Developing a New Simulation and Visualization Platform for Researching Aspects of Mobile Network Performance

C. Amaro[1] and T. Saraiva[2]
[1]*Instituto Superior Técnico,*
University of Lisbon
Lisbon, Portugal
catarina.p.amaro@tecnico.ulisboa.pt
thaina.saraiva@celfinet.com

D. Duarte[2,4] and P. Vieira[3,4]
[2]*CELFINET,*
Consultoria em Telecomunicações, Lda.
[3]*Instituto Superior de Engenharia de Lisboa*
Lisbon, Portugal
david.duarte@celfinet.com
pedro.vieira@isel.pt

M. P. Queluz[1,4] and A. Rodrigues[1,4]
[4]*Instituto de Telecomunicações*
Lisbon, Portugal
[paula.queluz, ar]@lx.it.pt

2021 24th International Symposium on Wireless Personal Multimedia Communications (WPMC) | 978-1-6654-2760-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/WPMC52694.2021.9700408

***Abstract*—Nowadays, mobile networks represent one of the most innovative and challenging technological and research-oriented fields of work. The growth on user subscriptions and the advances introduced by Artificial Intelligence (AI) and Internet of Things (IoT), greatly enhanced the complexity and potential of communication networks. The increase on variety of devices and exchanged mobile data traffic resulted in demanding requirements for the network providers. As networks tend to scale and data to increase, some problems start to arise. Traffic congestion, packet loss and high latency being some examples. Therefore, it is important to introduce powerful tools and methods to tackle these challenges. On this perspective, several studies have highlighted AI systems, mainly Machine Learning (ML) algorithms, as the most promising methods, in the context of wireless networks, by improving the overall performance and efficiency. This work proposes to integrate several network optimization algorithms, already developed, in a common and unified visualization platform. These algorithms were developed in C# and Python and some of them use supervised and unsupervised ML techniques. The proposed solution includes multi-threading processes to deal with concurrent simulations, a proxy to communicate between platforms and a dynamic visual interface.**

***Index Terms*—Mobile networks, machine learning, visualization platform, multi-threading, proxy.**

I. Introduction

In recent years, communication networks have grown in number of subscriptions, diversity of devices and amount of mobile data traffic. This pressure on the networks led network providers to step away from traditional methods and create innovative solutions to deal with these challenges. This growth was motivated by a series of new technologies, IoT based solutions being one of them. Nowadays, essentially any electrical device can be connected to the network, from mobile devices, to drones, fire detection systems, smart lights and autonomous cars. Each of these devices has very different network requirements, increasing network complexity. AI is another area of expertise that boosted this growth. More complex algorithms tend to consume more resources, which means the network must be able to handle very intensive tasks, without losing performance. Other technologies, that similarly to AI, consume a lot of resources are Virtual Reality (VR) and Augmented Reality (AR). Also, they are starting to appear in mobile applications along with other devices that connect with the network and they use massive data streaming [1]. The solutions to this demand can be categorized into two groups. The first is the integration of 5G infrastructures, technology designed to handle this new increase on data rates. The second is to introduce AI systems, mainly ML algorithms, to optimize and monitor the network data. Most of this data can be processed into Quality of Service (QoS) and Quality of Experience (QoE) metrics. Moreover, these metrics can be provided by the analysis of Performance Management (PM) data, using Key Performance Indicator (KPI) and Drive-Test (DT) information. It may also be provided by exploring network recording Traces, and also considering Configuration Management (CM) parameters and Energy Management (EM) measurements. The processing and analysis of these parameters can help to enhance network management, configuration and monitoring procedures. This work is integrated in the study of the second solution, the use of ML techniques to optimize the network. The proposed work is to create a unified visualization platform, that integrates several network optimization algorithms, previously developed. Concretely, it intends to create a solution that allows external tools to use and test the developed algorithms. Furthermore, it will allow a long-term study on the impact of both the algorithms and the developed tool, within the regular tasks performed by RAN engineers. Simultaneously, it will create an interface where new algorithms can be developed, tested, deployed and efficiently used.

This paper is organized as follows: in Section II the existing structures are briefly described, mainly the used platforms and some of the algorithms; in Section III the development of the new platform, architecture and implemented structures are presented; Section IV introduces some performance studies and respective results. Finally, conclusions are drawn in Section V.

II. Background

This work was conducted within the scope of Celfinet's research team. Which concentrates its main findings over a simulation platform called Research Innovation Studio (RINNOS). This work uses and enhances the RINNOS platform and the algorithms within, to create an improved and unified visualization platform. The resultant software destined to be used by Celfinet's operational teams and will support different engineering projects considering mobile network operator's. Moreover, the Automation Portal is the platform used by the operational team to support their work requirements. To create the unified solution, RINNOS (the platform oriented to research and innovation) will be linked to the Automation Portal. Because of the importance of RINNOS within this study, a brief introduction is provided.

RINNOS is a prototyping platform driven by the same motivation as this research, to study and create innovative solutions that deal with the growth and complexity of communication networks. It is used to support a group of diagnosis and optimisation algorithms aiming to guarantee an adequate QoS and QoE for end users. It is based on a modular structure, ensuring an easy integration of new algorithms and functionalities, through package installation or integration of micro services, via Hypertext Transfer Protocol (HTTP). Python was greatly used in the platform and its algorithms, since it is one of the most used programming languages in ML.

The developed algorithms were a result of multiple research studies that created models to analyse, predict and optimize parameters of the mobile network. Some of these algorithms are the Traffic Network Generator, Backhaul Open RAN Planner and Clutter Classification and will be described briefly in the sections below. To create these methods, data was collected from real mobile operator's networks.

There are currently some solutions that explore the use of ML algorithms and their impact in optimizing mobile networks such as Mapinfo, InfoVista and Metric. However, the objective of this research is to design a system capable of providing an easy interface for both research and corporate purposes. In this case, instead of having a standardized platform with well defined algorithms and capabilities, we have a flexible platform that supports research and takes advantage of the insight of the teams using it, with the additional advantage of continuous growth.

A. Traffic Network Generator Algorithm

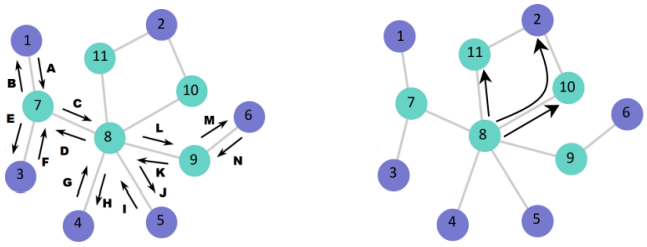
The methodology to inject the traffic on the network was based in a random function, where the routing algorithm was used in order to fill the network nodes and the connections between them [2] [3]. This random function can be stage in four phases that are presented as:

- **Charging Stage:** This phase involves an analogy from different nodes with related characteristics in terms of propagation environment. Having only traffic data for some access nodes in an analysing area, there is the need to extrapolate the information to access nodes that do not have traffic associated. For this situation and considering the classification of each access node, there is an attribution from the nodes with data to others of the same type with no traffic. By this, the result is a network with all access nodes charged with some levels of data traffic, according with the node classification.
- **Aggregate Load Stage:** This phase is introduced in order to get over the lack of data traffic information in the aggregation and backbone networks. The idea consists of analysing mostly terminal nodes (access nodes). There are connections in which the data flows directions can be determined, meaning that flows can only take one way when leaving a node. In order to illustrate, in Figure 1a, taking the connection 1-7, the traffic data injected in node 1 can only go to node 7, so node 7 aggregates the traffic from node 1. In a second stage node 7 and node 9 are in the same state as the terminal nodes were initially, so the same happens to those nodes, being this phase stopped at node 8, because the information can statistically flow through two different connections from it. In the end one should get Dispersion Points (DPs), from which the data could flow to another ones with the same characteristic through more than one path. Nodes 2, 8, 10 and 11 are considered as DP nodes since their traffic data can flow to more than one connection.
- **DP Random Interaction Stage:** In the random interaction phase, is considered the action of sending data between DP nodes. Considering the routing algorithm developed in [2], the idea is to iterate over the aggregated data previously split into services present in DPs, and rout it randomly to any other DP node.
- **Network Integrity Check Stage:** Finally, this phase is pretended to check whether the network had some traffic balancing inconsistencies once its generation was effectively random. Some non-redundant connections, meaning that were not protected, may be empty and the solution was to generate random traffic in the connection using the traffic in the correspondent nodes.

The presented algorithm was used to create a traffic busy-hour scenario, shown in Figure 2. The load calculation in each connection and node, considered the demand throughput of the several services flows in it. This was rather a random generation of a demand and traffic matrix, having into consideration a real network topology.

B. Backhaul Open RAN Planner

The Open-RAN (O-RAN) approach enables a new ecosystem of vendors to participate in the Radio Access Network (RAN) industry while leveraging the benefits of cloud-based architecture, including web-scale hardware and systems. Such degree of flexibility and vendor choice can greatly improve a service provider's supply chain posture, allowing for faster time to market and accelerated innovation cycles. The O-RAN concept is about disaggregating the RAN functionality by building networks using a fully programmable software-defined mobile network solution, based on open interfaces that



(a) Traffic aggregation from terminal nodes.

(b) Traffic being sent from 8.

Fig. 1: Network terminal nodes and connections.

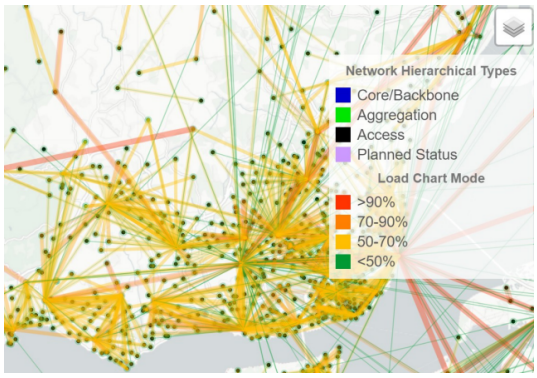


Fig. 2: Network busy-hour simulation.

runs on RAN hardware. O-RAN leverages on 2 key concepts: Virtual Radio Access Network (V-RAN), and Cloud Radio Access Network (C-RAN). V-RAN enables the disaggregation of hardware and software by abstracting the software application through the application of Network Function Virtualization (NFV) principles, allowing agile, and flexible RANs. C-RAN works like a V-RAN that coordinates Base Station (BS) functions, from a data center, using NFV and Software Defined Network (SDN) technologies, supporting dynamic capacity allocation within the RAN.

A O-RAN backhaul network algorithm developed in [5] is based on digraph theory to plan backhaul networks. In context of this work, the goal is to plan the locations for Core Clouds (CCs), Regional Clouds (RCs), and Edge Clouds (ECs), considering the a real network infrastructure and a star topology, in order to comply with the latency requirements associated to the 5^{th} Generation (5G) services slices. Additionally, a real mobile network topology was considered, including all access, aggregation and core nodes with their respective connections, but also a sample of the access nodes' main traffic KPIs. In this case, the backhaul planning algorithm considers the digraph nodes as corresponding to the full network topology while the edges coincide with the network links. Furthermore, the access nodes KPIs are used to measure each node latency, under distinct load levels. With these real network latencies, the O-RAN backhaul planning algorithm can evaluate if the O-RAN CCs, RCs, and the ECs can meet the associated latency requirements without requiring network latency simulations.

Initially, the O-RAN backhaul network planning algorithm was applied to an urban area, before being applied to the full network. The tested urban area, for the network busy hour load, is presented in Figure 3, alongside with the planned CCs, RCs, and ECs pictured as purple, red and dark green circles, respectively.

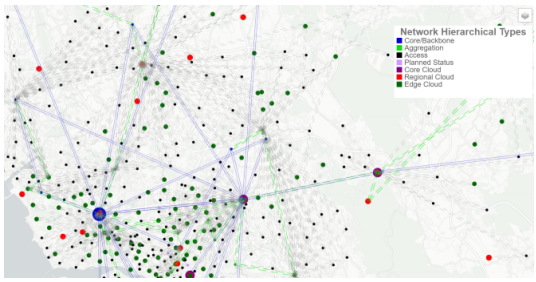


Fig. 3: O-RAN Planner in an urban area for busy hour.

The big blue symbol is the live core node containing data centers that provide services to the nodes in this urban area. As latency requirements were not so tight in (legacy) generations before 5G, the need for processing capacity close to the access nodes was less and, consequently, it was possible to centralize it in fewer locations. The algorithm's results provide a full picture of the cloud node density required to fulfil the high latency requirements in 5G. With that, Mobile Network Operators (MNOs) can evaluate which nodes should be prioritized for being upgraded using a cost-effective network investment strategy. Moreover, connections and congestion zones can be identified where an upgrade may be required in order to meet latency requirements. As an example, considering an aggregation node that connects five access nodes and one of them is set both as RC and EC, it can be concluded that from that access node to its aggregation node, the latency is higher than 1 and 4 ms but lower than 15 ms. In case a connection upgrade is performed, the latency requirements can be fulfilled, and that access node will no longer be set as both RC and EC.

C. Clutter Classification Algorithm

Nodes are basically element units being capable to deal with data that traverses it. It can be a router, a switch a base station, a data centre, etc. Regarding the access nodes, there is a second classification in terms of its environmental propagation conditions. In general, it is possible to classify the correspondent area of the node as: rural, sub-urban, urban, and even dense urban.

Therefore, a geographical characterisation reflects the broad range of radio environments and data traffic requirements and taking into consideration collected data from the Portuguese population density statistics of 2011, the parish granularity is classified into four geotypes: dense urban, urban, suburban and rural. Also, the model uses granularity at the level of parishes, being the areas classified into the geotypes above mentioned. In order to get the correct information about each

node location it was used the python package for geocoding denominated as Nominatim complemented with a json file from Correios de Portugal (CTT). Thus, having all access nodes the characteristic of the density of population that itself serves, it was possible to classify each one of the access nodes accordingly to the geotype scale. In Figure 4, node classification made by the algorithm is presented.

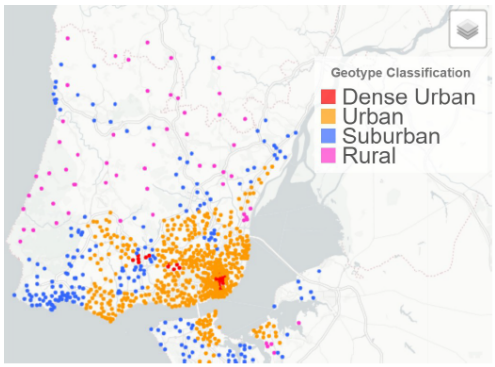


Fig. 4: Access nodes considered geotype classification.

III. Platform Development

The proposed solution aims to create a visualization platform that integrates a series of ML algorithms (as the ones previously shown) to help to optimize and detect network errors, as already stated. For this reason, the existent platforms should be improved, and new structures should be created in order to fulfil the operational team needs. The implementation of this new approach will be mainly split in two sections: improve the current structures used in RINNOS, allowing an efficient, parallel execution of the simulations, and to create a visual interface in the Automation Portal, to interact with the RINNOS backend. More specifically, the steps and requirements can be described as:

- **Interface Automation Portal**: The user must be able to perform simulations through a graphical interface in the Automation Portal. Additionally, the output of the simulations should be visualized as maps, tables and graphs.
- **Backend Automation Portal**: It should be able to handle simulation requests by redirecting them to the Proxy, connecting both platforms, and receive status updates, until they are completed.
- **Backend RINNOS**: It should be able to receive simulation requests through an Application Programming Interface (API), perform the simulation and send back the result to the proxy.
- **Simulations**: The performed simulations must run simultaneously with the backend server of RINNOS without blocking it. In addition, there should be periodical communication between RINNOS and the server executing the simulations to give user updates on their status.
- **Proxy**: It should be able to implement tools like caching to enhance performance, introduce a layer of abstraction between platforms and allow automatic integration of new algorithms.

A. Architecture

To achieve the proposed objectives and requirements an architectural pattern was developed and can be seen in Figure 5. The new structures, represented in the figure, are the Frontend Automation Portal, the Web API .NET Core (integrated in the Automation Portal) and the Task Control. The new frontend will follow the scheme of the already developed RINNOS frontend, with some user friendly improvements and additional features, such as a user simulation history and a simulation progress bar. The Task Control module is responsible for the communication between the Backend Server and the Algorithms. With this module, the platform can efficiently perform simulations (a simulation corresponds to the execution of a given algorithm applied to a specific input) in parallel, *i.e* multiple simulation requests, performed by the users, are executed concurrently and handled independently. In addition, the task module allows a periodic update of the simulation status, which was not yet possible.

In this schema, the data flow works as such: the user has access to an interactive User Interface (UI) where it is possible to select a certain geographic area, cell technology and even specific cells, through a maps and graphs; from this selection a request to the Web API .NET Core, in the Automation Portal Backend, is made; this request is redirected to the RINNOS backend and processed by the RINNOS Simulator; if the request is for a new simulation, the Task Control will be notified and will send a new task to an asynchronous task queue, the communication between the RINNOS backend and the queue is made through a message-oriented middleware; the task will be executed concurrently on one or more workers, given the resources available; and finally, on completion, the output is stored in .json format and presented to the user in the UI. Furthermore, the proxy component acts as a middleman between the communication of the two platforms and will be introduce in later phase. It can bring advantages such as abstraction between platforms, performance enhancements and automatic integration of new algorithms.

In the previous architecture, RINNOS was taking advantage of Django packages that allowed multi-threading processes. However, when performing long simulations, sometimes longer than one hour, keeping the server waiting for the response is not efficient. To solve this problem, the integration of an external Task Controller, in the current architecture, is necessary.

B. Implementation

In order to implement a system that represents the designed architecture and follows all the proposed objectives and requirements, a few technologies and components are needed. The main platforms use two different languages and frameworks, Django and Python for RINNOS and .NET Core and C# for the Automation Portal. Also, in both UI the React framework is used. Regarding the Task Control, two

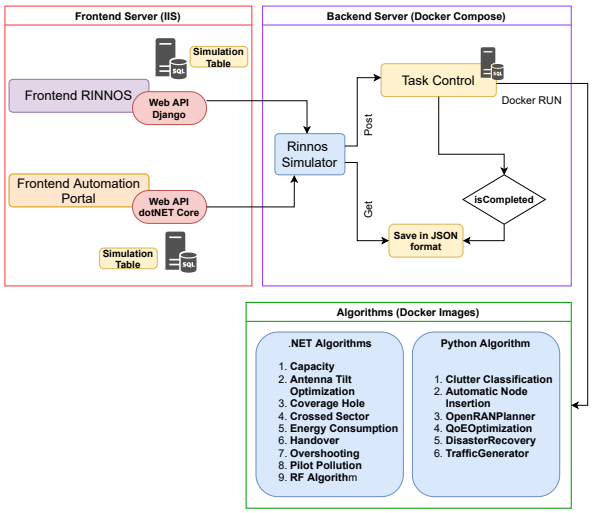


Fig. 5: Architectural pattern of the implemented solution.

technologies were used, Celery as the asynchronous task queue and RabbitMQ as the message-oriented middleware. By definition, Celery is a distributed system used to process messages that are referred as tasks, *i.e* a task queue with real-time processing, used to distribute work across threads. It also supports multiple queues and workers, responsible for executing the tasks. The combination of these features increases the tool performance. The results regarding the tool performance and robustness are explained in further detail in the section below. In addition, Celery only fully supports two message brokers Redis and RabbitMQ. Accordingly, Redis can handle bigger rates of small messages, while RabbitMQ works best for larger messages, but struggles with scaling up to bigger data rates [4]. Since the specifications for the performance of both brokers regarding the message size is not quantitative, both brokers were tested and their performance assessed (results shown in the section below). Lastly, it is possible to see the dynamic between the technologies used, in Figure 6.

Fig. 6: Django, Message Broker and Celery message flow.

In order to solve problems of cross-platform portability and versioning, docker was used. Docker is essentially a container visualization technology. It is similar to a Virtual Machine (VM) but much light-weighted, because it does not have an OS layer, instead it uses the machine OS and adds a visualisation layer on top. Further research and information regarding docker solutions and the problems it tries to solve are described by C. Boettiger [6]. In this implementation, the docker technology was used for two different purposes. The first for the RINNOS Backend Server itself, combining three different services, Django, Celery and Redis, in one docker container. This approach was used for two reasons: it was easier to configure the interactions between services and, natively, Celery uses multi-threading when running in a Linux Operating System (OS). Since the project will run in a Windows machine it was useful to use Linux containers, through Windows Subsystem for Linux (WSL). The second usage of docker was in the algorithms itself. For each algorithm a Dockerfile was produced and an image created. This way, each one of them can be configured with their own requirements and running environment, without affecting the tool or the other algorithms. Other advantages are less space usage, the tool only needs to have access to the docker image, not the code itself, and being able to run in multiple OS. For the tested python algorithms it was also created a Conda environment inside the Docker container. With this method, all the packages and configurations were installed within Conda environment. With this setup it was detected a 20% to 50% improvement on the overall execution time of the algorithms. The interaction between the Docker images of the algorithms and the Docker container with the three services running is as such: in the Django application the tasks, regarding each algorithm available, are registered in Celery; the task's code contains a subprocess function that creates and executes the corresponding docker container; the task communicates the inputs, given by the user, and receives the output; this output is stored in the local machine by means of two docker volumes (one regarding .net algorithms and the other the python algorithms); finally, the container is removed and the output is displayed to the user. Through docker run is possible to create more than one instance of a given image and run it in parallel if the resources allow it.

IV. Results

The previously mentioned algorithms were used as use cases to test the efficiency of this new architecture. Mainly, in terms of the simulation execution time and latency in the communication between the RINNOS backend and the other structures. The latency of the algorithms is primarily determined by the communication delay in the backend server and the Celery server, executing the tasks. This delay is examined in further detail in the section below.

A. Brokers Performance

To test the efficiency of the available brokers, RabbitMQ and Redis, a simple experiment was performed, to send 100 continuous tasks (*i.e* messages) to Celery, through the broker. The purpose of this experiment is to analyse the behaviour of both brokers, in conditions of high message rates and different message sizes. From the available documentation, RabbitMQ has an upper bound of 128 MB, for messages size, and Redis has an upper bound of 512 MiB. Each test was performed with a specific combination of broker, message size and message rate. Furthermore, the obtained results, refer to the brokers running on a local machine and configured as a service in the Docker container. This layout was chosen since it is the closest to the real deployment condition.

From the Celery documentation [4] two hypothesis were identified regarding the behaviour of the two brokers. The first is that Redis has a better performance for high rates and small messages. The second is that RabbitMQ deals better with lower rates and larger messages. The brokers were tested in the resource intensive situation *i.e* a larger flow of messages. The results, in Table I, show that RabbitMQ has a better performance in all of the tests, contrary to the proposed hypothesis. However, these experiments do verify that RabbitMQ handles better large sized messages, when comparing to Redis. It also corroborates that Redis works best for small messages, when comparing to larger ones. The Ratio column, in table I, represents how much quicker one broker is, in comparison with the other, *e.g* for 50 bytes Redis RabbitMQ is 27.29% faster than Redis. With this metric, we can conclude that the difference in performance between the two brokers increases with the size of the file. The last entry on the result Table, *i.e* for 50 MB, was performed with only 10 messages, instead of the proposed 100, hence the small execution time for Redis. For RabbitMQ, no value was obtained since the maximum message size was reached. This result is expected since Redis has an upper bound bigger than RabbitMQ.

This experiment was used to simulate the message exchange between Django and Celery and, therefore, the content of this messages will be mainly composed by the information of each algorithm. This data ranges from 1 KB to 1 MB, so it would be adequate to use each one of the brokers. However, in terms of efficiency, RabbitMQ is a better choice.

TABLE I: Performance tests between available brokers.

File Size	RabbitMQ [s]	Redis [s]	Ratio [%]
50 Bytes	0.16	0.22	27.29
5 KB	0.18	0.25	28.28
500 KB	1.97	4.14	52.32
5 MB	15.85	43.02	63.16
50 MB*	-	43.99*	-

B. Algorithms Performance

To make an assessment in the impact of the new architecture performance, three algorithms were selected to be analysed. More specifically, they were executed in two different environments: the new architecture, using Docker Images, RabbitMQ and Celery, and the previous (old) architecture, using the Django packages and the algorithms code. The execution time for each situation was measured and is presented in Table II. The Clutter Classification was tested with very few nodes, so the execution time was very small and showed some overhead in the case of the Docker architecture, when the communication time is higher than the algorithm execution time. The other algorithms have similar results in both architectures, which is expected, since the execution is being optimized but, at the same time, a communication delay is introduced. However, preliminary tests demonstrate a clear efficiency improvement when the architecture is dealing with multiple simulation requests.

TABLE II: Execution time in different architectures.

Clutter Classification		Open RAN Planner		QoE Optimization	
Docker [s]	RINNOS [s]	Docker [s]	RINNOS [s]	Docker [s]	RINNOS [s]
2.194	0.017	5.014	7.398	69.96	69.24

V. Conclusion

In order to improve a legacy backend (RINNOS), some architectural design changes were proposed. From the already implemented changes some conclusions can be extracted: more extensive and long simulations can be executed; the progress of the simulations is more easily tracked, due to the Celery infrastructure; Docker allows a fluid integration of new algorithms and they can be executed independently from their source code, which solves problems of portability and versioning. Furthermore, a proxy is being developed, creating a gateway to access RINNOS backend and introducing a layer of abstraction to the platform. Parallel to the developed structures, a performance analysis was conducted to the available message brokers, with RabbitMQ showing the best results.

During this work, the Docker performance was tested with the referred use cases, showing inconclusive results comparing to the previous architecture. This indicates more extensive testing most be performed. Moreover, one of the improvements in the new approach is the execution of concurrent simulations, that is expected to show better performance execution times. Further testing also needs to be performed under these conditions. Using more resource intensive algorithms, other Celery configurations and tuning the docker resource configurations, may also lead to better results.

Acknowledgment

This work was funded by COMPETE/FEDER, under the project Artificial Intelligence for Green Networks (AI4GREEN) 16/SI/2019 - I&DT Empresarial (Projetos Copromoção), through the international project CELTIC-NEXT/EUREKA (C2018/1-5). Moreover, an acknowledgment is due to CELFINET and Instituto de Telecomunicações (IT) for the support to this work.

References

[1] Y. Sun, M. Peng, Y. Zhou, Y. Huang, and S. Mao, "Application of machine learning in wireless networks: Key techniques and open issues, "IEEE Communications Surveys Tutorials, vol. 21,no. 4, pp. 3072–3108, 2019

[2] F. Dias, "Access and Core Transmission Joint Planning Including Interconnection Disaster Recovery," Master's thesis, Instituto Superior Técnico (IST), Portugal, 2019.

[3] F. Dias, D. Parracho, P. Vieira, M. P. Queluz, and A. Rodrigues, "A Method for Wireless Network Backhaul Re-planning in a Disaster Recovery Context", 13.º Congresso do Comité Português da URSI "Espaço: Desafios e Oportunidades", December 2019.

[4] Celery User Manual, Ask Solem, 2016, accessed 24 Oct 2021. [Online]. Available: https://docs.celeryproject.org/en/stable/

[5] R. Matos, P. Vieira, D. Parracho and M. Sousa, "Backhaul Planning Open RAN based on Real Network Data", 14.º Congresso do Comité Português da URSI "As telecomunicações na crise pandémica: caminhos para a virtualização", December 2020.

[6] C. Boettiger. 2015. "An introduction to Docker for reproducible research." SIGOPS Oper. Syst. Rev. 49, 1 (January 2015), 71–79. DOI: https://doi.org/10.1145/2723872.2723882