PyWiSim: Python wireless simulation framework for multislice systems

Pablo Belzarena*, Victor Gonzalez-Barbone* and Claudina Rattaro*
*Facultad de Ingeniería, Universidad de la República, Uruguay

Emails: {belza,vagonbar,crattaro}@fing.edu.uy

***Abstract*—This paper introduces PyWiSim, a Python-based simulation framework designed for wireless systems that falls somewhere between a link simulator and a system simulator. Link simulators model all communication layers in detail, making large-scale simulations computationally expensive. On the other hand, system simulators typically perform throughput calculations for a given simulation scenario, allowing simulations with many devices but providing little detailed information. With this compromise between these two classes of simulators, PyWiSim seeks a simulator that enables simulations with a large number of devices but modeling the most relevant aspects of the system with a certain level of detail. This framework is well-documented and allows for the easy addition of new wireless channel models, traffic generators, scheduling algorithms, etc. Being built in Python—a language widely used in artificial intelligence (AI) applications—PyWiSim facilitates the natural integration of AI-based algorithms into wireless simulations. To demonstrate this versatility, we present an example of a scheduler developed using deep reinforcement learning, specifically the Deep Q-Network (DQN) algorithm. It natively supports multislice, a fundamental feature of modern networks like 5G, and provides a flexible architecture that allows extensions to various wireless technologies, as demonstrated in this paper. Finally, we also present some graphical results obtained from PyWiSim to illustrate its capabilities.**

***Index Terms*—Simulation, Wireless Netwoks, Framework**

I. Introduction

The requirements of mobile services have changed over the years. 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, new technologies are launched every few years. In mobile cellular technologies, the fifth generation of mobile communications (5G) is a reality using the new 3GPP technology (3rd Generation Partnership Project). Today, researchers are working on next-generation mobile technologies, usually called sixth generation and beyond. In the same way, 802.11 new standards are also proposed every few years. Additionally, new proposals are discussed for other technologies such as: UAV (Unmanned Aerial Vehicle) networks, satellite constellations communications, etc. In addition, artificial intelligence (AI) is increasingly used in different parts of the network. These emerging and future technologies demand advanced simulation tools to design novel algorithms and evaluate upcoming network deployments.

All of the above creates substantial restrictions on simulation tools. On the one hand, link level network simulators implement all layers of communication (for example, ns-3 [1]). These tools are excellent for simulating a link or a few links but are impractical if a massive simulation of devices is required. On the other hand, system-level simulators allow massive simulations but have highly simplified system representations, typically using a few formulas to calculate the expected throughput in a specific simulation scenario. A simulator is required that allows large-scale simulations but incorporates the most relevant aspects of the system without implementing all the communication layers.

Furthermore, adding a new technology in many simulators implies rewriting all the communication layers one by one, which represents a significant effort and a very high set-up time. However, many relevant architectural aspects are common to all technologies. A simulator is required that is easy to extend to incorporate new technologies, allows for massive simulations, and incorporates essential aspects of wireless communications.

As previously mentioned, with AI playing an increasingly central role in networks—particularly in monitoring, resource allocation, and fault detection—future network simulation tools should be developed in programming languages that facilitate seamless integration with AI frameworks. Moreover, support for managing multiple slices in the wireless access network is essential, as this capability is expected to be a key feature of most emerging technologies.

To address these requirements, we developed PyWiSim: a Python-based wireless simulator. This simulator is based on a previous simulator explicitly developed for 5G: Py5cheSim (Python 5G Scheduler Simulator) [2]. PyWiSim incorporates the functionalities of Py5cheSim as an extension of the simulator but presents a general architecture that allows an easy extension for 5G; extensions can also be made for other technologies. The focus is on developing a framework to simulate wireless networks with a simple and clean architecture that is well documented, allows it to be easily extended, is written in Python, and, therefore, has simple interfaces to incorporate artificial intelligence libraries. PyWiSim is a free and open-source project under GNU license [3]. In this article, we present the main features of the tool, which is suitable for both educational and research purposes, and illustrate its flexibility through several usage examples. The open-source repository of PyWiSim is available at: https://gitlab.fing.edu.uy/vagonbar/simnet.

The rest of the paper is organized as follows. Section II provides a brief overview of existing wireless network simulators, with an emphasis on those supporting modern technologies such as 5G. In Section III, we describe the architecture of the PyWiSim framework. Section IV details how PyWiSim handles data traffic through its packet queue mechanism, including transmission, retransmission, and packet loss management. Section V explains how different wireless channel models can be integrated into PyWiSim. In Section VI, we present an example scheduler developed using deep reinforcement learning (specifically, the Deep Q-Network algorithm), showcasing the framework's compatibility with AI-based approaches. Section VII demonstrates how PyWiSim can be extended to support new wireless technologies, using 5G as a case study. In Section VIII, we present simulation results that illustrate PyWiSim's capabilities. Finally, Section IX concludes the paper.

II. Related work

One of the main open network simulation tools is ns-3 [1]. Ns-3 is a network simulator, very rich in terms of technologies supported: 802.11, LTE, 5G, etc.. 5G-LENA [4] is a GPLv2 simulator designed as a pluggable module for ns-3. Given its nature as a network simulator, 5G-LENA can take a considerable amount of time to simulate even a few users, making it impractical for certain scenarios. Concerning system-level simulators, one piece of software that stands out is the Vienna Simulator [5] [6]. This MATLAB tool, which is available for download under an academic use license, permits link-level and system-level simulations.

Simu5G is a 5G implementation in OMNeT++ [7]. It simulates the data plane of the 5G Radio Access Network (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 latest version 1.2.2, it does not simulate all possible features in relation to resource allocation like network slicing or mini-slot. Unlike Vienna, which is well tailored for the evaluation of lower-layer procedures, including signal-processing techniques, Simu5G is a discrete-event, application-level simulator.

Other examples of open-source simulation tools include 5G-air-simulator [8], SyntheticNET [9], and 5G-K-Sim [10]. Additionally, there are emulation platforms such as UERANSIM [11] and OpenAirInterface [12], which can provide realistic data. However, they face hardware limitations that hinder their scalability for large-scale simulations.

To the best of our knowledge, none of the open-source existing simulators support Network Slicing at the RAN level. In this context, PyWiSim stands out by enabling not only the management of dynamic slices, but also the analysis of both inter slice and intra slice scheduling mechanisms. Unlike network simulators, PyWiSim does not require layer-by-layer implementation of all procedures defined in the standards, making it a lightweight, faster, and more accessible alternative to many existing open-source tools. Moreover, by leveraging Python's extensive ecosystem of AI libraries, PyWiSim facilitates the integration of AI-based algorithms with minimal effort.

III. PyWiSim design and architecture

PyWiSim includes a general-purpose library called *libsimnet*, which serves as a common foundation for all wireless technologies implemented within the framework. Each technology extends *libsimnet*, inheriting from its classes and overriding the methods it needs. In turn, certain *libsimnet* classes support different models, and these models can be shared between different technologies. New models can be created to add to those already existing in *libsimnet*, such as wireless channel models, scheduler models, traffic generator models, etc. This separation between *libsimnet*, the models, and the extensions is shown in the package diagram of Figure 1. We will now focus on describing the main functionalities and architecture of *libsimnet*. In later sections, we describe the 5G extension and some wireless channel models as examples of how PyWiSim can be extended with new technologies and models. We also illustrate the framework's versatility for incorporating AI-based algorithms.

Figure 2 shows the conceptual view of the main *libsimnet* functionalities. We will begin with a informal description and then look more formally at the fundamental aspects of its architecture. In wireless technologies, there are generally one or more base nodes. In *libsimnet* each radio base has a set of air resources (time, frequency, codes, etc.). In turn, each radio base is divided into a set of slices. Each slice manages a subset of the air resources owned by the radio base. In addition, the radio base is assigned a set of users. According to different criteria (QoS, required delays, etc.), each user is associated with one of the radio base's slices. A central aspect of *libsimnet* is the hierarchical allocation of air resources. First, resources are provisioned across slices (inter slice scheduling), determining how much of the total capacity each slice receives. Then, within each slice, an independent scheduler (intra slice) allocates resources to individual users. Each intra slice scheduler operates at a fixed Transmission Time Interval (TTI), which can differ from one slice to another, and each slice may implement its own resource allocation algorithm. This design allows for flexible, slice-specific QoS strategies and user-level differentiation.

Users will be assigned resources whenever they have data to transmit, downlink or uplink. These data are generated by traffic generators, which can be of different types for each user. The generated traffic is aggregated into packet queues for each user. Transport Blocks (TBs) are built with the assigned resources, which are the Layer 1 data structures that will be sent over the air. A key element in the transition from packets to TBs is the coding and modulation used at Layer 1. Depending on the modulation and coding used, the N bits of a packet will be transformed into M bits of the TB. A key element in determining this N-to-M ratio is the state of the channel in each TTI for each user. If the channel has a high Signal-to-Noise Ratio (SNR) at a given time, one modulation

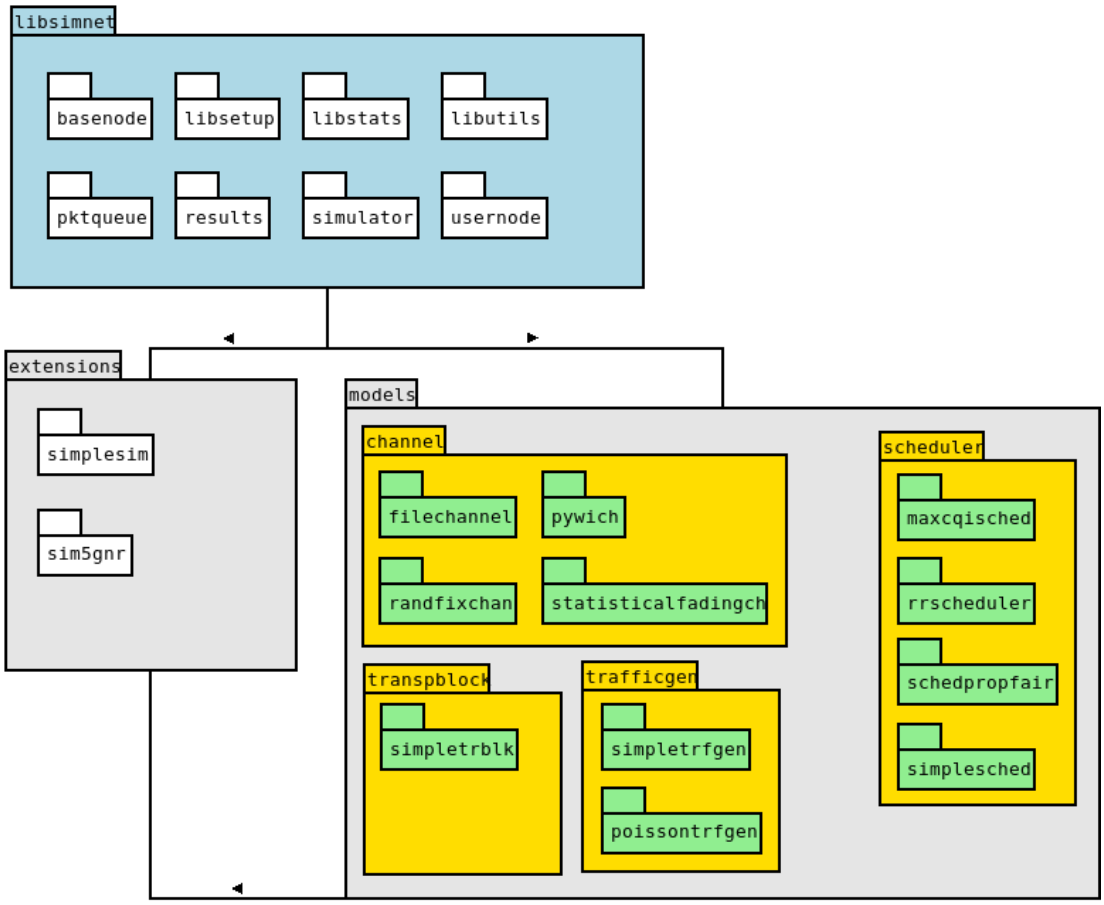

Fig. 1: PyWiSim UML packages diagram.

and coding will be used; if the SNR is low, another will be used, resulting in the same packet having more bits in a TB. Finally, once the TB is transmitted, it may arrive successfully at its destination or be lost due to errors in the air. In the latter case, the TB packets are entered into a retransmission queue and will be sent again in the next TTI.

The *libsimnet* library has five main packages: basenode, usernode, pktqueue, simulator, and results. The main *libsimnet* classes are shown in Figure 3. The simulator package has two main classes: Simulation and Setup. Setup is where the simulation scenario setup is processed. The scenario is specified in a file, and the class SetUp creates the set of objects needed to run the simulation. Simulation is the class responsible for running the simulation. Specifically, it contains the discrete event engine. The results package is responsible for storing the simulation results and the necessary statistics (Section VIII showcases several default graphical outputs available in the current version of the framework).

The basenode package contains, among other things, the BaseStation class. Each BaseStation contains a set of Slice objects and also an InterSliceSched object. The InterSliceSched class is responsible for periodically assigning BaseStation resources to the Slices (inter slice scheduler). Each Slice contains a set of resources (Resource class) assigned by the InterSliceSched algorithm. It will also contain an element of the Scheduler class. The Scheduler class assigns Slice resources to the users associated with the Slice in each TTI (intra slice scheduler). The users associated with the Slice will belong to a user group with their characteristics defined in the UserGroup class.

The usernode package has the UserEquipment class as its main class. The UserEquipment class will have an associated Channel class, which is responsible for obtaining the status of that user's wireless channel with the radio base. It will also have an associated PacketQueue class from the pktqueue package. PacketQueue is the class responsible for storing the packets generated by the TrafficGenerator class. PacketQueue is also responsible for extracting packets to build Transport Blocks (TransportBlock class from the usernode package) and keeping them in the retransmission queue in case they are lost. The next section provides further details on queue management.

The Figure 4 shows a *libsimnet* object diagram showing the relationships between objects in the different classes.

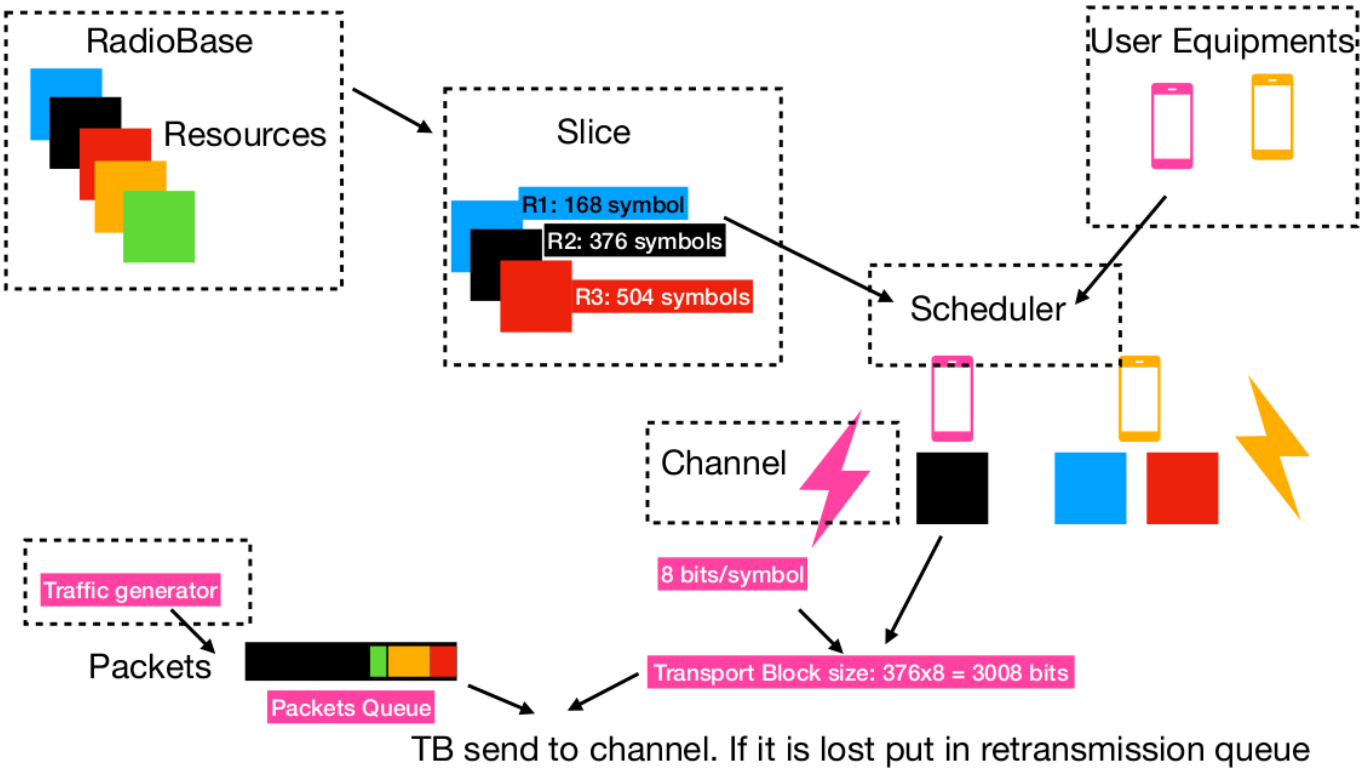

Fig. 2: Conceptual diagram of libsimnet functionalities.

IV. Data traffic handling

In PyWiSim, traffic is simulated in the form of data packets. A data packet is a tuple (identifier, time received, time sent, packet object or number of bits). The packet object may be a pointer to an object of a user defined class, a sequence of bytes or a string, or just the size in bits of the data in the packet. Times of reception and transmission of packets may be recorded as the number of simulation time units since the simulation started, or as real time.

Data packets are generated by a traffic generator, and added to a data packet queue. The PacketQueue class (see Figure 3) is responsible not only for receiving and sending data packets, but also for their transmission in transport block units, and their retransmission in the eventuality that the transport block is reported as lost in transmission. Each user equipment has its own packet queue. Figure 5 shows how the packet queue handles transmission.

The processes of reception and transmission work as follows:

1) Data packets produced by a traffic generator are added to the packet queue, in the order they arrive, into a receive list ("receive list" in Figure 5).
2) Upon request, the packet queue extracts packets from the receive list and adds them into a unit called a "transport block" ("TB-7" in Figure 5). A transport block is implemented as a list of packets related to a transport block unique identifier. To build a transport block, a packet queue function receives the transport block size as a number of bits. The block size is determined according to the resources assigned to the user equipment owning the queue.
3) Each newly built transport block is returned to the caller of the function, but is also kept in a dictionary of pending transport blocks ("dc_pend" in Figure 5). The pending transport block dictionary acts as a storage of "on air" packets, packets that have been transmitted but not confirmed as successfully arrived to destination, and which may be lost.
4) When a transport block is confirmed as successfully sent by the entity responsible for transmission, the transport block is extracted from the dictionary of pending transport blocks ("dc_pend" in Figure 5), its packets are marked as sent with a sent timestamp, and optionally moved to a list of sent packets ("sent list" in Figure 5). Sent packets may also be discarded, once counters have been updated; this may be required by long simulations. The state after successful transmission is shown in Figure 5 under title "Transport block sent".
5) If a transport block is lost, which is informed by the entity responsible for the simulation of transmission, the transport block is extracted from the dictionary of pending transport blocks ("dc_pend" in Figure 5) and its packets are inserted into a dictionary of packets to be retransmitted ("dc_retrans" in Figure 5). The state after a transmission failure is shown in Figure 5 under title "Transport block lost").

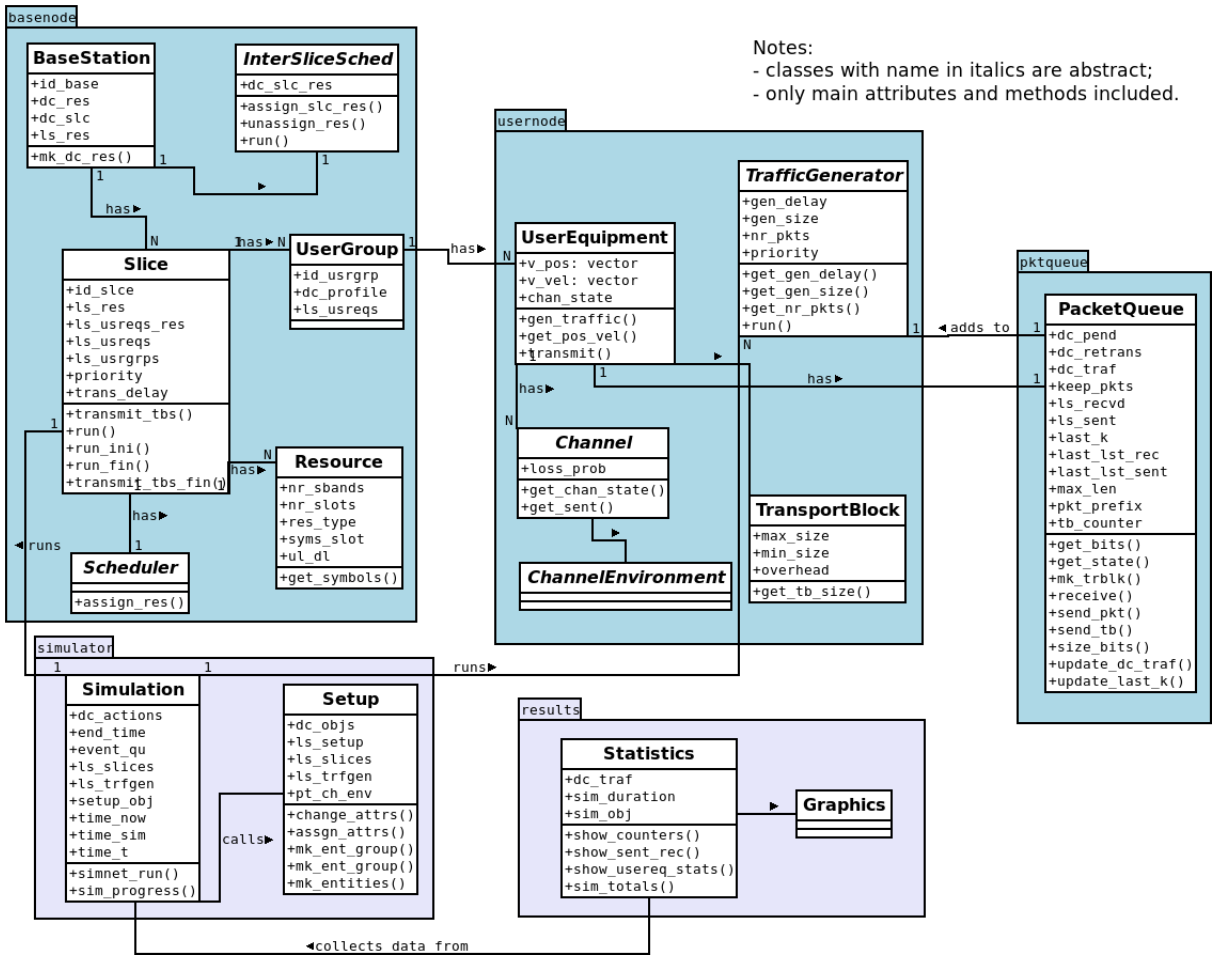

Fig. 3: PyWiSim UML classes diagram.

6) When the packet queue is instructed to create a new transport block, packets in the retransmission dictionary ("dc_retrans" in Figure 5) have priority over packets in the receive list ("receive list" in Figure 5). Hence, when building a new transport block, packets in the retransmission dictionary are considered first, and if the number of bits accepted for transmission have not been exhausted, packets from the receive list are considered.

A maximum size may be assigned to the list of received packages. When the number of received packages exceeds its maximum, packets are discarded (dropped) and not entered in the received list. The packet queue class updates counters of received, dropped, sent, and lost packets, both as number of packets and as number of bits.

V. Wireless channel models

As mentioned above, *libsimnet* has objects that can adopt different functionalities, and a user may often need to create a new model for these objects. An example is traffic generators. Traffic generators can have different packet sizes and inter-packet time models. PyWiSim currently has some traffic generators implemented that generate fixed-size packets at a fixed time or packets of size or time with different probability distributions. A user can build another traffic generator model by inheriting from *libsimnet* and overriding some methods. Another example is schedulers that allocate resources to users. PyWiSim currently has several scheduling algorithms: a round-robin scheduler, a max CQI scheduler, a proportional fair scheduler, etc. As with traffic generators, users can easily create a new scheduler with the desired functionality. The last example is about wireless channel models. In this section, we will explore this model in more detail because it allows us to see the integration of PyWiSim with external packages that simulate a wireless channel.

The *libsimnet* package has two abstract classes to model the wireless channel: Channel and ChannelEnvironment. ChannelEnvironment is an auxiliary class that stores channel scenario characteristics with parameters necessary to calculate the channel state. Channel is the main class and is responsible for providing the user with the state of their channel with the base station whenever needed. The simplest example of a wireless channel model implemented by PyWiSim is the random/fixed channel. In this case, each time users query their channel state, they are given a random (with a certain distribution) or fixed SNR value, depending on the config-

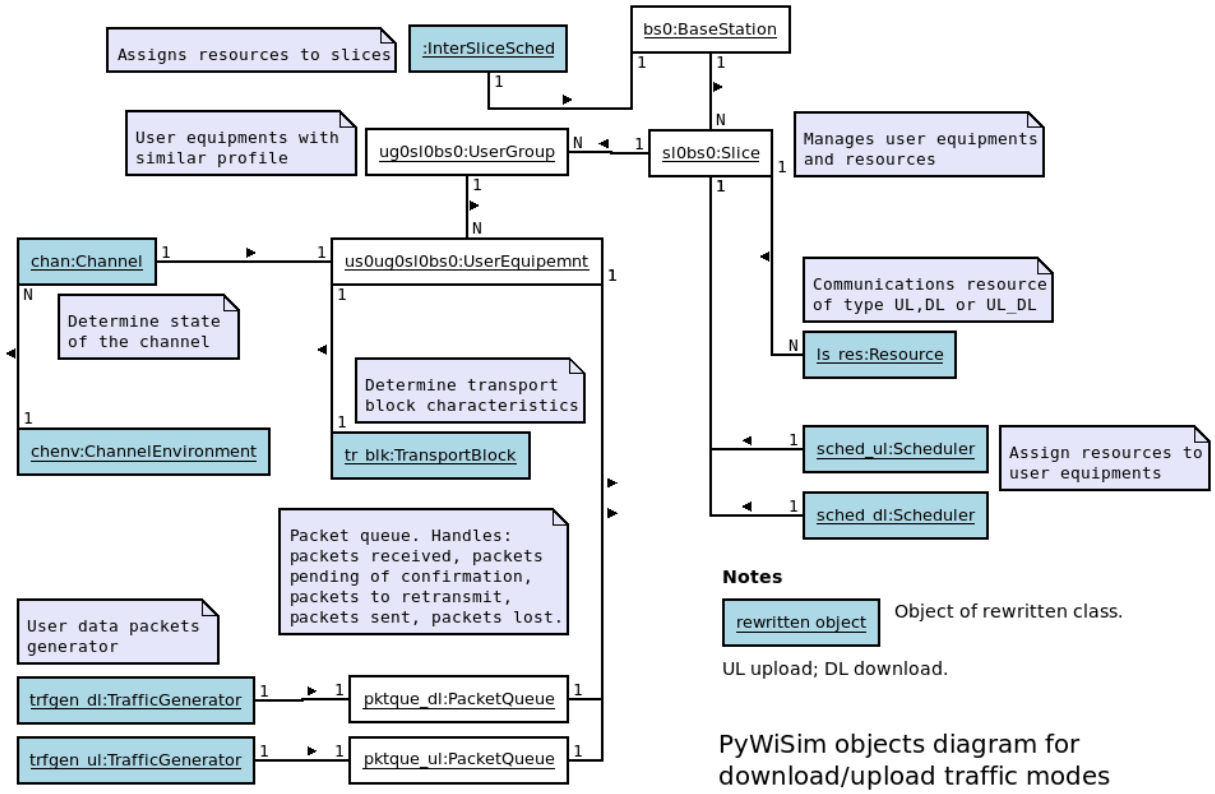

Fig. 4: PyWiSim UML objects diagram.

uration. With a slightly more complex approach, PyWiSim implements a Rayleigh or Riccian channel model. However, it should be noted that there are many channel models and channel simulators, and new ones frequently emerge with new technologies. For this reason, PyWiSim can be integrated with third-party channel simulators. This integration can be done in two ways, depending on the capabilities of the external channel simulator. The simplest way is through a file. PyWiSim has a channel model called FileChannel. In this case, a file format is defined, specifying the simulation time, the user, and the channel status. The desired simulator must be output in that file format (or another if the FileChannel class is modified or extended). When PyWiSim is accessed by a user at a given time, it will query the file and return the channel state. A file can also be associated with the channel positions of each user at each time.

The second form of integration depends on the desired simulator's ability to run as a Python package. In addition, the wireless channel simulator must have an interface that can be queried online about a user's channel state at a given time. As an example of this type of integration, PyWiSim integrates with a channel simulator developed by our group and called PyWiCh [13]. As shown in Figure 6, a Channel class in package pw_channel was created that inherits from the *libsimnet* Channel class in package usernode. In addition, different ChannelEnvironment classes (for indoor, outdoor, etc. environments) were created to allow the creation of PyWiCh simulation objects according to the desired scenario. Later, when PyWiSim runs, the channel state is queried from the pw_channel class to PyWiCh package online, informing it of the simulation time, the user's position in the scenario, and a series of other necessary parameters.

VI. Artificial Intelligence integration example

In the previous section, we described the tool's capabilities for incorporating new channel models. We now turn to a similar demonstration focused on schedulers. In particular, we highlight this feature to showcase how naturally Python supports the integration of AI libraries into algorithm design; in this case, within the intra slice scheduler.

This example develops a scheduler using deep reinforcement learning, specifically through the Deep Q-Network (DQN) algorithm. Figure 7, shows the overall framework for AI integration and is structured as follows. The neural network is implemented using the Python Torch library within the DQN class. The Agent class is responsible for training the neural network, selecting actions, and storing interaction data with PyWiSim for each state and action taken.

The Scheduler class allocates resources based on the action selected by the Agent class. The Scheduler measures the reward, observes the next state after resource allocation, and stores the information (state, action, reward, next state) in the Agent class.

In the specific example implemented, there are N users and the goal is to keep the size of the user packet queues as small as

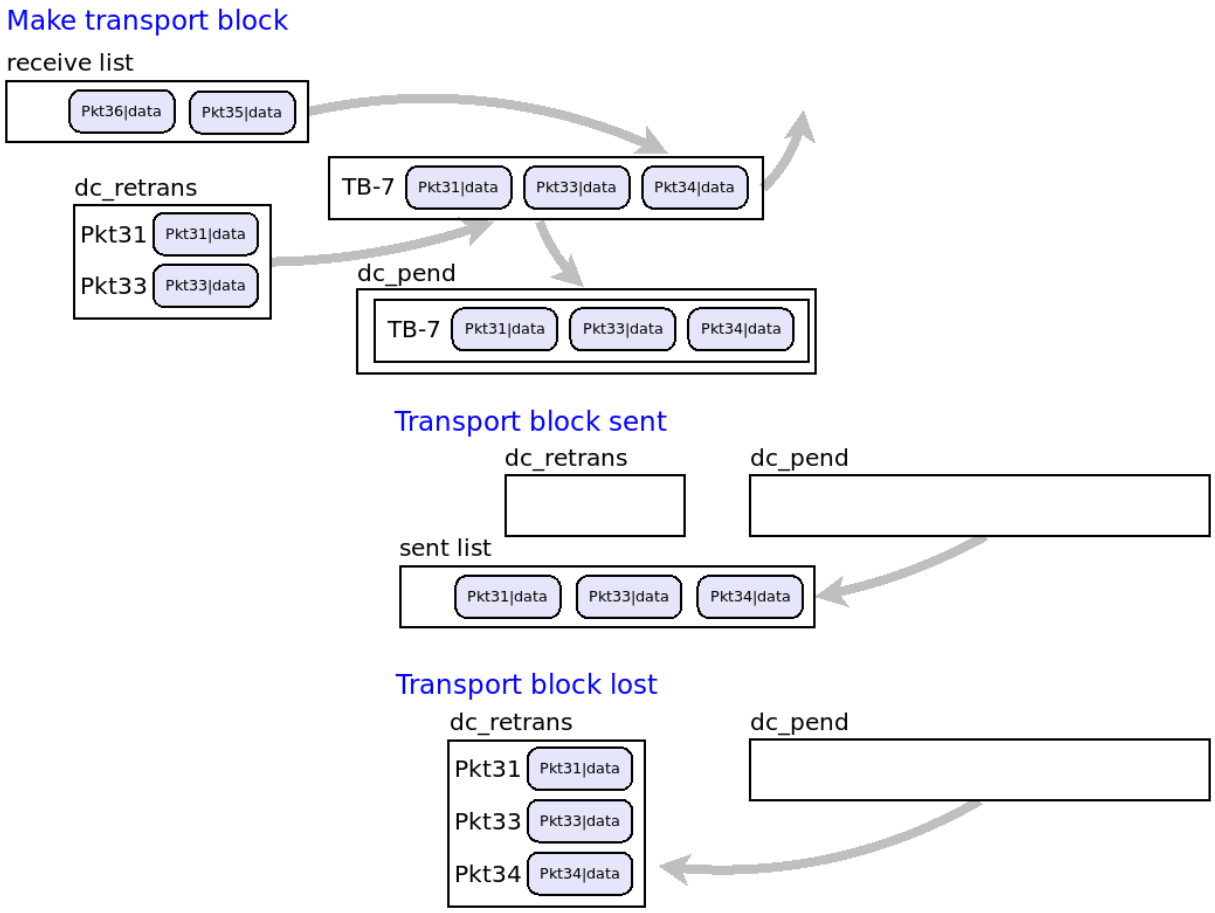

Fig. 5: PyWiSim transmission process.

possible throughout the simulation; an approach that indirectly contributes to reducing transmission delays.

The action consists of selecting the user (or users) to whom resources will be allocated. Once the selected user's packets are transmitted, the queue sizes of all users are measured. The scheduling policy is designed to minimize the cumulative maximum queue length across users over time, formally expressed as $\sum_t \max_u(Q_t^u)$, where Q_t^u denotes the queue length of user u at time t after resource allocation. The system's state is defined by the size of the queues of all users before resource allocation at time t, as well as the Signal-to-Noise Ratio (SNR) of the users at that time. The instantaneous reward is defined as $\max_u(Q_t^u)$.

VII. 5G Extension and Graphical User Interface

This section will briefly describe the extension of the *libsimnet* library to implement 5G. First, we will briefly describe the main extensions made to the *libsimnet* classes. Later, we will describe some examples of 5G configuration with the 5G Graphical User Interface.

One of the most important aspects of 5G is variable numerology. Numerology defines the subcarrier spacing (SCS). It also defines the timing of an OFDM symbol and, consequently, the TTI used by the scheduler. Each slice in 5G can have different numerologies, and, therefore, different TTIs at which its scheduler runs. For this reason, the Slice class is overridden by adding the numerology attribute, which defines different aspects, such as the TTI of that slice's scheduler. Slice methods that allow storing 5G-specific information are also overridden. The same applies to the InterSliceSched class, which is overridden only in one method to store 5G-specific information. The rest of the Slice and InterSliceSched functionality is inherited from *libsimnet*.

It is also necessary to override the Resource class. A resource in 5G is called a Physical Resource Block (PRB). A PRB comprises one slot and 12 subcarriers. Each slot contains 14 symbols, except when using a long cyclic prefix in OFDM, where each slot contains 12 symbols. In Frequency Division Duplexing (FDD), the uplink and downlink PRBs are independent. The number of symbols in each PRB is obtained by multiplying 14 symbols per slot (or 12 if a long prefix is used) by 12 subcarriers. In contrast, in Time Division Duplexing, the symbols in a PRB are shared between uplink and downlink, and therefore, depending on the configuration, part of the symbols will be used for the downlink and the other part for the uplink. All this logic is implemented in the 5G Resource class.

It is also necessary to override the TransportBlock class,

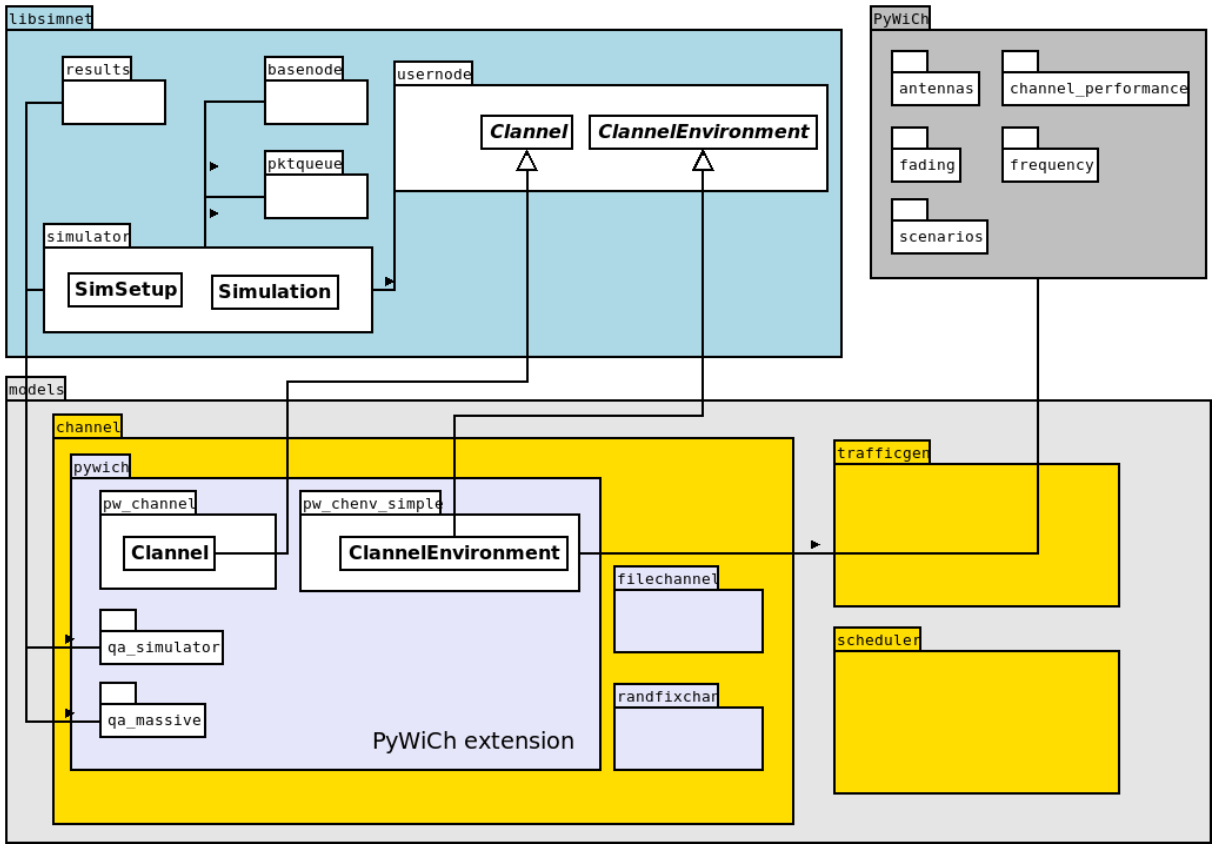

Fig. 6: PyWiSim - PyWiCh integration.

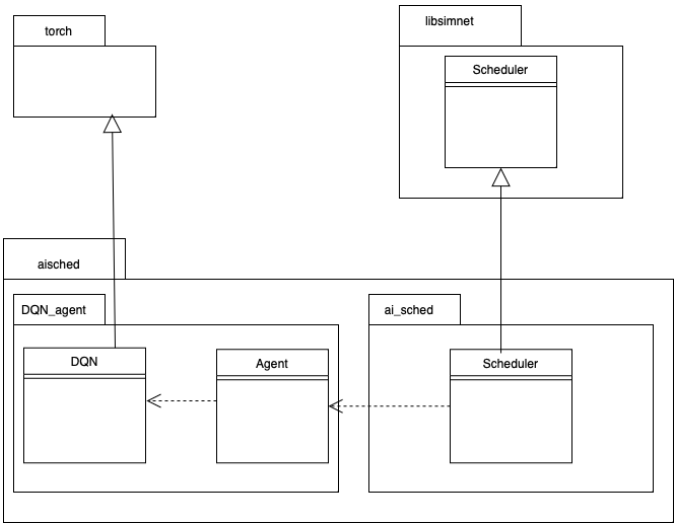

Fig. 7: PyWiSim artificial intelligence scheduler example.

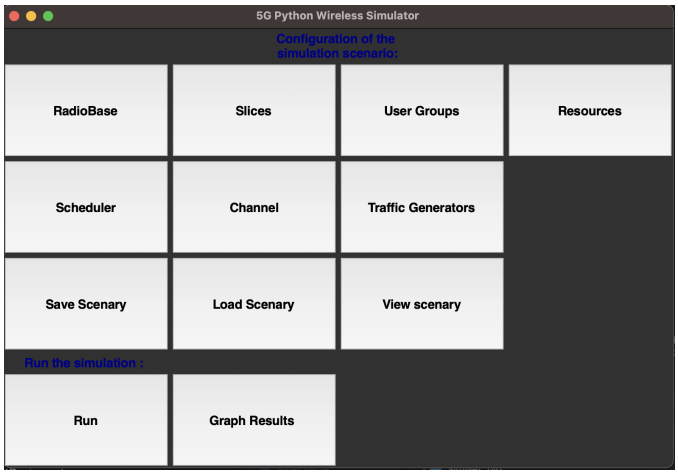

Fig. 8: PyWiSim GUI main window.

since this class calculates the number of bits a TB can carry, and this depends on the modulation and coding used in 5G, which in turn depends on the state of the wireless channel.

By overriding these four classes, actually some of these classes' methods, and inheriting the rest of the behavior from these classes and those not overridden, 5G is implemented.

Next, we will see how some of the 5G parameters discussed above are configured. The PyWiSim GUI can be used for this. Figure 8 shows the main window of this interface. This window has a menu that allows the configuration of a scenario's components: radio base stations, slices, user groups, resources, schedulers, channel models, and traffic generators. In addition, the user can save a configured scenario or load a previously saved one. After a scenery is configured, the user

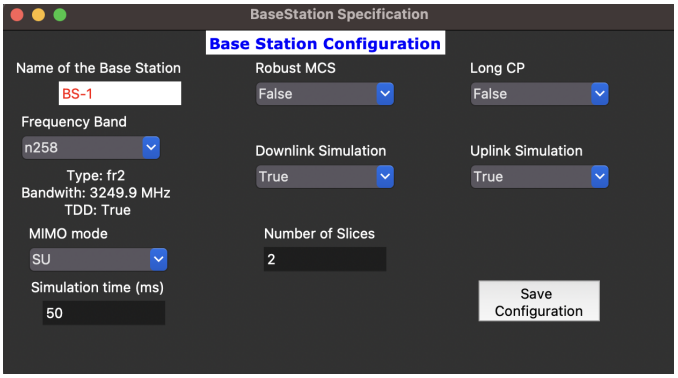

Fig. 9: BaseStation configuration.

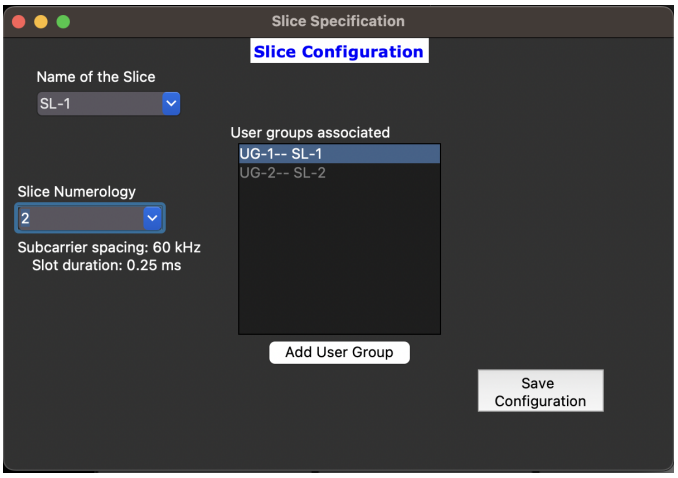

Fig. 10: Slice configuration.

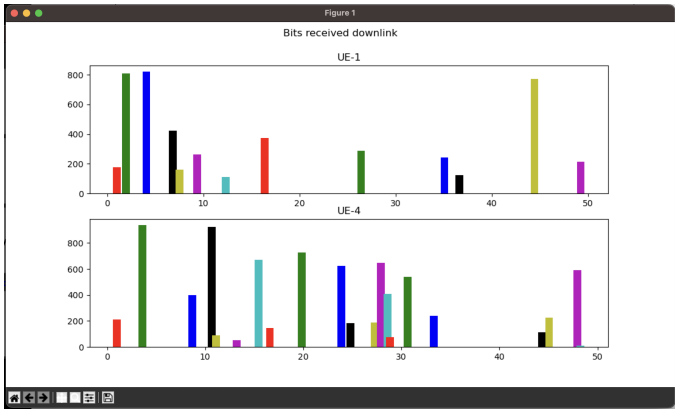

Fig. 11: Bits received.

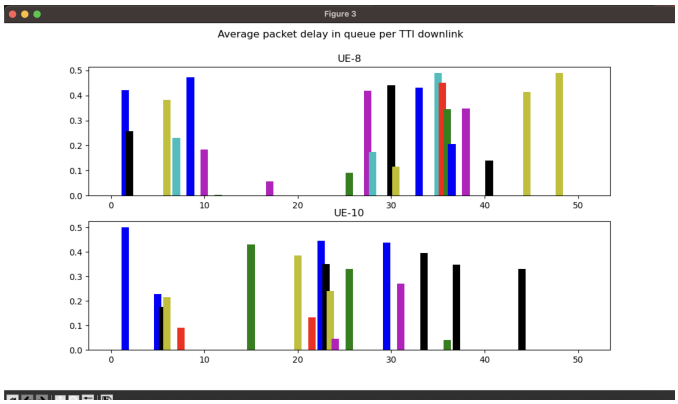

Fig. 12: Queue delay.

can run the simulation and graphically analyze the results. To illustrate the configuration of the scenario elements, Figure 9 and Figure 10 show the configuration windows for a radio base station and a slice as examples.

In the radio base configuration window, the user can configure some operating parameters, such as the frequency band, the MIMO type, the use or not of a long cyclic prefix, etc. The user can also configure general simulation characteristics, such as whether the simulation is uplink, downlink, or both, the simulation time, etc.

The slice configuration window allows the creation and association of user groups and slices (the characteristics of the user groups are defined in another window) and the configuration of the numerology for each slice.

As shown in Figure 8, once the scenario is fully defined, the simulation can be executed using the Run button in the GUI. Afterwards, the tool provides several predefined plots accessible through the "Graph Results" menu.

VIII. Main Simulation Results

In this section, we will show some results that can be obtained with PyWiSim. We will only show a few examples.

First, PyWiSim stores the packets received, transmitted, and lost over the air and the corresponding bits for each user queue. This information is stored at each TTI, that is, each time one or more transport blocks are transmitted. Figure 11 shows the bits received from the packet generators at each TTI for two users.

All the information about the state of each user's queue is also stored. For example, Figure 13 shows the packets in the queue at each TTI for two selected users. Figure 12 shows the average delay for each user queue at each TTI.

Other important information stored is information related to the scheduler. For example, Figure 14 shows the transport blocks generated at each TTI for two selected users. Likewise, Figure 15 shows the PRBs assigned at each TTI to three selected users.

IX. Conclusions

In this paper, we introduced PyWiSim, a Python-based simulation framework for wireless networks that bridges the gap between detailed link-level simulators and large-scale system-level simulators. By offering a modular architecture, PyWiSim enables the simulation of a large number of devices while preserving a reasonable level of detail in modeling key system components such as channel behavior, traffic handling, and scheduling.

One of PyWiSim's key strengths lies in its extensibility: new wireless channel models, traffic generators, and scheduling

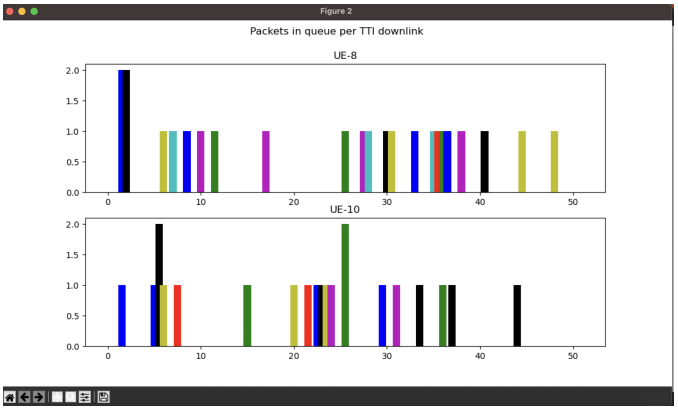

Fig. 13: Packets in queue.

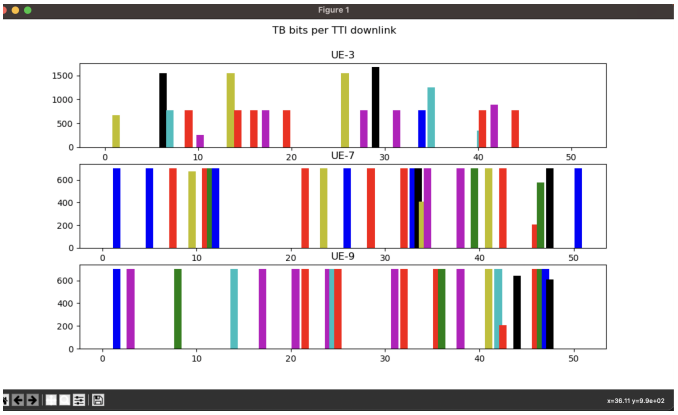

Fig. 14: Transport blocks generated in downlink.

strategies can be easily integrated. Particularly for channel models, it supports both built-in models (e.g., fixed, Rayleigh, Rician) and external integrations via files or live Python interfaces, exemplified by its integration with the PyWiCh simulator for dynamic, scenario-based channel state queries. Its implementation in Python also makes it particularly suitable for research involving artificial intelligence, allowing seamless incorporation of AI-based algorithms. We demonstrated this capability through the implementation of an intra-slice scheduler using the Deep Q-Network (DQN) algorithm.

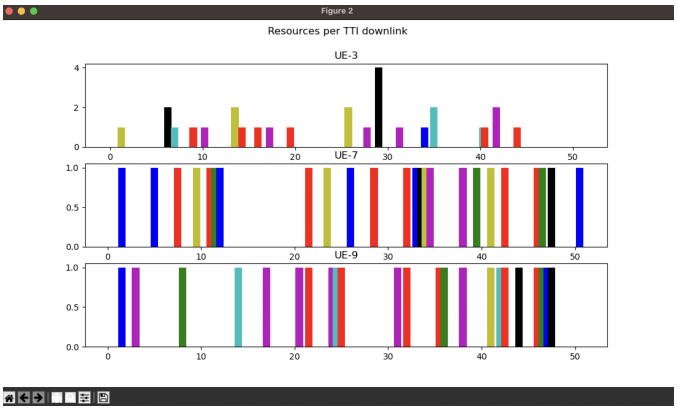

Fig. 15: Resources assigned.

Moreover, PyWiSim supports multi-slice operation, a central feature in 5G and beyond, making it a relevant tool for simulating modern network architectures. The framework's clean design, extensive documentation, and open-source availability position it as a practical resource for both educational and research purposes. So far, the tool has been used in undergraduate and graduate theses, as well as in academic courses — including a doctoral-level summer school held at IMT-Atlantique (https://github.com/ramonaas/summer-school).

Future work will focus on expanding the library of built-in models, improving simulation efficiency, and incorporating more advanced AI-driven optimization mechanisms for resource allocation and mobility management.

X. Acknowledgments

This work has been partially funded by CSIC R&D project: 5/6G Optical Network Convergence: an holistic view.

References

[1] G. Riley and T. Henderson, "The ns-3 network simulator." in *Wehrle, K., Güneş, M., Gross, J. (eds) Modeling and Tools for Network Simulation. Springer, Berlin, Heidelberg*, 2010.

[2] G. Pereyra, C. Rattaro, and P. Belzarena, "Py5chesim: a 5g multi-slice cell capacity simulator," in *2021 XLVII Latin American Computing Conference (CLEI)*, 2021, pp. 1–8.

[3] Free Software Foundation, "GNU General Public License, version 3," https://www.gnu.org/licenses/gpl-3.0.html, 2007.

[4] 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

[5] 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.

[6] 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.

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

[8] S. Martiradonna, A. Grassi, G. Piro, and G. Boggia, "5g-air-simulator: An open-source tool modeling the 5g air interface," *Computer Networks*, 2020.

[9] 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.

[10] 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.

[11] A. Gungor and contributors, "Ueransim: Open source 5g ue and ran simulator," 2021, accessed: March 16, 2025. [Online]. Available: https://github.com/aligungr/UERANSIM

[12] 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.

[13] P. Belzarena, "Pywich-python wireless channel simulator, version 1.0, release date 29/7/2022,doi:10.5281/zenodo.6941434," 2022.