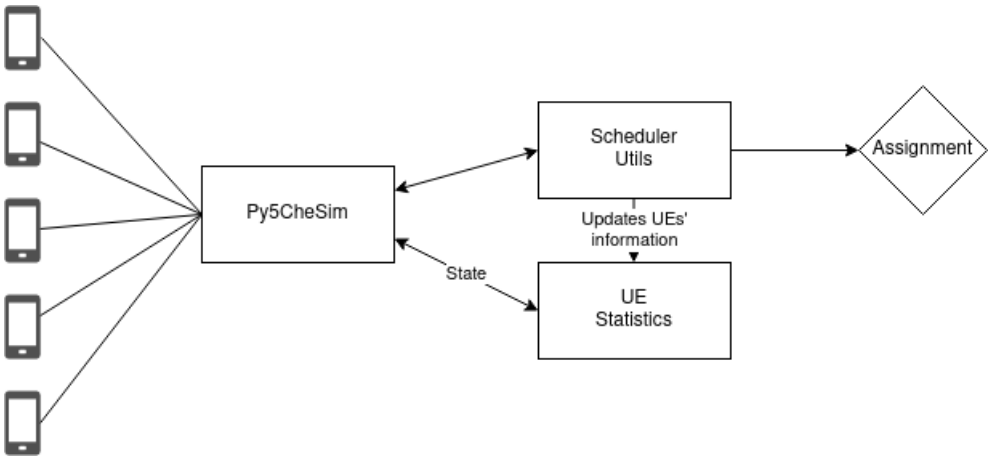


Figure 8: Allocation system.

Throughput, SINR, size of the packet queue of devices are some examples of what this module can obtain. This module has potential to continue growing, adding new statistics in future versions.

4.1.3 Introducing AI

Scheduling involves solving the problem of allocating a finite resource among a set of users. This problem is compounded when considering different traffic profiles, as in 5G, which has different service requirements. As a consequence, two possible approaches are handled for the development of an algorithm:

- solving the concern as a conventional optimization problem. In this case, an assignment would be obtained through a metric given by the solution, which could be implemented through the Scheduler Utils module; or
- addressing the problem through AI-algorithms.

Given that the first option can be achieved using the modules already presented, we decided to incorporate a module that eases the development of AI-algorithms. This module contemplates the development of schedulers based on supervised and unsupervised learning. Once the module structure is known, there are no restrictions for the development of AI-based schedulers. We will present a scheduler based on Support Vector Machines (SVM) in the examples section.

4.2 Py5cheLiSA inter-slice

So far, we have described resource allocation for end devices. The Py5cheLiSA Inter-Slice library has three modules: Scheduler Utils, Slice Statistics, and AI Scheduler Utils. Still, the resource scheduling between slices is as crucial as the first presented in previous subsection. Thereby, considering how relevant this assignment is, we have also developed a set of tools so that the end-user can develop new inter-slice assignment algorithms without dwelling on other details of the simulator. The inter-slice scheduling scenario shares many characteristics with intra-slice scheduling, so we will not delve into repeated details from the modules in the previous section.

4.2.1 Utils for slice scheduling

A parallelism can be drawn between devices and slices. The allocation of resources between devices must be done with the resources of the slice. The resources of the latter are distributed from the cell. To decide the allocation between slices, we can assign each of them a metric in the same way as in the UE case. The metric to follow is calculated as a function of layer statistics. Therefore, it is convenient to have a mechanism that facilitates obtaining the metric and the allocation.

The whole set can also be modeled as a system with measurement parameters as inputs and the assignment for each slice as output. In the inter-slice Scheduler Utils module, we provide support to the end-user through the approach introduced in the subsection 4.1.1. In this regard, as shown in Figure 9, we first create the inter-slice scheduler object, which contains the respective processing layers for the slices. Based on them, we obtain a metric for each slice. Lastly, we carry out the assignment from the slice or slices that maximize the metric.

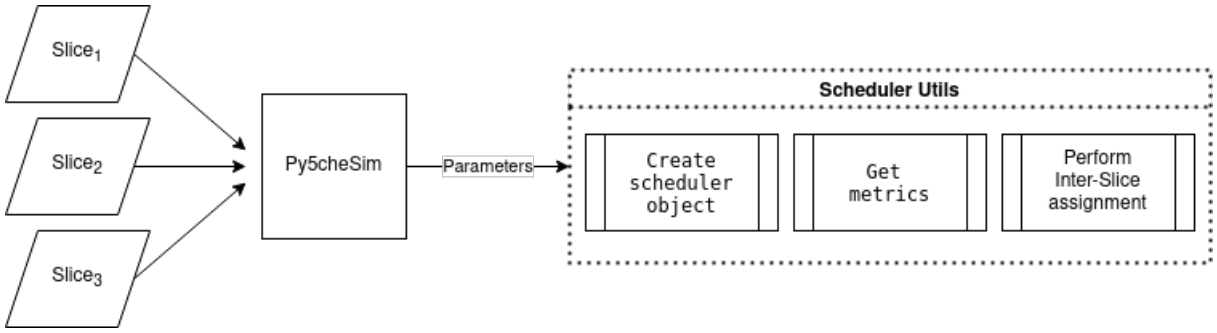


Figure 9: Allocation system for slices.

4.2.2 Measures for slices

Measurements and statistics are essential pieces to carry out the assignment problem. The Slice Statistics module contains an extensive and rich repertoire of functions to obtain them. These measures are directly related to the performance of the slice towards users. Some measurements that the module provides are the number of arrived packets within a specific time window, the number of devices in the slice reaching a certain user-defined level of service, mean spectral efficiency, and others.

4.2.3 AI between slices

At this point, we have the necessary elements in order to develop the inter and intra-slice scheduler. Taking advantage of the Slice Statistics module, we can design models that learn an optimal allocation strategy based on artificial intelligence. To encourage the end-user to develop this type of scheduler, we have created the AI Scheduler Utils module for slices. The end-user can also develop algorithms based on supervised and unsupervised learning with this module.

It is possible to think of the development of algorithms based on supervised learning in two stages. In the first one, the model learns to allocate resources based on a defined algorithm, and the second one predicts the allocation given the statistics. On the other hand, unsupervised learning allows us to find new allocation algorithms that maximize a statistic (reward) defined by the end-user. The system's state will also be given by measurements obtained from the Slice statistics module (or metrics obtained by Utils module), and the action to be carried out will be the assignment.

5 Py5cheLiSA' practical considerations and usage examples

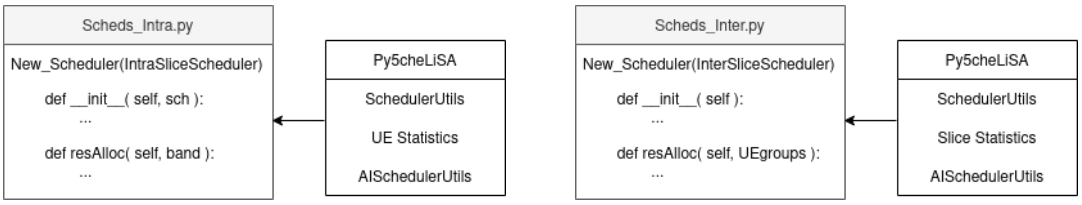
Up to this point we have described the simulator qualitatively, introducing each of the modules and their functionalities. In this section, we present practical aspects for the implementation of new resource allocation algorithms and we introduce some examples that are incorporated into the current version of the simulator.

5.1 Aspects to consider for implementation

To begin with, any new implementation of inter or intra-slice scheduler must be carried out as a new class within the simulator. These classes must maintain a minimum structure, being these inherited from a general class (as is explained in Section 3). For each case, we have the following configuration:

- Intra-Slice schedulers: Any new resource allocation algorithm is hosted in the "Scheds_Intra.py" file of Py5cheSim. The corresponding class inherits from the parent class IntraSliceScheduler. Finally, a "resAlloc" function must be implemented which overwrites a function from its parent class and allows us to develop our own algorithm.
- Inter-Slice schedulers: Every new scheduler has to be implemented in the simulator's "Scheds_inter.py" file. Every new scheduler must be defined as a class inherited from InterSliceScheduler class. Then, in the "resAlloc" function is where the algorithm is implemented, using the library modules as appropriate.

The link between these new classes and the library is outlined in the Figure 10.



(a) Intra-Slice scheduler design (b) Inter-Slice scheduler design

Figure 10: Implementation of new schedulers.

5.2 Some implemented algorithms

In order to boost the end-user experience with Py5cheLiSA, we have made some demonstrative examples that are incorporated in the current version of Py5cheSim. From now on, we will introduce some of these algorithms demonstrating how the library can be used.

5.2.1 Intra-Slice Proportional Fair

The first algorithm that we present is a vanilla Proportional Fair scheduler. Proportional fair seeks to allocate resources efficiently, maintaining a notion of fairness among devices that did not transmit in the recent past. In other words, allocation depends on the relation between the possible throughput a UE can obtain and the past throughput it had; so if no resources have been allocated for a UE in the past, there is more chance to allocate in the present. The metric for every device is chosen as follows:

$$metric_{UE_i} = \frac{actualThroughput^n}{pastThroughtput^m} \tag{1}$$

Where UE_i stands for user i, and n and m weight the final value of the metric. High values of m prioritize those devices who have not transmitted in the recent past, while high values of n prioritize the device with current high throughput.

Using Py5cheLiSA, specifically the corresponding Utils and Statistics modules, we follow these steps to build a Proportional Fair scheduling algorithm:

1. We firstly define a scheduler object with a single layer of type P with coefficients $[n, -m]$. Where n and m are arbitrary values defined by the user. This layer mimics the equation 1.
2. Then, we collect the two measurements from every device. We achieve this through the functions "get_throughput" and "average_past_throughputs" from UE Statistics module.
3. After that, from the defined scheduler and the measurements obtained, we call the "assignUEfactor" function of the Scheduler Utils module. This function gets the metric for each user. The Figure 11 shows the process of obtaining the metric.
4. Last but not least, we look for the maximum metric of the users and finally assign the resources to each user through the function "prbs_allocate" (from Scheduler Utils).

This algorithm is simple and efficient, but it also serves as the basis for developing any new scheduler modifying the input parameters and the processing layers.

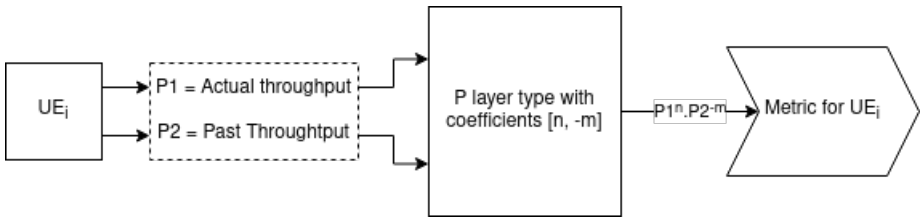


Figure 11: Proportional Fair layer.

5.2.2 Intra-Slice "Proportional Fair" with AI

As mentioned in Section 4 Py5cheLiSA provides a module for the development of schedulers based on artificial intelligence. It may be interesting to apply supervised learning to obtain a model that predicts allocation only by observing some measures of the UEs. One of the great advantages is that having a learned model accelerates the allocation times, since a metric does not have to be obtained through calculations to later determine the allocation, but rather is predicted based on the learned model.

As a design example we have developed a supervised learning algorithm. The implemented example is based on a Support Vector Machine (SVM) algorithm and learns from a vanilla Proportional Fair scheduler. The pseudocode can be seen in Algorithm 1

Algorithm 1 Pseudocode for AI scheduling

```
i ← 0
N_p ← n                                              ▷ learning episodes
scheduler ← framework_UEfactor object
svm_scheduler ← supervised_scheduler object

while Simulation continues do
    new_metric ← scheduler.get_metric( parameters )
    if i ≤ N_p then
        svm_scheduler.data ← parameters
        svm_scheduler.labels ← new_metric
    end if
    if i = N_p then
        svm_scheduler.mlmodel ← learning method
        svm_scheduler.mlmodel.fit(svm_scheduler.labels, svm_scheduler.data)
    end if
    assing_prbs( svm_scheduler.get_index() )
    i ← i + 1
end while
```

Note that svm_scheduler.get_index() always returns the index of the UE with the highest metric regardless of the episode number. If it is less than the prediction episode (N_p), it simply returns the maximum value learned by the scheduler object. Otherwise, it returns the prediction. It is worth mentioning that the choice to use SVM was arbitrary, but other techniques such as logistic regression or similar could be explored with the same library making slight modifications.

5.2.3 Inter-Slice Q-Learning scheduler

Classical scheduling approaches may be impractical in some 5G systems, where complex and dynamic scenarios are involved. Hence, new reinforcement learning techniques are increasingly applied in this context. In Py5cheSim we incorporate an implementation of a Q-Learning based scheduler. This scheduler is based on the work in [29] as we follow the same (state - action - reward) tuple (see Table 1).

Table 1: State, action, reward tuple.

State	Number of arrived packets in each slice within a specific time window
Action	Allocated bandwith to each slice
Reward	Weighted sum of SE and QoE among all slices

In Algorithm 2 we present the pseudocode of our implementation. The comments on the right refer to the library's module from which the function is obtained. As mentioned above, the code for this implementation is in Schds_inter.py. It uses the functionalities provided by the Py5cheLiSA modules but does not restrict the user from using any type of measure for state or reward. Thus, the user has complete freedom to experiment with new algorithms. Furthermore, observing the need to use this scheduler for large numbers of slices and UEs, we also developed a version based on deep q-learning, that can also be found in the included examples.

Algorithm 2 Pseudocode for Q-Learning approach

```
ql_model ← qlearning_scheduler object
reward, state, new_state, action ← init

while Simulation continues do
    current_state ← get_state()                          ▷ Slice Statistics
    action ← ql_model.EpsilonGreedyPolicy()              ▷ AI Scheduler Utils
    resource_allocate( action )                          ▷ Scheduler Utils
    new_state ← get_state()
    reward ← calculate_reward()                          ▷ Slice Statistics
    ql_model.update( state, action, reward, new_state)
end while
```

6 Validation and results

The validation was made through the 5G-LENA module and the throughput calculator web tool from https://5g-tools.com/5g-nr-throughput-calculator/ (the last one represents a quick comparison with known analytical results). There is also no free tool to compare ourselves in multi-slice scenarios, we validated the performance emulating scenarios and proposals of related previous works.

The general idea behind this validation was to test the developed simulator's operation and compare performance results with the references in terms of the main KPI considered, using the same configuration scenarios. Py5cheSim AMC (Adaptive Modulation and Coding) and TBS calculation was adjusted to LENA-5G's as much as possible for this purpose. However as Py5cheSim makes a high degree of simplification of NR procedures compared to LENA-5G, an error margin should be expected. The goal is to not exceed a 10% error margin when comparing simulations considering the same configuration parameters.

Simulator validation was made in two levels: an intra-Slice validation level, and an inter-Slice validation level. In the first case, validation was made in terms of: AMC operation and resulting TBS, throughput measures, different implemented intra-slice schedulers operation and supported features. In the second case, validation was made in terms of slice management and different inter-Slice schedulers operation. The validation and calibration process has been a comprehensive check verifying in a wide variety of scenarios. Following, we present some representative results of both levels.

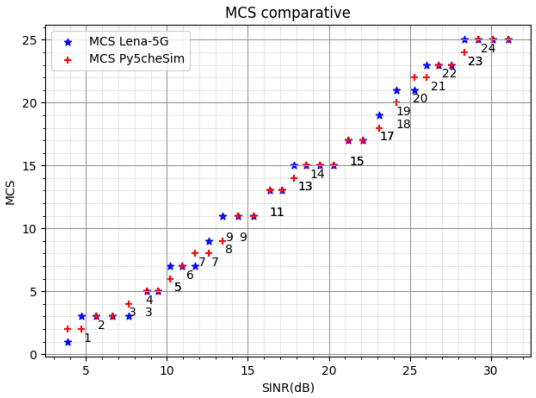
6.1 Intra-Slice validation

6.1.1 NR AMC and TBS Validation

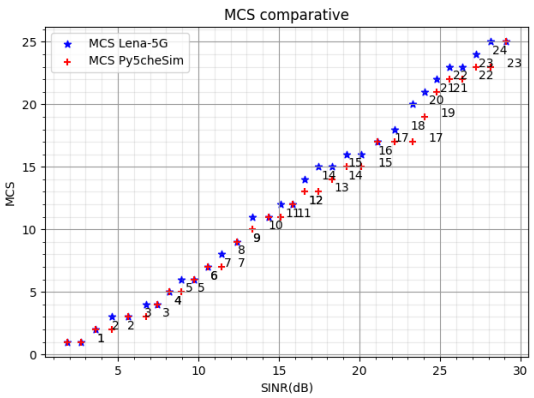
On the one hand, we compare MCS allocation for a wide range of SINRs, using the same band and bandwidth, no CA nor MIMO, and only one UE with full buffer DL and UL traffic profile. Graphics of Figure 12 show some validation results comparing Py5cheSim and the results of script cttcnrdemo.cc of 5G-LENA. On the other hand, using the same idea as above, TBS comparative results can be seen en Figure 13. As it is shown, MCS and TBS are allocated mainly according to *5G-LENA* results, as expected considering how AMC was implemented in the developed simulator.

6.1.2 Throughput Validation

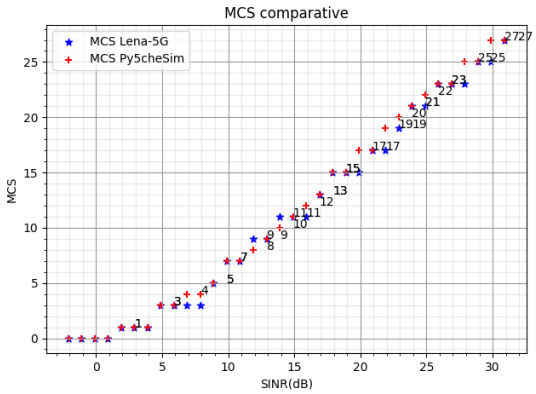
Continuing with intra-Slice level validation, we validate throughput in two ways: comparing the former script results with Py5cheSim's and comparing the latter with the web throughput calculation tool. In Figure 14 we show some throughput results for a downlink and uplink scenario (a complementary analysis of the case of Figures 12 (a) and (b)). Differences observed respond mainly to different MCS allocated, due to the implementation's high degree of simplification in NR procedures. AMC implementation can be improved rewriting the *setMCS* method. Differences can also be present for the same MCS, due to the differences in



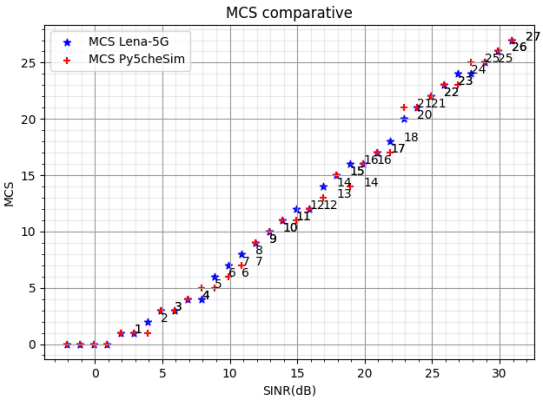
(a) 10 MHz FDD cell with DL full buffer traffic.



(b) 10 MHz FDD cell with UL full buffer traffic.



(c) 100 MHz TDD cell with DL full buffer traffic.



(d) 100 MHz TDD cell with UL full buffer traffic.

Figure 12: Py5cheSim vs 5G-LENA MCS comparison.

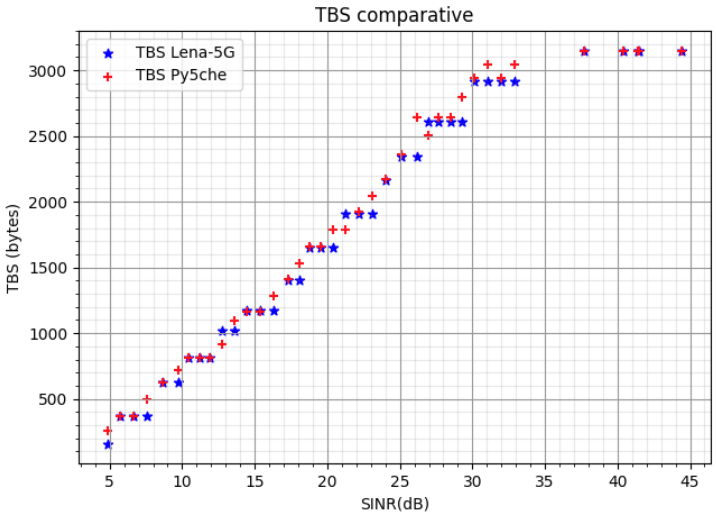
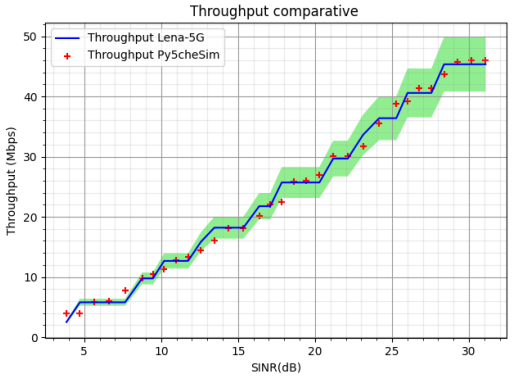
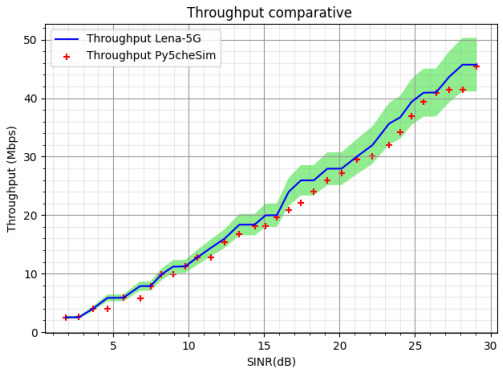


Figure 13: Py5scheSim vs 5G-LENA TBS example.



(a) 10 MHz FDD cell with DL full buffer traffic.



(b) 10 MHz FDD cell with UL full buffer traffic.

Figure 14: Py5cheSim vs 5G-LENA Throughput comparison.

Table 2: DL TDD Throughput comparison for 60 kHz SCS.

Bandwidth	Py5cheSim	Throughput Calculator	Relative error
50 MHz	255 Mbps	270 Mbps	6%
100 MHz	509 Mbps	538 Mbps	5%
200 MHz	999 Mbps	1078 Mbps	7%

the TBS calculation method. However, as can be observed in throughput figures, it remains under the error margin considered.

Tables 2 and 3 show the result of maximum throughput comparison with the web throughput calculation tool (note that in this case we show a TDD escenario). Throughput is calculated considering the highest MCS in use, and no CA nor MIMO, for different bandwidths and SCS, covering FDD and TDD cases. Error is calculated as the difference between results relative to Throughput calculator's value.

Table 3: UL TDD Throughput comparison for 60 kHz SCS.

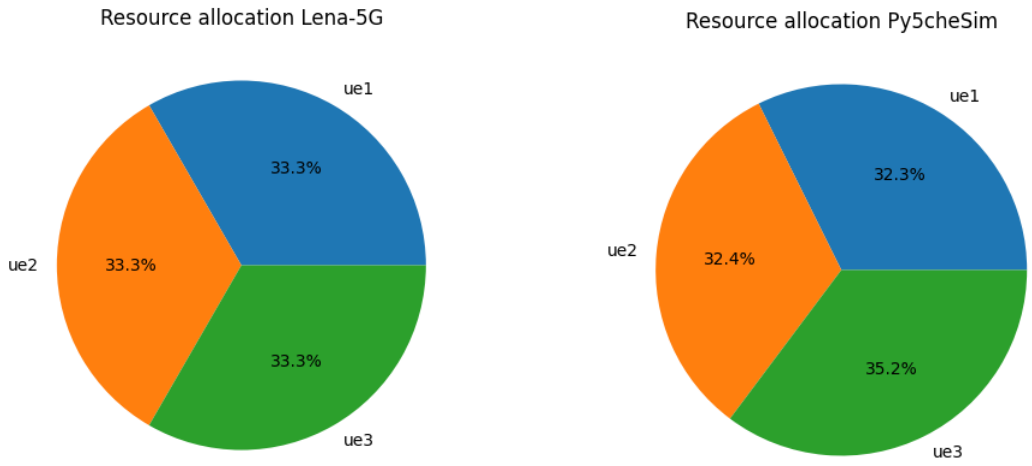
Bandwidth	Py5cheSim	Throughput Calculator	Relative error
50 MHz	282 Mbps	296 Mbps	4%
100 MHz	559 Mbps	592 Mbps	6%
200 MHz	1093 Mbps	1182 Mbps	8%

6.1.3 Basic Schedulers Validation

In this validation we consider one cell (with one slice) and more than one UE with a full buffer traffic profile. We tested Round Robin (RR) and Proportional Fair (PF) considering different exponent values schedulers. Figure 15 and 16 present representative results; in the first one the three UEs have very different SINR. Left, resource usage for the Proportional Fair of 5G-LENA and right the same but by Py5cheSim using numerator and denominator exponent equal to 1. On the other hand, Figure 16 presents an example using Round Robin scheduler.

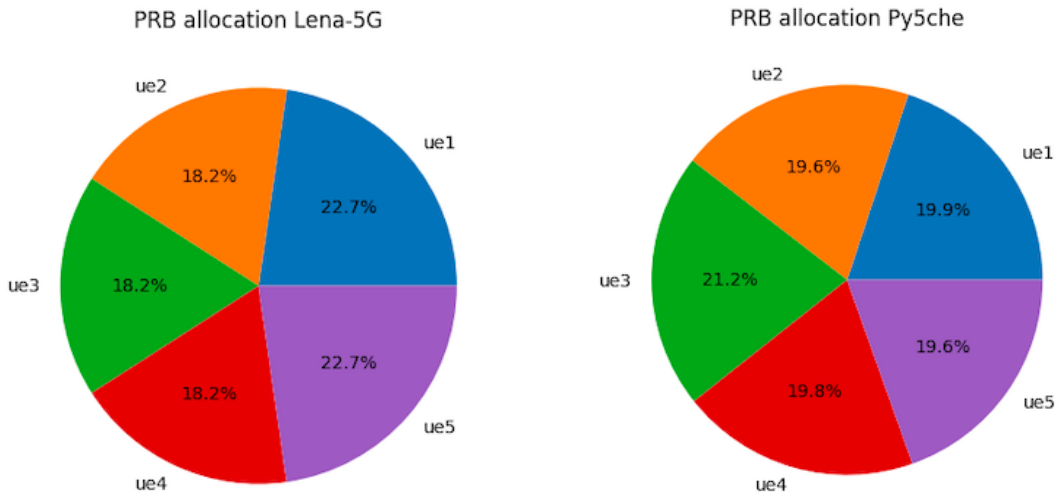
6.1.4 Features Validation

In the following, MIMO and CA features' validation results are presented. As an example we present a complete analysis for SU-MIMO validation. One UE full buffer traffic simulation with different number of supported layers was executed, and obtained results were compared with those of the throughput web calculation tool, under the same conditions (Qm, Rmax, SCS, bandwidth, and useful symbols by slot). Tables 4 and 5 show obtained results for a 10 MHz FDD cell, and 6 and 7 for a 100 MHz TDD cell. On the other hand, Tables 8 and 9 present some obtained results of CA validation.



(a) (b)

Figure 15: Py5cheSim vs 5G-LENA FDD PF11 resource use distribution.



(a) (b)

Figure 16: Py5cheSim vs 5G-LENA FDD RR uplink resource use distribution.

Table 4: FDD DL Throughput comparison using SU-MIMO for 4 and 8 layers, and 10 MHz bandwidth.

Layers	Py5cheSim	Throughput Calculator	Relative error
4 layers	210 Mbps	222 Mbps	5%
8 layers	422 Mbps	446 Mbps	5%

Table 5: FDD UL Throughput comparison using SU-MIMO for 4 and 8 layers, and 10 MHz bandwidth.

Layers	Py5cheSim	Throughput Calculator	Relative error
4 layers	224 Mbps	238 Mbps	6%
8 layers	450 Mbps	476 Mbps	6%

Table 6: TDD DL Throughput comparison using SU-MIMO for 4 and 8 layers, and 100 MHz bandwidth.

Layers	Py5cheSim	Throughput Calculator	Relative error
4 layers	2033 Mbps	2154 Mbps	6%
8 layers	4026 Mbps	4310 Mbps	7%

Table 7: TDD UL Throughput comparison using SU-MIMO for 4 and 8 layers, and 100 MHz bandwidth.

Layers	Py5cheSim	Throughput Calculator	Relative error
4 layers	2226 Mbps	2366 Mbps	6%
8 layers	4429 Mbps	4730 Mbps	6%

Table 8: FDD DL Throughput comparison using different CA combination examples.

Configuration	Py5cheSim	Throughput Calculator	Relative error
10 + 10 MHz	106 Mbps	112 Mbps	5%
10 + 10 + 10 MHz	159 Mbps	166 Mbps	4%
20 + 20 MHz	216 Mbps	226 Mbps	4%

Table 9: FDD UL Throughput comparison using different CA combination examples.

Configuration	Py5cheSim	Throughput Calculator	Relative error
10 + 10 MHz	115 Mbps	120 Mbps	4%
10 + 10 + 10 MHz	172 Mbps	178 Mbps	3%
20 + 20 MHz	232 Mbps	242 Mbps	4%

6.2 Inter-Slice validation

From here on, multi-slice validation is described. Note that, as at the moment there is no free multi-slice simulator to compare with, validation will be made considering the expected results according to theory.

6.2.1 Slice Management Validation

It is important to note that, except of resource allocation, Slice configuration is made at the moment of creation, and remains unchanged during the simulation. It is assumed that some service requirements will not change during the simulation. However, as the number of UEs and traffic intensity could change, resource allocation between slices can be updated, with a configurable time granularity. In this implementation Slice configuration is actually made in terms of:

- SCS: According to required delay
- AMC: Normal MCS allocation according to SINR or Robust
- Signalling Load: low or high
- Scheduling Algorithm: Set manually for each UE group in the simulation script
- DL/UL symbol allocation in TDD case
- Allocated PRBs: Resources are equally allocated to different Slices by default, but can change with other InterSlice scheduler implementations.

Table 10: Delay requirement to SCS mapping.

Delay Requirement (ms)	≤ 2.5	≤ 5	≤ 10	> 10
SCS (kHz)	120	60	30	15

Table 10 shows the configured mapping between RAN Delay requirements and SCS configuration for a Slice. Note that the considered thresholds were defined taking into account the delay analysis and results presented in [25] assuming average values for the scheduling timings, and the direct dependence with the slot duration and SCS as a consequence. Different thresholds can be configured modifying the *dly2scs* method in Slice class.

As for signalling load, two levels were considered:

- Normal Signalling Load: used for eMBB and URLLC Slice types
- Low Signalling Load: used for mMTC Slice types

If the service requires high availability AMC, the algorithm is modified to allocate an MCS index lower than the expected for that SINR.

For Slice Management validation simulations were run using different service requirements, and Slice configuration is checked to be consistent with the criteria explained before (see Table 11).

Table 11: Simulation with 3 UE groups in a 20 MHz cell. Service requirements vs Slice configuration.

Slice Name	eMBB-1	mMTC-1	URLLC-1
Delay Requirement (ms)	10	20	2
Reliability Requirement			High
Allocated PRBs	17	35	8
SCS (kHz)	30	15	60
Signalling Load	Normal	Low	Normal
Robust MCS allocation	False	False	True

As can be seen, Slices are configured according to service requirements and the established mapping in the *Slice* class. Resource Allocation between slices is equal in terms of bandwidth, but the number of PRB in each case respond to the numerology used. Note that even though 2 ms delay is required for the URLLC-1 group, 60 kHz SCS is configured. This is because in this simulation the configured band is in FR1, and at least in R15, there is no support for 120 kHz SCS in FR1 bands. When running the same simulation in a TDD cell SCS for URLLC-1 UEs is 120 kHz, and 60 kHz for the other UE groups, as expected.

Inter-Slice Scheduler validation is made by running simulations with more than one UEgroup, and checking that resource allocation between Slices is according to the scheduler algorithm used. Note that at the moment of writing this document, there are no multi-Slice simulators to compare with.

6.3 Inter-Slice Scheduler Validation

At the moment Py5cheSim presents four inter-slice schedulers:

- **Round Robin:** allocates the same amount of resources to the different slices, even if there is no traffic on UE bearer queues.
- **Round Robin Plus:** allocates the same amount of resources to the different slices, only with traffic in UE bearer queues. For example, if there are three Slices, but at the moment of scheduling resources between them, only two has UEs with traffic, band resources will be equally distributed between the other two.
- **Proportional Fair:** with configurable exponents: allocates cell's resources to the Slice with the highest metric.
- **deep-AI:** proposal of [29] based on Deep Reinforcement Learning algorithm.

Note that if slices have different numerology, even if the resources are distributed equally between them, PRBs available will be according to the configured numerology. However we can say that resources are equally distributed because Slices with higher SCS, will have proportionally shorter slots (in time duration).

As an example, we show the results obtained using Round Robin. It was tested by running a simulation with 3 different UE groups with good SINR. UE groups have the same requirement shown in table 11 and Slice configuration is according to that.

Figure 17 shows resource allocation between the three slices. As can be seen, from the cell's 52 PRB (in the 15 kHz reference numerology) 17 are allocated to mMTC-1 slice (15 kHz SCS), 8 to eMBB-1 slice (30 kHz SCS), and 4 to URLLC-1 (60 kHz SCS). As the different slices have different numerologies, allocated

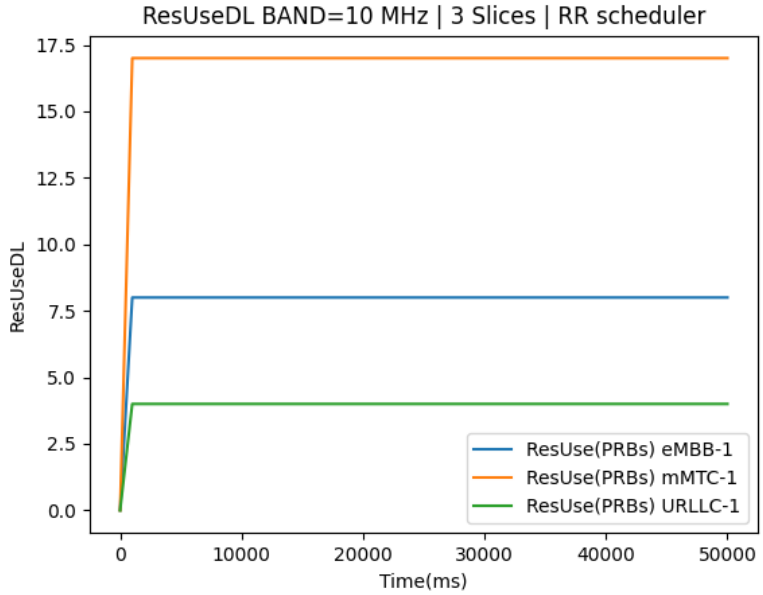


Figure 17: Resource allocation between slices using default Round Robin algorithm, in a 10 MHz FDD cell.

PRB are different between them, but the resources are almost the same. The difference is explained by the trunking made by the inter slice scheduler implementation. The next thing to note is that PRB allocation doesn't change during the simulation, even when mMTC-1 slice has a very relaxed traffic profile.

Figure 18 shows total throughput for each slice. Note that only eMBB-1 slice is using the entire available slice bandwidth. In this case, obtained throughput is according to the expected for a 8 PRB allocation using 30 kHz SCS (17.3 Mbps from web tool). URLLC-1 and mMTC-1 traffic profiles do not have enough intensity to consume their respective slices resources.

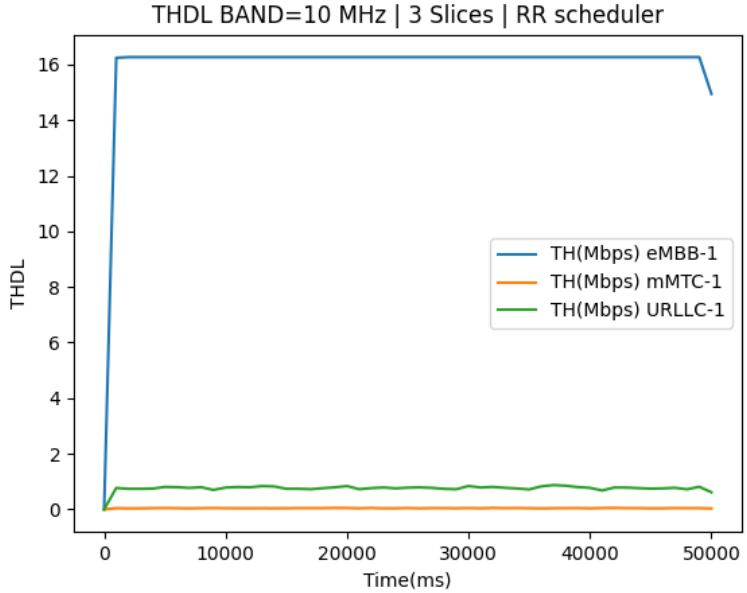


Figure 18: Slices throughput using default Round Robin algorithm, in a 10 MHz FDD cell.

The obtained validation results (see more examples at [27]) show that even with the high level of simplification made on this implementation, differences with those obtained from 5G-LENA and theoretic values are under the considered error margin. Furthermore, different scheduler algorithms were tested at intra and inter slice levels showing results according to the expected ones. In this way, the developed tool is validated and considered adequate for different scheduling algorithms analysis, as long as the defined error margin

could be tolerated. More accurate results could be reached enriching the defined models, adding, however, more complexity to the implementation.

7 Conclusions and future work

In this work we have introduced Py5cheSim, a new discrete event Python simulator focused on cell capacity analysis. Py5cheSim, and Py5cheLiSA, constitutes a simple environment to develop and test new 5G scheduler algorithms (inter and intra-slice). We have presented validation results that show near compliance with 3GPP requirements and the reference simulator.

The architecture and capabilities of Py5cheSim have been presented in order to help researches understand the level of detail and get a clear idea of its functionalities. The simulator is available on our web page [30] [1]. We are working on a new version of the simulator, improving some features, and including new ones as mini-slots support, user mobility possibilities, intelligent link-adaptation decisions, massive MIMO support, etc.

Acknowledgements

This work has been partially supported by the Agencia Nacional de Investigación e Innovación, Uruguay (Project FMV_1_2019_1_155700, "Artificial Intelligence applied to 5G networks") and Fondo Carlos Vaz Ferreira, Convocatoria 2021, Dirección Nacional de Innovación, Ciencia y Tecnología, Ministerio de Educación y Cultura, Uruguay (Project FVF-2021-128– DICYT). The authors would like to thank Víctor González Barbone for helpful discussions.

References

[1] ITU-R, "Imt vision – framework and overall objectives of the future development of imt for 2020 and beyond," International Telecommunication Union, Geneva, Recommendation M.2083-0, Set 2015.

[2] "Description of network slicing concept," NGMN P1 WS1 E2E Architecture Team,NGMN Alliance, White Paper, Jan 2016.

[3] "Network slicing for 5g networks & services," 5G Americas, White Paper, 2016.

[4] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, and T. Braun, "Network slices toward 5g communications: Slicing the lte network," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 146–154, 2017.

[5] S. Zhang, "An overview of network slicing for 5g," *IEEE Wireless Communications*, vol. 26, no. 3, pp. 111–117, 2019.

[6] 3GPP, "NR; NR and NG-RAN Overall description; Stage-2," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.300, 03 2021, version 16.5.0.

[7] ——, "NR; Physical layer; General description," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.201, 01 2020, version 16.0.0.

[8] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, "5g: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 6, pp. 1201–1221, 2017.

[9] G. Pereyra, C. Rattaro, and P. Belzarena, "A 5g multi-slice cell capacity framework," Aug 2021, poster presented at ACM SIGCOMM 2021 Networking Networking Women Professional Development Workshop (N2Women'21).

[10] ——, "Py5chesim: a 5g multi-slice cell capacity simulator," in *2021 XLVII Latin American Computing Conference (CLEI)*, 2021, pp. 1–8.

[11] N. Patriciello, S. Lagen, B. Bojovic, and L. Giupponi, "An e2e simulator for 5g nr networks," *Simulation Modelling Practice and Theory*, vol. 96, p. 101933, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X19300589

[12] "An open source model for the simulation of lte handover scenarios and algorithms in ns-3," p. 289–298, 2013. [Online]. Available: https://doi.org/10.1145/2507924.2507940

[1] https://iie.fing.edu.uy/investigacion/grupos/artes/proyectos/inteligencia-artificial-aplicada-a-redes-5g/

[13] M. Mezzavilla, M. Zhang, M. Polese, R. Ford, S. Dutta, S. Rangan, and M. Zorzi, "End-to-end simulation of 5g mmwave networks," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2237–2263, 2018.

[14] M. K. Müller, F. Ademaj, T. Dittrich, A. Fastenbauer, B. R. Elbal, A. Nabavi, L. Nagel, S. Schwarz, and M. Rupp, "Flexible multi-node simulation of cellular mobile communications: the Vienna 5G System Level Simulator," *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, no. 1, p. 17, Sep. 2018.

[15] S. Pratschner, B. Tahir, L. Marijanovic, M. Mussbah, K. Kirev, R. Nissel, S. Schwarz, and M. Rupp, "Versatile mobile communications simulation: the Vienna 5G Link Level Simulator," *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, no. 1, p. 226, Sep. 2018.

[16] G. Nardini, G. Stea, A. Virdis, and D. Sabella, "Simu5g: A system-level simulator for 5g networks," 07 2020.

[17] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis, "Simu5g–an omnet++ library for end-to-end performance evaluation of 5g networks," *IEEE Access*, vol. 8, pp. 181 176–181 191, 2020.

[18] Y. Kim, J. Bae, J. Lim, E. Park, J. Baek, S. I. Han, C. Chu, and Y. Han, "5g k-simulator: 5g system simulator for performance evaluation," in *2018 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2018, pp. 1–2.

[19] J. Baek, J. Bae, Y. Kim, J. Lim, E. Park, J. Lee, G. Lee, S. I. Han, C. Chu, and Y. Han, "5g k-simulator of flexible, open, modular (fom) structure and web-based 5g k-simplatform," in *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2019, pp. 1–4.

[20] S. M. A. Zaidi, M. Manalastas, H. Farooq, and A. Imran, "Syntheticnet: A 3gpp compliant simulator for ai enabled 5g and beyond," *IEEE Access*, vol. 8, pp. 82 938–82 950, 2020.

[21] F. Kaltenberger, G. d. Souza, R. Knopp, and H. Wang, "The openairinterface 5g new radio implementation: Current status and roadmap," in *WSA 2019; 23rd International ITG Workshop on Smart Antennas*, 2019, pp. 1–5.

[22] F. Kaltenberger, A. P. Silva, A. Gosain, L. Wang, and T.-T. Nguyen, "Openairinterface: Democratizing innovation in the 5g era," *Computer Networks*, vol. 176, p. 107284, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128619314410

[23] T. SimPy, "Documentation for simpy," *https://simpy.readthedocs.io/en/latest/contents.html*, 2020.

[24] 3GPP, "NR; User Equipment (UE) radio access capabilities," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.306, 03 2021, version 16.4.0.

[25] ——, "NR; Physical layer procedures for data," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.214, 03 2021, version 16.5.0.

[26] G. Bounous, L. Silva, and M. Rodríquez, "Modelado y planificación de redes lte ltest," oct 2012. [Online]. Available: https://iie.fing.edu.uy/publicaciones/2012/BSR12/

[27] G. Pereyra, "Scheduling in 5g networks : Developing a 5g cell capacity simulator," Master's thesis, Universidad de la República (Uruguay). Facultad de Ingeniería. IIE, 2021. [Online]. Available: https://iie.fing.edu.uy/publicaciones/2021/Per21/

[28] E. Davyt, A. Muzante, and M. Rizzo, "A5ignator : Framework para la implementación de algoritmos de asignación de recursos en 5g," feb 2021. [Online]. Available: https://iie.fing.edu.uy/publicaciones/2021/DMR21/

[29] R. Li, Z. Zhao, Q. Sun, C.-L. I, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74 429–74 441, 2018.

[30] G. Pereyra, L. Inglés, C. Rattaro, and P. Belzarena, "Py5chesim," https://github.com/ClaudinaRattaro/Py5cheSim, 2021.