UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

CONTRIBUTION TO THE DEVELOPMENT OF A HYPERVISOR IN A VIRTUALIZED MOBILE COMMUNICATION NETWORK

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Irene Vilà Muñoz

In partial fulfilment

of the requirements for the degree of

MASTER IN TELECOMMUNICATIONS ENGINEERING

Advisor: Anna Umbert Juliana

Barcelona, July 2017

Title of the thesis:

CONTRIBUTION TO THE DEVELOPMENT OF A HYPERVISOR IN A VIRTUALIZED MOBILE COMMUNICATION NETWORK

Author:

Irene Vilà Muñoz

Advisor:

Anna Umbert Juliana

Abstract

Software Defined Networking (SDN) and Network Function Virtualization (NFV) are two promising technologies that together provide a more efficient utilization of the network resources and a reduction of operational costs. SDN and NFV enable the Radio Access Network (RAN) slicing, in which the radio resources are shared, which can be controlled through a hypervisor. In this thesis, a virtualized RAN Slicing simulator (ViRANsim) programmed in Python and based on the 5G-EmPOWER, has been designed, implemented and tested to validate and foresee the performance of two novel algorithms before applying them in a real environment: the Air-Time Deficit Round Robin (ADRR) algorithm, which is a time variant scheduling mechanism and will be used by the hypervisor, and the weight compensation algorithm, which is placed in the network controller and pretends to maximize the Access Points (APs) resource usage in order to satisfy the traffic demand fluctuations in the short-term, while at the same moment assuring the Service Level Agreement (SLA) of the different tenants in the long – term perspective. Through this thesis, the performance of these algorithms has been studied, providing different analysis based on simulation results.

Acknowledgements

For the development of this project it has been fundamental the support of different people, who have given me the strength to fulfil this thesis.

In first place, I would like to thank Dr. Anna Umbert, the tutor of this thesis, for giving me the chance to develop this project. I express my gratitude for her technical advices, her patience when correcting the documentation and her encouraging and motivating attitude during the course of the project. Moreover, I want to acknowledge Dr. Katerina Koutlia, who has been involved during all the thesis development giving me technical and mental support as well as correcting the present document. I would also like to thank Dr. Ferran Casadevall for his clarity when introducing the different algorithms and Sergio García for the informatics support.

Last but not least, I would like to thank my parents, my couple, my brother and friends for supporting me and cheering me up during these months in which I have been committed to this thesis.

Thank you very much all of you.

Revision history and approval record

Revision	Date	Purpose
0	12/07/2017	Document creation
1	14/07/2017	Document revision
2	17/07/2017	Document correction

Written by:		Reviewed and approved by:	
Date	12/07/2017	Date	17/07/2017
Name	Irene Vilà Muñoz	Name	Anna Umbert Juliana
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

1. Introduction

Today's wireless networks challenges include the necessity of being capable of managing high levels of data traffic and providing a widespread connectivity to users while being cost effective. In addition to this, the traffic and connectivity demands are expected to keep growing during the following years. Because of the mentioned reasons, the traditional network concept based on hardware components (i.e. switches, routers…) with complex protocols, is evolving to a more flexible and efficient model. This evolution goes hand in hand with Software Defined Networking (SDN) and Network Function Virtualization (NFV), two promising technologies that together have the potential to provide the network with the management and operation required to fulfil the current and future connectivity demands.

SDN and NFV implementation allow the Radio Access Network (RAN) to share dynamically the available resources and slice the RAN into different virtual slices. RAN slicing can be the key to manage the traffic demand in wireless networks, as different Mobile Virtual Network Operators (MVNO), also called tenants, could provide connectivity to their users through a common network infrastructure by using the same Access Points (APs). This would imply a reduction of network expenditures as well as an improvement in terms of network efficiency, performance and user experience. This way, each tenant can have its own logically isolated slice of resources with its own desired set of services and the complete control of them. Slicing a RAN becomes particularly challenging due to the inherently shared nature of the radio channel and the potential influence that any transmitter may have on any receiver. In order to guarantee resource isolation between tenants, a hypervisor must be introduced at the virtualization layer. Usually a fix share between tenants of the available resources in every AP is assumed.

5G-EmPOWER is a Mobile Network Operating System for SDN and NFV research and experimentation in heterogeneous mobile networks. The Mobile Communications Research Group (GRCM) of UPC is developing a project together with CREATE-NET[1i] using this platform. GRCM is in charge of the design and implementation of a new hypervisor that exploits the concept of virtualization and considers a flexible resource allocation per AP. This new hypervisor, along with a weight compensation algorithm (explained later on) is focused on the RAN and its main objective is to maintain the tenants Service Level Agreement (SLA) in a long-term perspective considering all the APs of the network while satisfying traffic demand fluctuations in the short term in the individual APs. With this purpose, two novel algorithms are introduced: a new scheduling algorithm for the hypervisor in the Wi-Fi AP, called Air-Time Deficit Round Robin (ADRR), and a weight compensation algorithm, capable of maximizing the resource usage of the different APs of the network while assuring the SLA of the different tenants in a long – term perspective in the network. The weight compensation algorithm is located in the controller and together with the new hypervisor can provide the desired network performance.

Based on the above, it arises the need of a simulator capable of validating and foreseeing the network performance before the implementation of the new hypervisor and the weight

[1] *CREATE-NET (Center for REsearch And Telecommunication Experimentation for NETworked communities) (http://create-net.fbk.eu/) is a research center established in Trento (Italy) since 2003. It is part of the FONDAZIONE BRUNO KESSLER (FBK), a research non-profit public interest entity.*

compensation algorithm in the real-time 5G-EmPOWER testbed. In this Thesis, a virtualized RAN Slicing simulator, called ViRANsim, that is based on the EmPOWER test-bed and that contains the hypervisor and the controller simulators has been designed, implemented and tested. The simulator consists of different Python programmed classes, which correspond to the different elements defined in the 5G-EmPOWER architecture. Throughout the implementation of this thesis, both the ADRR and the weight compensation algorithms have been analysed using the developed simulator, as well as different parameters related to these algorithms have been studied carefully in order to set them to the most appropriate values.

This Thesis consists of 6 chapters, including this first introductory chapter, which also includes the detail of the Thesis' objectives and a project plan with its deviations.

The second chapter explains the technological context of the Thesis, starting with a description of the concepts of SDN and NFV. After this, the 5G-EmPOWER architecture and principle of operation is explained. To conclude the chapter, it has been included an explanation of the scheduling algorithms in which ADRR is based on: Round Robin (RR) and Weighted Deficit Round Robin (WDRR).

The third chapter contains the motivation for the ADRR and weight compensation algorithm as well as a general description of the simulator python classes. Moreover, there has been explained the exportation of simulation results as well as a justification of the time management in the simulator.

The fourth chapter contains different case studies performed through the ViRANsim simulator. The studies have been divided in two groups. The first group considers a single AP in all the scenarios. The studies performed in this first section are focused in the evaluation of the convergence of the ADRR algorithm, the comparison with reference algorithms such as the RR and WDRR and the analysis of the different parameters related to the ADRR. In addition, the performance of the different traffic generators created in the simulator is also verified. The second group of studies involve a multi-AP scenario and is focused on the weight compensation algorithm.

Finally, in the fifth chapter it can be found the cost assessment of the Thesis while in the sixth chapter the conclusions of the Thesis and a discussion about future development of the simulator are provided.

1.1. Objectives

The main objectives of the project are the following:

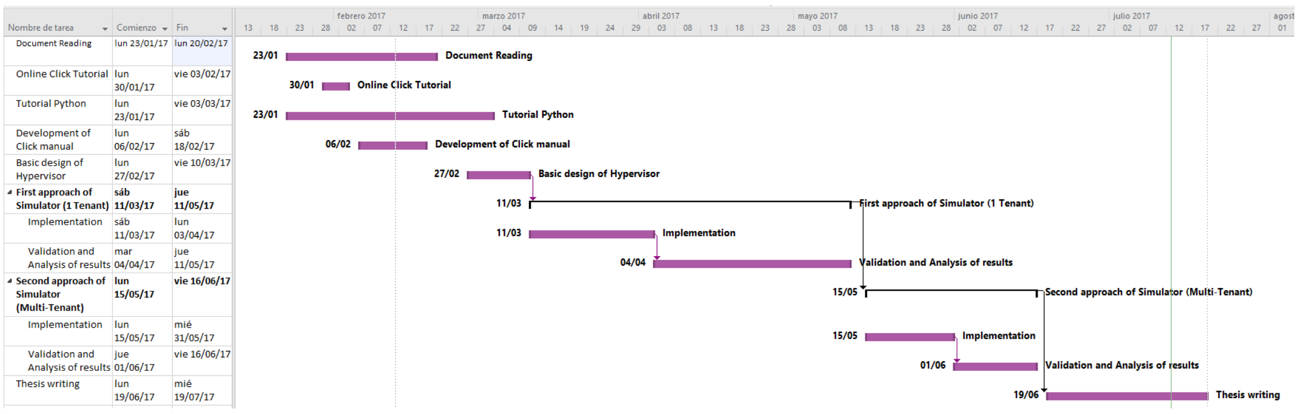
- To design a virtualized RAN slicing simulator, with main target to evaluate the performance of ADRR and the weight compensation algorithms under different scenarios.
- To implement the simulator using Python programming language in a modular style to ease future upgrades.
- To be able to obtain measurable results from the simulations for their later analysis.
- To study the performance of the ADRR and the weight compensation algorithms in order to foresee possible issues about them before its implementation in the

real-time 5G-EmPOWER test-bed and contrast the real results with theoretical ones.

- To compare the ADRR operation with already existing algorithms in the state of the art.

1.2. Project Plan and deviations

The project has been developed in different phases, as it can be observed in the following Gantt diagram.



Nombre de tarea	Comienzo	Fin
Document Reading	lun 23/01/17	lun 20/02/17
Online Click Tutorial	lun 30/01/17	vie 03/02/17
Tutorial Python	lun 23/01/17	vie 03/03/17
Development of Click manual	lun 06/02/17	sáb 18/02/17
Basic design of Hypervisor	lun 27/02/17	vie 10/03/17
First approach of Simulator (1 Tenant)	**sáb 11/03/17**	**jue 11/05/17**
Implementation	sáb 11/03/17	lun 03/04/17
Validation and Analysis of results	mar 04/04/17	jue 11/05/17
Second approach of Simulator (Multi-Tenant)	**lun 15/05/17**	**vie 16/06/17**
Implementation	lun 15/05/17	mié 31/05/17
Validation and Analysis of results	jue 01/06/17	vie 16/06/17
Thesis writing	lun 19/06/17	mié 19/07/17

febrero 2017
marzo 2017
abril 2017
mayo 2017
junio 2017
julio 2017
agost
23/01 Document Reading
30/01 Online Click Tutorial
23/01 Tutorial Python
06/02 Development of Click manual
27/02 Basic design of Hypervisor
11/03 First approach of Simulator (1 Tenant)
11/03 Implementation
04/04 Validation and Analysis of results
15/05 Second approach of Simulator (Multi-Tenant)
15/05 Implementation
01/06 Validation and Analysis of results
19/06 Thesis writing

Figure 1. Project plan Gantt diagram

The first phase included the literature review related to the topic of the Thesis and the study of Click For Routers and Python programming, as they were two language-programming tools that had to be applied during the project. Moreover a Click manual was developed, as the APs used in the EmPOWER architecture are programmed using this language. Initially, the scope of the project was to develop the hypervisor in Click, but it was finally considered that it would be more convenient to implement the ViRANsim simulator in Python, so the Click programming has not been applied to the thesis.

The second phase of the thesis consists in designing the main blocks of the simulator and specifying its requirements. After this, a first approach of the simulator was implemented focusing in a single WTP scenario. Then, the simulator operation was validated, readjusting the necessary parameters and algorithms and the first studies were performed. The main focus of this section was the ADRR algorithm evaluation. This phase required more time than the expected since it has been required the development of different traffic generators in the simulator, which were not included in the initial planning of the Thesis.

In the third phase, the simulator was upgraded to a Multi-WTP scenario, including the implementation of the weight compensation algorithm. Further validations and studies regarding the upgrades were performed.

The last phase of the development of the Thesis was the writing of this document.

2. State of the art

This chapter explains the context of this project. It starts with the concepts of Software Defined Network (SDN) and Network Function Virtualization (NFV). After this, it is explained the 5G-EmPOWER, the project in which this thesis is based on. Finally, the explanation of the scheduling algorithms Round Robin (RR) and WDRR (Weighted Deficit Round Robin) is carried out, as they are the basis of the algorithm implemented in the thesis.

2.1. Software Defined Networking (SDN)

Traditional networks are built from a large number of network devices like routers, switches and numerous types of middleboxes with many complex protocols implemented on them, resulting in a complex and hard to manage network. Operators are responsible for configuring policies to respond to the huge demand of network events and applications. Providing the lack of flexibility of the traditional network, operators have to manually transform the high-level policies into low-level commands to configure each of their network devices [2].

In this context, Software Defined Networking (SDN) tries to change the limitations of the current network infrastructures, providing it with more flexibility and promoting its evolution. SDN is referred to as network architecture defined by the next four features [4]:

- **Decoupled control and data planes.**
  In current networks the control and data plane are tightly coupled inside the networking devices. In SDN, control functionality is removed from the network devices, which become simple forwarding devices for data plane.
- **Forwarding decisions are flow based.**
  Instead of forwarding packets individually to a certain destination, SDN proposes to define flows of packets that fulfil a certain filter with a set of actions. All the packets in the flow receive identical service policies by the forwarding devices. In this way, it is possible to unify the behaviour of different types of devices like routers, switches, firewalls and middleboxes. Flow forwarding provides high flexibility, only limited by flow tables' capabilities.
- **Control logic moved to SDN controller or Network Operating System (NOS).**
  The controller or NOS is a software platform that can be run on a server and provides the essential resources and abstractions to facilitate the programming and management of forwarding devices. The existence of the controller implies a logically centralized network view.
- **Programmable network.**
  The network can be programmable through software applications running on top of NOS, which interacts with the forwarding devices.

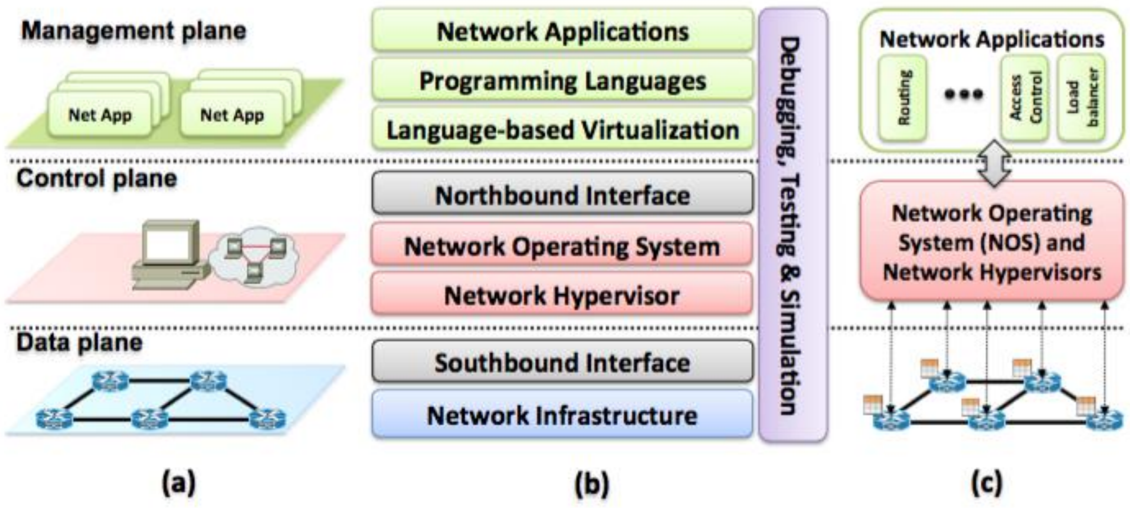


Figure 2. SDN architecture. (a) planes, (b) layers and (c) system design architecture

Figure 2 shows a scheme with the principal elements of the SDN architecture. It can be seen from three different points of view:

a) **Forwarding:** It allows the forwarding behaviour desired by the network application while hiding details of the underlying hardware. In this part, it is relevant to mention *OpenFlow* [12], which is a standard to exchange information between the data and control plane.
b) **Distribution:** SDN applications should have the sensation that the system is not distributed, making transparent the distributed physical network. This abstraction needs common distribution layer, which in SDN resides in the SDN controller or NOS. The controller is the one in charge of installing the control commands on the forwarding devices and collecting status information from the forwarding layer to offer a global network view to network applications.
c) **Specification:** It should be possible that a network application can express the desired network behaviour without being responsible for implementing that behaviour itself. It can be achieved by virtualization of solutions as well as network programming languages.

Notice that in Figure 2 it appears the concept of Hypervisor, which enables distinct virtual machines to share the same hardware resources, what is known as network slicing. This concept is quite important for this Thesis.

In terms of wireless networks, different attempts have been made to apply SDN to them. This is the case of OpenRoads [13] project, which proposes a wireless architecture that is backwards compatible and where is possible to share the network between different operators. Moreover, there is Odin [14], which introduces programmability in enterprise wireless LAN environments and OpenRadio [15], which focuses on deploying a programmable wireless data plane that provides flexibility at the PHY and MAC layers [2].

2.2. Network Function Virtualization (NFV)

Another important concept related to this Thesis is the so-called Network Function Virtualization (NFV). NFV takes advantage of the IT[2] virtualization and cloud computing techniques and applies them to telecommunication networks. It virtualizes the networking functions calling them Virtualized Network Functions (VNF). The concept is to transfer the functions from dedicated hardware appliances to software-based applications running on commercial off-the-shelf (COTS) equipment, without affecting the functionality. These applications are then executed in datacentres, network nodes an end-user premises as network requires [7].

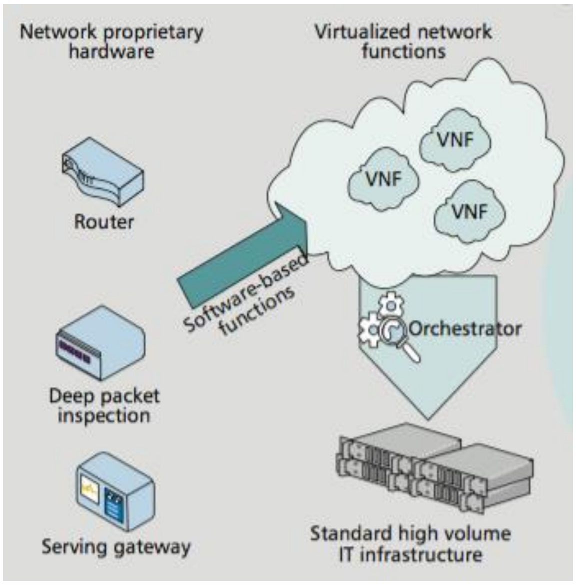

Network proprietary hardware
Virtualized network functions
VNF
VNF
VNF
Router
Software-based functions
Orchestrator
Deep packet inspection
Serving gateway
Standard high volume IT infrastructure

Figure 3. NFV virtualization concept

The benefits of NFV [7] to the telecommunications industry are the following ones:

- Openness of platforms.
- Scalability and flexibility.
- Operating performance improvement.
- Shorter development cycles.
- Reduced CAPEX and OPEX investments.

The NFV framework is built of the following components, in which NFV is deployed:

- **Physical server**: it is the machine that has all the physical resources: CPU, storage and RAM.
- **Hypervisor**: is the software that enables distinct virtual machines to share the same hardware resources. It provides the virtual environment on which the guest virtual machines are executed.
- **Guest virtual machine**: piece of software that emulates the architecture and functionalities of a physical platform on which the desired application is executed.

[2] IT calls for Information Technology

The virtual machines (VM) can be located in high-volume servers (datacentres, network nodes or end-user facilities) but also from the cloud using them as an Infrastructure as a Service (IaaS).

While SDN and NFV are two different technologies, they are complementary to each other. SDN can serve NFV by providing the programmable connectivity between VNF. A VNF orchestrator can manage these connections, which is a homologous entity as the SDN controller. Moreover, NFV can be used by SDN by using NFV network functions in a software manner on COTSs servers. It can virtualize the SDN controller to run on cloud, which could be easily migrated depending on the network needs.

NFV can be applied to wireless and mobile networks, improving them in terms of flexibility and scalability [1]. Furthermore, the deployment of new applications and services will be quicker and different network functions will be able to share the same resources. Because of this, NFV can play a fundamental role in the fifth-generation mobile networks. In this context, the Network-as-a-Service business model can provide operators with new revenue strategies by slicing the network into services operated by different MVNOs.

Providing this, future wireless and mobile networks will further rely on virtualized resources and on dynamic service orchestration. However, this implies that NVF also needs to reach the radio access of the network, which is a challenge in terms of resource provisioning and logical isolation.

2.3. 5G-EmPOWER

5G-EmPOWER [10] is an open Mobile Network Operating System for SDN and NFV research and experimentation in heterogeneous mobile networks. It has a flexible architecture and high-level programming APIs that allow a fast prototyping of novel services and applications.

The EmPOWER is built upon a single platform that consists of general-purpose hardware with operating system (Linux) in order to provide three types of virtualized network resources: the forwarding nodes, the packet processing nodes and the radio processing nodes.

The EmPOWER architecture can be observed in Figure 4. In EmPOWER, the Wi-Fi Access Points are called Wireless Termination Points (WTP), the forwarding nodes with packet processing capabilities are called Click Packet Processors (CPPs) and the virtual LTE eNodeB(s) are called Virtual Base Stations (VBS). In the platform, the radio access is treated as a VNF. In order to do so, a Light Virtual Access Point (LVAP) is created for all wireless clients and runs on the WTPs. The LVAP concept facilitates the handover mechanism between WTPs.

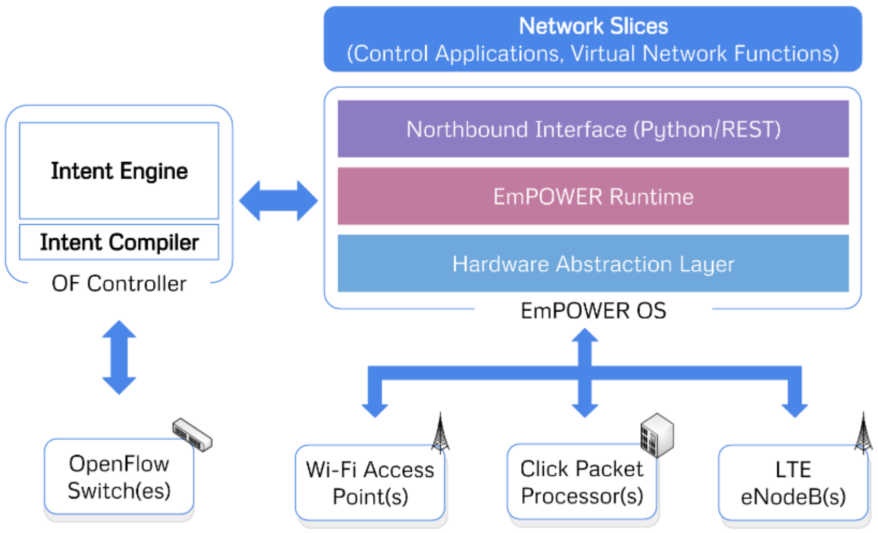

Network Slices
(Control Applications, Virtual Network Functions)
Intent Engine
Intent Compiler
OF Controller
Northbound Interface (Python/REST)
EmPOWER Runtime
Hardware Abstraction Layer
EmPOWER OS
OpenFlow Switch(es)
Wi-Fi Access Point(s)
Click Packet Processor(s)
LTE eNodeB(s)

Figure 4. EmPOWER architecture

Moreover, the platform also supports general purpose VNF named Light Virtual Network Functions (LVNFs), which are an instance of the Click Modular Router with a particular configuration.

The main elements in the architecture are described in the following points [5]:

- **Controller**
  It is responsible for the deployment of LVAP/LVNF on the network devices. It supports multiple virtual networks, also called Tenants, working on top of the same physical infrastructure. Network apps run on top of the Controller in their own slice of resources and exploit the controller programming primitives by using a REST API or a native Python API. The controller ensures that Network Apps are just presented with a view of the network related to its slice. The main features of the controller are:
  a) **Soft State**. The persistent information stored in the controller is the clients' authentication method and the list of network slices currently defined. LVAP/LVNF is kept using a distributed model and is synchronized when the WTP connects to the controller. In this way, the network can operate using the last known state even if the controller becomes unavailable.
  b) **Modular Architecture**. Apart from the login subsystem, all the tasks in the controller are implemented as plug-in that can be loaded at runtime.
  c) **Slicing.** Multiple logical virtual networks can be instantiated on top of the controller.
- **Wireless Termination Points (WTPs)**
  The WTPs are the physical devices handling the low-level communication with the clients. They consist of two components: one OpenvSwitch instance managing the communication over the wired backhaul and one Click modular router instance implementing WiFi. Click Modular Router [11] is a software architecture for building flexible and configurable routers. The main features regarding Click are the following ones:
  - **Modular architecture:** Click architecture is focused on small components, called elements that are interrelated or linked between them. A set of linked

elements defines a configuration, which allows having control over the forwarding path.

- **Declarative language:** configurations are based on definitions of elements and their links. Element classes are written in C++ using an extensive support library.
- **Programmability and flexibility:** Click language has been designed with low restrictions so new methods in the way of how elements are programmed could be invented.

Connection between Click and Controller take place over persistent TCP connection. The Click instance can run over standard Wi-Fi devices.

Moreover, EmPOWER includes a Software Development Kit (SDK), which is Python-based, available for application developers. The SDK is specifically tailored for Wi-Fi technology. The SDK includes the tools to generate network applications and control the behaviours of the different elements in the system.

The simulator designed in this thesis is based on the 5G-Empower architecture and operation.

2.4. Scheduling Algorithms

A scheduler is an element that serves packets from different queues (or flows) with a certain criterion. In this thesis, a new scheduling algorithm has been defined, which is based on already existing schedulers: Round Robin (RR) [8] and Weighted Deficit Round Robin (DRR). In this section, both of them are explained.

2.4.1. Round Robin (RR) Scheduler

To start, the packets coming from different flows are stored in different queues. After this, the scheduler serves the queues in a Round Robin manner. The scheduler defines an order in which it looks at the queues circularly. Each time it checks a queue, if there are packets, it just transmits one of them.

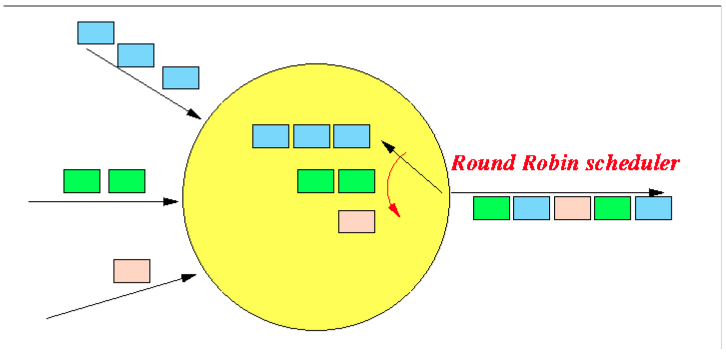


Figure 5. Scheduler algorithm

In each round, if all the queues have packets, the same number of packets is sent from each of the queues. RR achieves fairness when the packets in the queues have the same size. If not, the number of bytes transmitted from each queue would not be the same.

Another important fact to comment about the Round Robin Scheduler is that it gives the same chance to transmit to all the queues. This can be a disadvantage when it is desired that a certain flow should have more chances to transmit than another.

2.4.2. Weighted Deficit Round Robin (WDRR) Scheduler

An upgrade of the RR scheduler is presented in this section, the Weighted Deficit Round Robin (WDRR) [8]. This upgrade pursues to provide a fair scheduling algorithm when there are packets of different sizes in the queue and it is desired to assign weights to each of the queues. In this way, different resource allocation can be assigned to queues if required.

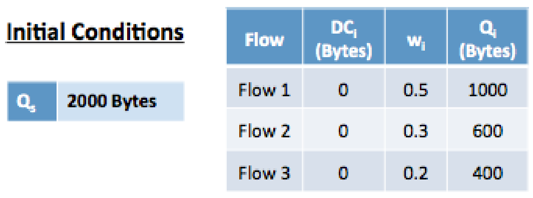
For this type of scheduling it is necessary to define the following concepts:

- **Deficit Counter (*DC_i*)**: number of bytes that flow *i* can transmit when it is its turn.
- **System Quantum (*Q_s*):** number of total bytes that the scheduler can serve during a round.
- **Weight (*w_i*):** proportion from *Q_s* assigned to a certain flow *i*. It is in parts out of one.
- **Quantum (*Q_i*):** number of bytes added to the deficit counter of flow *i* in each round. It can be also explained as the amount of credit per round. The *Q_i* can be computed through equation (1).

$$Q_i = w_i \cdot Q_s \tag{1}$$

As in RR, WDRR defines an order to check if there are packets in the different queues or flows. At the start of each round, it is updated the *DC* of each flow by adding *Q_i* and each time the system can transmit a packet it is also reflected in its *DC*. The following example explains the operation of this scheduling algorithm.

The starting point consists of defining the weights of each of the flows, the system quantum (*Q_s*) and the quantum of each of the flows (*Q_i)*. In this example, three flows with the conditions shown in Figure 6 have been considered.



Initial Conditions

Q_s	2000 Bytes

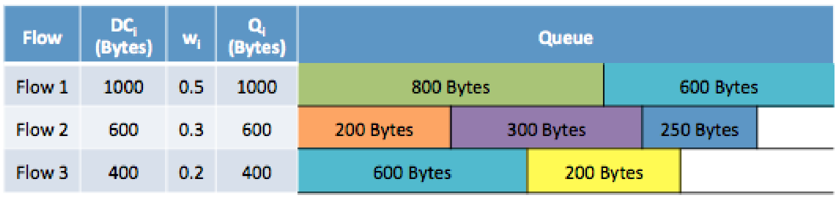
Flow	DC_i (Bytes)	w_i	Q_i (Bytes)
Flow 1	0	0.5	1000
Flow 2	0	0.3	600
Flow 3	0	0.2	400

Figure 6. Initial conditions

After this, packets start arriving to the different flow queues. The *DC_i* of each flow is updated by adding its *Q_i* value. In Figure 7, it is possible to observe the update of the *DC_i* of each of the flows and the packets in the queue of each flow, which have different lengths.

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

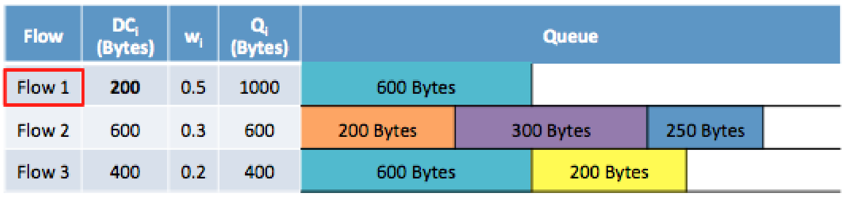
telecom BCN



Flow	DC_i (Bytes)	w_i	Q_i (Bytes)	Queue		
Flow 1	1000	0.5	1000	800 Bytes	600 Bytes	
Flow 2	600	0.3	600	200 Bytes	300 Bytes	250 Bytes
Flow 3	400	0.2	400	600 Bytes	200 Bytes	

Figure 7. Starting point. *DCᵢ* initialization with *Qᵢ* value.

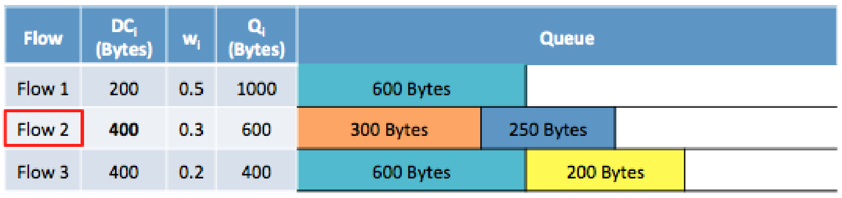
After the initialization of the *DCᵢ*, the scheduler starts with the first flow. It looks at the first packet in the queue, which has 800 Bytes. As the *DCᵢ* of flow 1 is greater than the packet length (1000 Bytes > 800 Bytes), it can be transmitted.



Flow	DC_i (Bytes)	w_i	Q_i (Bytes)	Queue		
Flow 1	**200**	0.5	1000	600 Bytes		
Flow 2	600	0.3	600	200 Bytes	300 Bytes	250 Bytes
Flow 3	400	0.2	400	600 Bytes	200 Bytes	

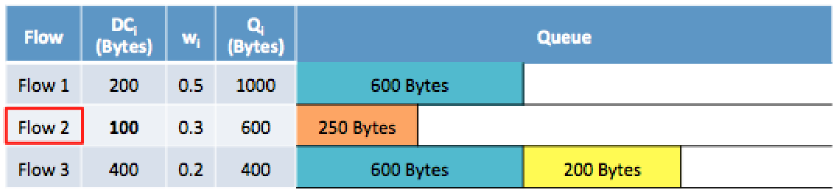
Figure 8. Flow 1 turn. Transmission of first packet

After transmitting the packet, the packet length is deducted from the *DCᵢ* of the flow, so the current *DCᵢ* is 200 Bytes (1000 Bytes – 800 Bytes). After this, it is checked if the following packet can be transmitted. As the packet length is greater than the DC_1, the packet cannot be transmitted and the scheduler moves to check flow 2. It operates in the same manner.



Flow	DC_i (Bytes)	w_i	Q_i (Bytes)	Queue	
Flow 1	200	0.5	1000	600 Bytes	
Flow 2	**400**	0.3	600	300 Bytes	250 Bytes
Flow 3	400	0.2	400	600 Bytes	200 Bytes

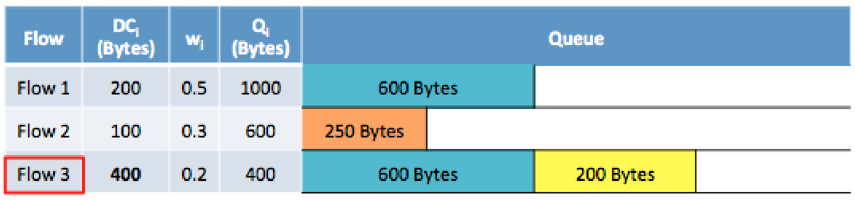
Figure 9. Flow 2 turn. Transmission of first packet



Flow	DC_i (Bytes)	w_i	Q_i (Bytes)	Queue	
Flow 1	200	0.5	1000	600 Bytes	
Flow 2	**100**	0.3	600	250 Bytes	
Flow 3	400	0.2	400	600 Bytes	200 Bytes

Figure 10. Flow 2 turn. Transmission of second packet

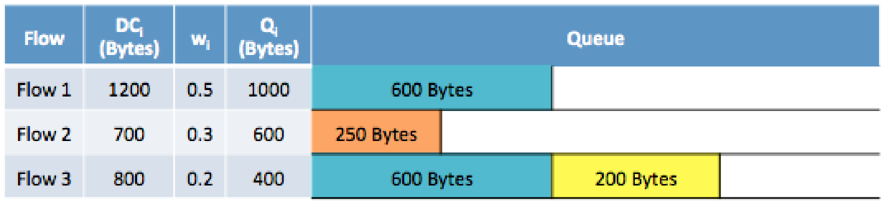
As observed in Figure 9 and Figure 10, the scheduler transmits two packets from flow 2 and each time a packet is transmitted, the DC_2 value is updated. Then the turn passes to flow 3, which is not capable of transmitting any packet since its *DCᵢ* is smaller than the packet size, as it is also shown in Figure 12.



Flow	DC_i (Bytes)	w_i	Q_i (Bytes)	Queue	
Flow 1	200	0.5	1000	600 Bytes	
Flow 2	100	0.3	600	250 Bytes	
Flow 3	**400**	0.2	400	600 Bytes	200 Bytes

Figure 11. Flow 3 turn. It does not transmit any packet.

Finally, when the next round starts the values of Q_i are added to the DC_i and the scheduler starts again to look at the different flows to transmit packets, if possible. Notice that, with this algorithm, the residual DC_i of a round is taken into account for the following round. In addition to this, if a certain queue is empty, the DC_i is reset.



Flow	DC_i (Bytes)	w_i	Q_i (Bytes)	Queue	
Flow 1	1200	0.5	1000	600 Bytes	
Flow 2	700	0.3	600	250 Bytes	
Flow 3	800	0.2	400	600 Bytes	200 Bytes

Figure 12. Start of the following round. Upgrade of the DC_i of all flows with Q_i

When using WDRR, it is possible to achieve weighted fairness, as it is assigned a different flow quantum according to the specified weights of different flows. When all the packets have the same length and the flow quantum Q_i is the same for all flows, the performance would be the same as in RR.

WDRR provides a fair sharing between flows, in terms of transmitted bytes. However, when sharing the resources by using 802.11 WiFi protocol, it is more convenient to share the resources in terms of time. In Wireless LAN, the available resources cannot be shared with respect to the assigned bandwidth (BW), but must be shared in time [16]. This is a challenge has been faced in this Thesis.

3. Project development

In this section, the different aspects studied during the project development will be explained. As a first step, the system architecture and the thesis motivation will be presented, in order to be able to analyse later on and in detail the novel Air-Time Deficit Round Robin (ADRR) scheduling algorithm and the weights compensation algorithm. Finally, the requirements and an overview of the ViRANsim simulator design will be given. This ViRANsim simulator contains the hypervisor and the controller simulator.

3.1. System architecture

Before explaining the implemented simulator and the different algorithms designed, it is important to understand the different elements in our system and how they are related. The following image shows a map of the whole system and the relation between the different elements:

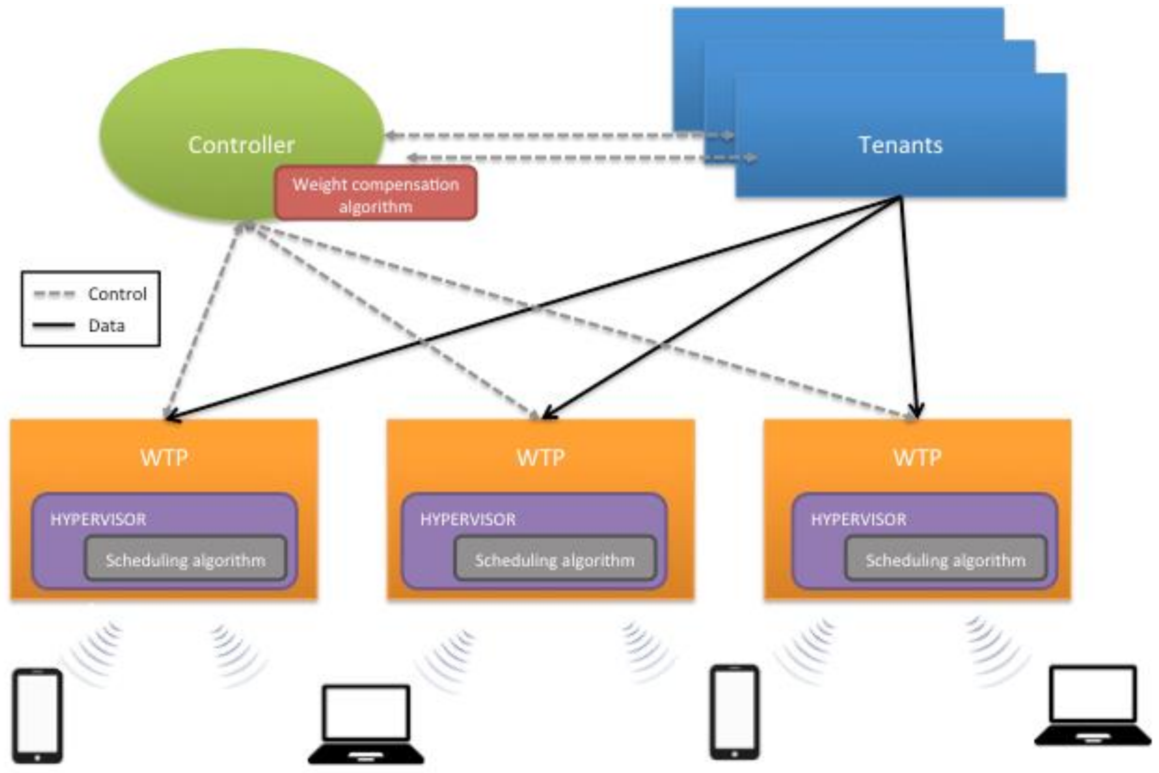


Figure 13. Overall system architecture with the different elements

The main elements and their functions are the following ones:

- **Tenants:** Virtual operators giving a service in the network. Tenants have a SLA (Service Level Agreement) guaranteed in the network.

- **Wireless Termination Points (WTP):** The network element that provides client with wireless connectivity, i.e. an Access Point in IEEE 802.11 terminology. [3]
- **Controller:** Responsible of the management of tenants in the different WTPs of the network. It also assures that the SLA is accomplished for all tenants in average in the network.
- **User Terminals:** They are clients of the different tenants that want to use the network.

The different elements are interconnected between them. Tenants provide their services to their users or clients through the different WTPs in the network. The controller manages the resources associated to each tenant in each WTP in the way that the SLA is accomplished in average in the network.

3.2. Thesis motivation

In scenarios where several tenants use the same common network infrastructure, a fix share between them of the available resources in every AP (WTP) is usually assumed, however traffic demands from the different tenants are not constant. So the aim of this thesis is to demonstrate the capabilities of a new strategy that exploits the concept of virtualization and considers a flexible resource allocation per WTP. The introduction of this new strategy in the controller aims at maintaining the SLA in a long term perspective and considering all the WTPs of the network, while satisfying traffic demand fluctuations in the short term in the individual WTPs, to which end a new hypervisor using the ADRR has been defined. In order to validate the performance of the hypervisor and the proposed algorithm for the controller with further objective the proper implementation of it in the 5G-EmPOWER test-bed, a simulator part of the 5G-EmPOWER platform has been developed.

In order to achieve the above-mentioned simulator, the following system operation is desired. Initially, the different tenants negotiate the SLA with the controller. The controller communicates the weights of each of the tenants to each WTP and the system operation starts. Each WTP, through the hypervisor, shares its radio resources between the different tenants according to the assigned weights. Every certain period of time, the controller receives from each WTP the traffic that has been demanded from each of the tenants during the last period. With this information and taking into account the SLA, the controller adjust the weights of the tenants in each WTP, trying to optimize the network resource usage. This information is communicated back to the WTP. In this way, it is possible to satisfy the traffic demand fluctuation in the short-term while ensuring the SLA in the long-term perspective.

In order to achieve the operation described above, the ADRR scheduling algorithm is necessary for the hypervisor located in each WTP to share the resources focusing on the transmission times. Furthermore, the weights compensation algorithm is required for the controller to modify the weights assigned to each of the tenants in each WTP in response of the tenant's traffic demands while maintaining the SLA in the long-term perspective.

[3] EMPOWER WIKI https://github.com/5g-empower/5g-empower.github.io/wiki/Glossary-of-Terms

3.3. ADRR scheduling algorithm

This project's objective is to create a system capable of sharing the network resources between different tenants given a Wireless LAN context. As it has already been mentioned, the sharing of resources in this context is not possible to be performed with respect to the assigned bandwidth to each of the tenants but it must be shared in terms of transmission time. This is the main reason why a new scheduling algorithm has been proposed in the Mobile Communications Research Group (GRCM) of UPC. This algorithm has been tested through the ViRANsim simulator.

The new scheduling algorithm is called Air-Time Deficit Round Robin (ADRR) and it is based on the principles of the WDRR. The most important feature of this algorithm is that it is capable of sharing the resources based on the transmission time, while considering a given weight for each of the tenants in the system. This algorithm will be used by the WTP since it is responsible for the management of its tenant's traffic.

In order to fully understand the algorithm, it is necessary to define some variables: the system quantum (Q_s), which is a system variable that corresponds to the time needed to transmit the packet of maximum size at the lowest bit rate, and the tenant quantum (Q_i), which is the proportional part of the Q_s taking into account the agreed SLA's weight for the tenant i (w_i). So, the tenant quantum Q_i can be computed as in equation(2). Moreover, each tenant is assigned with a Deficit Counter (DC_i), which is the variable that is actually used for controlling the tenant's available time for transmitting packets.

$$Q_i = w_i \cdot \; Q_s \tag{2}$$

Figure 14 consists of a scheme describing the ADRR algorithm. Notice that its operation is really similar to that of the WDRR, however the ADRR instead of considering bytes it considers time units. When the system starts, the Q_s, the Q_i and the DC_i are initialized. Notice that the initial value of the DC_i is the same as the Q_i for all tenants. After initializing the variables, the first tenant is chosen and it is checked if it has packets in its queue. If the queue is empty, its DC_i is set to 0, as it will not require time to transmit any packets. In the contrary case, when the queue has packets, the first packet in the queue is selected and the system computes a theoretical expected value for the packet transmission time (t_p). In the simulator, this expected value (t_p) is computed considering the packet length and a data rate chosen randomly considering the different available data rates probabilities, which are based on Minstrel [17], a 802.11 rate control algorithm. Then, the t_p time is compared to the DC_i of the tenant, and if DC_i is greater or equal the theoretical time, the packet can be transmitted, as the tenant has enough available credit time to transmit it. After the packet transmission, the DC_i is reduced considering equation (3).

$$DC_i = DC_i - \; t_p \tag{3}$$

The following step in the algorithm is to check if there are more packets in the queue. If so, the same procedure is repeated and if not, the next tenant queue is considered for transmission. In the case that the DC_i is smaller than t_p, the packet is not transmitted and the following tenant turn starts. When all the tenants have had the chance to transmit their packets, a new iteration starts and the DC_i is updated for all tenants, according to equation (4).

$$DC_i = DC_i + Q_i \tag{4}$$

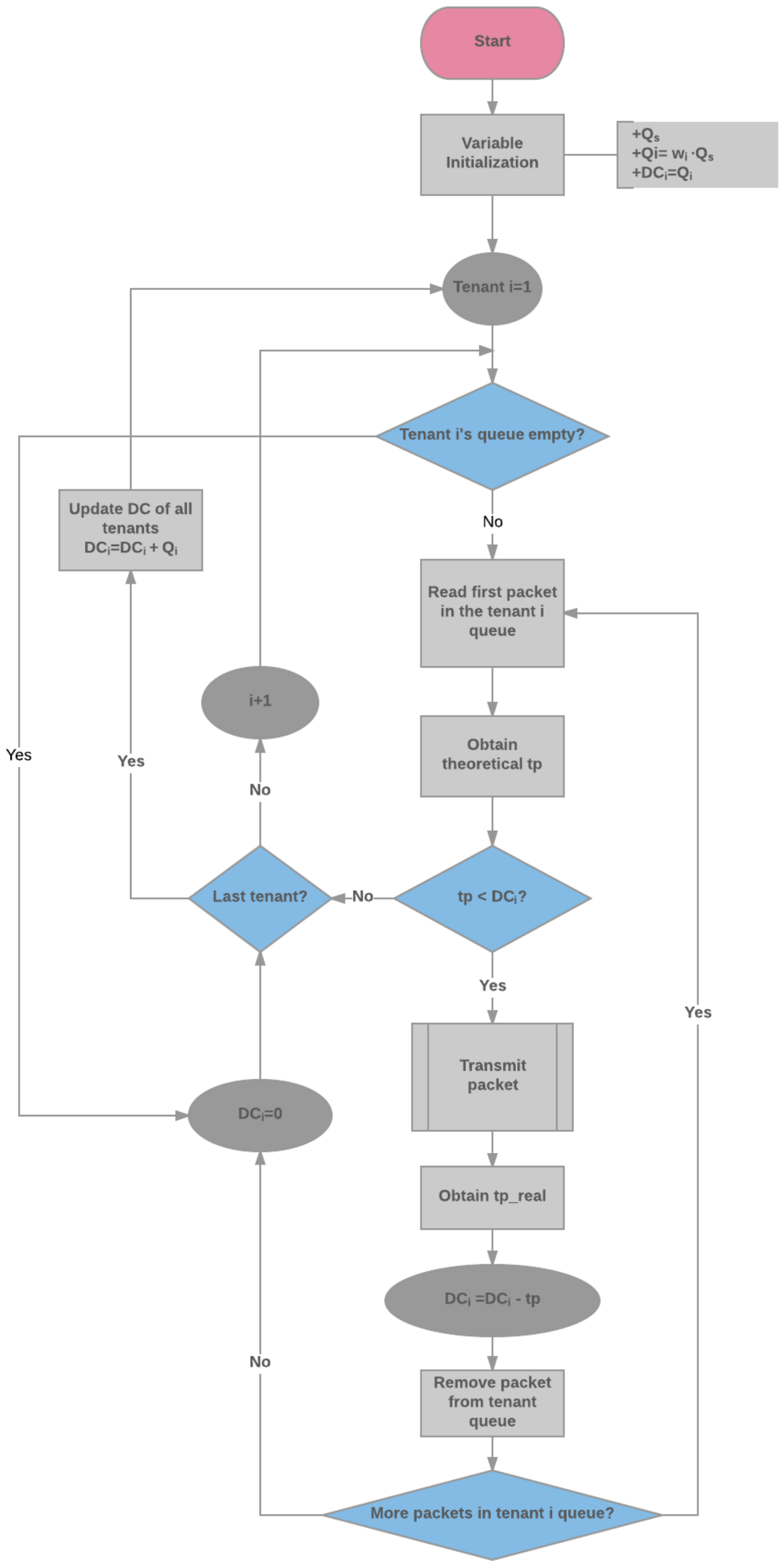


Figure 14. ADRR operation scheme

Notice that with the defined algorithm, the value of the *DC$_i$* will never be negative. If in a certain iteration the final *DC$_i$* is not null, so not all the available time has been utilized, that time will be used in the following iteration. However, let us notice that the *DC$_i$* adjustment was initially designed like in equation (5), using the real packet transmission time ($t_{p,real}$). In that case, the *DC$_i$* could have been negative in the case the tp_{real} was greater than the *tp,* for example due to unexpected retransmissions. Considering this, the *DC$_i$* in the following iteration would compensate the extra time used.

$$DC_i = DC_i - t_{p,real} \tag{5}$$

The initial design was modified, as in the real hypervisor it is not feasible to use the real transmission time because of implementation reasons.

Concluding, with the proposed algorithm it is possible to share satisfactorily the time resource according to the weight specified in each tenant's SLA. Different studies have been performed for the analysis of the algorithm, with respect to variations of the value of the system quantum (Q_s), the convergence time, as well as through comparisons with the RR and WDRR. All these studies can be found in the Studies section of this Thesis.

3.4. Weight compensation algorithm

Our system not only wants to share the resources of the network between different tenants but also to exploit the network resources, so that the WTP resources utilization is maximized. That is why we define a weight compensation algorithm, which ensures that the tenants SLA weights are satisfied in average considering the whole network in a long-term perspective while satisfying the temporary traffic needs of the tenants in the WTPs. It is supposed that the tenants respect their SLA, so no more traffic than contracted will be generated in average over the entire network.

Figure 15 shows an example to justify the need of the weight compensation algorithm. At the top, it can be observed the initial weights assigned by the controller to each of the tenants in each WTP and the incoming average traffic demand for each of the tenants in each WTP.

If no weight compensation algorithm is applied (middle part in the figure), it is not possible to fulfil the traffic demands in all the WTPs considering the initial allocation of resources. However, in WTP1 and WTP3 not all the resources are being used, which could be used to serve the traffic of the two tenants.

In the second case (bottom part of the figure), when a weight compensation algorithm is used, the WTP resource usage is maximized. In the case of the WTPs where there was an excess of resources (WTP1 and WTP3), it has been possible to assign more resources to the other tenant. Although it is not possible to fulfil the traffic demand of tenant 2 in WTP2 as the traffic demands coming from both tenants is greater than 100% of the available resources, the global average capacities of both tenants in the network improve with the use of this weight compensation algorithm.

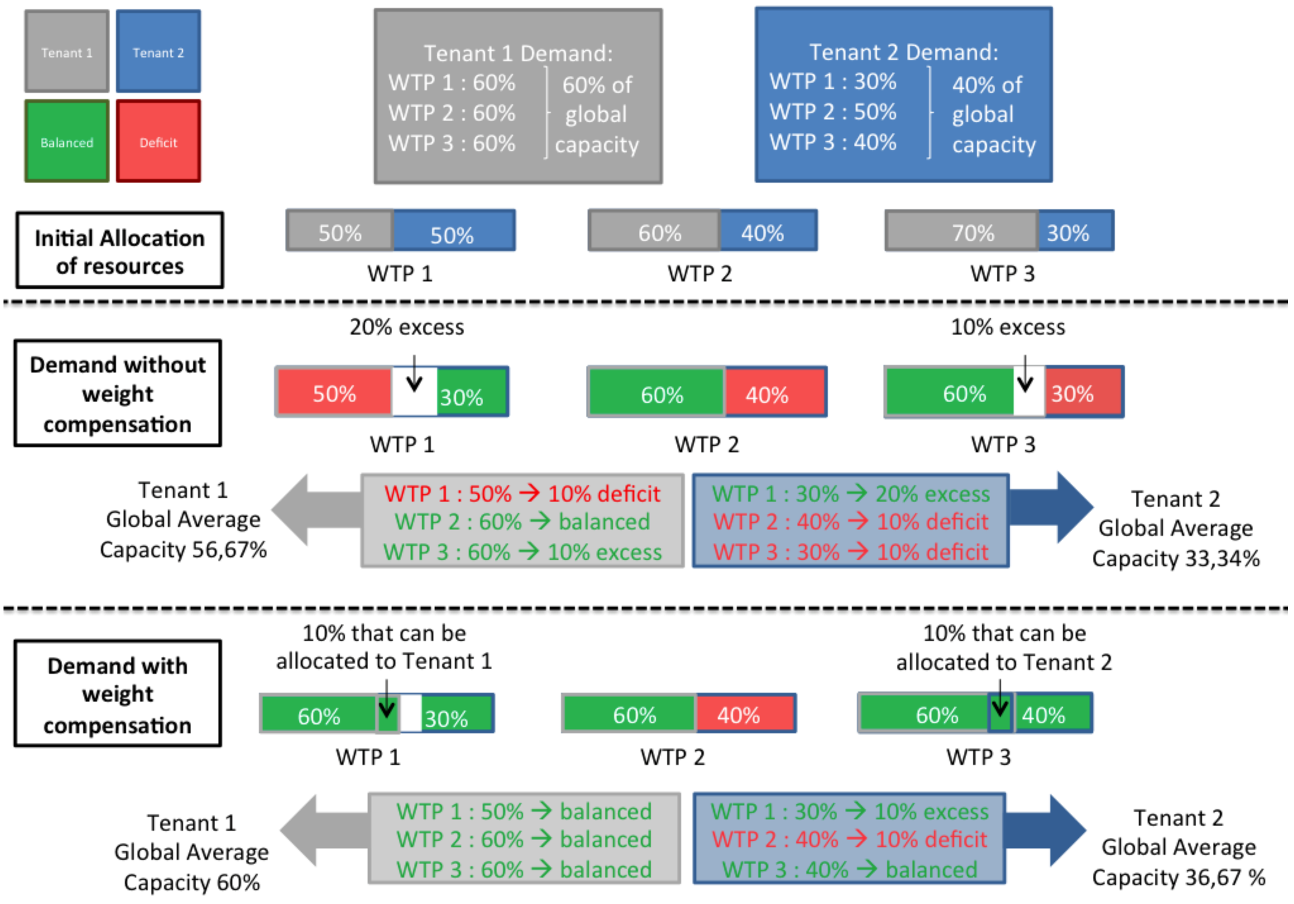


Figure 15. Weight compensation motivation

It has to be considered that the traffic percentages have to be in average considering the weight compensation in every certain period. The reason for this is that the weight compensation algorithm does not pursue to compensate instantaneous traffic peaks but to optimize the traffic allocation in a mid-term temporary scale.

The controller runs the weight compensation algorithm, since it has vision of the entire network. The controller can obtain information from the different WTPs and modify the weights applied in each of them to fulfil the traffic requirements and the SLA, maximizing the resources utilized.

Considering a network with *N* WTPs, each one with an average data rate $\overline{Rb(n)}$, the global transmission capacity of the network is $\overline{R_{b,global}(n)} = \sum_{n=1}^{N} \overline{Rb(n)}$.

Each of the *S* Tenants contracts a certain capacity, so the SLA capacity in parts out of one considering the requested capacity $\overline{R_{Ten}(s)}$ is formulated in equation (6). It has to be satisfied that $\sum_{n=1}^{N} C_{SLA}(s) \leq 1$. In the case that a tenant exceeds its C_{SLA} the controller should notify it to the tenant.

$$C_{SLA}(s) = \frac{\overline{R_{Ten}(s)}}{\overline{R_{b,global}(n)}} \qquad (6)$$

Considering that, every certain period of time, the average capacity requested to each of the WTPs by each of the Tenants is measured. The measured capacity in parts out of one is computed using equation(7), where *G_{Ten}(s,n)* is the generated traffic by tenant *s* in WTP *n*. Notice that $C_{measured}$ is measuring the traffic requested to a WTP during the last period in relation to the average traffic the WTP can serve.

$$C_{measured}(n, s) = \frac{G_{Ten}\,(s, n)}{\overline{R_{b,}(n)}} \qquad (7)$$

It has to be considered that $\overline{R_{b,}(n)}$ has to be the average effective transmission rate that the WTP can really support. In the studies of the weight compensation algorithm it is discussed how to fix this value. In addition, the period in which the $C_{measured}$ is computed has also been studied. All these studies can be found in the Studies section.

Once the controller has measured the traffic level of each tenant in each WTP, it is computed the capacity requested by each of the tenants in each WTP using equation (8). Notice that the $\Delta C_{req}(l,j)$ is also in parts out of one and can be positive or negative.

$$\Delta C_{req}(l,j) = C_{measured}(l,j) - C_{SLA}(j) \qquad \forall j \in \{1, \dots S\} \; and \; \forall l \in \{1 \dots N\} \qquad (8)$$

Considering the value of $\Delta C_{req}(l,j)$, two different situations can result:

a) $\boldsymbol{\Delta C_{req}(l,j) \leq 0}$. If $\Delta C_{req}(l,j)$ is negative, it means that the tenant *j* in WTP *i* has not generated all the capacity it has contracted in its SLA. In this case the weight of the tenant is $w(l,j) = \Delta C_{req}(l,j)$ as the WTP can proportionate the requested capacity.

b) $\boldsymbol{\Delta C_{req}(l,j) > 0}$. When $\Delta C_{req}(l,j)$ is positive, it means that the tenant *j* has generated more traffic than the agreed in the SLA in WTP *i*. When this happens, it is checked if in WTP *i* there is capacity that is not being used. This way the capacity excess of the WTP *i* is computed through equation (9) .

$$\Delta C_{exc}(l) = 1 - \sum_{j=1}^{S} C_{SLA}(j) \; - \sum_{\substack{j=1 \\ j \to \Delta C_{req}(l,j) \leq 0}}^{S} \{\Delta C_{req}(l,j)\} \qquad (9)$$

Then, it is checked if $\Delta C_{exc}(l)$ is enough to satisfy all the requested capacity in WTP *i*. In order to do so the total requested capacity in the WTP is computed using equation (10)

$$\Delta C_{sol}(l) \equiv \sum_{\substack{j=1 \\ j \to \Delta C_{req}(l,j) > 0}}^{S} \{\Delta C_{req}(l,j)\} \qquad (10)$$

Once $\Delta C_{exc}(l)$ and $\Delta C_{sol}(l)$ are computed, two more situations can arise:

- **$\Delta C_{sol} \leq \Delta C_{exc}$.** All the required capacity can be assigned to all tenants in the WTP so the assigned weights for the tenants with $\Delta C_{req}(l,j) > 0$ will be the ones in equation (11).

$$w(l,j) = \Delta C_{req}(l,j) \tag{11}$$

- **$\Delta C_{sol} > \Delta C_{exc}$.** The excess capacity is not enough to satisfy the requested capacity in the WTP. In this case the excess of capacity is shared proportionally through equation(12).

$$w(l,j) = C_{SLA}(j) + \Delta C_{req}(l,j).\frac{\Delta C_{exc}}{\Delta C_{sol}} \quad \forall j \rightarrow \quad \Delta C_{req}(l,j) > 0 \tag{12}$$

Additionally, it has been defined a concept called **proportional sharing**, which deals with the cases when the sum of the required capacity is smaller than 1, so not all the capacity of the WTP is assigned to the tenants. With proportional sharing, the remaining capacity not assigned to tenants is added proportionally to its SLA according to equation

$$w(l,j) \equiv w(l,j) + \left(1 - \sum\nolimits_{i=1}^{S} w(l,i)\right) \cdot C_{SLA}(j) \qquad \forall j \ \in \{1, \dots S\} \tag{13}$$

3.5. <u>ViRANsim Simulator</u>

This section explains the ViRANsim simulator's principles of operation, as well as its implementation details. The purpose of this simulator is to study the operation of the different algorithms proposed, as well as to be proactive in addressing the issues related to them before applying them in the real EmPOWER testbed. Firstly, the requirements of the hypervisor will be listed and then an overview of the different simulator classes will be explained. For interested readers more details are included in ANNEX 1.

3.5.1. Requirements

Regarding the desired scenario and operability, the ViRANsim simulator has the following set of requirements:

- **Python programming**. The main part of the EmPOWER test-bed (i.e. the Controller) is programmed using Python[4], so it is convenient that the simulator is also programmed in this language.
- **Modular implementation**. Giving that the simulator's purpose is to test the functionality of the different proposed algorithms, it needs to be easy to upgrade in order to include new functionalities, but also having the different functionalities as independent as possible. Because of this a modular implementation is required.

[4] Python programming. https://en.wikipedia.org/wiki/Python_(programming_language)

- **Object based**. Considering the different elements in the system (tenants, controller, WTPs…) the simpler way to implement the simulator in order to be configurable is to be object based.
- **Configurable**. Different scenarios may be analysed to check the performance of the system, so it is necessary that the simulator is easy to configure.
- **Different scheduling algorithms.** Taking into account that our system pretends to check the performance of ADRR, it is necessary to contrast it with WDRR and RR. So, the three scheduling algorithms need to be included in the simulator.
- **Weight compensation algorithm.** The simulator has to include the weight compensation algorithm to test its operation.
- **Different traffic generators modes.** The algorithms may have different behaviours depending on the traffic from the tenants, so different and configurable traffic modes need to be included.
- **Time management.** The system needs to be synchronized in order to apply the algorithms correctly and as close as possible to the reality.
- **Exportation of results.** Data results from simulations need to be provided by the simulator to evaluate the performance of the algorithms. In addition, data has to be effectively analysed and managed.

3.5.2. Simulator Classes

The simulator has been programmed using Python programming language, specifically using the version 3. One important characteristic of the simulator is that it is modular, so new functionalities can be easily added to the program.

Python is a programming language that lets you develop your programs quickly in a readable way. It supports multiple programming paradigms, including object-oriented, imperative, functional programming and procedural styles apart from relying on a wide standard library.

Taking advantage of the object-oriented features of Python, the following classes have been created in the simulation, representing different entities of the system:

- Channel Model
- Tenant General
- WTP
- Tenant WTP
- Controller
- Scenario

Notice that the user terminal has not been implemented as only the downlink is used. Each of the classes are presented In the following sections, while their functions have been detailed in ANNEX 1. Figure 16 shows a diagram of the different classes created and their relations between them. Moreover, it is included the scenario script, which is related to all classes. In ANNEX 3 it is given a detailed scheme of the classes with all its functions and attributes. Moreover, in ANNEX 4 it is included an example of how to run the simulator.

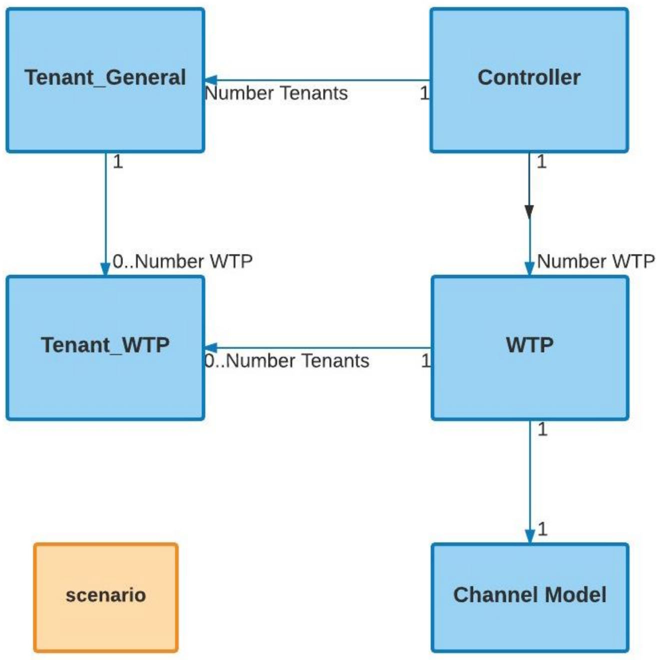


Figure 16. Block diagram of the hypervisor simulator with the different classes and scripts

3.5.2.1. Channel Model

The channel model class simulates the effects of a wireless channel when packets are transmitted from the WTP to the final user’s terminal. It gives a random behaviour to the packet transmission, giving the possibility of packet retransmissions. As a result, the channel model allows us to compute the time required to transmit a certain packet.

In order to do so, the algorithm in Figure 17 has been developed. For each packet, the algorithm randomly selects a modulation scheme for the packet to be transmitted. After this, it is checked if the packet has been transmitted successfully or not. If so, the algorithm exits but if not, the algorithm checks how many times the packet has been retransmitted. If the number of retransmissions is greater than 3, the modulation scheme is decreased to a more robust modulation scheme (slower one) and then the packet is retransmitted. If the number of retransmissions is less than 3, the packet is retransmitted without modifying the modulation scheme. All transmission and retransmission modulation schemes used during the packet transmission are stored in a list.

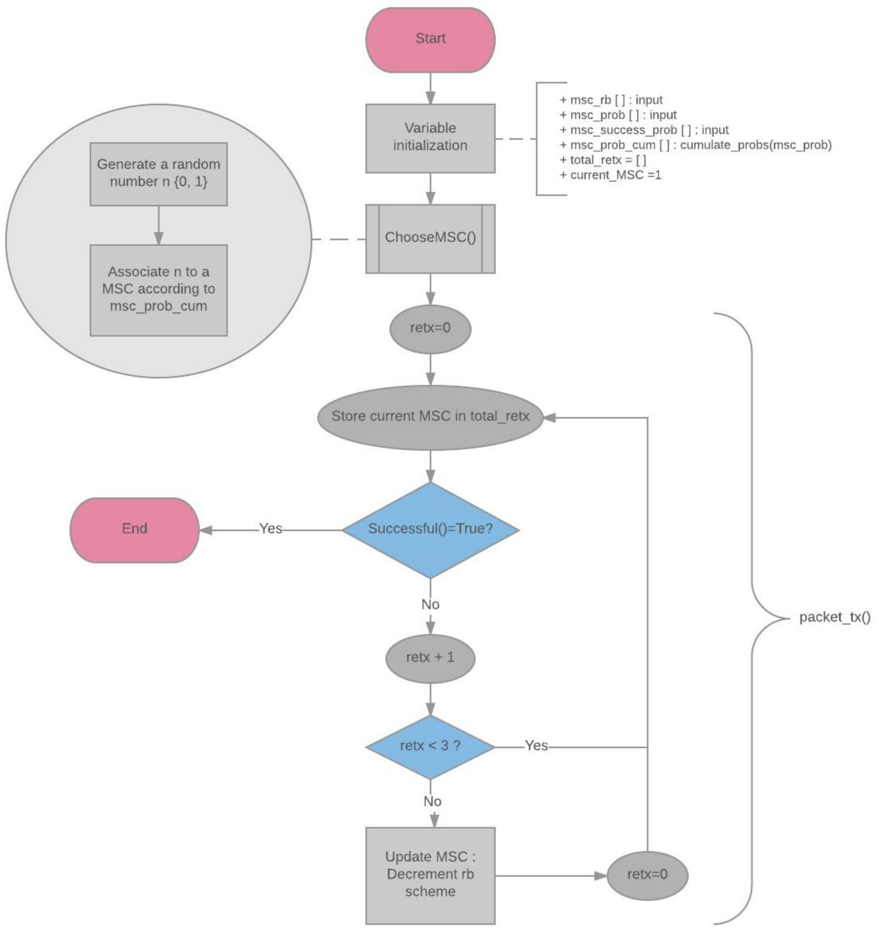

Start
Variable initialization
+ msc_rb [ ] : input
+ msc_prob [ ] : input
+ msc_success_prob [ ] : input
+ msc_prob_cum [ ] : cumulate_probs(msc_prob)
+ total_retx = [ ]
+ current_MSC =1
Generate a random number n {0, 1}
Associate n to a MSC according to msc_prob_cum
ChooseMSC()
retx=0
Store current MSC in total_retx
End
Yes
Successful()=True?
No
retx + 1
retx < 3 ?
Yes
No
Update MSC : Decrement rb scheme
retx=0
packet_tx()

Figure 17. Channel Algorithm

For testing, the following modulation schemes had been defined. In addition, these are the default values used if no others are specified for a simulation.

Index	MCS transmission rate	MCS probability	MCS cumulated probability	MCS success probability
1	**54 Mbps**	0.8	0.8	0.9
2	**48 Mbps**	0.1	0.9	0.95
3	**24 Mbps**	0.05	0.95	0.98
4	**12 Mbps**	0.03	0.98	0.99
5	**6 Mbps**	0.02	1	0.999

Table 1. Modulation schemes data rates, occurrence probability, cumulated probability and success probability

3.5.2.2. Tenant General

The concept of tenant in the simulator has been split into two parts: the class Tenant General and class Tenant WTP. The class Tenant General is the class in which the global properties of a certain tenant in the system are specified, while the class Tenant WTP consists of the instance of a certain tenant in a WTP. This has been divided in this way in order to easily generate and manage the traffic of each of the tenants in each WTP, but maintaining the entity of the Tenant as a general element in all the system.

As mentioned, the class Tenant General defines a certain virtual operator in the entire network. This class has been designed in a way that allows that all the Tenants WTP instances of a certain Tenant General belong to it and can be accessed by it.

3.5.2.3. Tenant WTP

The Tenant_WTP class represents the instance of a certain Tenant_General in a certain WTP, as it has already been mentioned.

An important variable that belongs to the tenant_WTP is the queue of packets of the tenant in that WTP, which consists of a FIFO (First In First Out) queue.

The traffic generated by a certain tenant is not generated in the Tenant_General and then passed to the Tenant_WTP, but instead it is generated in the Tenant_WTP. This decision was taken for simplicity reasons, as it is easier to manage the queue locally, generating its traffic and processing their packets in the Tenant_WTP. Considering this, different traffic generators are included in the class Tenant_WTP. These traffic generators generate traffic considering the SLA, so in average the traffic generated will not be greater than the agreed.

The first traffic generator designed (Figure 18) was focused on having a certain amount of bytes in the queue but had not into account the time synchronization between the packet generation and the packet transmission. Moreover, for the studies of the different algorithms, it was convenient to be able to establish a certain traffic generation rate during the simulation, which is not possible with this first traffic generator.

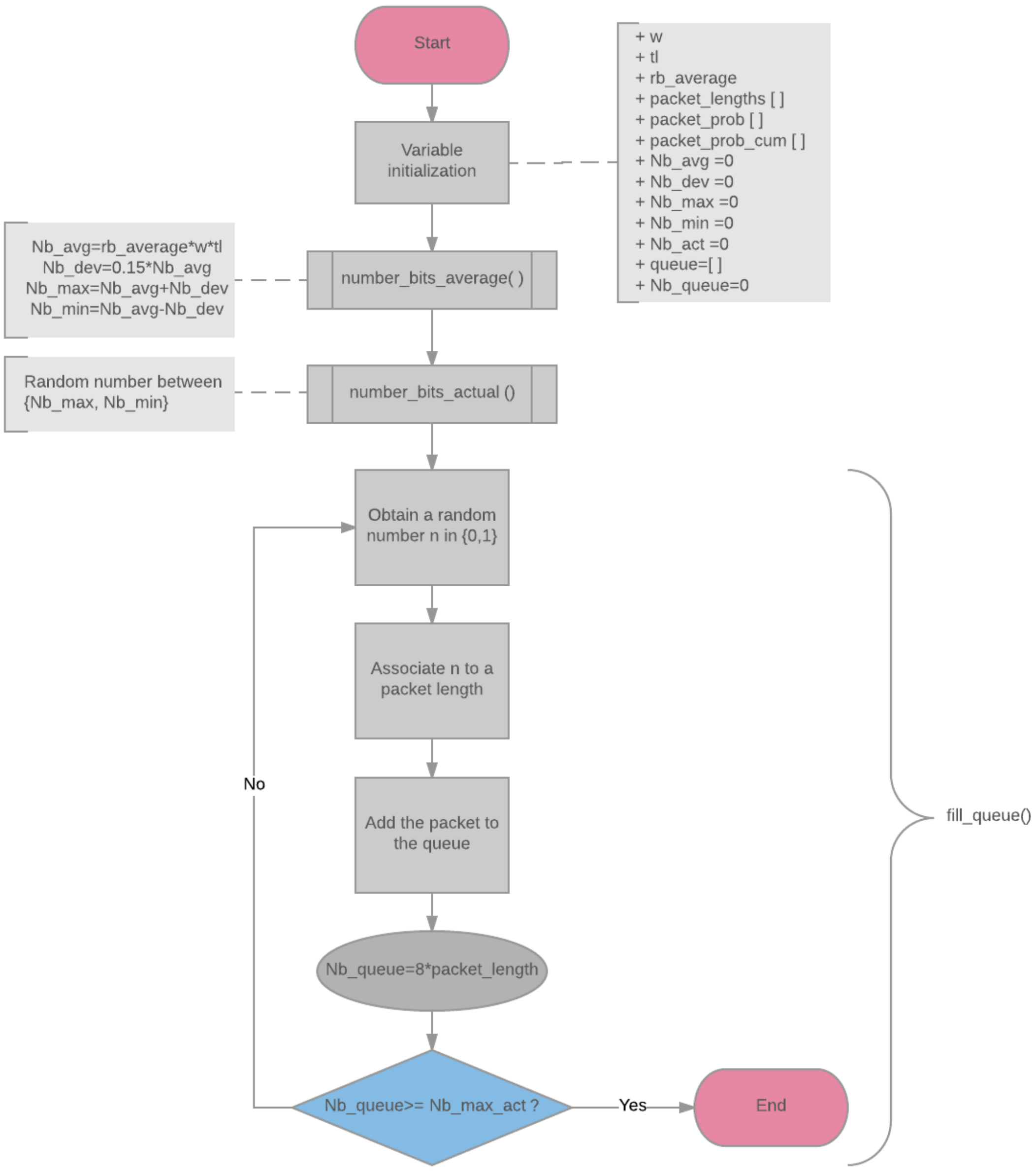
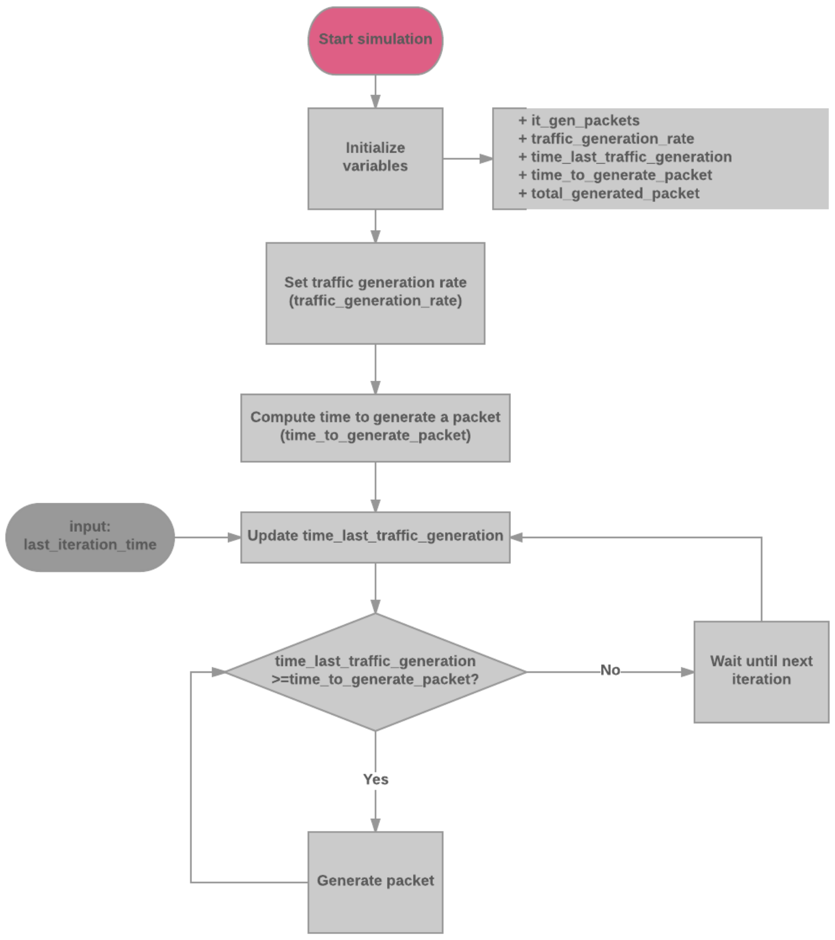


Figure 18. Initial traffic generator scheme

Considering the disadvantages of the initial proposed traffic generator, it was defined a new traffic generator, which is time based and works at a fixed traffic rate. For this reason, this second generator is called fixed generator and its principle of operation is described in Figure 19.



Start simulation
Initialize variables
+ it_gen_packets
+ traffic_generation_rate
+ time_last_traffic_generation
+ time_to_generate_packet
+ total_generated_packet
Set traffic generation rate (traffic_generation_rate)
Compute time to generate a packet (time_to_generate_packet)
input: last_iteration_time
Update time_last_traffic_generation
time_last_traffic_generation >=time_to_generate_packet?
No
Wait until next iteration
Yes
Generate packet

Figure 19. Fixed traffic generator scheme

A critical issue for the fixed generator is how to select the traffic generation rate. In the traffic study of the section 4.1.5, different measures that have been carried out in order to set the rate, are discussed. Through the study developed, it has been possible to formulate the following equations that allow establishing a traffic generation rate for a certain tenant, while avoiding the queues to indefinitely increase and use the maximum of the capacity associated to the tenant. In order to establish the data rate, it is necessary to compute the capacity of a WTP and then the capacity associated to the tenant in that WTP.

$$Capacity_{WTP_{Rbi}}[bits] = Rb_i\ [bps] \cdot Q_s[s] \quad for\ i\ in\ 1. \tag{14}$$

$$Capacity_{Tenant\ k_{Rb_i}}[bits] = w_k \cdot Capacity_{WTP_{Rbi}}[bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}} \tag{15}$$

The capacity of the tenant depends on the capacity of the WTP, the SLA weight of the tenant and the factors r_i, which is used to compensate differences added by 802.11g delays, and f_{retx}, that compensates the effect of retransmissions. The f_{retx} factor formula has been fixed through simulations. Both factors are defined as:

$$r_i = \frac{time_{delays_{Rb_i}}[s]}{time_{total\ packet\ transmission_{Rb_i}}[s]} \quad (16)$$

$$f_{retx_{rbi}} = p_{retx_{rb_i}} - 0.01 \quad (17)$$

The capacity is computed for each possible data rate in the WTP. After this, the resultant capacities are used to compute the traffic generation rate, considering the probabilities associated to the data rate in which the capacity was initially computed.

$$G_{T_{Tenant\ k}} = \frac{\sum_{i=0}^{N} Capacity_{Tenant\ K_{Rb_i}} \cdot p_i\ [bits]}{Q_s[s]} \quad (18)$$

$$G_{T_{Tenant\ k}} = \frac{\sum_{i=0}^{N} w_k \cdot Rb_i[bps] \cdot Q_s[s] \cdot p_i\ [bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}}}{Q_s[s]} = w_k \cdot \sum_{i=0}^{N} Rb_i[bps] \cdot p_i\ [bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}} \quad (19)$$

$$G_{T_{Tenant\ k}} = w_k \cdot Rb_{effective_{average}} \quad (20)$$

The effective data rate in the last equation corresponds to equation (21).

$$Rb_{effective_{average}} = \sum_{i=0}^{N} Rb_i\ [bps] \cdot p_i \cdot (1 - r_i) \cdot\ f_{retx_{rbi}} \quad (21)$$

As one of the requirements of the simulator was to be able to have different traffic generator modes, it has been designed a uniform traffic generator, and two Gaussian traffic generators.

The **uniform traffic generator** has a uniform distribution, which in our case, the minimum traffic generation rate is 0 Mbps and the maximum generation rate is computed as in equation (22). This equation considers the maximum data rate that the WTP is working with and the weight of the tenant. Notice that r_{Rbmax} corresponds to the delays compensation r_i at the maximum data rate Rb_{max}.

$$G_{T_{max}}[bps] = \frac{w \cdot\ Capacity_{Rb\_max}[bits]}{Q_s[s]} = \frac{w \cdot\ Rb_{max} \cdot (1 - r_{Rbmax}) \cdot SQ[s]}{Q_s[s]} = w \cdot Rb_{max} \cdot (1 - r_{Rbmax}) \quad (22)$$

It is relevant to mention that the retransmission compensation factor is not considered in the computation of Gt_{max} as it is computed as a peak value, so when no retransmissions occur it could happen that the generation rate is the computed without that factor.

Moreover, two Gaussian generators have been designed. For the first Gaussian generator, the mean of the Gaussian distribution has been set to half Gt_{max}. Gt_{max} is computed in the same way as in the uniform generator. Considering Figure 20, most of the probability is concentrated into the interval (μ-3σ, μ+3σ). So, it has been truncated the function making this interval coincide with (0, Gt_{max}). The standard deviation (σ) and mean (µ) of our Gaussian distribution are computed in equations (23) and (24) respectively.

$$\mu = \frac{Gt_{\max}}{2} \qquad (23)$$

$$\mu - 3\sigma = 0 \rightarrow \sigma = \frac{Gt_{\max}}{6} \qquad (24)$$

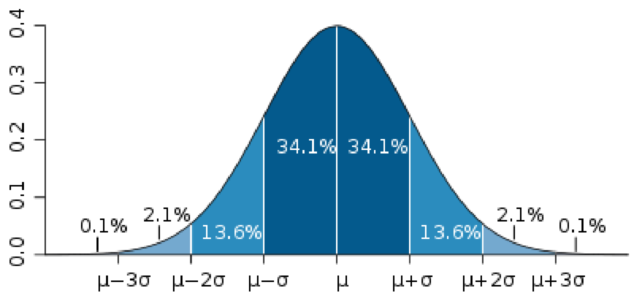


Figure 20. Gaussian distribution

The second Gaussian generator is called shifted Gaussian generator. In this case, the mean has been shifted according to equation (25) and the standard deviation has been defined as the 15% of the mean value, although it can be tuneable.

$$\mu = w_{tenant\ i} \cdot Capacity_{average,wtp} \qquad (25)$$

$$\sigma = 15\% \cdot \mu \qquad (26)$$

The need of this last Gaussian generator has resulted during the weights algorithm tests, as it was needed a generator in which sometimes the traffic requested was greater than the capacity assigned to a tenant.

3.5.2.4. WTP

As it has already been defined previously, the WTP is the element in charge of providing wireless connectivity to the different tenants' users. In addition, in the WTP is where the hypervisor is placed and the scheduling of packets is performed.

The WTP in the simulator is capable of scheduling packets by using the ADRR algorithm as well as RR and WDRR. These two last algorithms have been included to be able to compare them with the ADRR operation.

Considering that the WTP is responsible for running the scheduling algorithm, it contains different variables related to the time spent in each of the iterations, as well as during all the simulation. Because of this, in this class it has been required to compute different times: the packet transmission time and the empty queue time, explained below.

For the computation of the packet transmission time, the simulator uses equation (27), in which the 802.11g delays values shown in Table 2 are considered. It has not been considered the back-off times in 802.11g as we are just focusing on the downlink so collision avoidance is not needed. It is important to point out that in equation (27), it is taken into account the transmission of the MAC data packet and MAC ACK packet like in Figure 21. This is the reason why the physical layer and signal extension delays are multiplied by 2. Figure 22 shows the 802.11g frame format, where is possible to identify the fields of physical layer and signal extension. It has to be considered that the column physical layer in the table coincides with the preamble plus the signal in Figure 22.

$$time_{packet\_tx} = t_{DIFS} + 2 \cdot t_{phy_{layer}} + \frac{Packet_{length}}{R_b} + 2 \cdot t_{signal_{extension}} + t_{SIFs} + \frac{ACK_{length}}{Rb} \qquad (27)$$

DIFS(us)	Physical layer (us)	Signal extension (us)	SIFS	ACK length (bytes)
28	20	6	10	14

Table 2. 802.11g delays considered

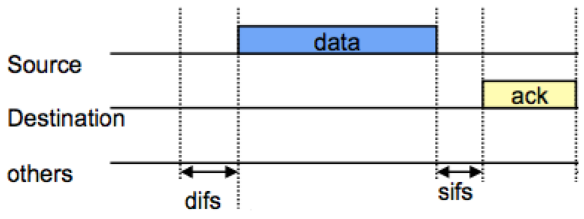


Figure 21. 802.11 error control in radio medium

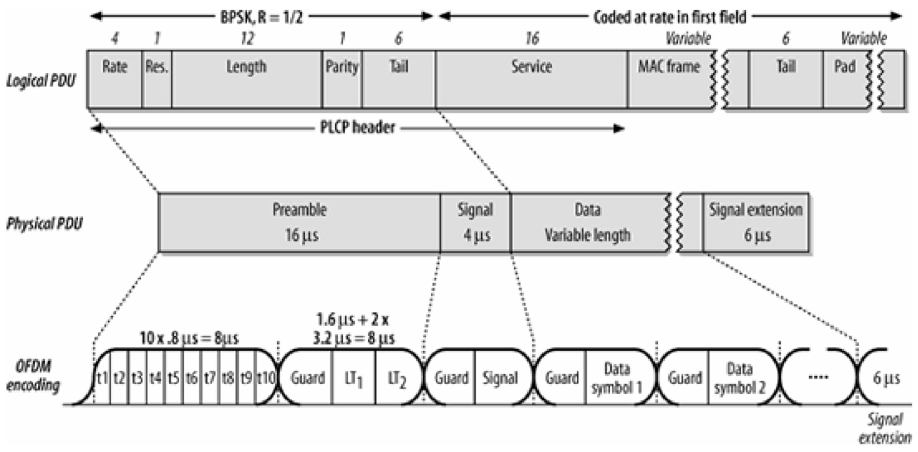


Figure 22. 802.11g ofdm frame format

As commented, this class considers an empty queue time, which is added to the iteration time when the queue of a certain tenant is empty. This decision was taken in order to simulate that when the queue is empty the time keeps running, as well as to consider the processing time when looking at the queue.

Finally, the WTP class includes some computations required for the weight compensation algorithm. It includes the computation of the WTP average capacity, which is defined as the capacity that the WTP can serve, and the excess and solicited capacities in the WTP, in which the traffic demands of the tenants in that WTP are considered.

The average capacity that a WTP can serve is computed taking into account equations (28), (29) and (30). The capacity average computation uses the effective data rates of the WTP (including 802.11g delays), its probabilities and a new compensation factor f_{comp}, explained below.

Equation (29) shows how the average time to transmit given a certain data rate Rb_i is computed. Notice that it is an approximation, as it just considers three retransmissions while the programmed algorithm decreases the data rate if a packet is sent incorrectly three times, which will have a low probability with the values given. Equation (30) computes the new compensation factor, needed to avoid long queues. This is caused by the difference of maximum and minimum data rate allowed in the WTP, as when a low data rate is used it processes fewer packets, while new packets continue arriving to the system, which results in an increase of the queue. Therefore, we use the relation of times of the maximum data rate and the minimum data rate in the system to avoid packets to be accumulated in the queue.

$$C_{average} = f_{comp} \cdot \sum_{i=0}^{N-1} \frac{Packet_{length_{average}}(bytes) \cdot 8 \frac{bits}{Byte}}{tp_{average_{Rbi}}} \cdot prob(Rb_i) \quad (28)$$

$$tp_{average_{Rb_i}} = tp(Rb_i) \cdot prob(packet_{ok}) + 2 \cdot tp(Rb_i) \cdot prob(packet_{ok}) \cdot prob(packet_{Nok}) + 3 \cdot tp(Rb_i) \cdot prob(packet_{ok}) \cdot prob(packet_{Nok})^2 \quad (29)$$

$$f_{comp} = 1 - \left(\frac{tp\left(MAX\ Rb, packet\ lenght_{average}\right)}{tp\left(MIN\ Rb, packet\ lenght_{average}\right)}\right) \quad (30)$$

3.5.2.5. Controller

The Controller is the element in charge of managing the different WTPs and assuring the SLA of the different tenants in the network. The controller has a global vision of all the elements in the system, being capable of acceding to all the WTPs and the tenants' information.

In the simulator, the controller has three main responsibilities: creating the instances of each of the tenants in the WTP, performing the weight compensation algorithm, whose operation has already been described, and running all the system.

The controller creates the instances of the tenants in the WTP as it controls in which WTP a certain tenant operates. It might be desired that a certain tenant does not operate in a part of the network. Moreover, the controller has been chosen to run all the system functions as it has accessibility to all the elements.

3.5.2.6. Scenario

The scenario is a python script, not a class, which contains different global system variables and generic functions. It was decided to use a separated script for these auxiliary functions, so that all the system classes can access them in a shared manner. Moreover, some of these functions are used for checking the expected values of other functions in a faster way. This is the case of some traffic generation rate functions.

The different functions included in the scenario can be found in ANNEX 1.

3.5.3. Exportation results files

A relevant requirement for the simulator was the capability of exporting files with the simulation results to be analysed and checked. To do so, it has been decided to generate files using the format csv (comma-separated values format), which can be analysed using excel.

Different exportation files are created during the simulation. Three different types of exportation files have been designed are the following:

a) **Packet transmission results aggregated by iterations.** These files contain the abstract of the data transmitted per iteration[5] for each tenant in each WTP. In

Table 3 the results of the packets results exportation file during iterations 68 and 69 of a simulation is shown. For each of the iterations, it is possible to consult the simulation time reference, the iteration time, the transmitted bytes, the number of packets transmitted and its type, the packets generated in the iteration and the packets in the queue.

Iteration	Time	Total it time	Tx bytes	Packet Tx	Tx Type 1	Tx Type 2	Tx Type 3	Retx Type 1	Retx Type 2	Retx Type 3	Generated _packets	Packets in queue
68	67191.7226	1898.22222	6056	4	4	0	0	2	0	0	4	0
69	69722.6855	1265.48148	6056	4	4	0	0	0	0	0	5	1

Table 3. Packet transmission results example

b) **Packet transmission results detailed.** In this file, it is possible to obtain more details about the data obtained in the file of *packet transmission results* aggregated by iterations. Table 4 depicts in detailed iterations 68 and 69 of the same simulation as before. Notice that in this case, the details of each packet transmission are exported: the packet length, the packets in the queue in that instant, the generated packets in the iteration, the traffic generation rate, the initial and final *DC*, the data rate used, and the theoretical and real times.

Iteration	Time	Packet Lenght	Packets in queue	Generated _packets	Gt	Initial DC	Rb used	Tp teoric	Tp_real	Final DC	Transmitted?
68	67191.7226	1514	4	4	23851083.1	1400	[1]	316.37037	316.37037	1083.62963	YES
68	67191.7226	1514	3	4	23851083.1	1083.62963	[1]	316.37037	316.37037	767.259259	YES
68	67191.7226	1514	2	4	23851083.1	767.259259	[1, 1, 1]	316.37037	949.111111	450.888889	YES
68	67191.7226	1514	1	4	23851083.1	450.888889	[1]	316.37037	316.37037	134.518519	YES
68	67191.7226	0	0	4	23851083.1	0	0	0	0	0	NO PACKET
69	69722.6855	1514	5	5	23851083.1	1400	[1]	316.37037	316.37037	1083.62963	YES
69	69722.6855	1514	4	5	23851083.1	1083.62963	[1]	316.37037	316.37037	767.259259	YES
69	69722.6855	1514	3	5	23851083.1	767.259259	[1]	316.37037	316.37037	450.888889	YES
69	69722.6855	1514	2	5	23851083.1	450.888889	[1]	316.37037	316.37037	134.518519	YES
69	69722.6855	1514	1	5	23851083.1	134.518519	0	316.37037	0	134.518519	NO

Table 4. Packet transmission detailed example

c) **Weight compensation exportation files.** It contains all the information about the weight compensation algorithm. Table 5 shows an example of the *weights algorithm exportation* file. In this file, it is detailed the computed variables $C_{measured}$, C_{req}, C_{exc} *and* C_{sol} for each tenant in each WTP for each of the times the weights are modified.

[5] An iteration is defined as the amount of time that a tenant_wtp has the opportunity to transmit, so that it is checked its DC to see if it is possible to transmit a packet until it is not possible to transmit more packets.

Change weights number	Iteration	Time (us)	Time Passed(us)	WTP	Tenant	WTP Avg Capacity (Mbps)	Csla	Generated Bytes tenant	Total bytes WTP	Cmeasured	Creq	Cexc	Csol	w
0	1	3584.29461	0	0	0	28159887.6	0.7	0	0	0	0	0	0	0.7
0	1	3584.29461	0	0	1	28159887.6	0.3	0	0	0	0	0	0	0.3
1	5032	10005130	10001545.7	0	0	28159887.6	0.7	34505574	35205300.4	0.9801244	0.2801244	0.18001636	0.2801244	0.88001636
1	5032	10005130	10001545.7	0	1	28159887.6	0.3	4224060	35205300.4	0.11998364	-0.1800164	0.18001636	0.2801244	0.11998364
2	9134	20005042.2	9999912.22	0	0	28159887.6	0.7	34493462	35199550.5	0.97994041	0.27994041	0.17999676	0.27994041	0.87999676
2	9134	20005042.2	9999912.22	0	1	28159887.6	0.3	4224060	35199550.5	0.12000324	-0.1799968	0.17999676	0.27994041	0.12000324
3	13258	30006488.4	10001446.1	0	0	28159887.6	0.7	34487406	35204949.9	0.9796181	0.2796181	0.17984315	0.2796181	0.87984315
3	13258	30006488.4	10001446.1	0	1	28159887.6	0.3	4230116	35204949.9	0.12015685	-0.1798431	0.17984315	0.2796181	0.12015685
4	22267	40003999.1	9997510.74	0	0	28159887.6	0.7	14785724	35191097.3	0.42015524	-0.2798448	0.27984476	0.18021429	0.48989657
4	22267	40003999.1	9997510.74	0	1	28159887.6	0.3	16899268	35191097.3	0.48021429	0.18021429	0.27984476	0.18021429	0.51010343
5	26876	50005564.2	10001565.1	0	0	28159887.6	0.7	14782696	35205368.7	0.41989891	-0.2801011	0.28010109	0.17984761	0.49007635
5	26876	50005564.2	10001565.1	0	1	28159887.6	0.3	16893212	35205368.7	0.47984761	0.17984761	0.28010109	0.17984761	0.50992365

Table 5. Weights compensation exportation file example

3.5.4. Time management

Even though the time management has been commented when explaining the different classes, it deserves to be explained separately to fully understand its operation.

Provided that in the simulator the packets are not transmitted at a real time, it has been necessary to simulate the time dimension. The time management is fundamental for generating traffic, running different WTPs and compensating the weights of the different WTPs. Two main strategies for the time management were proposed:

a) **System clock**. This first option consists of creating a general clock in the controller and defining a certain small period of time that could be passed to the WTPs for giving them a reference time to operate. The problem of this solution is that, in our system, it is not possible to interrupt the packet transmission. So, it is really complex to have an entity controlling all the timers. For this solution, it would be needed to have time compensation variables to compensate the differences in the timers and the time period given to WTPs would have been critical.

b) **WTP time controlled.** In this mode, each WTP has its own time counter. By defining a simulation time, the WTP will run until each of the WTP timer reaches the simulation time. Differences in each WTP timers can be produced, as packet transmissions cannot be stopped. However, these differences are really small. With this option it is not needed to stop the WTP operation in the middle of iterations or packet transmissions.

The strategy finally selected is the second one in order to keep the complexity of the simulator at low levels. The key fact of this option is to count the time spend by each WTP in each iteration. Thanks to this, it is not just possible to count the total time run in a WTP, but also to generate traffic according to the time spent in the previous iteration. In this way, it is possible to manage both the packet generation and transmission.

An important aspect of the synchronization is the parallel operation of the WTPs. Although tenants in a WTP do not operate at the same time, different WTPs do. So, different solutions regarding this have been tested:

a) **Multiprocessing**. An interesting approach is the use of threads through a Python class called multiprocessing. Thanks to this class, it is possible to run the different WTPs in parallel, giving them the time they need to run. The main disadvantage of this solution is that the class multiprocessing does not allow to access the variables inside the class being run, as it creates its own runnable objects. These objects' variables have to be accessed using other complex functions. In other words, this means that it is not possible to easily access the variables of class

WTP and consequently class Tenant_WTP, which is needed for the weight algorithm. Giving the complexity that this solution adds to the simulation, its application has been reconsidered.

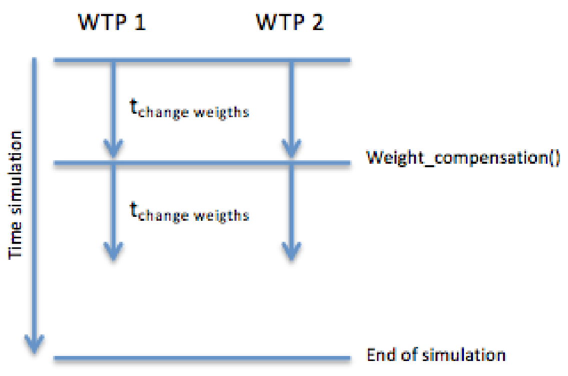


Figure 23. Parallelism by using multiprocessing

b) **Sequential parallelism.** This solution tries to simulate a fictitious parallelism, running the WTP operation of different WTPs at the same time but sequentially. Each of the WTPs run the same amount of time and, as the different WTPs do not interact, the same results than in the multiprocessing solution are found, being capable of applying the weights compensation algorithm. The simulations with this solution last longer than with multiprocessing solution, but it allows us to evaluate the weight compensation algorithm.

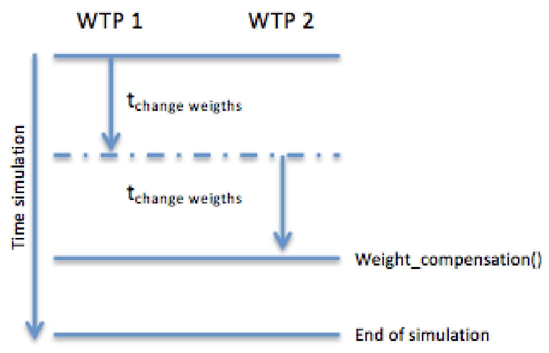


Figure 24. Parallelims using sequencial parallelism

Considering the sequential parallelism, the different WTPs run during the defined period before a weights change and then the weights are modified. Then the WTPs run during this period as many times until the simulation time is reached. It has to be considered that the WTPs will run exactly during the period before the weights change, so differences between WTP timers may occur. However, the differences are extremely small so it can be considered that they run the same time.

4. Studies

During the development of the ViRANsim simulator, several studies have been performed in order to evaluate the performance of the simulator and analyze the operation of the different algorithms implemented. Through the studies, it has been possible to take important decisions about the implementation of the simulator as well as to foresee possible behaviors before applying the algorithms in the real EmPOWER testbed.

Figure 25 shows how the different studies performed are organized. In the first block, there are included the simulations where a single WTP is considered while in the second block scenarios with multiple WTPs have been analyzed. In addition, the different studies have been classified depending on the topic they study: the simulator operation, the ADRR algorithm operation or the weight compensation algorithm performance.

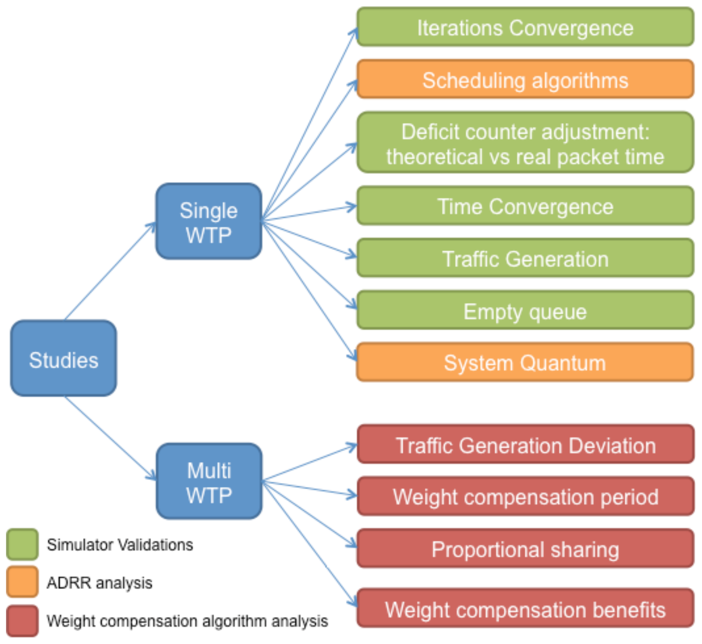


Figure 25. Studies performed

4.1 Single WTP studies

In this section different studies in which a single WTP is considered are presented. In section 4.1.1 it can be found a convergence study in terms of iterations, while in section 4.1.2 a comparison between the ADRR, WDRR and RR is provided. Moreover, in section 4.1.3 a comparison between the use of the expected or real packet transmission time for the *DC* adjustment is analyzed. Regarding the needs of studying the algorithm convergence focusing on time, an analysis of the simulation results has been developed in section 4.1.4. In section 4.1.5 the operation of the different traffic generators is tested. The last two studies consist in the justification of the empty queue time in section 4.1.6 and a brief analysis of the system quantum value determination in 4.1.7.

4.1.1. Iterations Convergence Study

In this first study we focus on the convergence of the algorithm in order to determine the necessary number of iterations, with main purpose to obtain valid results and minimize the size of the result simulation files. It is worth of mentioning that this is the first study performed with the ADRR and it is being considered that the number of iterations is an input parameter, as it is the initial implementation. This study is also focused to get a first approach of the behavior of the ADRR and the simulator performance.

It is also important to point out that the DC adjustment in this section is performed by reducing the real packet transmission time from the DC after the transmission, as in equation**(31)**.

$$DC_{Tenant\ i} = DC_{Tenant\ i} - tp_{real} \quad (31)$$

The scenario used for the simulations in this section consists in two tenants in one WTP that share the available resources based on the given weights. The Modulation Coding Scheme (MCS) and the packet length are chosen randomly, according to their associated probabilities in Table 6 and Table 7, respectively.

Packet Length (Bytes)	Packet probability
1514	**0.7**
512	**0.3**

Table 6. Packet lengths and probabilities

MCS transmission rate	MCS probability	MCS success probability
54 Mbps	0.8	0.9
48 Mbps	0.1	0.95
24 Mbps	0.05	0.98
12 Mbps	0.03	0.99
6 Mbps	0.02	0.999

Table 7. Data rate, probability and success probability of each modulation scheme

The study has been developed focusing on the ADRR and it has been analyzed for both cases of using and not using delays from the 802.11g protocol (SIFS, DIFS…) and when varying the weights associated to the tenants. With this, it has been possible to prove that the convergence is not affected by these parameters.

In order to determine the minimum number of iterations required to converge, for each of the cases, different simulations of different number of iterations have been run. For each of the simulations, the average time and transmitted bytes over all the iterations have been computed and the percentage over the total time and bytes of both tenants has been obtained. By doing this, it has been possible to study how both parameters are shared when using ADRR. As ideally, for both tenants, the percentages of time and bytes should be equal to its weight, it has been computed the dispersion in relation to specified weight (i.e. for the case of 50%-50%, the dispersion is computed around 50%). With the dispersion is possible to evaluate how close to the desired weight is sharing the scheduler the resources to the different tenants.

In Figure 26, the dispersion when focusing on the time sharing obtained from a simulation with 50%-50% sharing and considering 802.11g delays is depicted. It can be seen how after 500 iterations, the dispersion decreases drastically to dispersions of the order of 1E-4. However, when looking at the bytes sharing in Figure 27, higher dispersion values are observed.

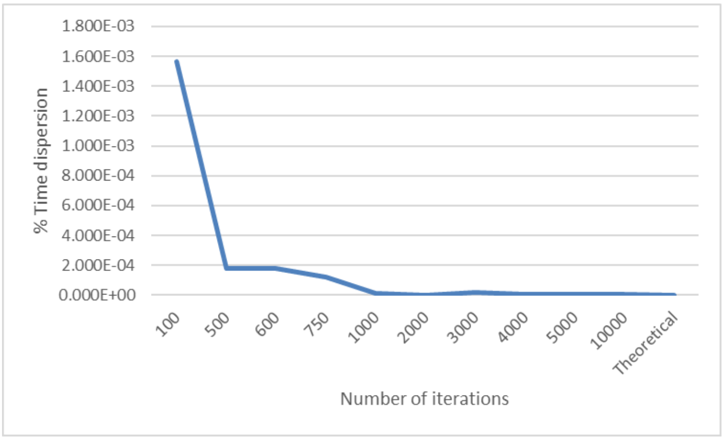


Figure 26. Time dispersion over 0.5 for different number of iterations. Case considering 802.11g delays

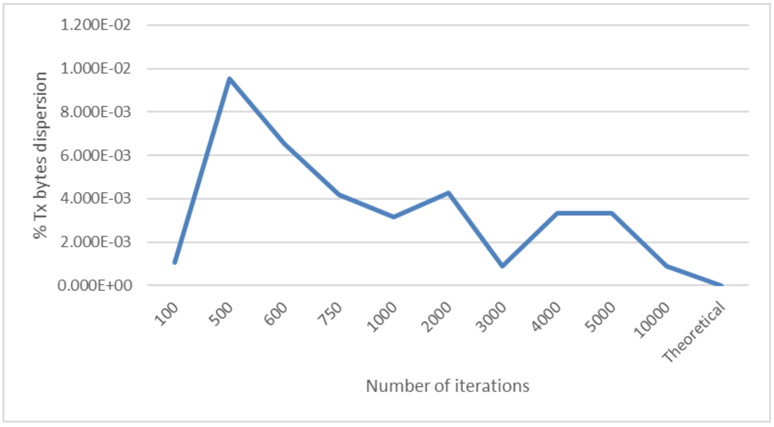


Figure 27. Transmitted bytes dispersion over 0.5 for different number of iterations. Case considering 802.11g delays

The difference in the dispersion between the two graphs is due to the fact that the ADRR quantum is set in time units, so it adjusts the time used by each tenant. However, the algorithm does not limit the number of bytes transmitted directly, but through the time quantum.

Similar results have been observed when the 802.11g delays are not considered and with different weight repartition. However, when performing the same simulations but using the WDRR algorithm, the dispersion was lower for the transmitted bytes sharing than for the time sharing, as the system quantum is set in bytes. The details of those simulations can be found in ANNEX 2.

Regarding the dispersion threshold defined at 1E-4, it can be concluded that with more than 500 iterations it is possible to achieve reliable simulations.

4.1.2. Scheduling algorithms comparison

The simulation results obtained with the ViRANsim simulator with purpose to verify and study the operation of the employed scheduling algorithms are presented in this section.

The scenario consists in two tenants and one WTP, managed by the controller. The traffic for the two tenants is generated with a fixed rate of 1Mbps for all simulations. Tenant 1 sends packets of 1514 bytes while Tenant 2 sends packets of 512 bytes.

For each of the studied cases, the results presented correspond to 700 iterations, time that is sufficient for the algorithm to reach convergence.

4.1.2.1. 54 Mbps without 802.11g delays: Tenant 1 50% - Tenant 2 50%

As a first approach, the transmission rate has been fixed to 54Mbps and the headers in 802.11g have been omitted. Moreover, the transmission error probability has been set to 0. In this case, for the WDRR and ADRR the weights of each tenant have been set to 50%.

Table 8 depicts the results for 700 iterations. The first three columns show the total of time in microseconds, the transmitted bytes and the number of packets transmitted, respectively. The fourth column is the computation of the bandwidth, which can be understood as the effective data rate associated to a tenant in a WTP. This is computed according to the equation **(32)** and it is the throughput that the tenant perceives when it is its turn in the scheduling algorithm.

$$BW = \frac{Tx_{bytes} \cdot 8\ bits}{Total_{time}(s)} \left[\frac{bits}{s}\right] \qquad (32)$$

Moreover, the fourth last columns compute the percentage of time used, the bytes transmitted, the number of packets transmitted and the bandwidth of each tenant over the total obtained by both tenants of each corresponding parameter. The following expression shows how the percentage of time has been computed. The rest of the percentages have been computed in a similar way.

$$\%Time_{Tenant\ 1} = \frac{Total\ Time_{Tenant\ 1}}{Total\ Time_{Tenant\ 1} + Total\ Time_{Tenant\ 2}} \qquad (33)$$

$$\%Time_{Tenant\ 2} = \frac{Total\ Time_{Tenant\ 2}}{Total\ Time_{Tenant\ 1} + Total\ Time_{Tenant\ 2}} \qquad (34)$$

RR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	%Time	%Bytes	%Tx Packets	%BW
Tenant 1	158407.41	1059800	700	53522749.59	0.74403	0.7473	0.5000	0.50429
Tenant 2	54496.30	358400	700	52612749.76	0.25597	0.2527	0.5000	0.49571

WDRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	%Time	%Bytes	%Tx Packets	%BW
Tenant 1	78503.70	529900	350	54000000.00	0.50023	0.5002	0.2529	0.5000
Tenant 2	78430.81	529408	1034	54000000.00	0.49977	0.4998	0.7471	0.5000

ADRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	%Time	%Bytes	%Tx Packets	%BW
Tenant 1	78728.00	531414	351	54000000.00	0.49998	0.49998	0.2527	0.5000
Tenant 2	78734.22	531456	1038	54000000.00	0.50002	0.50002	0.7473	0.5000

Table 8. Results obtained for RR, DRR and ADRR for the case of fixed 54Mbp without 802.11 g delays and 50%-50% sharing for both tenants.

In order to correctly understand the results obtained, it is necessary to know the time required to send each of the packet types, when no headers are considered. This is shown in Table 9.

	Packet length (Bytes)	Time to transmit a packet (us)
Tenant 1	1514	224.3
Tenant 2	512	75.85

Table 9. Time needed to transmit each type of packet

In the case of **Round Robin (RR)**, in each step, each tenant will send one packet if there is one in its queue. As the simulation consists of 700 iterations, the total number of packets transmitted by each tenant is 700 packets. Moreover, if we look at the percentage of time and bytes transmitted by each tenant, it can be observed the relation between the packet lengths of each tenant in Table 10.

	Packet Length (Bytes)	% Packet Length
Tenant 1	**1514**	**0.747285291**
Tenant 2	**512**	**0.252714709**

Table 10. Percentage that represents each packet length over the sum of both lengths

In the case of the percentage of time, in order to send a packet of 1514 bytes, it will take the 74,7% of the time and to send a packet of 512 it will take the 25,27% of the time since in RR each tenant sends one packet at each iteration and retransmissions are not considered in this first simulation. For the same reason, the percentage of transmitted packets follows the same relation, as Tenant 1 will transmit the same number of packets as Tenant 2 but with its associated lengths.

In the case of the **Weighted Deficit Round Robin (WDRR),** the weights for both tenants are 50%, and the system quantum is set to 1514 bytes, so each tenant can transmit up to 757 bytes (1514 bytes ·50%) per iteration. In the simulation results, it is shown how the percentage of time and bytes used by each tenant is set as desired (50-50%). The reason why the time used by each of the tenants is the same, is that both have to send the same amount of bytes and both transmit at the same rate (54Mbps) even though the packets of each tenant have different sizes. However, the percentage of packets transmitted is the inverse of the packet relation, shown before. This is because Tenant 1, which sends packets of 1514 bytes, has 714 bytes to transmit in each iteration, so it will transmit fewer packets than Tenant 2, which sends packets of 512 bytes, but both will transmit the same amount of bytes.

As already explained, **Air Time Deficit Round Robin (ADRR)** works like WDRR but the quantum is set in units of time. In this case, the system quantum is set to 225us. Considering that each tenant has a weight of 50%, each of them will have 112,5us to send its packets. The selected system quantum assures the transmission of the larger packet (1514 bytes) at a data rate of 54Mbps. Equation **(35)** computes the minimum required system quantum for this system.

$$Q_S = \frac{1514\ bytes \cdot 8\frac{bits}{byte}}{54\ Mbps} = 224.29\ \mu s \sim 225\ \mu s \qquad (35)$$

The number of packets transmitted in each of the iterations in ADRR depends on the system quantum, which is set to 225us (112,5us for each tenant). Considering the times to transmit packets of 1514 bytes and 512 bytes, the number of packets expected to be send in an iteration of each of the lengths is the following one, which is the same as obtained through simulation.

$$number_{packets}(packet\ 1514\ bytes) = \frac{112.5\ us}{224.3\ us} = 0.5015\frac{packets}{iteration}$$
$$\rightarrow 700\ iterations \sim 351\ packets \quad (36)$$

$$number_{packets}(packet\ 512\ bytes) = \frac{112.5\ us}{75.85\ us} = 1.48\frac{packets}{iteration}$$
$$\rightarrow 700\ iterations \sim 1038\ packets \quad (37)$$

4.1.2.2. 54 Mbps without 802.11g delays: Tenant 1 80% - Tenant 2 20%

In this case, for WDRR and ADRR, the weight of tenant 1 has been set to 80% and tenant 2 to 20%.

WDRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	%Time	%Bytes	%Tx Packets	%BW
Tenant 1	165754.96	1118846	739	54000000	0.8001	0.8001	0.5751	0.5000
Tenant 2	41415.11	279552	546	54000000	0.1999	0.1999	0.4249	0.5000

ADRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	%Time	%Bytes	%Tx Packets	%BW
Tenant 1	125830.22	849354	561	54000000	0.7999	0.7999	0.5748	0.5000
Tenant 2	31478.52	212480	415	54000000	0.2001	0.2001	0.4252	0.5000

Table 11. Results obtained for DRR and ADRR for the case of fixed 54Mbp without 802.11 g delays and 80%-20% sharing for Tenant 1 and Tenant 2.

This case has not been applied to RR, as there is no possibility to apply weights to the different tenants. For WDRR and ADRR, the weights of each tenant are shown in the percentages of time and bytes. However, the percentage of transmitted packets does not follow the relation between packet sizes as before. This percentage can be obtained as follows (example with WDRR) considering a system quantum of 1514 bytes. Equations **(38)**, **(39)** and **(40)** compute the parameters for tenant 1, while equations **(41)**, **(42)** and **(43)** for tenant 2.

$$Quantum_{tenant1} = 1514\ bytes \cdot 80\% = 1211.2\ bytes \quad (38)$$

$$number_{packets_{tenant1}} = 1211.2\ bytes \cdot \frac{1\ packet}{1514\ bytes} = 0.8\frac{packets}{iteration} \quad (39)$$

$$\%\ total\ packets_{tenant1} = \frac{0.8}{0.8 + 0.59} \cdot 100 = 57\% \quad (40)$$

$$Quantum_{tenant2} = 1514\ bytes \cdot 20\% = 302.8\ bytes \quad (41)$$

$$number_{packets_{tenant2}} = 302.8\ bytes \cdot \frac{1\ packet}{512\ bytes} = 0.59\frac{packets}{iteration} \quad (42)$$

$$\%\ total\ packets_{tenant2} = \frac{0.59}{0.8 + 0.59} \cdot 100 = 42\% \quad (43)$$

In this way, it can be justified the percentages observed in the simulation results.

4.1.2.3. 54 Mbps with 802.11g delays: Tenant 1 50% - Tenant 2 50%

As a second approach, the transmission rate has been fixed to 54Mbps and the header times in 802.11g have been considered. Moreover, the transmission error probability has been set to 0. In this case, for the WDRR and ADRR the weights of each tenant have been set to 50%.

RR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	%Time	%Bytes	%Tx Packets	%BW
Tenant 1	**221459.26**	**1059800**	**700**	**38284242.57**	**0.6533**	**0.7473**	**0.5000**	**0.6108**
Tenant 2	**117548.15**	**358400**	**700**	**24391707.10**	**0.3467**	**0.2527**	**0.5000**	**0.3892**

WDRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	%Time	%Bytes	%Tx Packets	%BW
Tenant 1	110729.63	529900	350	38284242.57	0.3894	0.5002	0.2529	0.6108
Tenant 2	173635.41	529408	1034	24391707.10	0.6106	0.4998	0.7471	0.3892

ADRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	%Time	%Bytes	%Tx Packets	%BW
Tenant 1	111995.11	535956	354	38284242.57	0.5003	0.6112	0.3471	0.6108
Tenant 2	111838.67	340992	666	24391707.10	0.4997	0.3888	0.6529	0.3892

Table 12. Results obtained for RR, DRR and ADRR for the case of fixed 54Mbp with 802.11 g delays and 50%-50% sharing for both tenants.

The headers added by 802.11g affect to the final performance, since for each packet transmission they add 92.07us. Equation (27) shows how the packet transmission time is computed and Table 2 contains the values of the 802.11g delays considered in this study. In this section, we will see how these headers impact to the different algorithms.

In the case of RR, the percentage of transmitted packets is preserved, as in each of the iterations each tenant sends one packet. The main difference we find in comparison to Table 8 is in the bandwidth and time percentages, which is caused by the addition of the headers. The following table shows the contribution of the 802.11g signaling and data to the total packet transmission time, which explains the results in terms of time percentage obtained in the simulation:

	Packet length (Bytes)	Total Packet Transmission (us)	Signalling Time (us)	Body Time (us)	% Signalling/Packet transmission	% Body / Total Packet Transmission
Tenant 1	1514	316.37	92.07	224.3	0.2910	0.709
Tenant 2	512	167.92	92.07	75.85	0.5483	0.4517

Table 13. Contributions of body and 802.11g signalling to the total packet transmission time.

In order to justify the percentage of time obtained in Table 12, equations **(44)** and **(45)** compute the relation between the total packet transmission times for each of the tenants.

$$\%Time_{Tenant\ 1} = \frac{316.37us}{316.37us + 167.92us} = 0.6533 \qquad (44)$$

$$\%Time_{Tenant\ 1} = \frac{167.92us}{316.37us + 167.92us} = 0.3467 \qquad (45)$$

Moreover, the percentage of bandwidth or effective data rate was expected to be equal for both tenants, but they are not. This occurs since in order to transmit a packet of 1514 bytes, the headers have less weight in front of the body of the message than when sending a packet of 512 bytes. In this way, the bandwidth is greater when transmitting packets of 1514 bytes than when a short 512 bytes packet is sent. This can be proven with the following equation:

$$Rb_{theoretical} * \%time_{body} = Rb_{effective} \tag{46}$$

The results obtained using the previous equation and the percentages obtained in Table 13 are written in Table 14.

	BW theoretical (bps)	% Body	BW effective (bps)
Tenant 1	54000000	0.7090	38286000
Tenant 2	54000000	0.4517	24391800

Table 14. BW effective for Tenant 1 and Tenant 2

Taking the effective BW obtained, it is possible to prove the percentage of BW in Table 12, which represents the contribution of each tenant to the total effective bandwidth to users.

$$\%BW_{Tenant\ 1} = \frac{38286000 \text{ bps}}{38286000 \text{ bps} + 24391800 \text{ bps}} = 0.3873 \tag{47}$$

$$\%BW_{Tenant\ 2} = \frac{24391800 \text{ bps}}{38286000 \text{ bps} + 24391800 \text{ bps}} = 0.6127 \tag{48}$$

Because of this, the bandwidth or effective data rate has decreased considerably when considering the headers for the three algorithms.

For WDRR, the results show that the algorithm preserves the desired weight (50% each tenant) in the number of transmitted bytes while the percentage of time is the same as the bandwidth percentage but inverted. This is because in WDRR the bytes to be sent are fixed so the additional header time will be less when transmitting larger packets.

For ADRR what is preserved is the percentage of time (50%) but the percentage of transmitted bytes and bandwidth are equally affected as in the previous algorithms. The percentage of time can be justified also by the body percentage used in RR, although applied to the number of packets to be sent in each of the iterations. For this case, the minimum required system quantum has been computed considering the 802.11g delays as in equation **(49)**, where the time to transmit the larger packet using the lowest data rate is taken into account.

$$Q_s = \frac{1514\ bytes \cdot 8\,\frac{bits}{byte}}{54\ Mbps} + 92.07us = 316.37\ \mu s \sim 320\ \mu s \tag{49}$$

Considering the computed system quantum and that both tenants share the time equally, each tenant has a quantum of 160us. The number of packets sent in each iteration can be computed as in equations **(50)** and **(51)**.

$$Packets_{Tenant\ 1}(iteration) = 160us \cdot \frac{1\ packet}{316.37\ us} = 0.5057\,\frac{\text{packets}}{\text{iteration}} \tag{50}$$

$$Packets_{Tenant\ 2}(iteration) = 160us \cdot \frac{1\ packet}{167.92\ us} = 0.9528 \frac{\text{packets}}{\text{iteration}} \quad (51)$$

From these values, it is possible to compute the percentage of packets transmitted obtained in Table 12, which represents the contribution of each tenant to the total of packets transmitted in the system.

$$\%Packets\ transmitted_{Tenant\ 1} = \frac{0.5057\ \text{packets}}{0.5057\ \text{packets} + 0.9528\ \text{packets}} = 0.347 \quad (52)$$

$$\%Packets\ transmitted_{Tenant\ 2} = \frac{0.9528\ \text{packets}}{0.5057\ \text{packets} + 0.9528\ \text{packets}} = 0.653 \quad (53)$$

It is worth to point out that even having into account the 802.11g delay times, RR adjusts well the percentage of number of packets transmitted to the initially specified weights, while WDRR adjusts the number of transmitted bytes and ADRR adjusts the time to transmit all the packets.

4.1.2.4. 54 Mbps with 802.11g delays: Tenant 1 80% - Tenant 2 20%

In this case, for WDRR and ADRR the weight of Tenant 1 has been set to 80% and Tenant 2 to 20%. 802.11g delays have been taken into account.

WDRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	%Time	%Bytes	%Tx Packets	%BW
Tenant 1	177167.41	847840	560	38284242.6	0.7187	0.8004	0.5755	0.6108
Tenant 2	69353.41	211456	413	24391707.1	0.2813	0.1996	0.4245	0.3892

ADRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	%Time	%Bytes	%Tx Packets	%BW
Tenant 1	179065.63	856924	566	38284242.6	0.8004	0.8629	0.6803	0.6108
Tenant 2	44668.30	136192	266	24391707.1	0.1996	0.1371	0.3197	0.3892

Table 15. Results obtained for DRR and ADRR for the case of fixed 54Mbp with 802.11 g delays and 80%-20% sharing for Tenants 1 and 2 respectively.

As shown before, WDRR and ADRR adjust the desired weights in terms of percentage of transmitted bytes and time, respectively. The effective BW follows the same reasoning as in the previous case.

For WDRR, the percentage of transmitted packets follows the same tendency as in the case without headers, but the relation between the times is quite different. It is shown as follows (considering that the system quantum is 1514 bytes):

$$Quantum_{Tenant\ 1} = 1514 \cdot 0.8 = 1211{,}2\ bytes \quad (54)$$

$$Time_{iteration_{Tenant1}} = 1211{,}2\ bytes \cdot \frac{1\ packet}{1514\ bytes} \cdot \frac{316.37\ \mu s}{1\ packet} = 253{,}096\mu s \quad (55)$$

$$Quantum_{Tenant\ 2} = 1514 \cdot 0.2 = 302{,}8\ \ bytes \quad (56)$$

$$Time_{iteration_{Tenant\ 2}} = 302{,}8\ bytes \cdot \frac{1\ packet}{512\ bytes} \cdot \frac{167.92\ \mu s}{1\ packet} = 99.31\ \mu s \qquad (57)$$

From the iteration times obtained through the previous expressions, it is possible to justify the contribution of each tenant to the total simulation time:

$$\%Time_{Tenant\ 1} = \frac{253.096\ \mu s}{253.96\ \mu s + 99.31\ \mu s} = 0.718 \qquad (58)$$

$$\%Time_{Tenant\ 1} = \frac{99.31\ \mu s\ \mu s}{253.96\ \mu s + 99.31\ \mu s} = 0.282 \qquad (59)$$

For ADRR, the percentage of transmitted bytes follows the same tendency as in the case without headers but, in this case, the affected parameter is the relation of transmitted packets as it can be similarly shown in the case of WDRR:

$$Packets_{iteration_{Tenant\ 1}} = 320\ \mu s\ \cdot 0.8\ \cdot \frac{1\ packet}{316.37\ \mu s} = 0.8092 \frac{packets}{iteration} \qquad (60)$$

$$Packets_{iteration_{Tenant\ 2}} = 320\ \mu s\ \cdot 0.2\ \cdot \frac{1\ packet}{167.92\ \mu s} = 0.3811 \frac{packets}{iteration} \qquad (61)$$

With these values, it is possible to compute the contribution of each tenant to the total of packets transmitted during the simulation.

$$\%Packets\ transmitted_{Tenant\ 1} = \frac{0.8092\ packets}{0.8092\ packets + 0.3811\ packets} = 0.68 \qquad (62)$$

$$\%Packets\ transmitted_{Tenant\ 2} = \frac{0.3811\ \ packets}{0.8092\ packets + 0.3811\ packets} = 0.32 \qquad (63)$$

4.1.2.5. Random Rb with 802.11g delays: Tenant 1 50% - Tenant 2 50%

As a third approach, the transmission rate is chosen randomly and the 802.11g delay times have been considered. In addition, the transmission error probability depends on the modulation scheme used (data rate chosen), which allows packet retransmissions. In this case, for WDRR and ADRR the weights of each tenant have been set to 50%. The modulation schemes used and its probabilities are the same as specified in Table 7.

It is important to point out that in this simulation we consider retransmissions. Table 16 show the results obtained through simulation.

RR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	Retx	% Time	% Bytes	% Tx packets	% BW	% Retx	Retx/ Packet Tx
Tenant 1	287148.44	1059800	700	29526191.6407	58	0.66206	0.7473	0.5000	0.6015	0.4531	0.0829
Tenant 2	146573.30	358400	700	19561544.1042	70	0.33794	0.2527	0.5000	0.3985	0.5469	0.1000

DRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	Retx	% Time	% Bytes	% Tx packets	% BW	% Retx	Retx/ Packet Tx
Tenant 1	153255.48	529900	350	27661000.8270	27	0.41303	0.5002	0.2529	0.5872	0.2500	0.0771
Tenant 2	217794.11	529408	1034	19446182.3527	81	0.58697	0.4998	0.7471	0.4128	0.7500	0.0783

ADRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	Retx	% Time	% Bytes	% Tx packets	% BW	% Retx	Retx/ Packet Tx
Tenant 1	118721.63	393640	260	26525242.3659	26	0.49958	0.5734	0.3125	0.5738	0.3171	0.1000
Tenant 2	118919.19	292864	572	19701715.8867	56	0.50042	0.4266	0.6875	0.4262	0.6829	0.0979

Table 16. Results obtained for RR, DRR and ADRR for the case of random MSC with 802.11 g delays and 50%-50% sharing for Tenants 1 and 2.

The results are similar to the ones obtained in Table 12. When comparing, it is easy to notice that there is a clear decrease of the effective bandwidth. This is because different modulation schemes are used. Moreover, it can be observed a difference in the percentages of transmitted packets that is due to retransmissions. However, in this case it can be seen again that RR adjusts the sharing specified percentages (50%-50%) for the transmitted packets (%Tx packets), WDRR adjusts the number of transmitted bytes (%Bytes) and ADRR adjusts the time (%Time).

Since in this case retransmissions are possible, it is also presented the number of retransmissions occurred during the simulation, the percentage of retransmissions of each tenant over the total of retransmissions of both tenants and the relation between the number of retransmitted packets over the total of packets transmitted. It is shown that for the three algorithms, the number of retransmissions is around the 10% of the packet transmission, which makes sense, as the most probable modulation scheme is the one operating at 54Mbps and its success probability is 90%. Moreover, the percentage of retransmissions of each tenant over the total of retransmissions during the simulation is similar to the percentage of transmitted packets, as when more packets are transmitted more retransmissions will occur.

In this case, the system quantum used for the ADRR has been set to 340us and it has been computed having into account the average data rate and the maximum packet length. The average data rate is 49.68Mbps and it has been calculated from the data in Table 7 using equation **(64)**.

$$R_{b,avg} = R_{b1} * p_{54} + R_{b2} * p_{48} + R_{b3} * p_{24} + R_{b4} * p_{12} + R_{b5} * p_6 \tag{64}$$

The system quantum has been computed with equation **(65)**, where it has been considered the data rate average in the denominator.

$$Q_s = \frac{1514\ bytes \cdot 8\,\frac{bits}{byte}}{49.68\ Mbps} + 92.25\ \mu s = 336.05\ \mu s \sim 340\ \mu s \tag{65}$$

Notice that, in this case, the delay times considered also uses the data rate average for the transmission of the ACK packet, so it is a slightly greater than the one considered in the previous section.

We can conclude that the best scheduling algorithm to share resources in time (as it happens in Wi-Fi devices) is the proposed novel algorithm ADRR.

4.1.2.6. Random Rb with 802.11g delays: Tenant 1 80% - Tenant 2 20%

In this case, for WDRR and ADRR the weight of Tenant 1 has been set to 80% and Tenant 2 to 20%. 802.11g headers are taken into account and the data rate is chosen randomly.

WDRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	Retx	% Time	% Bytes	% Tx packets	% BW	%Retx	Retx/ Packet Tx
Tenant 1	351913.30	847840	560	19273838.390	54	0.6716	0.80038	0.5755	0.6622	0.5567	0.0964
Tenant 2	172084.74	211456	413	9830319.601	43	0.3284	0.19962	0.4245	0.3378	0.4433	0.1041

ADRR	Total time (us)	Tx bytes	Packet Tx	BW (bps)	Retx	% Time	% Bytes	% Tx packets	% BW	%Retx	Retx/ Packet Tx
Tenant 1	1115995.48	2688864	1776	19275088.795	180	0.7996	0.88761	0.7276	0.6644	0.6923	0.1014
Tenant 2	279754.81	340480	665	9736525.900	80	0.2004	0.11239	0.2724	0.3356	0.3077	0.1203

Table 17. Results obtained for DRR and ADRR for the case of random MSC with 802.11 g delays and 80%-20% sharing for Tenants 1 and 2.

Once again, the results are really similar to the case with fixed MSC in Table 15, but with lower bandwidth. The results related to retransmissions follow the same tendency as in the previous case (50%-50%).

4.1.3. Deficit counter adjustment: theoretical vs real packet time.

In this study, it is discussed the difference between adjusting the deficit counter (DC) using the real time in which packets are transmitted or using a theoretical time in the ADRR algorithm.

The first approach of the ADRR algorithm for the simulator assumed that the DC was reduced using the real time in which the packet was transmitted, as in equation **(66)**. In this way, the exact time that the tenant had spent transmitting the packet was taken into account, considering the data rate in which it was transmitted and the retransmissions.

$$DC_{Tenant\ i} = DC_{Tenant\ i} - tp_{real} \qquad (66)$$

Although this solution is the fairer, when thinking of the real implementation of the hypervisor it is not a feasible option. The reason of this is that in the real hypervisor computing the real time in which a packet is transmitted involves having to read the ACK packets. This would increase the complexity of the algorithm, introducing latency to the system when serving packets. Because of this, it was decided to change the initial assumption and use the initial expected time as in equation **(67)**.

$$DC_{Tenant\ i} = DC_{Tenant\ i} - tp_{theoretical} \qquad (67)$$

Moreover, in the first approach the expected time was always computed using the most probable data rate without considering retransmissions so the expected time was always the same. In our case, the expected time was always considering the 54Mbps rate. After deciding to use the expected time, it was opted to modify the computation of the expected theoretical time, to include the possibility of different data rates in its computation. In this way, the DC reduction, in average, is nearer the first approach, since the expected packet transmission time is not always the same and has similar statistical behavior to the real packet transmission time.

Considering this, it is required to check the differences between the two approaches. The main difference between them is reflected in the convergence study, as their behavior differs. In order to compare both approaches, it has been considered the simulation conditions in Table 18. The explanantion of the field empty queue is provided in section 4.1.6.

Number of WTP	1
Tenants	2
Tenant 1 SLA	0.7
Tenant 2 SLA	0.3
Packet lengths	1514 Bytes (both)
System quantum	2ms
802.11g delays	ON
Empty queue time	100us
Traffic generation mode	Fixed
Simulation time	60 s

Table 18. Simulation conditions

The same simulations have been performed, but in this the WTP is allowed to transmit at different data rates, as specified at Table 7. Big differences can be observed when comparing the performances in terms of convergence. In Figure 28 and Figure 30, the dispersion of the time percentage used by each tenant is depicted. It can be seen how, in the case of using the tp real, the dispersion reaches smaller values than when using the tp theoretical. The same happens in Figure 29 and Figure 31 for the transmitted bytes dispersion. This is caused by the differences between the tp theoretical and tp real as different data rates are possible. When considering the real time, the algorithm is being more accurate in the time sharing between the different tenants. However, when using the theoretical time, the accuracy is decreased in both the time and transmitted bytes sharing.

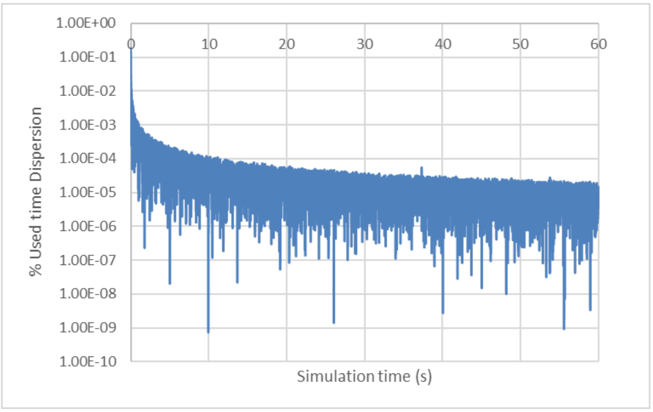


Figure 28. Used time dispersion around expected weight for the case of random data rate and tp real

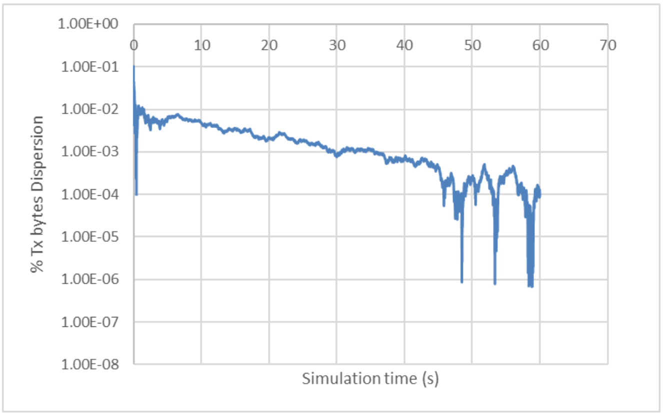


Figure 29. Transmitted bytes dispersion around expected weight for the case of random data rate and tp real

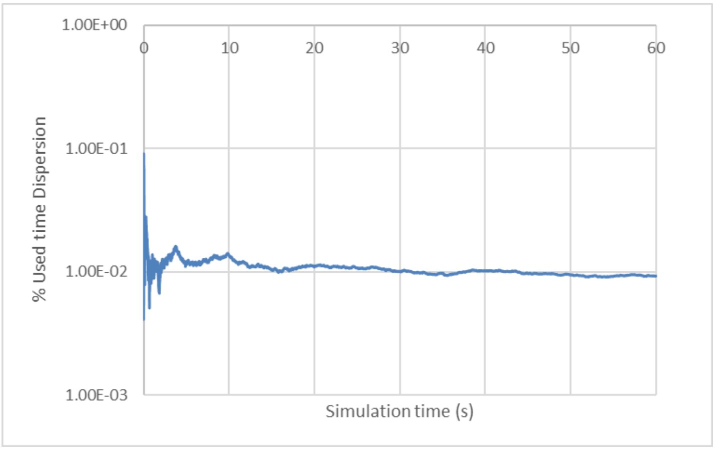


Figure 30. Used time dispersion around expected weight for the case of random data rate and tp theoretical

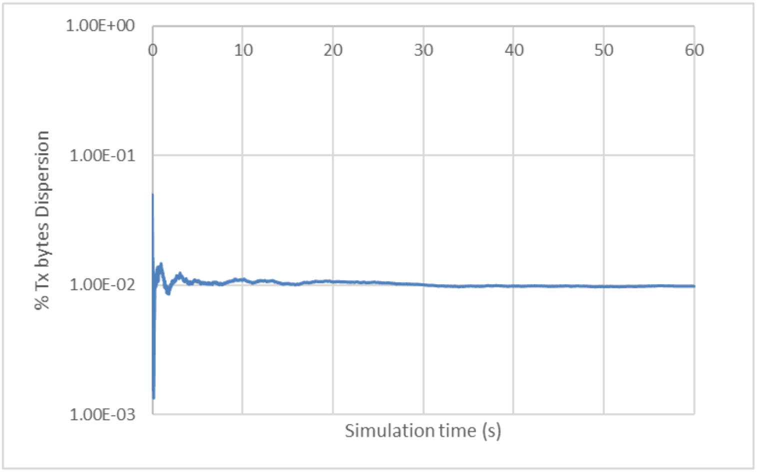


Figure 31. Transmitted bytes dispersion around expected weight for the case of random data rate and tp theoretical

In conclusion, in the case of using different data rates in a WTP, the resource sharing is more accurate and fairer when using the real transmission time in the DC adjustment. Although this, it is needed to use the expected or theoretical packet transmission time because of implementation issues for the real hypervisor so a dispersion of 1E-2 will be achieved focusing in both used time and transmitted bytes. Giving this, the time convergence has been analyzed in the following section.

A more detailed analysis about this study can be found in ANNEX 2.

4.1.4. Time Convergence Study

A key factor when studying the performance of the simulator is to determine the amount of time needed to obtain reliable results, what means to study the convergence of the system.

This convergence study focuses on the convergence time, while in the previous studies the focus was on the necessary number of iterations. The reason of this is that for the EmPOWER testbed it would be much useful to know the convergence in time units, to set a coherent simulation time. Moreover, in the study of this document, the expected theoretical packet transmission time is subtracted from the DC instead of the real time, as in equation **(67)**.

For all the tests performed, the simulation scenario considers two tenants with a SLA of the 60% for Tenant 1 and 40% for Tenant 2 and a single WTP. Both tenants just transmit packets of 1514 Bytes and the system quantum is set to 2ms. Moreover, the queues are initially empty. Different generators and transmission rates have been used in the following sections to study how the convergence is affected. It has been observed that the convergence behavior in all the studies is quite similar.

In order to see how the convergence behaves, from all the simulation performed, in this document it is included the simulation in which a fixed generator was used and the WTP can transmit using different data rates, chosen randomly according to the probabilities in Table 7. The fixed generation rate is computed as concluded in section 4.1.5.1, where it is considered an average of the different rates, as well as the delays and the related data rates probabilities.

In Figure 32, it can be seen the dispersion in terms of time-sharing when running a simulation of 60s. It can be found that the dispersion is stabilized after 20s approximately. The dispersion converges to 0.6E-3. In Figure 33 it can be observed that the transmission bytes dispersion has the same behavior as the time dispersion. This difference in comparison to the first convergence study, when tp real was used for the DC, is due to the fact that in order to compute the theoretical time, it is selected a data rate randomly according to the Rb probabilities. Nevertheless, the real data rate used can be totally different, which can introduce big differences in the number of packets sent during an iteration.

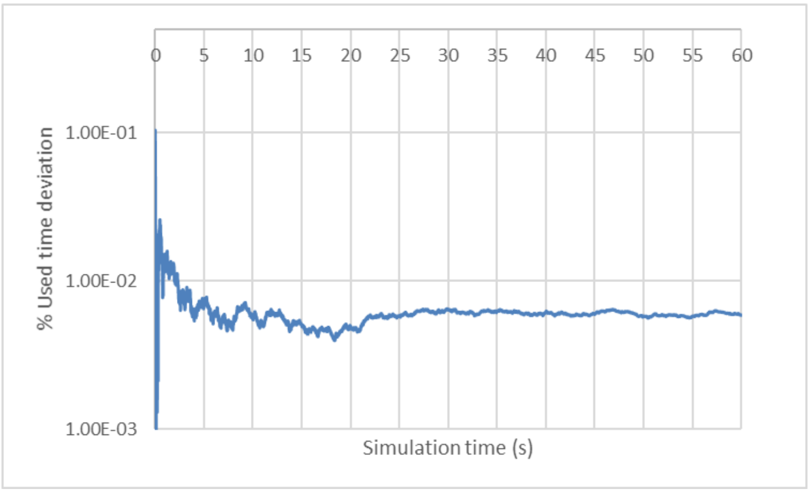


Figure 32. Time deviation with logarithmical vertical axis for the case of Rb random and fixed generation rate.

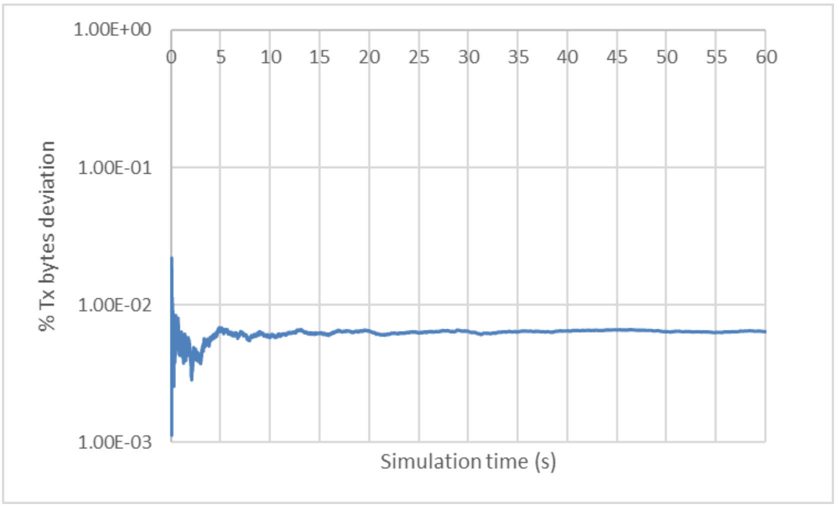


Figure 33. Transmitted bytes deviation with logarithmical vertical axis for the case of Rb random and fixed generation rate.

It has to be considered that in this simulation, the queue had no packets when the simulation started. It has been performed a simulation starting the system with packets and the algorithm converged faster, in about 5s.

In the ANNEX 2, it is possible to find more details about the different simulations performed to study the time convergence, when using a single data rate in the WTP, a Gaussian generator or starting the queue with packets.

4.1.5. Traffic Generation Analysis

In this study, a description of how the traffic is generated in the ViRANsim simulator is presented. One requirement for the simulator was to be able to control the traffic generation of the different tenants but also to simulate the real traffic expected from them. In order to do so, different traffic generators have been designed with this purpose.

All the designed simulators work using the same principle of operation, which will be explained in the first sub-section of this section. Moreover, different issues corresponding to the generators design as well as its results will be discussed. It has to be commented that this section consist in an overview of the study performed. The complete analysis can be found in ANNEX 2.

4.1.5.1. Fixed Traffic Generator Rate

The fixed traffic generator rate sets a rate at the beginning of the simulation that is not modified again during the simulation.

An important issue related to this generator is how to fix the traffic rate. The traffic generation rate of each of the tenants is related to the traffic that a given WTP can manage. This is why for each WTP its capacity (in bits) is computed . As a first approach, it was supposed that a WTP was just serving at a single Rb, so the WTP capacity in a system quantum (Q_s) has the expression in equation **(68)**:

$$Capacity_{WTP}[bits] = Rb\ [bps] \cdot Q_s[s] \qquad (68)$$

According to this, each of the tenants in a WTP could transmit the following amount of bits, considering its weight in the WTP, as it is specified in equation **(69)**:

$$Capacity_{Tenant\ i}[bits] = w_i \cdot Capacity_{WTP}[bits] \qquad (69)$$

So the traffic generation rate (G_T) for a given tenant in a WTP follows equation **(70)**, considering that we are working in a system quantum interval of time.

$$G_{T_{Tenant\ i}}[bps] = \frac{Capacity_{Tenant\ i}[bits]}{Q_s[s]} \qquad (70)$$

If expressions **(68)**, **(69)** and **(70)** are written together, the traffic generation rate (G_T) of a given tenant in a WTP could also be written as in equation **(71)**.

$$G_{T_{Tenant\ i}}[bps] = \frac{Capacity_{Tenant\ i}[bits]}{Q_s[s]} = \frac{w_i \cdot Capacity_{WTP}[bits]}{Q_s[s]} = \frac{w_i \cdot Rb\ [bps] \cdot SQ[s]}{Q_s[s]} = w_i \cdot Rb\ [bps] \qquad (71)$$

In the simulator, it is more convenient to consider the capacity of the WTP so we will work with equations **(68)**, **(69)** and **(70)** but for validation purposes it will be also useful equation **(71)**.

In order to evaluate the performance of the proposed algorithm a simulation was performed. The conditions for the simulation are specified in Table 19.

Simulation Conditions	
Rb	54Mbps
Probability of error	0.1
Generation rate	Fixed mode
802.11g delays	ON
System Quantum	2 ms
Retransmissions	ON
Simulation time	60s
Number of WTP	1
Number of tenants	2
Weight Tenant 1	0.6

Weight Tenant 2	0.4
Packet length (Both tenants)	1514 Bytes
Time empty queue	100us

Table 19. Simulation conditions for fixed generator

The results of the simulations performed can be observed in Table 20.

	Tenant 1	Tenant 2
Total Time (us)	59998936	59998936
Utilized Time (us)	36021614	23978659.48
Transmitted bytes	155106272	103403172
Number of packets generated	160499	106999
Number of packets transmitted	102448	68298
BW (Mbps)	20.68	13.78
Gt (Mbps)	32.399	21.599
Number of packets in queue	58051	38701

Table 20. Simulation results with fixed generator and single Rb

However, the effective throughput[6] (referred as BW) provided to users is smaller than the traffic generation rate. This causes that the number of packets in the queue at the end of the simulation is really high, as it can be observed in Figure 34.

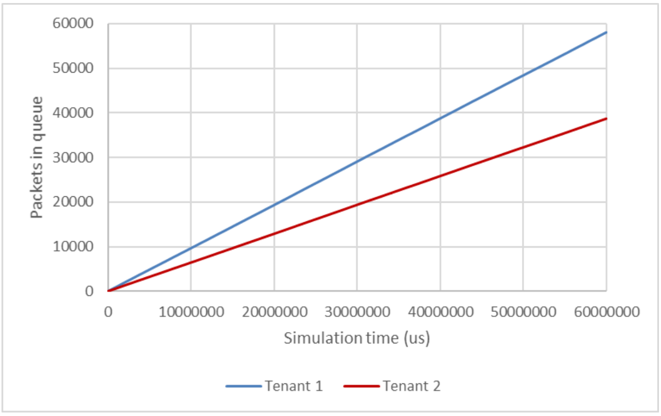


Figure 34. Packets in queue during time

One of the reasons of the growth of the queue is that the system does not transmit packets at a single nominal data rate (54Mbps in the simulation) and that 802.11g Wi-Fi protocol introduces delays that affect to the data rate. In order to face this fact it has been introduced a **delay compensation factor** considering the average packet length and all the nominal data rates. The delay compensation factor (r_i) is the relation of the delays in front of the time required to send a packet, at a certain nominal rate i. This factor will be computed for each of the tenants in each WTP as it depends on the packet length as well as on the effective data rate. Equation **(16)** shows how to compute it:

[6] *The effective bandwidth is the real data rate provided to users considering retransmissions, packet transmission delays, the total simulation time and the bytes transmitted.*

$$r_i = \frac{time_{delays_{Rb_i}}[s]}{time_{total\ packet\ transmission_{Rb_i}}[s]} \tag{72}$$

Another reason why the queue grows is that packets retransmissions are possible. In order to solve this issue, it is proposed to incorporate a **compensation factor for retransmissions** ($f_{retx,\ rbi}$) that will be different for each data rate. After doing some tests, it has been concluded that the best way to include the compensation factor is to multiply the generation rate by a factor slightly smaller than the probability to transmit a packet correctly. It has been observed that it is important to set the compensation factor slightly smaller than the probability of success, since giving that a retransmission occurs, another retransmission could occur.

The expressions used to obtain the generation rate of tenant k considering the delay and retransmission compensation factors are the following ones:

$$Capacity_{Tenant\ k_{Rb_i}}[bits] = w_k \cdot Capacity_{WTP_{Rbi}}[bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}} \tag{73}$$

$$G_{T\,Tenant\ k} = \frac{\sum_{i=0}^{N} Capacity_{Tenant\ K_{Rb_i}} \cdot p_i\ [bits]}{SQ[s]} \tag{74}$$

$$G_{T\,Tenant\ k} = \frac{\sum_{i=0}^{N} w_k \cdot Rb_i[bps] \cdot SQ[s] \cdot p_i\ [bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}}}{SQ[s]}$$
$$= w_k \cdot \sum_{i=0}^{N} Rb_i[bps] \cdot p_i\ [bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}} \tag{75}$$

$$G_{T\,Tenant\ k} = w_k \cdot Rb_{effective_{average}} \tag{76}$$

The average effective data rate can be expressed like in equation **(21)**.

$$Rb_{effective_{average}} = \sum_{i=0}^{N} Rb_i\ [bps] \cdot p_i \cdot (1 - r_i) \cdot\ f_{retx_{rbi}} \tag{77}$$

There have been performed different simulations studying which would be the convenient value for the retransmission compensation factor, setting it to 0.9, which is the probability of correct packet, to 0.89 which is slightly smaller than the probability of correct packet and to 0.8, which is a much lower value. All these simulation results can be found in ANNEX 2. It has been concluded that the best option is using a value slightly smaller than the probability packet (0.89) as it allows maximizing the traffic we are sending to the system without accumulating packets in the queue.

This is why the compensation factor has been defined as in equation **(17)**. The packets in queue for Tenant 1 are represented in Figure 35. It can be observed how the system is capable of processing the packets without accumulating them in the queue.

$$f_{retx_{rbi}} = p_{retx_{rb_i}} - 0.01 \tag{78}$$

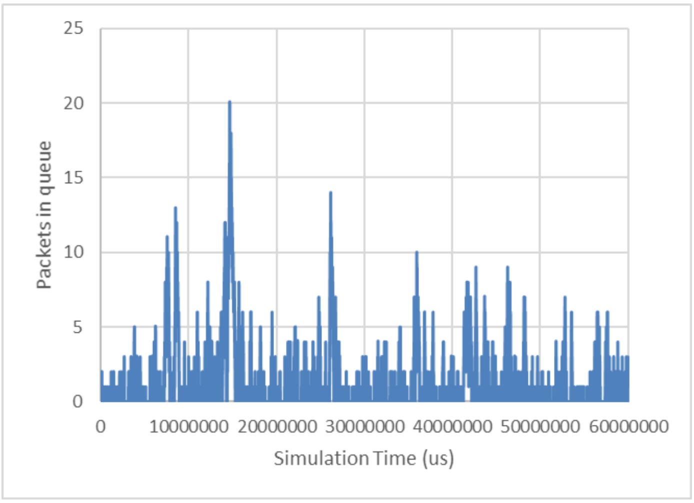


Figure 35. Packets in queue during time with retransmission and delays compensation. Retransmission compensation set to 0.89

Moreover it has been studied the performance of the system when the WTP can operate using different data rates available. However, in the simulations performed, it has been found that the queue grow when more than a single Rb value are considered in a WTP. This is because it is more difficult to control the traffic generation so it will be more likely the queue to grow. It has to be taken into consideration that in a real scenario, a tenant would not be generating traffic constantly, so this generator is just to study the performance of the algorithm as an experiment.

4.1.5.2. Uniform generator

Another requirement for the simulator was to test its performance when different traffic generation rates are taken into account and the traffic is not constant. This is why a generator that chooses the traffic rates using a uniform distribution has been designed.

This generator has been tested doing a simulation where the traffic generation rate is modified in every iteration. After this the time to generate a packet is computed and, if possible, packets are generated. The conditions of the simulation are the ones in Table 21. In this case, different modulation schemes are possible in the WTP, so different data rates are available with its success probabilities, which are the ones specified in Table 7.

Simulation Conditions	
Rb	54Mbps, 48 Mbps, 24 Mbps, 12 Mbps, 6 Mbps
Generation rate	Uniform mode
802.11g delays	ON
System Quantum	2ms
Retransmissions	ON
Simulation time	500ms
Number of WTP	1
Number of tenants	2
Weight Tenant 1	0.6
Weight Tenant 2	0.4
Packet length (Both tenants)	1514 Bytes
Time empty queue	100us

Table 21. Simulation conditions for uniform generation test

The traffic generation rates of the first 35ms of the simulation for Tenant 1 have been represented in Figure 36. It has not been represented all the simulation as it was clearer to represent less time. It has to be considered that the maximum generation rate for this tenant is computed in equation **(79)** :

$$Gt_{max} = 54\ Mbps \cdot (1 - 0.291) \cdot 0.6 = 22.97Mbps \tag{79}$$

In the simulation, the values range from 0 to 22.97Mbps, so the uniform generator is working properly.

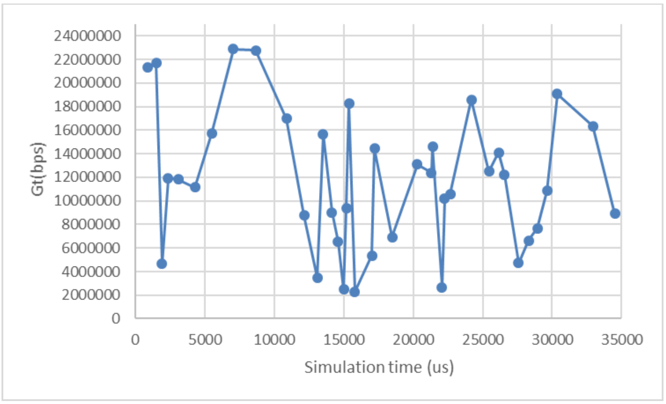


Figure 36.Traffic generation rate (Gt) using a uniform generator.

It has also been analysed the consequences of using a uniform generator in the performance of the system. The results obtained with this simulator are the ones in Table 22.

	Tenant 1	Tenant 2
Total Time (us)	499931.678	499931.678
Utilized Time (us)	292560.963	208922.444
Transmitted bytes (us)	964418	654048
Number of packets generated	637	432
Number of packets transmitted	637	432
BW (Mbps) Average	15.4328	10.4662
Gt (Mbps) Average	15.4328	10.4662
Number of packets in queue	0	0

Table 22. Results from simulator with uniform distribution

From the results, it is observed how the generation rate is smaller than the one obtained with the fixed traffic generation. If we consider the average traffic generation rates for Tenant 1 and 2, it has a middle value between the maximum generation rate and 0. It is also interesting to see the evolution of the queue with this traffic generator, which is represented in Figure 37. It is observed that the number of packets in the queue is really low and that the system is capable of managing the queue when it grows.

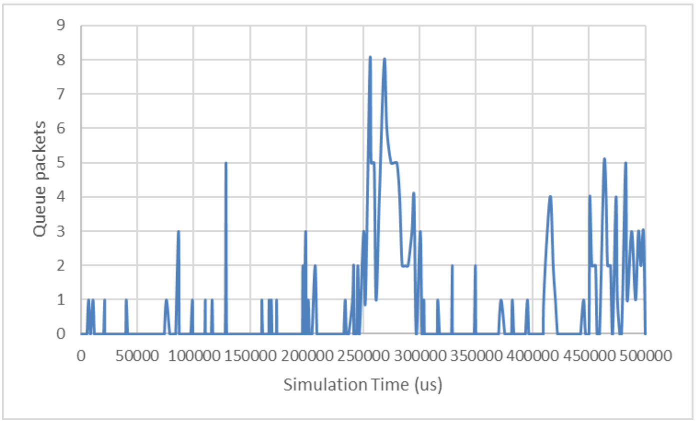


Figure 37. Packets in queue for the uniform generator simulation

4.1.5.3. Gaussian Generator

Moreover, it has been designed a Gaussian generator, as it is expected to generate traffic similarly to the reality.

After designing the Gaussian generator and implementing it, it has been tested with the conditions shown in Table 23. The nominal data rate probabilities are the ones in Table 7.

Simulation Conditions	
Rb	54Mbps, 48 Mbps, 24 Mbps, 12 Mbps, 6 Mbps
Generation rate	Gaussian mode
802.11g delays	ON
System Quantum	2 ms
Retransmissions	ON
Simulation time	500ms
Number of WTP	1
Number of tenants	2
Weight Tenant 1	0.6
Weight Tenant 2	0.4
Packet length (Both tenants)	1514 Bytes
Time empty queue	100us

Table 23. Simulation conditions for Gaussian generator

From the simulation results, it has been represented the traffic generation rates obtained by the Gaussian generator during the first 35ms for Tenant 1 (Figure 38), which has a maximum traffic generation rate of 22.97Mbps. As represented, the Gaussian generator sets traffic generation rates in its range of possible values (0, 22.97Mbps), generating more traffic generation rates around the mean.

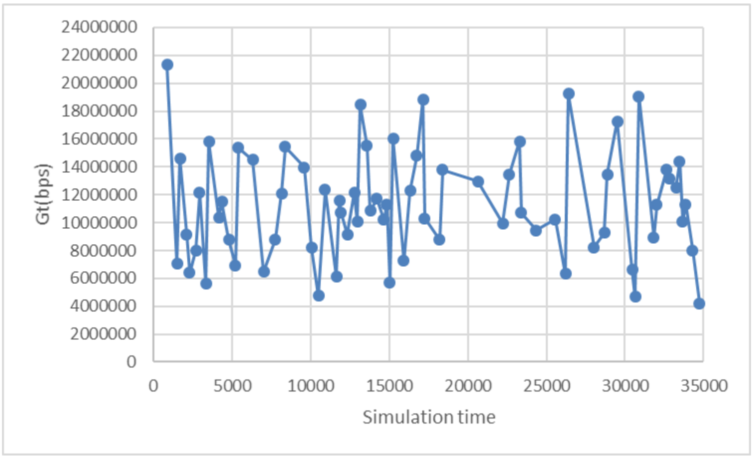


Figure 38. Traffic generation rate (Gt) using a Gaussian generator.

If we compare Figure 38 (Gaussian Generator) and Figure 36 (Uniform Generator), it can be observed that with the Gaussian Generator there are more values around the mean while with the Uniform Generator the traffic generator rates take more dispersed values.

Like in the uniform generator, it has been analysed the consequences of using a Gaussian Generator. Table 24 summarizes the obtained results when using Gaussian generator.

	Tenant 1	**Tenant 2**
Total Time (us)	498993.9	498993.9
Utilized Time (us)	285417.778	215122.741
Transmitted bytes (us)	823616	563208
Number of packets generated	544	372
Number of packets transmitted	544	372
BW (Mbps) Average	13.204	9.029
Gt (Mbps) Average	13.204	9.029
Number of packets in queue	0	0

Table 24. Results from simulator with Gaussian distribution

When we use a Gaussian generator, the average generation rate obtained is a bit lower than when a uniform generator is used (Table 22). Moreover, it has been analysed the evolution of packets in the queue for Tenant 1, which is represented in Figure 39. As fewer packets are generated, there are fewer packets in the queue and they do not accumulate.

Finally, it could be concluded that the Gaussian generator is the one that is thought to be nearer a real scenario, so problems in queue would not be a problematic issue.

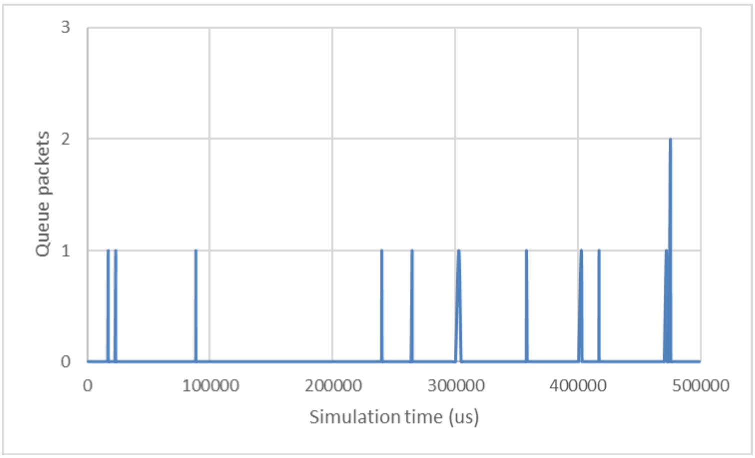


Figure 39. Packets in queue for the Gaussian generator simulation

For further information about the traffic generation, it is recommended to check ANNEX 2, where a more extended version of this study is included.

4.1.6. Study time for empty queue

In this section, it is discussed the performance of the ViRANsim simulator when the queue is empty. As a first approach, no additional time was taken into account when the queue was empty, which is not realistic because when the queue is empty the time keeps running. As the simulator tries to be as closer as possible to the reality, the management of the time when the queue is empty has been studied and designed.

The approach chosen for the empty queue issue, involves the definition of a fixed time to be added only when the queue is empty. Moreover, with this approach, instead of fixing a number of iterations per simulation, what is fixed is a time for the whole simulation because, if the fixed time for empty queue is too small, it could happen that the number of iterations is not enough to generate packets to transmit.

A key factor with this approach is to determine the fixed time value for the empty queue.. In this section, the most relevant abstractions of this study have been included but the complete study performed can be found in ANNEX 2.

To study the effect of the empty queue, three different empty queue times have been considered: 10us, 100us and 500us. Moreover it has also been taken into account a fixed generator with retransmissions compensation factor. Using the retransmission compensation factor, as showed in the previous study, the system is capable of managing the queue, in other words clearing it when it grows. The simulation conditions can be found in Table 25.

For each simulation, it has been obtained the total time of the simulation, the used time by each tenant, the number of bytes sent during the simulation, the number of packets generated and transmitted as well as the number of packets in the queue at the end of the simulation. Moreover, the effective bandwidth for each of the tenants and the time added by the empty queue have been computed.

Simulation Conditions	
Rb	54Mbps
Success probability	0.9
Generation rate	Fixed mode
802.11g delays	ON
System Quantum	2 ms
Retransmissions	ON
Simulation time	30s
Number of tenants	2
Weight Tenant 1	0.6
Weight Tenant 2	0.4
Packet length (Both tenants)	154 Bytes

Table 25. Simulation Conditions for fixed generator without retransmissions compensation factor

The results obtained are shown in Table 26, Table 27 and Table 28.

SIMULATION 10 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	29998343.9	17951836.6	76653820	50634	50630	4	34.1597673	131010
Tenant 2	29998343.9	12048721.9	51102042	33756	33753	3	33.9302659	173760

Table 26. Results with fixed generator and 10 us as empty queue time with retransmission compensation factor

SIMULATION 100 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	**29998767.1**	**17830188.1**	**76659876**	**50634**	**50634**	**0**	**34.3955433**	**21700**
Tenant 2	**29998767.1**	**12169844.4**	**51106584**	**33756**	**33756**	**0**	**33.5955545**	**258500**

Table 27. Results with fixed generator and 100 us as empty queue time with retransmission compensation factor

SIMULATION 500 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	**29998408.3**	**17756470.7**	**76656848**	**50634**	**50632**	**2**	**34.536975**	**500**
Tenant 2	**29998408.3**	**12244152.2**	**51103556**	**33756**	**33754**	**2**	**33.3896901**	**394500**

Table 28. Results with fixed generator and 500 us as empty queue time with retransmission compensation factor

When setting the empty queue time to 10 us the simulation takes a lot of time. This is because when the queue clears, a lot of iterations are required until the generator is capable of generating a new packet, as 10us is a really small value in comparison to the times to generate packets, which are computed as follows:

$$t_{packet_{T1}} = \frac{1514\ Bytes \cdot 8\frac{bits}{Byte}}{20.44\text{Mbps}} = 592.45\ us \qquad (80)$$

$$t_{packet_{T2}} = \frac{1514\,Bytes \cdot 8\,\frac{bits}{Byte}}{13.63\text{Mbps}} = 888.68\,us \tag{81}$$

In the case of setting the empty queue time to 100us, the time required to obtain the simulation results is reduced significantly in comparison to the 10us case, as the number of iterations to generate a packet when the queue is empty is much smaller. Another issue to point out is that the difference in time added by empty queue between both tenants is much greater than in the case of 10us. This is because, when the queue is empty and we add time, that time is added to the simulation time, so it affects both tenants. In this way, when tenant 2 queue is empty and 100us are added, in the following iteration, tenant 1 will be more capable of producing packets. As the generation rate of tenant 2 is lower, its queue will be empty more times, so it will add more empty time, that will make tenant 1 add more packets, which also avoids that tenant 1 empties its queue. This effect is accentuated when the time added by empty queue is of the same magnitude as the time to generate a packet.

In the third simulation, it is used 500us as empty queue time. The time required to obtain the simulations results is similar to the case of 100us. The difference in time added by empty queue between both tenants is extremely big. Tenant 1 just adds 500us in the first iteration, so the queue never empties. 500us is nearly the time that Tenant 1 needs to generate a packet (592.45 us), so when Tenant 2 is empty and adds 500us, in the following iteration, Tenant 1 nearly always will produce a packet. Considering that when a packet is correctly sent without retransmissions, the duration of the transmission is 316.37 us, adding 500us when the queue is empty is not really appropriate so a lower value would have more sense.

Concluding, using a time per empty queue of 100us would be appropriate considering the simulation results.

4.1.7. System quantum study

An important variable of ADRR is the system quantum (Q_S). In this section, a brief study about how different system quantum values affect the ADRR algorithm's performance is presented. The system quantum was initially defined in ADRR as the minimum time in order to transmit the largest packet in the system at the slower data rate. According to this, the system quantum should vary depending on the data rate and packet lengths in the system.

The simulation conditions of this section are found in Table 29. The simulations use a fixed generator with the assumptions taken in section 4.1.5.1. The possible data rates that the WTP supports are specified in Table 7.

Number of WTP	1		
Tenants SLA	Tenant 1	70%	
	Tenant 2	30%	
WTP Data rates (All WTP)	MCS RB (Mbps)	MCS Probability	MCS success probability
	54	0.8	0.9
	48	0.1	0.95
	24	0.05	0.98
	12	0.03	0.99
	6	0.02	0.999
802.11g Delays	ON		
Packet lengths	1514 Bytes for all tenants		
Simulation time	20s		
Scheduling algorithm	ADRR		
Time Empty queue	0.01ms		
Traffic generation mode	Fixed generator		

Table 29. Simulation conditions for system quantum study

The system quantum has been set to three different values. The first value is 340us, as a result of computing the packet transmission time considering the data rate average and the packet length. Secondly, it has been selected the value 2130us, result of the same computation but considering the slower data rate (6Mbps) and the packet size. Finally a larger value has been selected, 21300us, which is ten times the previous value and it has been chosen to be able to contrast the system quantum effect with larger values.

	Qs=340us		Qs=2130us		Qs=21300us	
	Tenant 1	Tenant2	Tenant 1	Tenant2	Tenant 1	Tenant2
Total time(us)	19999795.37	19999795.37	19998338.7	19998338.7	19995548.93	19995548.93
Used time (us)	14012368.44	6249758.296	13750008.3	6249758.296	13968613.78	6049588.519
Tx bytes	49577444	21350428	49015750	22122568	49616808	21742554
BW(Mbps)	19.5744	8.4297	19.6065	8.8491	19.8287	8.6891
%Used time	0.6916	0.3084	0.6875	0.3125	0.6978	0.3022
%Tx bytes	0.6990	0.3010	0.6890	0.3110	0.6953	0.3047

Table 30. Simulation results from testing different system quantum (Qs) values

In Table 30, it is shown that for the different system quantum values, the percentage of time used and the percentage transmitted bytes is really close to the weights specified for each of the tenants. Moreover, it has been computed the average bandwidth(BW) that each of the tenants perceives and no relevant differences are observed.

Although there is slightly no difference between the results obtained, the system quantum modification is relevant for the running simulation time. When a small system quantum value is used, as the simulator will process more iterations, the running simulation time is longer than when the system quantum is larger.

The criterion taken for the system quantum determination is to continue computing the system quantum as the minimum time in order to transmit the largest packet in the system at the slower data rate. The reason of this is that in the EmPOWER testbed this same assumption has been taken, so results can be compared easily. Another reason is that with this definition, it is prevented the use of large system quantum's, that in a real environment would cause a less dynamic system as the turns of each of the tenants would be slower.

4.2. Multi-WTP

The studies of this section consider more than a single WTP. As a result of this, the different studies included are related to analyse the performance of the compensation weight algorithm. In addition to running different WTPs in parallel, the purpose of these studies is to analyse the performance of the weight compensation algorithm as well as to determine which are the appropriate input values to perform the simulations, considering different scenarios: different traffic deviation, slow and fast traffic change.

For all the following simulations, the scenario chosen consists of two WTP and three Tenants with the weights shown in Table 31.

Number of WTP	2		
Tenants SLA	Tenant 1	50%	
	Tenant 2	30%	
	Tenant 3	20%	
WTP Data rates (All WTP)	MCS RB (Mbps)	MCS Probability	MCS success probability
	54	0.8	0.9
	48	0.1	0.95
	24	0.05	0.98
	12	0.03	0.99
	6	0.02	0.999
802.11g Delays	ON		
Packet lengths	1514 Bytes for all tenants		
Simulation time	180.1 s		
Scheduling algorithm	ADRR		
System Quantum	2 ms		
Time Empty queue	0.01ms		
Traffic generation mode	Gaussian Shifted		

Table 31 Simulation conditions

4.2.1. Traffic generation deviation

The first study performed consists of varying the deviation of the Gaussian shifted traffic generation. For this analysis, the traffic generation rate changes every 1-second for each of the tenants and WTP. The proportional sharing strategy is enabled. Moreover, the weights are compensated every 20 seconds.

The deviations for the Gaussian shifted generator tested are 15% and 40%. The capacity average for the WTP is 28.16Mbps, which is computed from the effective data rates of the WTP (including 802.11g delays), its probabilities and a compensation factor, as computed in equation **(66)**.

Equation (83) shows how the average time to transmit, given a certain data rate *Rb_{i,}* is computed. Notice that it is an approximation, as it just considers three retransmissions while the programmed algorithm decreases the data rate if a packet is sent incorrectly three times, which will have a low probability with the values given. Equation (2) computes a new compensation factor, needed to avoid long queues and its purpose is to compensate the effect of the difference of data rate allowed in the WTP. When a low data rate (6Mbps) is used, it processes fewer packets while new packets are arriving to the system, which results in an increase of the queue. Therefore, we use the relation of times of the maximum data rate and the minimum data rate in the system, to avoid packets accumulate in the queue.

$$C_{average} = f_{comp} \cdot \sum_{i=0}^{N-1} \frac{Packet_{length_{average}}(bytes) \cdot 8 \frac{bits}{Byte}}{tp_{average_{Rbi}}} \cdot prob(Rb_i) \qquad (82)$$

$$tp_{average_{Rb_i}} = tp(Rb_i) \cdot prob(packet_{ok}) + 2 \cdot tp(Rb_i) \cdot prob(packet_{ok}) \cdot prob(packet_{Nok}) + 3 \cdot tp(Rb_i) \cdot prob(packet_{ok}) \cdot prob(packet_{Nok})^2 \qquad (83)$$

$$f_{comp} = 1 - \left(\frac{tp(54Mbps, 1514Bytes)}{tp(6Mbps, 1514\ Bytes)}\right) = 0.85 \qquad (84)$$

The values of capacity average for each of the tenants and deviations for the case of 15% and 20% are specified in Table 32.

Tenant	Capacity average (bps)	Deviation (15%) (bps)	Deviation (40%) (bps)
Tenant 1 (50%)	14079943.8	2111991.57	5631977.521
Tenant 2 (30%)	8447966.281	1267194.942	3379186.513
Tenant 3 (20%)	5631977.521	844796.6281	2252791.008

Table 32. Capacity average and deviations for each of the tenants

The capacity average and the deviation are computed by using equations (85) and (86).

$$C_{average}(Tenant_i) = w_{SLA} \cdot Capacity_{WTP} \qquad (85)$$

$$Deviation(Tenant_i, Deviation(\%)) = C_{average}(Tenant_i) \cdot \frac{Deviation(\%)}{100} \qquad (86)$$

The traffic generated by the tenants in WTP for the deviation of 15% and 40% is represented in Figure 40 and Figure 41 respectively. More results can be found in ANNEX 2. In the figure it is also checked the performance of the Gaussian Shifted generator for tenant 1, but for tenant 2 and 3 the results would be similar but centred in their $C_{average}$. It can be validated that the traffic generator is working as expected.

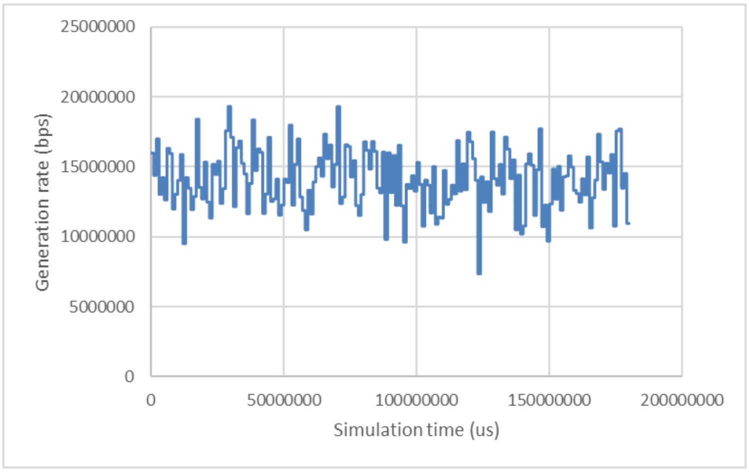


Figure 40. Traffic generation evolution for tenant 1 in WTP 1 with Gaussian deviation of 15%

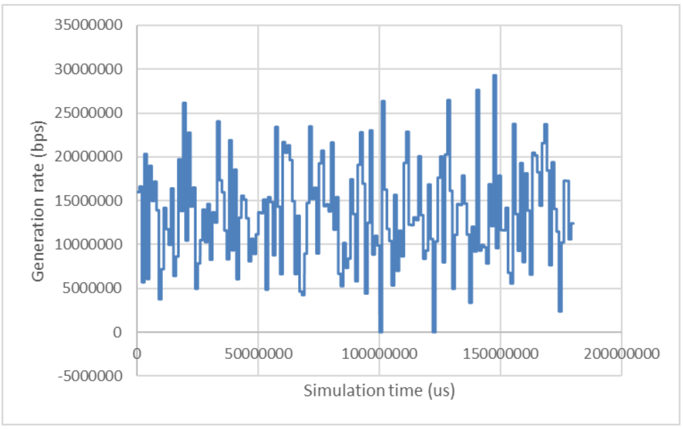


Figure 41. Traffic generation evolution for tenant 1 in WTP 1 with Gaussian deviation of 40%

After analyzing the generation rates, it has been analyzed how the weights are compensated during time. In order to do so, it has been represented the evolution of the weights during the simulation for WTP1, as it is represented in Figure 42 and Figure 43.

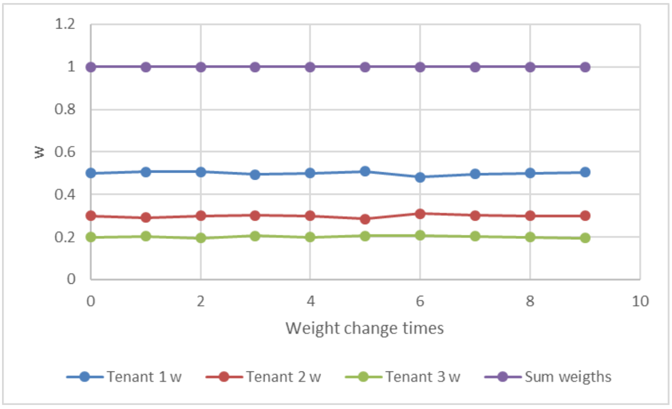


Figure 42. Weights evolution for the case of deviation of 15%

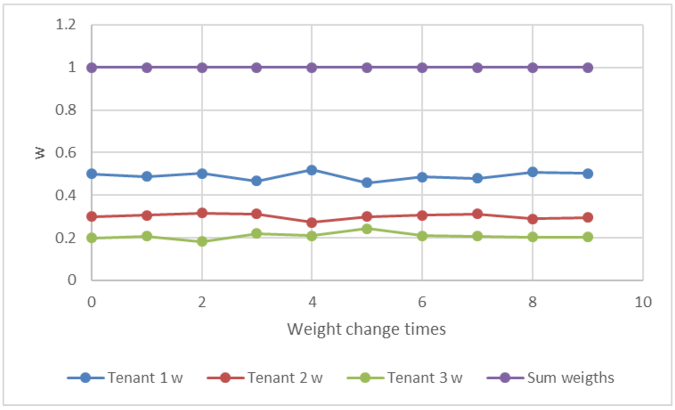


Figure 43. Weights evolution for the case of deviation of 40%

In Figure 42 and Figure 43, it is observed that when the weight of a tenant is increased, the weight of the other tenants is decreased. Moreover, it is showed how the sum of the weights of the system during all the simulations is always equal to 1, so we are assigning the total of the capacity in the WTP. In Figure 44, it can be observed the variation of the weights for tenants 1 in the case of deviation of 15% and 40%. For tenants 2 and 3 similar results are obtained. It can be stated that when a deviation of the Gaussian Generator is set to 40%, the weights change to more dispersed values.

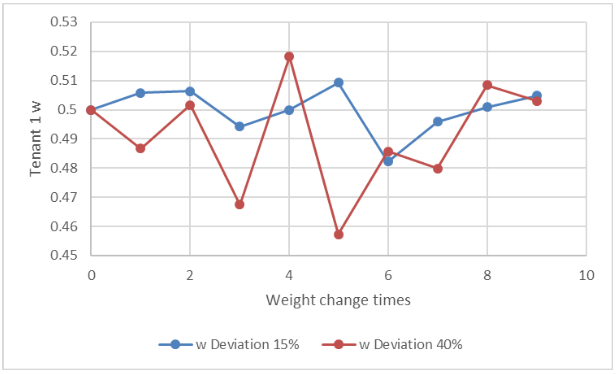


Figure 44. Comparison of the weights of tenant 1 when using a deviation of 15% and 40%

Finally, it has been analyzed if the SLA is accomplished in average for the whole network. Table 33 and Table 34 show the average of the used time and transmitted bytes in the whole network, for the case of 15% and 40% deviation. It can be observed how the SLA is accomplished for the three tenants.

Tenant	Network % Used Time	Network %Transmitted bytes
Tenant 1	0.4958	0.5011
Tenant 2	0.2979	0.2972
Tenant 3	0.2063	0.2016

Table 33. Performance of tenants over the whole network for the case of 15% of Gaussian deviation

Tenant	Network % Used Time	Network %Transmitted bytes
Tenant 1	0.4863	0.4931
Tenant 2	0.3016	0.3005
Tenant 3	0.2121	0.2064

Table 34. Performance of tenants over the whole network for the case of 40% of Gaussian deviation

4.2.2. Weight compensation period variation

The second study to evaluate the weights performance has been focused on the period of weights variation. To do so, the period in which weights are compensated has been varied. Moreover, it has also been modified the period in which the traffic generation changes, so that the relation of both period of traffic change and period of weights compensation can be analyzed.

With the purpose of visualizing variations in the weight algorithm easily, it has been chosen a Gaussian dispersion of 40% for all simulations of this section. In addition, the proportional sharing that will be analyzed later in this section is enabled. The rest of conditions for the simulations are the same shown in Table 31.

4.2.2.1. High traffic variability

Firstly, the time in which the Gaussian traffic generator changes its rate has been set to 1 second. With this period of traffic generation variation, it has been performed three simulations in which the period of weights compensation is set to 10s, 20s and 60s respectively.

It has been observed that the period of weights compensation affects the size of the queues of each of the tenants in each WTP, as the weight is applied to the tenant quantum in each WTP. For a given tenant in a WTP, if the weight is set to a lower value and then it has more traffic, the queue will grow and it would need a greater weight value. So, to evaluate the weight compensation period, it will be analyzed the evolution of the different queues.

Figure 45, Figure 46 and Figure 47 show the queue evolution for the three tenants in WTP 1. The queue evolution for the tenants in WTP 2 is similar so the same conclusions apply. Figure 48 shows how the weights vary during the simulation.

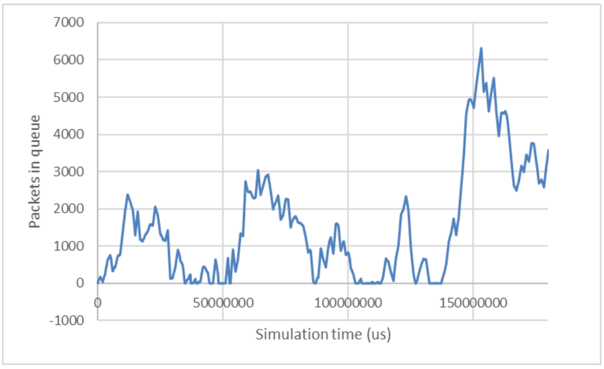


Figure 45. Queue evolution for tenant 1 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 10s.

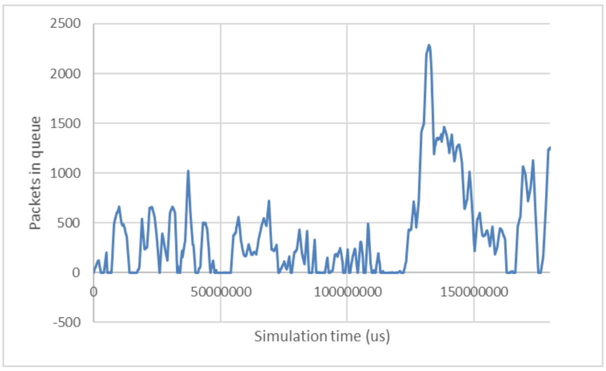


Figure 46. Queue evolution for tenant 2 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 10s.

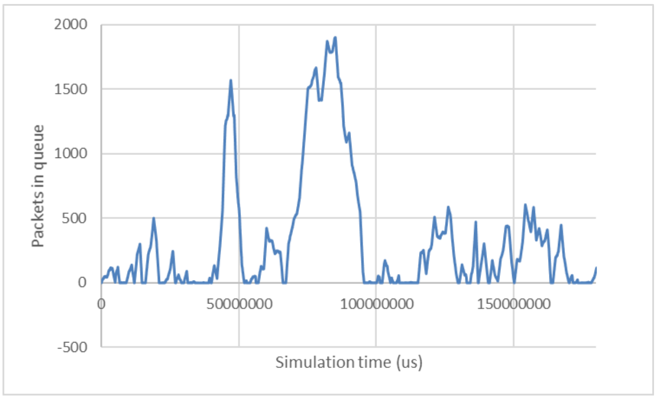


Figure 47. Queue evolution for tenant 3 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 10s.

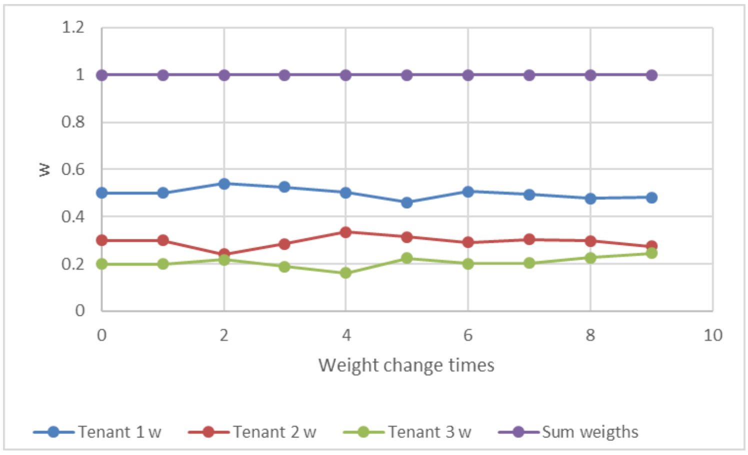


Figure 48. Weights evolution during simulation for each of the tenants. Weight compensation period to 10 seconds with traffic generation period to 1s.

The same graphics have been obtained for the case of setting the weight compensation period to 20s. Figure 49, Figure 50 and Figure 51 show the evolution of the queues during the simulation for each of the tenants in WTP 1. It can be observed how the queue lengths are smaller than in the case of weight compensation period to 10s. Moreover, Figure 52 contains the weights variation of each of the tenants for the same WTP, which is slightly lower than the variation in Figure 48.

The reason of this can be explained considering that the weight algorithm uses the average traffic generated by each tenant during the last period. So, if the weight compensation period is not the appropriate, the average traffic generated computed can be affected by punctual peaks of traffic. This can produce that the weight compensation algorithm computes weights that do not allow serving the traffic of each tenant satisfactorily.

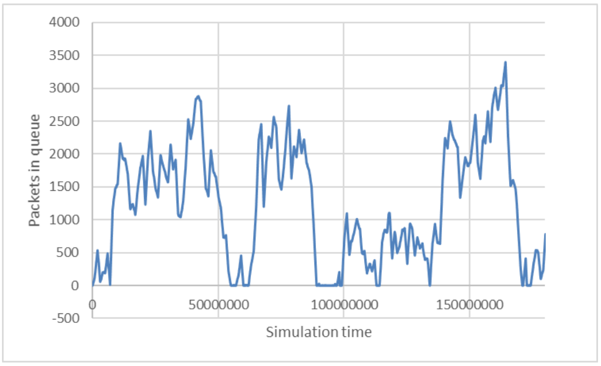


Figure 49. Queue evolution for tenant 1 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 20s.

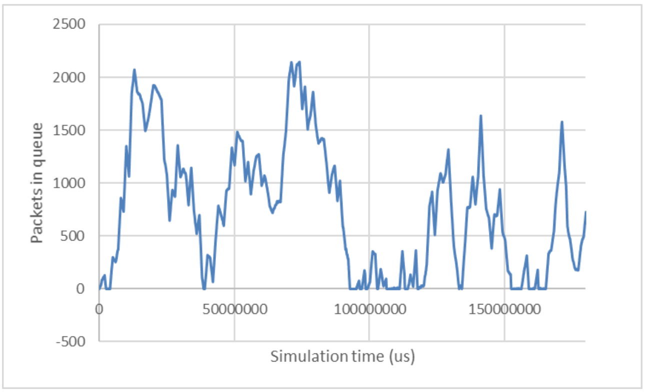


Figure 50. Queue evolution for tenant 2 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 20s.

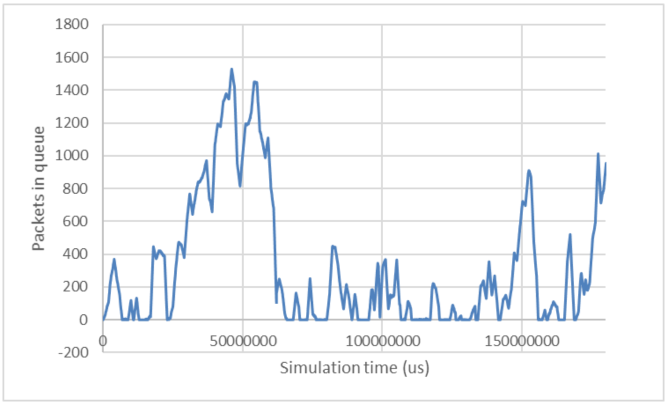


Figure 51. Queue evolution for tenant 3 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 20s.

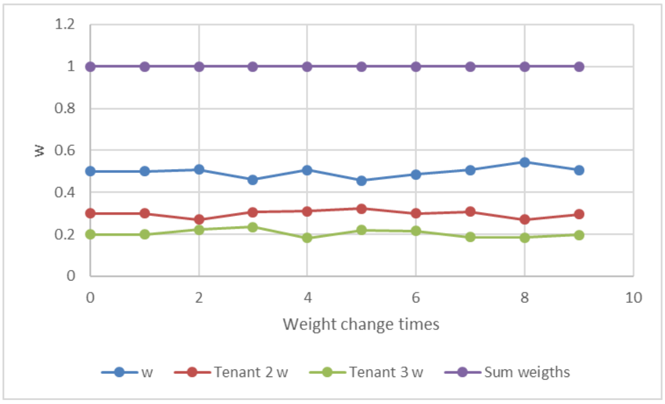


Figure 52. Weights evolution during simulation for each of the tenants. Weight compensation period to 20 seconds with traffic generation period to 1s.

The same representations have been generated for the case of setting the period of weights compensation to 60s. In Figure 53, Figure 54 and Figure 55, it is represented the queue evolution for all the tenants in WTP1. It can be observed how the queue sizes in this case have the same range of values than the case where the weights compensation period is 20s, but the queue size variation is more frequently.

In Figure 56, it is possible to observe how the weights nearly change during all the simulation. As the weights compensation period is higher, the traffic average takes more values so it will converge to the $C_{average}$, so the weights will not change much, as observed in the results.

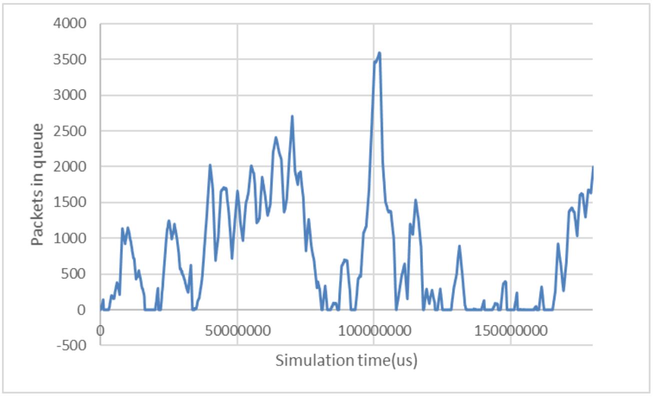


Figure 53. Queue evolution for tenant 1 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 60s.

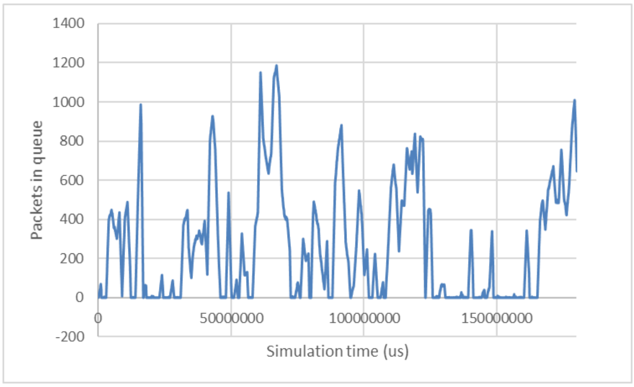


Figure 54. Queue evolution for tenant 2 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 60s.

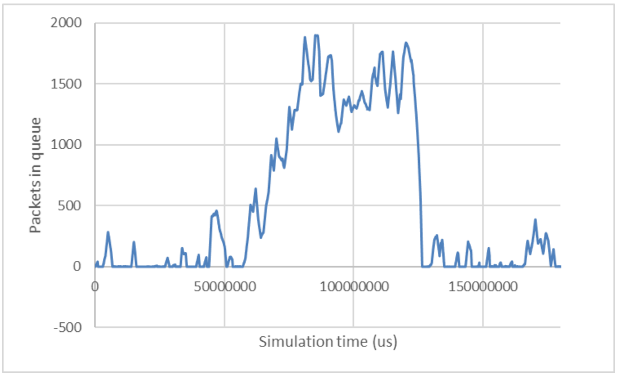


Figure 55. Queue evolution for tenant 3 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 60s.

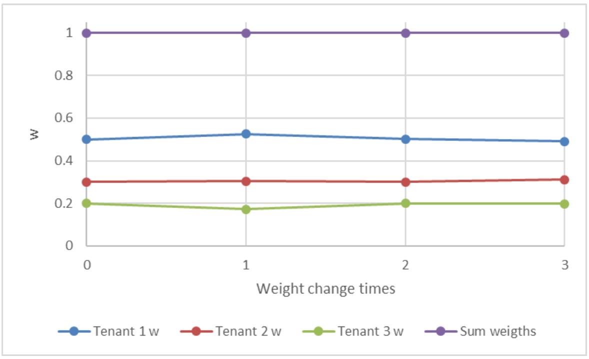


Figure 56. Weights evolution during simulation for each of the tenants. Weight compensation period to 60 seconds with traffic generation period to 1s.

4.2.2.2. Lower traffic variability

In second place, the same study has been performed when the traffic change period is set to 5s, so the traffic changes more slowly. Considering this, it will be tested the performance of the system when the weights compensation period is set to 20s and 60s. It will not be considered the case of a weights compensation period of 10s, as the traffic change period is set to 5s so the weight compensation algorithm would not be based on a reliable traffic average.

In Figure 57, Figure 58 and Figure 59, it is represented the queue evolution for the case of weight compensation period of 20s when the traffic varies every 5s. It can be observed how the queues are much larger than in Figure 49, Figure 50 and Figure 51, the homologous case with traffic variation every 1 second but without the curly trace in those graphs. This difference in the queue size is due to the relation between the weight compensation period and the traffic variation period is much lower in this case. This produces that the weight compensation values computed are more vulnerable to reach values that produce the increase of the queue, as the $C_{average}$ computed does not take into account sufficient values.

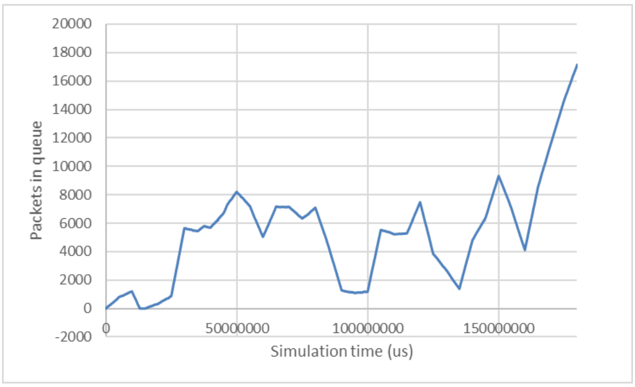


Figure 57. Queue evolution for tenant 1 in WTP 1. Case of period traffic change to 5 s and weights compensation period to 20s.

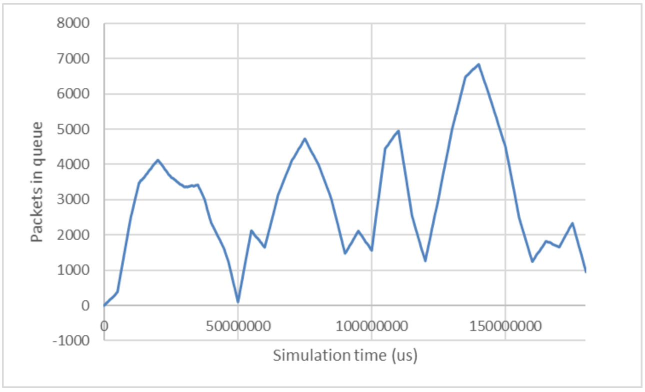


Figure 58. Queue evolution for tenant 2 in WTP 1. Case of period traffic change to 5 s and weights compensation period to 20s.

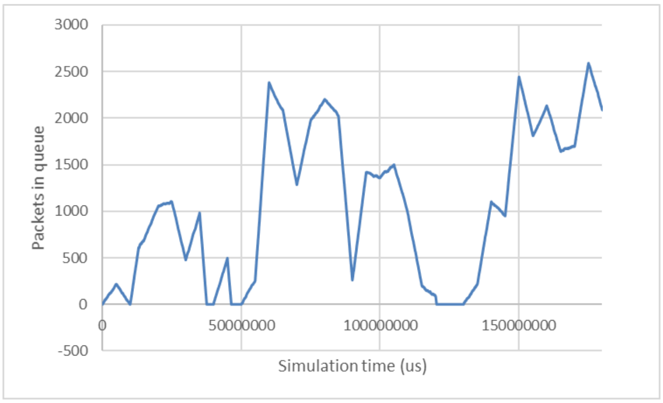


Figure 59. Queue evolution for tenant 3 in WTP 1. Case of period traffic change to 5 s and weights compensation period to 20s.

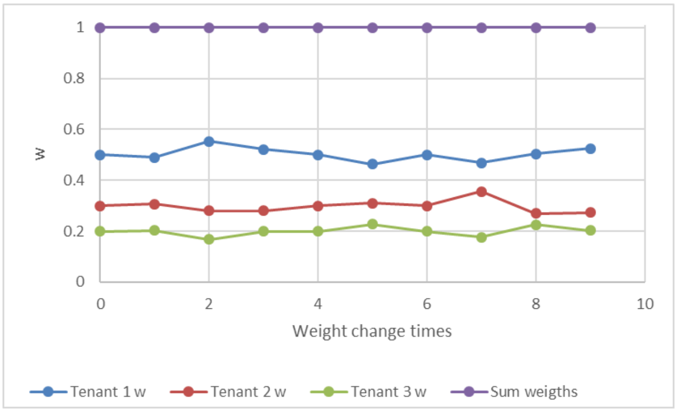


Figure 60. Weights evolution during simulation for each of the tenants. Weight compensation period to20 seconds with traffic generation period to 5s.

The same graphics have been created but setting the weight compensation period to 60 seconds. In Figure 61, Figure 62 and Figure 63 the queue evolution for each of the tenants is plotted and in Figure 64 the variation of the weights for each of the tenants is represented in WTP1. It can be observed how the queue lengths are slightly smaller but still large. This means that having a window of 60s is still not enough to compute a reliable $C_{average}$.

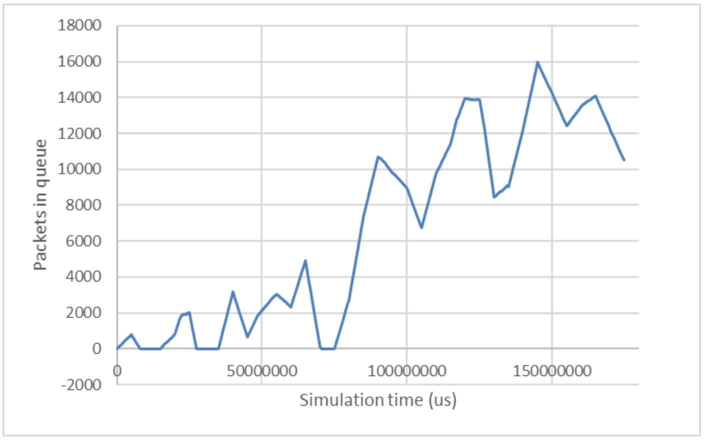


Figure 61. Queue evolution for tenant 1 in WTP 1. Case of period traffic change to 5 s and weights compensation period to 60s.

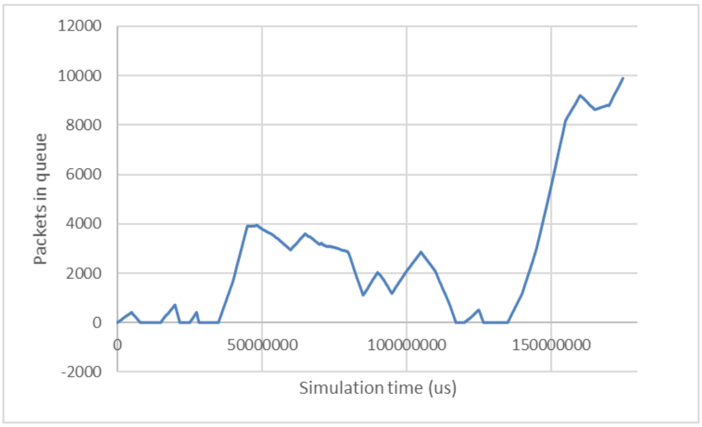


Figure 62. Queue evolution for tenant 2 in WTP 1. Case of period traffic change to5 s and weights compensation period to 60s.

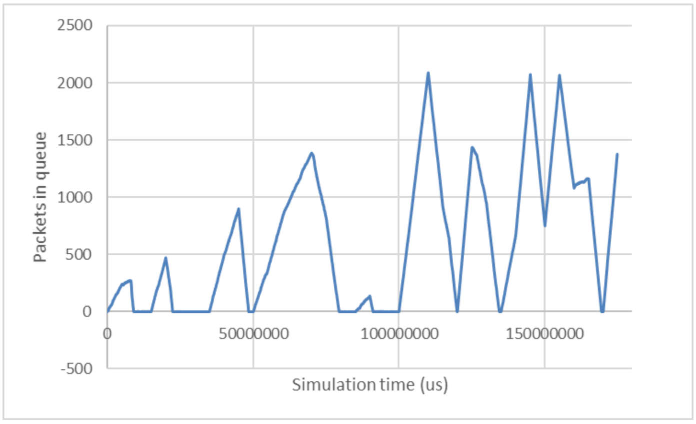


Figure 63. Queue evolution for tenant 3 in WTP 1. Case of period traffic change to 5 s and weights compensation period to 60s.

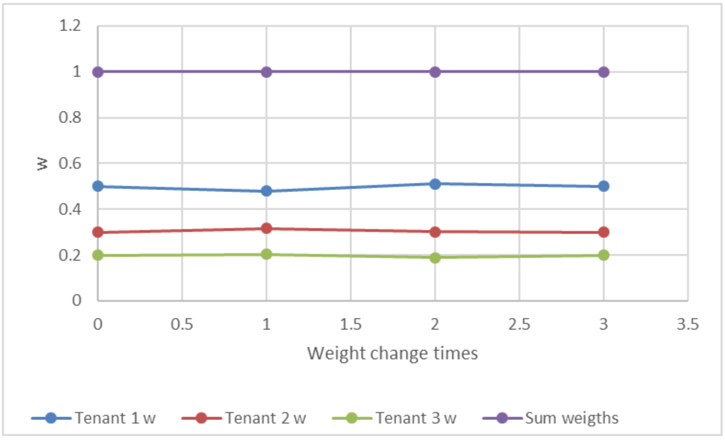


Figure 64. Weights evolution during simulation for each of the tenants. Weight compensation period to 60 seconds with traffic generation period to 5s.

Considering the different simulations performed, studying the effect of the weights compensation period in relation to the traffic generation period, it can be stated that they are two parameters that are strongly related. Considering the obtained results, it is recommendable to use a weight compensation period at least 20 times the traffic generation period. That means that for the second case studied, when the traffic generation period is set to 5s, the weights compensation time should have been set to 100s.

Moreover, another issue to be considered is that if the weight compensation period is too large, the weight compensation will not compensate periods of high traffic level, so a balance has to be found. It is needed to study which are the traffic requirements for our system and how relevant is to compensate high peaks of traffic.

4.2.3. Proportional sharing

Another study developed is the evaluation of the performance of the proportional sharing algorithm. The weight compensation algorithm was initially defined in the way that if during the previous period, the different tenants were not requesting the 100% of the WTP capacity, not all the resources were assigned. That means that the sum of the weights assigned to the different tenants in a WTP could be less than 1.

Regarding this issue, it was decided to add a mechanism that allow the proportional sharing of the non-assigned resources between the different tenants, having into account its SLA. In this section, some of the results comparing the usage and non-usage of this mechanism are provided. An extended analysis including more results can be found in ANNEX 2.

The simulation conditions used consist of a weight compensation period of 20s, a traffic generator with a Gaussian dispersion of 40% with a variation period of 1s. The simulations have been run with proportional sharing and without proportional sharing.

Figure 65 shows the weight evolution and the sum of weights when proportional sharing is enabled while in Figure 66 the proportional sharing is disabled. It can be observed that in the first case the addition of all the weights is always equal to 1 while in the second case sometimes the addition of weights is lower than 1. In both cases the sum of weights never exceeds 1.

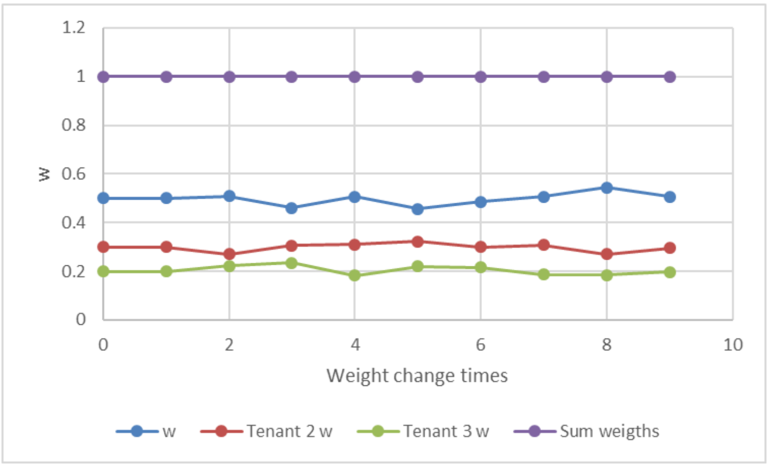


Figure 65. Weight evolution and sum of weights with proportional sharing

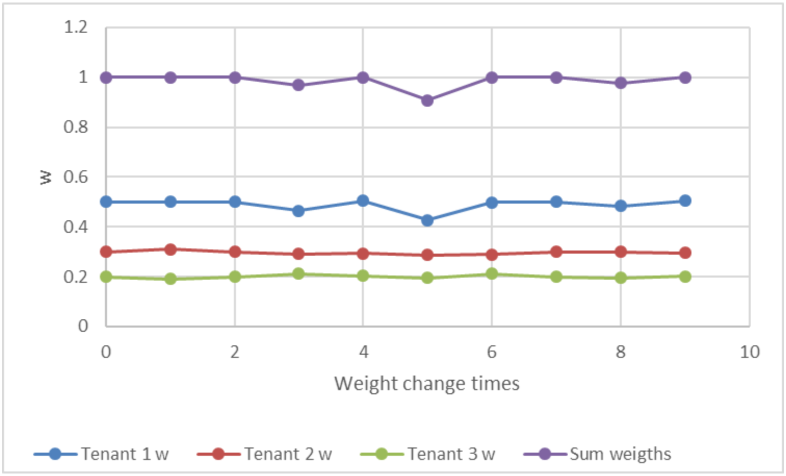


Figure 66. Weight evolution and sum of weights without proportional sharing

Figure 67, Figure 68 and Figure 69 show the queue evolution for each of the tenants without proportional sharing. If those figures are compared to Figure 49, Figure 50 and Figure 51 it can be observed how the queue lengths are much higher. This is caused by the fact that not all the available resources are considered.

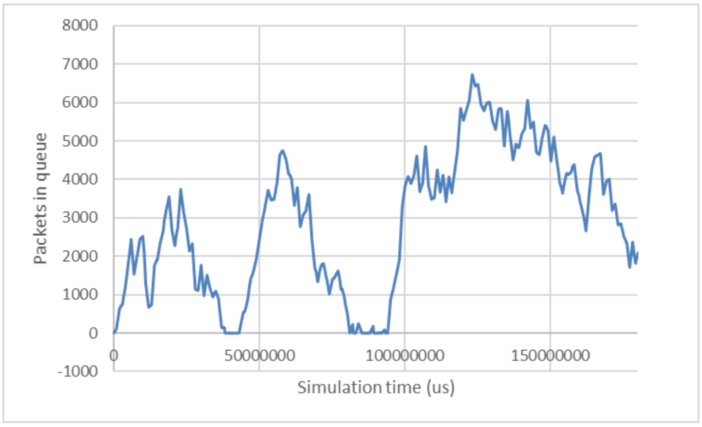


Figure 67. Queue evolution for tenant 1 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 20s without proportional sharing.

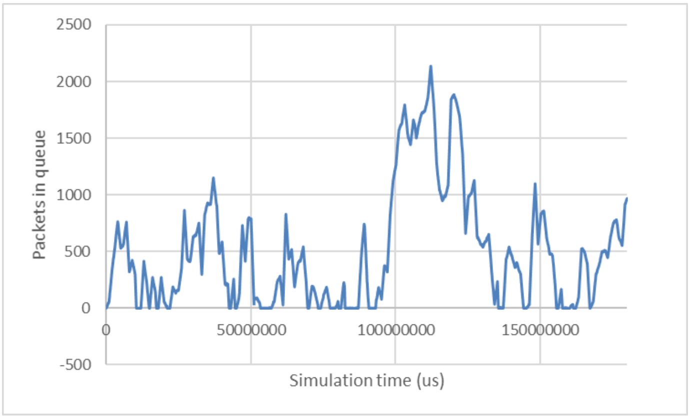


Figure 68. Queue evolution for tenant 2 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 20s without proportional sharing.

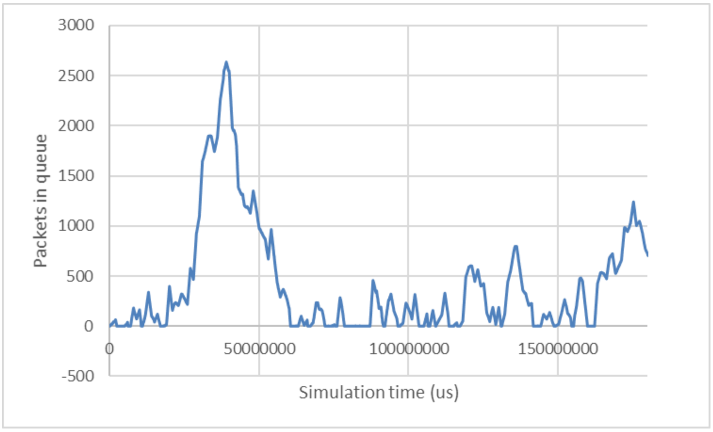


Figure 69. Queue evolution for tenant 3 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 20s without proportional sharing.

In a scenario where the capacity requested is low during a period, it could happen that the computed weights without considering proportional sharing are smaller. So, if during a period of low activity, a peak of traffic arrives, the fact that no proportional sharing is used could result in a bad management of the resources: larger queues, latency… Proportional sharing can provide a softer transition between periods of high and low traffic levels and a reduction of the queue lengths.

4.2.4. Benefits of using the weights algorithm

The last study performed has the purpose of demonstrating that the weights algorithm improves the quality of the tenant's service. To easily prove so, simpler simulation conditions have been considered.

Number of WTP	1		
Tenants SLA	Tenant 1	70%	
	Tenant 2	30%	
WTP Data rates (All WTP)	MCS RB (Mbps)	MCS Probability	MCS success probability
	54	0.8	0.9
	48	0.1	0.95
	24	0.05	0.98
	12	0.03	0.99
	6	0.02	0.999
802.11g Delays	ON		
Packet lengths	1514 Bytes for both tenants		
Simulation time	60 s		
Scheduling algorithm	ADRR		
System Quantum	2 ms		
Time Empty queue	0.01ms		
Traffic generator	Fixed traffic with pulse of 30s		
Time change weights	10s		
Proportional sharing	Enabled		

Table 35. Simulation Conditions pulse mode

As it can be observed in Table 35, to generate the traffic of each of the tenants it has been used a traffic generator that changes the traffic generator rate of both in the middle of the simulation. This way, it is possible to manually set the traffic generation rate of both tenants to force certain conditions to easily observe the effect of the weight algorithm.

For this simulation, it has been forced that in mean both tenants generate traffic according to its SLA, as it is shown in Table 36. Figure 70 show the traffic generation rates obtained from simulation results.

	Period 1 (0s - 30s)	Period 2 (30s - 60s)	Mean Traffic Rate
Traffic Generated Tenant 1	27.59 Mbps	11.83 Mbps	19.712 Mbps
Traffic Generated Tenant 2	3.38 Mbps	13.51 Mbps	8.45 Mbps

Table 36. Traffic generation rates for each tenant and period

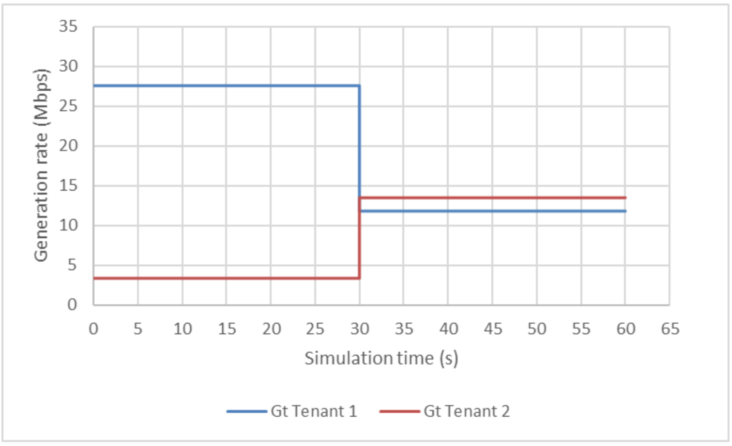


Figure 70. Traffic generation rates during simulation time

Considering the traffic generation rates of each tenant, two simulations have been run: one using the weight algorithm and the other without using it. The results of both simulations are represented in Table 37 and Table 38, where no important differences are found.

WTP	Tenant	Used Time (us)	Transmitted bytes	Packet transmitted	Packets generated	%Used Time	%Transmitted bytes
WTP 1	Tenant 1	41865457.56	147836044	97646	97646	0.6978	0.7000
	Tenant 2	18131896.44	63362414	41851	41851	0.3022	0.3000
	TOTAL	**59997354**	**211198458**	**139497**	**139497**	**1.0000**	**1.0000**

Table 37. Performance results when enabling the weight algorithm

WTP	Tenant	Used Time (us)	Transmitted bytes	Packet transmitted	Packets generated	%Used Time	%Transmitted bytes
WTP 1	Tenant 1	41563227.41	147837558	97647	97647	0.6928	0.7000
	Tenant 2	18433511.56	63362414	41851	41851	0.3072	0.3000
	TOTAL	**59996738.96**	**211199972**	**139498**	**139498**	**1.0000**	**1.0000**

Table 38. Performance results when disabling the weight algorithm

However, when looking at the queues evolution of each of the tenants when using and non-using the weight algorithm, differences are found.

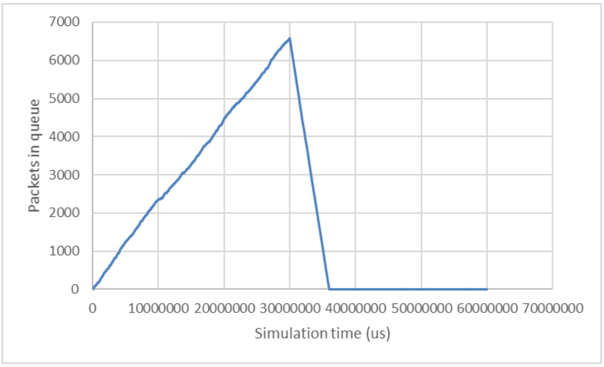


Figure 71. Queue evolution during simulation time when enabling the weight algorithm every 10s for tenant 1.

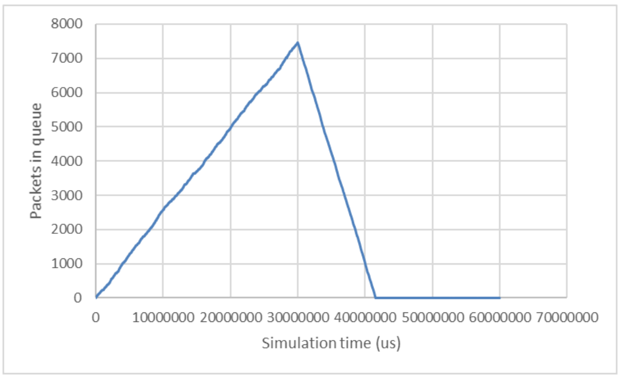


Figure 72. Queue evolution during simulation time when disabling the weight algorithm every 10s for tenant 1.

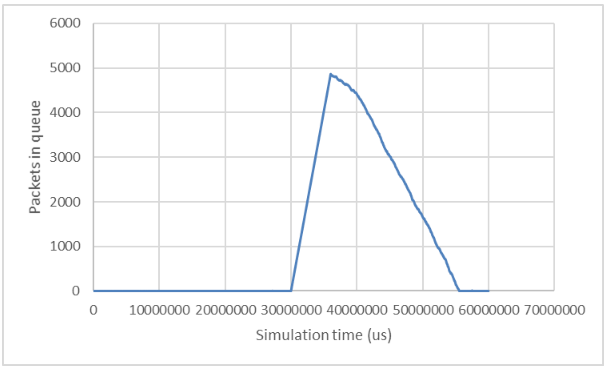


Figure 73. Queue evolution during simulation time when enabling the weight algorithm every 10s for tenant 2.

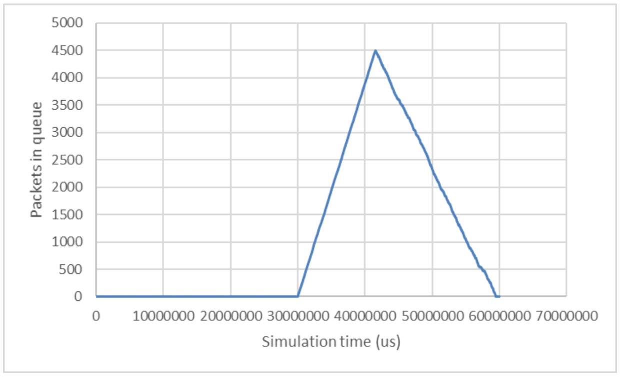


Figure 74. Queue evolution during simulation time when disabling the weight algorithm every 10s for tenant 2.

Looking at the graphics, it can be seen how in Figure 71 and in Figure 73, where the weight algorithm is enabled, the size of the queues are smaller and decrease faster than in Figure 72 and Figure 74. This proves that the weight compensation algorithm is useful to improve the performance of the system, as the packets will be processed faster.

In Figure 75 it can be observed that, in the first 30s when tenant 1 has a higher traffic level, the weight of tenant 1 is incremented while the weight of tenant 2, with lower traffic lever, is lowered. In the last 30s, the weights of both tenants are set approximately to 50%-50% as the traffic requirements are similar, as it can be observed in Figure 70.

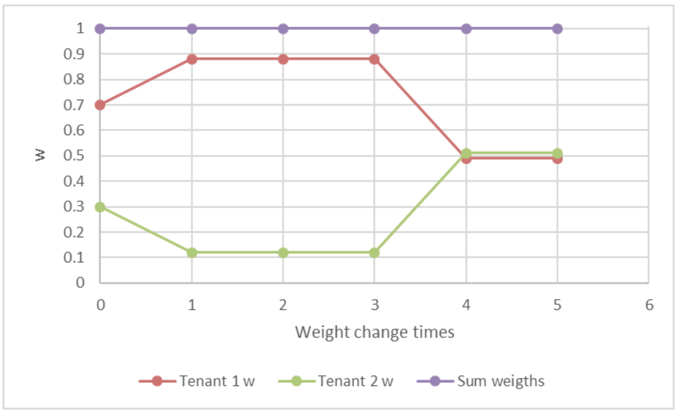


Figure 75. Weights evolution during simulation

5. Cost assessment

In this chapter it is included an estimation of the costs of the project. The different costs have been split into labour cost and development tools cost.

The labour costs have been computed considering a Junior Engineer receiving 10,30 €/hour. To this, has to be added a 30% of fees and social insurance, so the hour remuneration is 13,30€

Project stage	Hours	Cost (€)
Formation	120 h	1596 €
Development and analysis of single WTP solution	360 h	4788 €
Development and analysis of Multi-WTP solution	240 h	3192 €
Thesis Documentation	160 h	2128 €
Total	**880 h**	**11704 €**

Table 39 Labour costs of the project

The tools used during the project consist in the items presented in Table 40. For all the items, it has been considered a certain amortization time and that the project has lasted 6 months. The editor used of the simulator programming is Sublime Text, which is free.

Concept	Price	Amortization time (months)	Project cost (6 months)
Development PC	900€	36 months	150 €
Microsoft Windows 10 Pro Licence	279€	60 months	27.90 €
Microsoft Office 2016 Licence	149€	36 months	24.80 €
Total			**202.70€**

Table 40. Development tools costs

By considering both costs it can be obtained the total project cost.

Concept	Cost

Labour	11704 €
Development tools	202.70€
Total	**11906.7 €**

6. Conclusions and future development

In this Thesis, it has been designed, implemented and validated a Python hypervisor and controller simulator, called ViRANsim, which is based on the 5G-EmPOWER platform. The virtualized RAN slice simulator (ViRANsim) includes two novel algorithms: the ADRR, a time-based scheduling algorithm, and the weight compensation algorithm, which allows maximizing the use of the WTP resources while assuring the SLA in a long-term perspective. Thanks to the simulator, it has been possible to obtain a first theoretical analysis of the performance of these algorithms before implementing them in the real 5G-Empower testbed.

An important part of this project has been the development of the ViRANsim simulator that includes the hypervisor and the controller simulator. Python is a programming language that has ease the implementation of the simulator, making it simple to create classes and develop the different functions in a modular style. A key aspect of Python is the existence of a large number of libraries, which has been extremely useful and time saving for the project. In addition, the possibility of exporting files has facilitated the analysis of the results in a practical manner. However, in the development part, many problems have been faced related to the time management and synchronization, which has been finally solved by running the different WTPs the same amount of time sequentially, which give the sense of running them in parallel.

Thanks to the simulator, it has been possible to study the behaviour of the ADRR and the weight compensation algorithms. For the testing and validation of both algorithms, it has been fundamental to control the traffic generation, in order to correctly evaluate their performance in different case studies. Moreover, computing the capacity that a WTP can support has been challenging since 802.11g delays and packet retransmissions have introduced additional complexity to the computation. Nevertheless, the consideration of those parameters has helped to evaluate the algorithms in much more realistic conditions

In the case of the ADRR, different abstractions have been obtained from the different studies performed. An important consideration for the ADRR is the use of the computed theoretical packet transmission time for the Deficit Counter (*DC*) adjustment, which implies a reduction of the precision of the resource sharing percentage with respect to the case of using the real packet transmission time. However, as shown in the algorithm time convergence, in less than 20s is possible to obtain values around the desired weight with a very small dispersion (1E-2). Another important conclusion taken from the ADRR study is that the system quantum value (Q_s) in a simulation does not affect the final performance of the algorithm in terms of percentage of used time and bytes or bandwidth. However, in a real implementation, the value of the system quantum should be carefully chosen, as a large value could affect the dynamism of the algorithm. Overall, it can be stated that the ADRR algorithm is capable of sharing the resources by using them successfully.

For the weight compensation algorithm, it has been proved that it allows improving the system performance, as the WTP resources assigned to the users are maximized and the system is able to adjust the weights depending on the traffic demand. In order to reach this resource assignment maximization, the need to define a proportional sharing mechanism that modifies the weights of each of the tenants in a WTP assigning all the resources in the WTP has arisen. For this algorithm, it has been obtained that it is important to control the traffic variability and define the time in which the weights compensate it, as the algorithm will be more or less reactive depending on this. The weight compensation period is recommended to be set 20 times the time traffic variation period.

From the simulations, it has been possible to extract relevant information for the real implementation of the hypervisor and weight compensation algorithm in the EmPOWER platform. Part of the simulator information and results have been included in a paper submitted to the IEEE Conference on Network Functions Virtualization and Software Defined Networking (IEEE NFV-SDN 2017) [16].

In conclusion, thanks to the ViRANsim simulator, it has been possible to analyse both algorithms and verify their correct execution, providing relevant abstractions to its real implementation in the 5G-EmPOWER. Both algorithms operate as expected, making possible a fair weighted sharing of the resources in the WTP while exploiting the resources of the network, without wasting them.

6.1. Future development

This thesis offers the possibility of continuing using the simulator to study more aspects related to the RAN slicing. In the following, different proposals to keep extending the simulator are given:

- Create a User GUI to easily configure the simulation scenario before running simulations.
- Upgrade the simulator to enable different traffic modes in different tenants in a WTP. Study the consequences of different traffic modes in this case.
- Include new traffic shapes in the simulator to study the different algorithms in more situations: pulse, ramp…
- Incorporate an algorithm that controls that the tenants do not generate more traffic than agreed in their SLA. Current model assumes tenants generate the agreed traffic in average. The system could advise the tenant to request a lower traffic level.
- Include the concept of Active List (tenants with packets in its queue) in the hypervisor simulator. This concept is being implemented in the 5G-EmPOWER hypervisor. It would be required to study its consequences in the system performance.
- Propose and include alternative compensation algorithms with different criteria than proportional sharing.
- Extend the simulator to support LTE RAN slicing.

Bibliography

[1] A. Bradai, K. Singh, T. Ahmed, T. Rasheed. " Cellular Software Defined Network-a Framework". *IEEE Communications Magazine.* vol. 53, pg. 36-43, June 2015

[2] B-Astuto, M. Mendonca, X. Nguyen, K. Obrackzka, T. Turletti. "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks". *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, Third quarter 2014.

[3] R.Riggio, T. Rasheed, F. Granelli. "EmPOWER: A Testbed for Network Function Virtualization Research and Experimentation". *IEEE SDN for Future Networks and Services,* November 2013

[4] D. Kreutz, F. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig. " Software-Defined Networking: A Comprehensive Survey". *Proceedings of IEEE,* vol. 103, pg 14-76, Jan. 2015.

[5] R.Riggio, K. Mabell, T. Rasheed, J. Schulz-Zander, S. Kuklinski, M. K. Marina, "Programming Software-Defined Wireless Netorks". *10th International Conference on Network and Service Management (CNSM) and Workshop*. 17-21 Nov. 2014.

[6] R.Riggio, A. Bradai, T. Rasheed, D. Harutyunyan, T. Ahmed "Scheduling Wireless Virtual Networks Functions". *IEEE Transaction on Network and Service Management,* vol. 13, no. 2, June 2016.

[7] H. Hawilo, A. Shami, M. Mirahmadi, R. Asal "NFV: State of Art, Challenges, and Implementation in Next Generation Mobile Networks (vEPC)". *IEEE Network,* November/December 2014.

[8] "Deficit Round Robin". [Online] Available: http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/11-Fairness/DRR.html [Accessed: 13 July 2017].

[9] "Packet transmission time in 802.11" [Online] Available: https://sarwiki.informatik.hu-berlin.de/Packet_transmission_time_in_802.11 [Accessed 24 March 2017]

[10] "5G-EmPOWER Wiki" [Online] Available: https://github.com/5g-empower/5g-empower.github.io/wiki/Overview [Accessed 4 July 2017]

[11] "The Click Modular Router Project"[Online] Available: http://read.cs.ucla.edu/click/click [Accessed 1 February 2017]

[12] "OpenFlow" [Online] Available: https://www.opennetworking.org/en/sdn-resources/openflow [Accessed 16 July 2017]

[13] K.K. Yap, M. Kobayashi, R. Sherwood, T.Y. Huang, M. Chan, N. Handigol, and N. McKeown. "Openroads: Empowering research in mobile networks." *ACM SIGCOMM Computer Commun. Review*, 40(1):125–126, 2010.

[14] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao. "Towards programmable enterprise wlans with odin". In *Proc. 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 115–120, New York, NY, USA, 2012. ACM.

[15] M. Bansal, J. Mehlman, S. Katti, and P. Levis. "Openradio: a pro- grammable wireless dataplane." *In Proc. 1st workshop on Hot topics in software defined networks*, pages 109–114. ACM, 2012.

[16] K. Koutlia, A. Umbert, R. Riggio, I. Vilà and F. Casadevall. “RAN slicing for multi-tenancy support in WLAN scenario”. *IEEE NFV-SDN* [Submitted for revision].

[17] “Minstrel” [Online] Available: https://wireless.wiki.kernel.org/en/developers/documentation/mac80211/ratecontrol/minstrel [Accessed 16 July 2017]

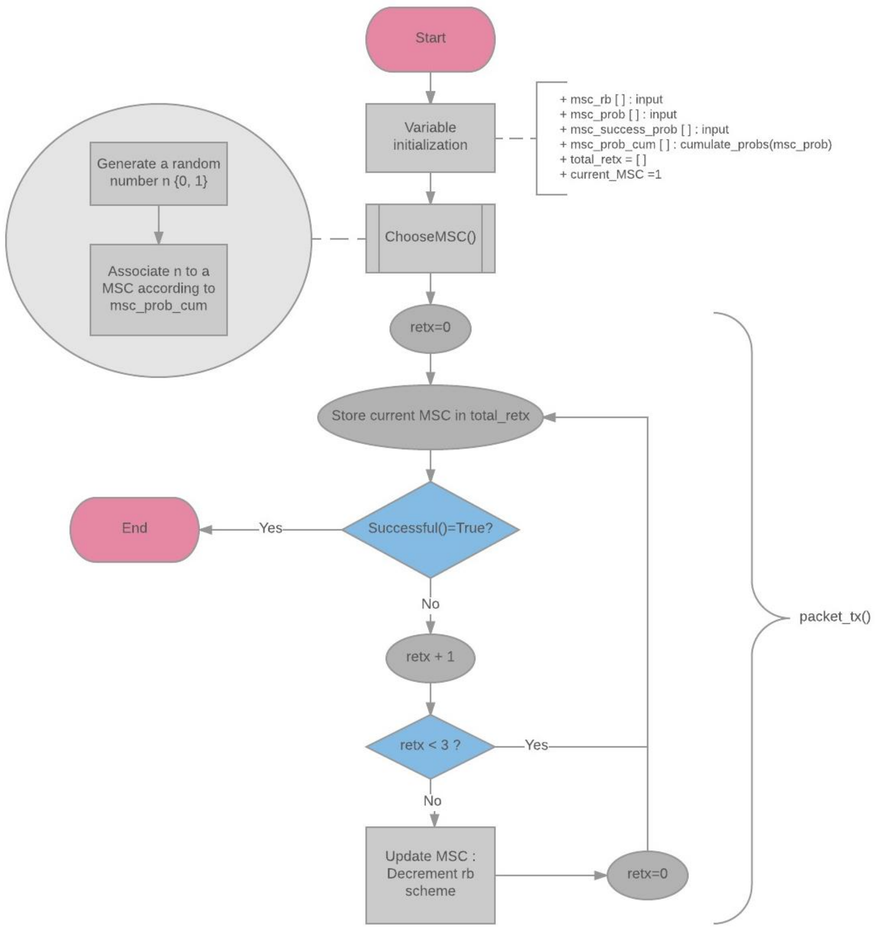
Annex 1. ViRANsim Simulator Classes description

In this annex is provided a detailed description of the ViRANsim simulator, including programming abstracts.

1. Channel Model

The channel model class simulates the effects of a wireless channel when packets are transmitted from the WTP to the final user's terminal. It gives a random behaviour to the packet transmission, giving the possibility of packet retransmissions. As a result, the channel model allows us to compute the time required to transmit a certain packet.

In order to do so, the following algorithm has been developed:



Start
Variable initialization
+ msc_rb [ ] : input
+ msc_prob [ ] : input
+ msc_success_prob [ ] : input
+ msc_prob_cum [ ] : cumulate_probs(msc_prob)
+ total_retx = [ ]
+ current_MSC =1
Generate a random number n {0, 1}
Associate n to a MSC according to msc_prob_cum
ChooseMSC()
retx=0
Store current MSC in total_retx
End
Yes
Successful()=True?
No
retx + 1
retx < 3 ?
Yes
No
Update MSC : Decrement rb scheme
retx=0
packet_tx()

Figure 76. Channel Algorithm

For each packet to be transmitted, the algorithm randomly selects a modulation scheme for the packet to be transmitted. After this, it is checked if the packet has been transmitted successfully or not. If so, the algorithm exits but if not, the algorithm checks how many times the packet has been retransmitted. If the number of retransmissions is greater than 3, the modulation scheme is decreased to a more robust modulation scheme (lower one) and then the packet is retransmitted. If the number retransmissions are less than 3, the packet is retransmitted without modifying the modulation scheme. All transmission and retransmission modulation schemes used during the packet transmission are stored in a list.

The class programmed has the following class variables and functions, which implement the algorithm:

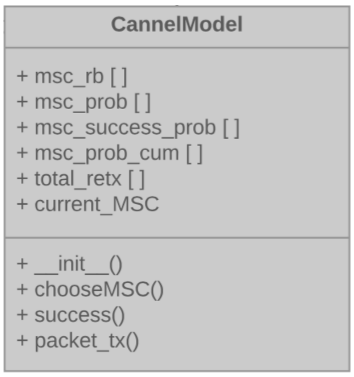


Figure 77. Channel Model class properties(variables) and methods

The first step that the algorithm does is to initialize the class variables. In the channel, different modulation schemes have to be defined, specifying its data rate, the probability of using that modulation scheme and its probability of error. Those values are the variables *msc_rb*[] , *msc_prob*[] and *msc_success_prob*[], respectively. Moreover, the modulation scheme cumulated probabilities are computed in order to easily select a MSC randomly and are stored in the variable *msc_prob_cum*[]. In these lists the modulation schemes have to be sorted from the faster to the lower modulation scheme and the same index in all the lists refer to the same modulation scheme (e.g. *msc_rb*[1], *msc_prob*[1], *msc_success_prob*[1] and *msc_prob_cum*[1] refer to the transmission rate, probability, success probability and cumulate probability of the first modulation scheme).

For testing, the following modulation schemes had been defined. In additions, these are the default values used if no others are specified for a simulation.

Index	MCS transmission rate	MCS probability	MCS cumulated probability	MCS success probability
1	**54 Mbps**	0.8	0.8	0.9
2	**48 Mbps**	0.1	0.9	0.95
3	**24 Mbps**	0.05	0.95	0.98
4	**12 Mbps**	0.03	0.98	0.99
5	**6 Mbps**	0.02	1	0.999

Table 41. Modulation schemes data rates, occurence probability, cummulated probability and success probability

In addition, the variables *total_retx* and *current_MSC* are defined. The *total_retx* variable is a list in which there are stored the modulation schemes used for the transmission and retransmission of the packet. The *current_MSC* variable is used to know which is the modulation scheme that the packet is using. For both variables, the modulation schemes stored in them are referred to its position in the *msc_rb* variable, which has to be sorted from the faster to the slower modulation scheme. For instance, if the value of *current_MSC* is 2, it will mean that the modulation scheme with 48Mbps is being used.

The three functions defined inside the ChannelModel class, apart from the initialization function, are the functions *chooseMSC*(), *success*() and *packet_tx*(). The function *packet_tx*() is the one that performs all the steps needed to perform the algorithm already explained. The other two functions are used inside *packet_tx*(). The function *chooseMSC*() generates a random number between [0,1]. The random number is compared to *msc_prob_cum* and a modulation scheme is associated to it. This is only done for the first transmission of the packet, not for the retransmissions. The function *success()* also generates a random number and giving the current modulation scheme it states if the packet has been successfully transmitted or not.

In order to better understand the algorithm, we will put an example of how it operates.

1. All the variables are initialized with the default modulation scheme values. The list *total_retx* is empty and the *current_MSC=0*.
2. The *current_MSC* is computed through *chooseMSC*() function. The random value obtained is 0.83, so the modulation scheme used will be the one operating at 48Mbps as 0.83 is between 0.8 and 0.9. Then, the *current_MSC=2* as the chosen modulation scheme is the second one in the list.
3. The packet is transmitted. The *current_MSC* is stored in the *total_retx* list. So, *total_retx= [2]*.
4. It is computed if the packet is sent successfully through *success()* function. The random number generated in *success()* is 0.98. As 0.98 is greater than 0.95 the packet is not transmitted successfully. The number of retransmissions is incremented. *Retx*=1
5. It is checked if the number of retransmissions is greater than 3. The packet has been sent once for the moment, so we go to the next step.
6. The packet is transmitted again. The *current_MSC* is stored in the *total_retx* list. So, *total_retx*= [2, 2].
7. The packet is not successfully transmitted again. The random number obtained is 0.96, which is greater than 0.95. The number of retransmissions is incremented. *Retx=2*
8. It is check if the number of retransmissions is greater than 3. It is not, so we go to the next step.
9. The packet is transmitted again. The *current_MSC* is stored in the *total_retx* list. So, *total_retx= [2, 2, 2]*.
10. The packet is not successfully transmitted again. The random number obtained is 0.97, which is greater than 0.95. The number of retransmissions is incremented. *Retx=3*
11. It is check if the number of retransmissions is greater than 3. It is so the current modulation scheme is decremented and the number of retransmissions with the current modulation scheme is set to 0. *Retx=0* and *current_MSC=3* (24Mbps).
12. The packet is transmitted again. The *current_MSC* is stored in the *total_retx* list. So, *total_retx*= [2, 2, 2, 3].
13. The packet is successfully transmitted. The random number obtained is 0.76, which is lower than 0.98.

The example has been forced to show how the retransmissions are managed as well as modulation schemes. With the default input values, most of the times the packet is successfully transmitted considering the probabilities introduced and the modulation scheme are rarely decreased, as the number of retransmissions is low.

2. Tenant General

The concept of Tenant in the simulator has been split into two parts: the class Tenant General and class Tenant WTP. The class Tenant General is the class in which the global properties of a certain tenant in the system are specified while the class Tenant WTP consists of the instance of a certain tenant in a WTP. This has been divided in this way in order to easily generate and manage the traffic of each of the tenants in each WTP but maintaining the entity of the Tenant as a general element in all the system.

As mentioned, the class Tenant General defines a certain virtual operator in the entire network. In Figure 78 it is shown the properties (variables of a class in python) and methods implemented in Tenant General class.

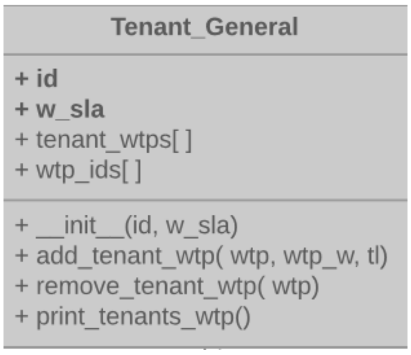


Figure 78. Tenant General class properties(variables) and methods

As shown in Figure 78 in bold, in order to create a Tenant_General object, it is necessary to give it an identifier, which usually will be a number, and the SLA agreed for the tenant in parts out of one (e.g. *w_sla*=0.6 instead of 60%). Moreover, each Tenant_general will have all the instances of tenant in each WTP, which means all the tenant_wtp objects associated to the tenant. All the objects of tenant_wtp associated to a Tenant, will have the same identifier as its Tenant_general. Moreover, the Tenant_general has the identifiers of the WTPs in which it has instances (tenant_wtp). It is important to point out that the indexes in the lists *tenant_wtps*[] and *wtp_ids*[] are related; the tenant_wtp object in the position n, *tenant_wtps*[n], is located in the WTP with identifier in the position n in the list *wtps_ids*[ ], so *wtp_ids*[n].

The methods *add_tenant_wtp*() and *remove_tenant_wtp*() are useful for adding and removing objects tenant_wtp to the list *tenant_wtps*[ ] and from *wtp_ids*[ ].

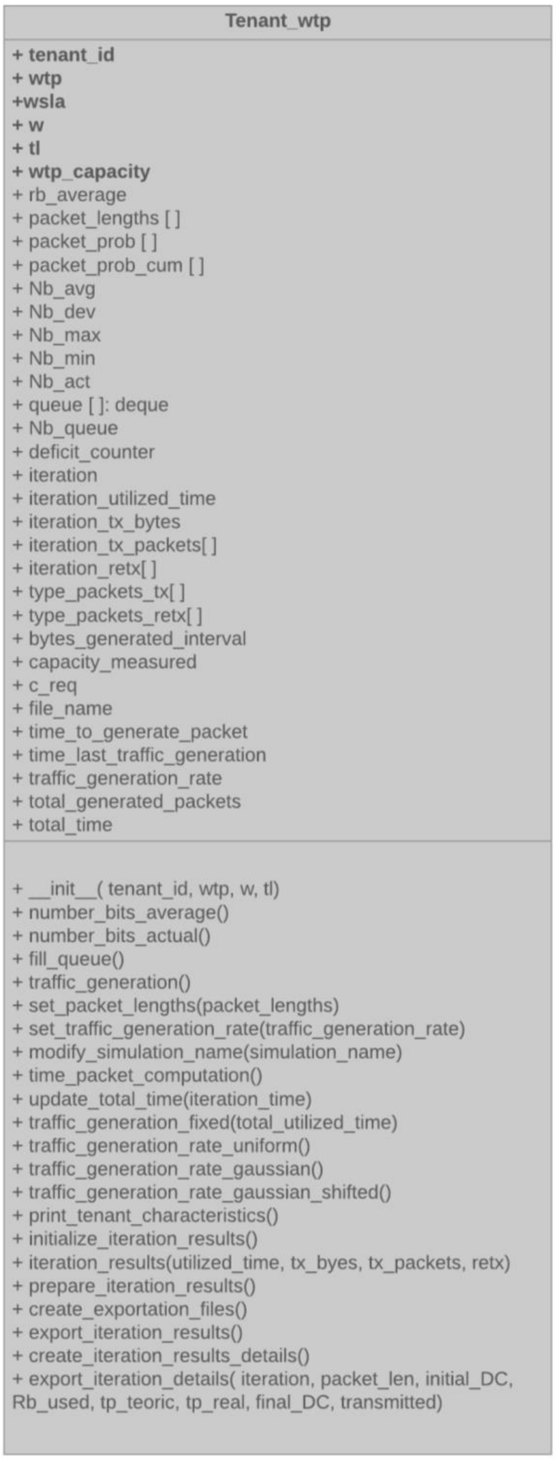
In this way, the Tenant_General allows us to control and access to all the information from the tenant_wtp objects associated to a certain Tenant_General object. Moreover, it allows us not to lose the entity of the general as a transversal element present in all the system.

3. Tenant WTP

The Tenant_WTP represents the instance of a certain Tenant_General in a determinated WTP, as has already been mentioned. This class contains all the variables and methods in Figure 79. In order to be initialized, the tenant_WTP needs to know to which Tenant_general it belongs (tenant_id) as well as the *wtp_id* in which is located (WTP). Moreover, it needs to know which is the SLA weight negotiated by the tenant and which is the capacity of the WTP it has been assigned (wtp_capacity). Moreover, for the initialization, it is also needed a variable related to the traffic level (*tl*), which has not finally been used during the studies but it could be used in future. The value of this variable is always set to 1, representing the 100% of the traffic level.

An important variable that belongs to the tenant_WTP is the queue of packets of the tenant in that WTP. The queues of tenant_WTP are FIFO (*First In First Out*) queues, which have been programmed through a Python class that is called *deque* from the module collections, which has to be imported in the python script. With this class it is possible to add packets at the end of the queue and to obtain and remove the first packet in the queue by using the functions *append()* and *popleft()*.

The traffic generated by a certain Tenant is not generated in the Tenant_General and then passed to the tenant_WTP, but instead it is generated in the tenant_WTP. This decision was taken for simplicity reasons, as it is easier to manage the queue locally, generating its traffic and processing their packets in the tenant_WTP. Considering this, different traffic generators are included in the class tenant_WTP. These traffic generators generate traffic considering the SLA, so in average the traffic generated will not be greater than the agreed.



```
Tenant_wtp
+ tenant_id
+ wtp
+wsla
+ w
+ tl
+ wtp_capacity
+ rb_average
+ packet_lengths [ ]
+ packet_prob [ ]
+ packet_prob_cum [ ]
+ Nb_avg
+ Nb_dev
+ Nb_max
+ Nb_min
+ Nb_act
+ queue [ ]: deque
+ Nb_queue
+ deficit_counter
+ iteration
+ iteration_utilized_time
+ iteration_tx_bytes
+ iteration_tx_packets[ ]
+ iteration_retx[ ]
+ type_packets_tx[ ]
+ type_packets_retx[ ]
+ bytes_generated_interval
+ capacity_measured
+ c_req
+ file_name
+ time_to_generate_packet
+ time_last_traffic_generation
+ traffic_generation_rate
+ total_generated_packets
+ total_time

+ __init__( tenant_id, wtp, w, tl)
+ number_bits_average()
+ number_bits_actual()
+ fill_queue()
+ traffic_generation()
+ set_packet_lengths(packet_lengths)
+ set_traffic_generation_rate(traffic_generation_rate)
+ modify_simulation_name(simulation_name)
+ time_packet_computation()
+ update_total_time(iteration_time)
+ traffic_generation_fixed(total_utilized_time)
+ traffic_generation_rate_uniform()
+ traffic_generation_rate_gaussian()
+ traffic_generation_rate_gaussian_shifted()
+ print_tenant_characteristics()
+ initialize_iteration_results()
+ iteration_results(utilized_time, tx_byes, tx_packets, retx)
+ prepare_iteration_results()
+ create_exportation_files()
+ export_iteration_results()
+ create_iteration_results_details()
+ export_iteration_details( iteration, packet_len, initial_DC,
Rb_used, tp_teoric, tp_real, final_DC, transmitted)
```

Figure 79. Tenant WTP class variables and methods

The first traffic generation approach was to implement a traffic generator that generates random traffic according to Figure 80. This first traffic generator is programmed in the method *traffic_generation*() and it uses the functions *number_bits_average*(), *number_bits_actual*() and *fill_queue*().

This traffic generator has the inconvenient that it was focused on having a certain amount of bytes in the queue but had not into account the time synchronization between the packet generation and the packet transmission. Moreover, for the studies of the different algorithms it was convenient to be able to establish a certain traffic generation rate during the simulation, which is not possible with the initial traffic generator.

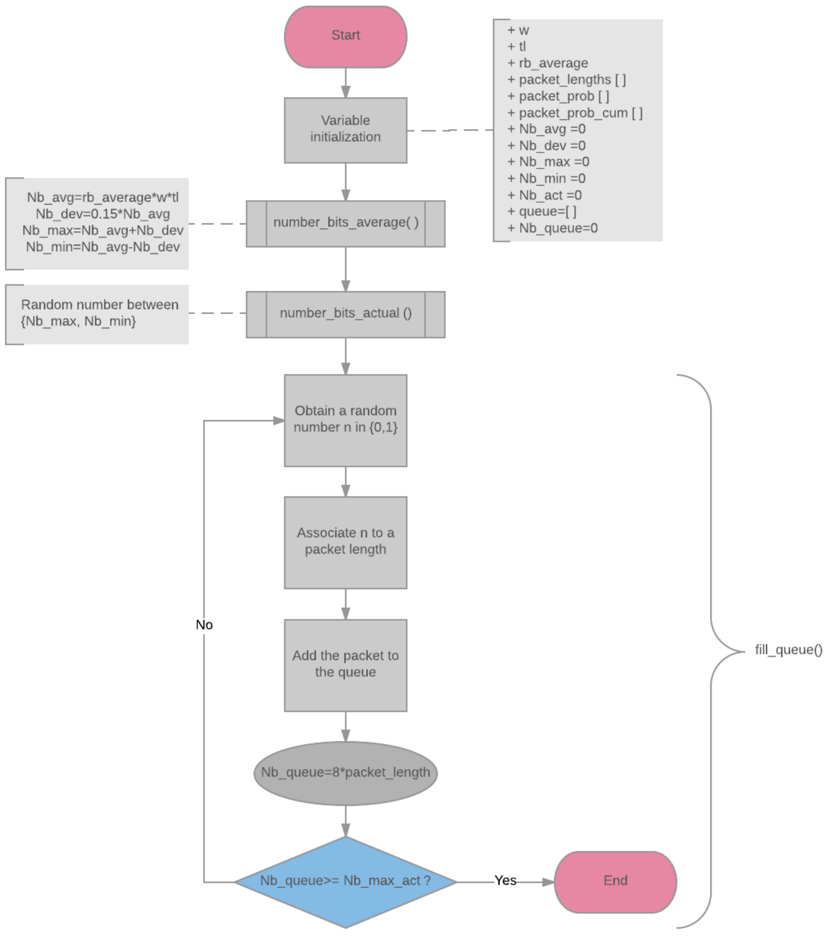


Figure 80. Initial traffic generator scheme

Considering the disadvantages of the initial proposed traffic generator, it was defined a new traffic generator, which is time based and works at a fixed traffic rate. The algorithm is called **fixed generator** and it is programmed inside the function *traffic_generation_fixed*(). Its operation can be found in Figure 80. Basically, the tenant_wtp computes the time required to generate a packet at the specified traffic generation rate and it waits until enough time has passed to generate a packet. As the algorithm works based on the time already spent by the WTP when sending packets, it is needed an initial sufficient time to generate a packet for the correct initialization of the system.

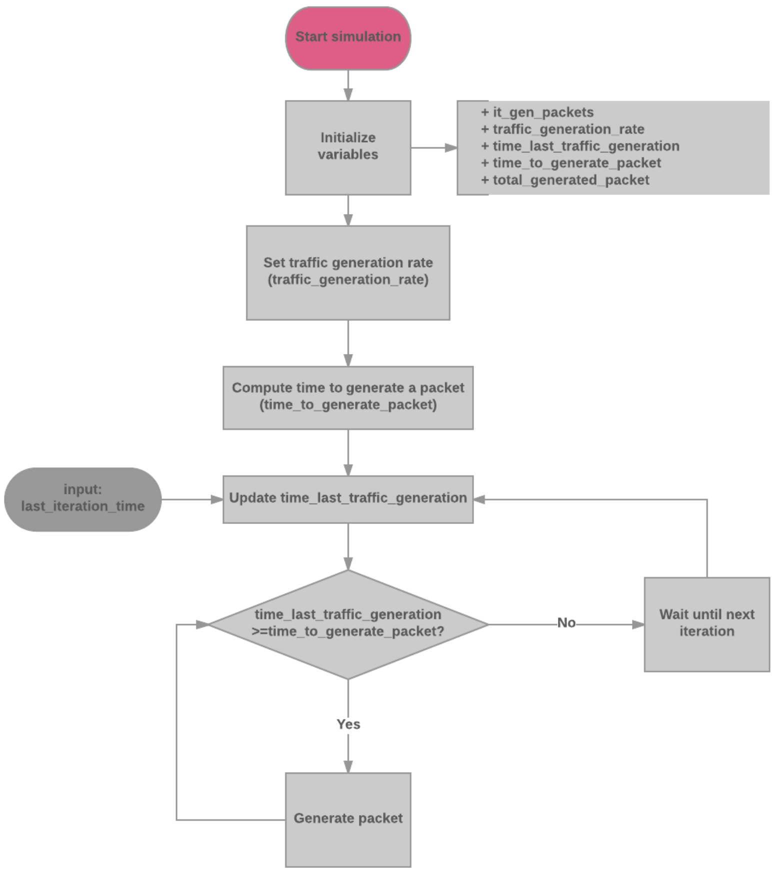

Start simulation
Initialize variables
+ it_gen_packets
+ traffic_generation_rate
+ time_last_traffic_generation
+ time_to_generate_packet
+ total_generated_packet
Set traffic generation rate (traffic_generation_rate)
Compute time to generate a packet (time_to_generate_packet)
input: last_iteration_time
Update time_last_traffic_generation
time_last_traffic_generation >=time_to_generate_packet?
No
Wait until next iteration
Yes
Generate packet

Figure 81. Fixed traffic generator scheme

A critical issue for the *traffic_generator_fixed* is how to select the traffic generation rate. In the traffic study, different measures that have been carried out in order to set the rate are discussed. Through the study developed, it has been possible to formulate the following equations, that allow establishing a traffic generation rate for a certain tenant while avoiding the queues to indefinitely increase and using the maximum of the capacity associated to the tenant. In order to establish the data rate, it is necessary to compute the capacity of a WTP and then the capacity associated to the tenant in that WTP.

$$Capacity_{WTP_{Rbi}}[bits] = Rb_i\ [bps] \cdot Q_s[s] \quad for\ i\ in\ 1. \tag{87}$$

$$Capacity_{Tenant\ k_{Rb_i}}[bits] = w_k \cdot Capacity_{WTP_{Rbi}}[bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}} \tag{88}$$

The capacity of the tenant depends on the capacity of the WTP, the SLA weight of the tenant and the factors *r_i*, which is used to compensate differences added by 802.11g delays, and *f_{retx}*, that compensates the effect of retransmissions. The *f_{retx}* factor formula has been fixed through simulations.

$$r_i = \frac{time_{delays_{Rb_i}}[s]}{time_{total\ packet\ transmission_{Rb_i}}[s]} \tag{89}$$

$$f_{retx_{rbi}} = p_{retx_{rb_i}} - 0.01 \tag{90}$$

The capacity is computed for each possible data rate in the WTP. After this, the resultant capacities are used to compute the traffic generation rate, considering the probabilities associated to the data rate in which the capacity was initially computed.

$$G_{T_{Tenant\ k}} = \frac{\sum_{i=0}^{N} Capacity_{Tenant\ K_{Rb_i}} \cdot p_i\ [bits]}{Q_s[s]} \tag{91}$$

$$G_{T_{Tenant\ k}} = \frac{\sum_{i=0}^{N} w_k \cdot Rb_i[bps] \cdot Q_s[s] \cdot p_i\ [bits] \cdot (1-r_i) \cdot f_{retx_{rbi}}}{Q_s[s]} = w_k \cdot \sum_{i=0}^{N} Rb_i[bps] \cdot p_i\ [bits] \cdot (1-r_i) \cdot f_{retx_{rbi}} \tag{92}$$

$$G_{T_{Tenant\ k}} = w_k \cdot Rb_{effective_{average}} \tag{93}$$

The effective data rate in the last equation corresponds to equation(94) .

$$Rb_{effective_{average}} = \sum_{i=0}^{N} Rb_i\ [bps] \cdot p_i \cdot (1-r_i) \cdot\ f_{retx_{rbi}} \tag{94}$$

As one of the requirements of the simulator was to be able to have different traffic generator modes, it has been designed a uniform traffic generator, and two Gaussian traffic generators. The three generators are more of the form of a traffic shaper than a traffic generator. This is because the three generators change the traffic generation rate every certain time period and then the packets are generated by the function *traffic_generation_fixed*(). So the responsible of generating packets in all modes is the function *traffic_generation_fixed*() while the other functions just shape the traffic according to a certain distribution.

The **uniform traffic generator** is programmed using a python function from class random that is called 'uniform'. This function, whose name is *traffic_generation_rate_uniform*(), needs to be called specifying in which range of values it has to generate a value. In our case, the minimum traffic generation rate is 0 Mbps and the maximum generation rate is computed as in equation (22), considering the maximum data rate that the WTP is working with and the weight of the tenant. Notice that r_{Rbmax} corresponds to the delays compensation and r_i at the maximum data rate Rb_{max}.

$$G_{T_{max}}[bps] = \frac{w \cdot\ Capacity_{Rb\_max}[bits]}{Q_s[s]} = \frac{w \cdot\ Rb_{max} \cdot (1-r_{Rbmax}) \cdot SQ[s]}{Q_s[s]} = w \cdot Rb_{max} \cdot (1-r_{Rbmax}) \tag{95}$$

It is relevant to mention that the retransmission compensation factor is not considered in the computation of Gt_{max} as it is computed as a peak value, so when no retransmissions occur it could happen that the generation rate is the computed without that factor. The delays compensation factor is included as the 802.11g delays are always considered in the transmission of packets.

There have been designed two gaussian generators. For the first **Gaussian generator** designed**,** whose function name is *traffic_generation_rate_gaussian(),*a python function from class random called 'gauss' is used, which needs the mean and the deviation of the

Gaussian distribution. The mean of the Gaussian distribution has been set to half Gt_{max}. Gt_{max} is computed in the same way as in the uniform generator. Considering Figure 82, most of the probability is concentrated into the interval (μ-3σ, μ+3σ). So, it has been truncated the function making this interval coincide with (0, Gt_{max}). The standard deviation (σ) and mean (μ) of our Gaussian distribution are computed in equations (96) and (97) respectively.

$$\mu = \frac{Gt_{\max}}{2} \qquad (96)$$

$$\mu - 3\sigma = 0 \rightarrow \sigma = \frac{Gt_{\max}}{6} \qquad (97)$$

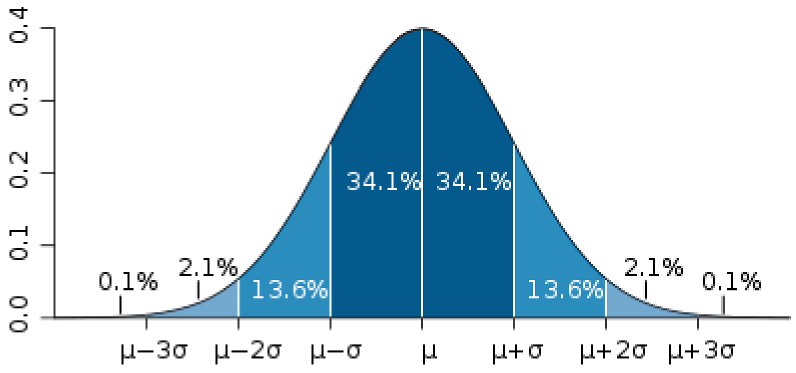


Figure 82. Gaussian distribution

The second gaussian generator has been called **shifted Gaussian generator** (*traffic_generation_rate_gaussian_shifted*()). This Gaussian generator works in the same way as the *traffic_generation_rate_gaussian*() but it shifts the mean of the Gaussian distribution to the average capacity that the tenant can request. The standard deviation has been defined as the 15% of the mean value, but it can be tuneable.

$$\mu = w_{tenant\ i} \cdot Capacity_{average,wtp} \qquad (98)$$

$$\sigma = 15\% \cdot \mu \qquad (99)$$

The need of this last Gaussian generator was caused by tests of the weights algorithm, as it was needed a generator in which sometimes the traffic requested was greater than the capacity average of the WTP.

Apart from the traffic generation, the class Tenant_wtp is in charge of generating some exportation files with the system performance data, which will be explained deeply in the Exportation Files section.

4. WTP

As it has already been defined previously, the WTP is the element in charge of providing wireless connectivity to the different tenants' users. Figure 83 contains all the WTP class variables and methods included in the class WTP.

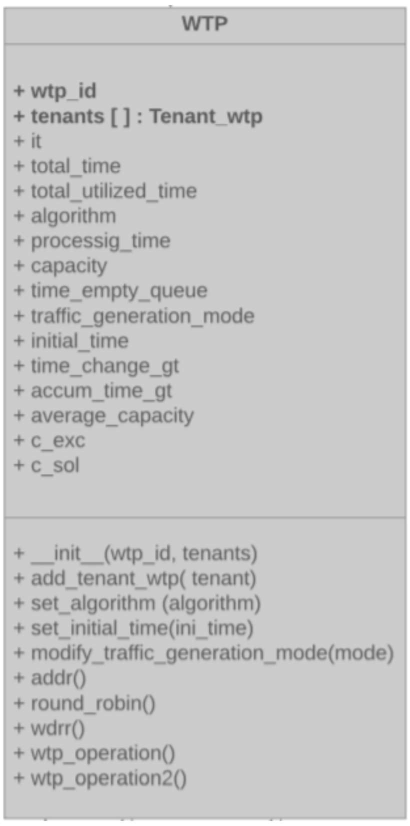


Figure 83. WTP class properties (variables) and methods

When a WTP is created, it is necessary to give to it an identifier (*wtp_id*) and it needs to be given a list of the instances of the WTP (*tenants*[]), which are tenant_wtp objects. However, the class also contains a function to add new tenant instances, called *add_tenant_wtp*(), so the WTP can be initialized with an empty list and then add the instances of the tenants. In this way, the WTP can access to tenant_wtp objects, which allows the scheduling of their packets and management of their traffic.

A relevant functionality of the WTP is the scheduling of packets. In this class, it can be found the functions *rr*() for Round Robin, *wdrr*() for Weighted Deficit Round Robin and *adrr*() for Air-Time Deficit Round Robin. The algorithm is stored in the variable algorithm and the possible values are ‘rr’, ‘wdrr’ and ‘adrr’. The default algorithm is ADRR but it can be changed by the function *set_algorithm*(). Each of the scheduling function access to the tenant_wtp queues to transmit its packets and modify its deficit counter (DC). Inside the three scheduling functions, it is created an object of ChannelModel and the packets are sent using the function *packet_tx*(), previously explained.

Besides the already explained operation of the algorithms, ADRR, WDRR and RR functions have to translate the retransmissions list returned by the function *packet_tx*() to time in microseconds. In order to do so, it uses a function located in the script scenario.py called *tp_real_computation*(), which needs as a input the list of retransmissions and the packet length. It has to be remembered that the list returned by the fuction *packet_tx*() is a list of the data rates used in each of the retransmissions. So, the function *tp_real_computation*() computes the time per each transmission/retransmission and adds all the times.

The time of a single transmission (or retransmission) is computed considering the 802.11g delays values in Table 42 and using the equation (100). It has not been considered the back-off times in 802.11g as we are just focusing on the downlink so collision avoidance is not needed. It is important to point out that in the equation it is taken into account the transmission of the MAC data packet and MAC ACK packet like in Figure 21. This is the reason why the physical layer and signal extension delays are multiplied by 2. Figure 22 shows the 802.11g frame format, where is possible to identify the fields of physical layer and signal extension. It has to be considered that the column physical layer in the table coincides with the preamble plus the signal in Figure 22.

DIFS(us)	Physical layer (us)	Signal extension (us)	SIFS	ACK length (bytes)
28	20	6	10	14

Table 42. 802.11g delays considered

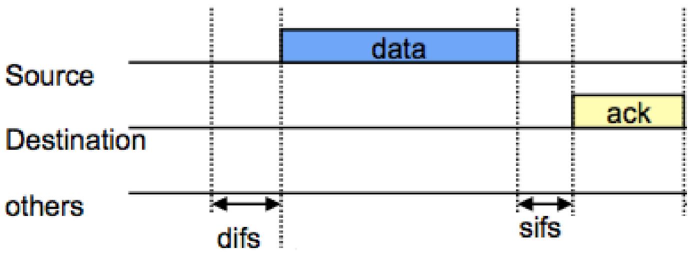


Figure 84. 802.11 error control in radio medium

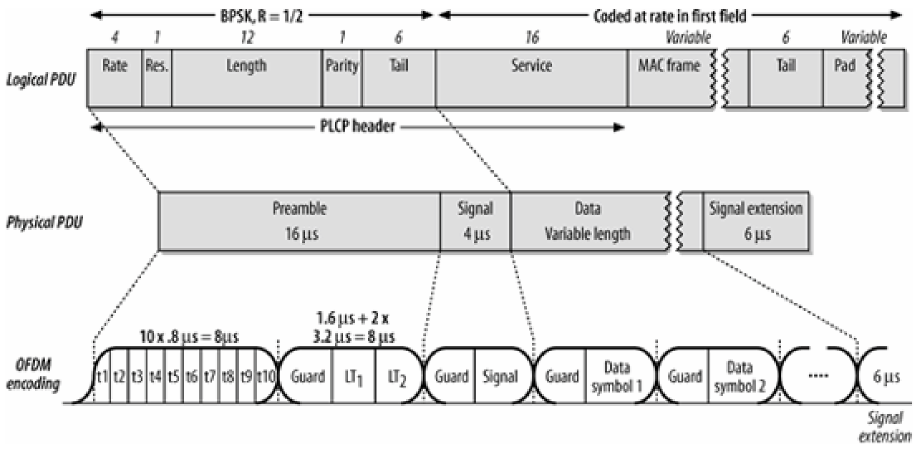


Figure 85. 802.11g ofdm frame format

$$time_{packet\_tx} = t_{DIFS} + 2 \cdot t_{phy_{layer}} + \frac{Packet_{length}}{R_b} + 2 \cdot t_{signal_{extension}} + t_{SIFs} + \frac{ACK_{length}}{Rb} \qquad (100)$$

Considering that the WTP is the responsible of running the scheduling algorithm, it contains different variables related to the time spent in each of the iterations as well as during all the simulation. Moreover, it contains the variable *time_empy_queue*, which is added to the iteration time when the queue of a certain tenant is empty. This decision was taken in order to simulate that when the queue is empty the time keeps running, as well as to consider the processing time when looking at queue. It also includes the variable *it* that corresponds to the current iteration number.

In addition, in the WTP there are called the functions to generate traffic of each of the tenant instances in the WTP. In order to do so, it is necessary to specify which traffic generation mode is desired. The traffic mode is stored in the variable *traffic_generation_mode* and the default value is 'Fixed'. The traffic generation mode can be modified by the function *modify_traffic_generation_mode*(), and the other modes that the function accepts are 'Uniform', 'Gaussian' and 'Gaussian Shifted'. In the WTP there is a variable called *initial_time* needed for the tenant_wtp objects to generate traffic, as they generate traffic from the time that has already been spent.

All the WTP operation is runnable from the functions *wtp_operation*() and *wtp_operation2*(). The difference between them is that the function *wtp_operation*() generates the traffic randomly using the tenant_wtp function *traffic_generation*() while the function wtp_operation2() allows to generate the traffic using the functions *traffic_generation_fixed*(), *traffic_generation_rate_uniform*(), *traffic_generation_rate_gaussian*() and *traffic_generation_rate_gaussian_shifted*(). The three last functions are possible to be called just once every *time_change_gt*, so the traffic generation rate is just modified every certain period of time. In order to do so, a variable called *accum_time_gt* is created and counts the time since the last traffic generation rate update. When accum_time_gt is equal or greater than time_change_gt, one of these functions is called when selected.

Finally, the WTP class includes some variables used by the weight compensation algorithm. These variables are the capacity_average that specifies which is the average capacity that the WTP can afford in bytes, the c_exc, which is the excess of capacity and the c_sol, which is the solicited capacity.

The capacity_average of a WTP is computed through the function *compute_rb_average_WTP_v2*() in the script scenario.py and takes into account the equations (101), (102) and (103). The capacity average computation uses the effective data rates of the WTP (including 802.11g delays), its probabilities and a new compensation factor f_{comp}, explained below.

Equation (101), shows how the average time to transmit given a certain data rate Rb_i is computed. Notice that it is an approximation, as it just considers three retransmissions while the programmed algorithm decreases the data rate if a packet is sent incorrectly three times, which will have a low probability with the values given. Equation (103) computes the compensation factor, needed to avoid long queues. This is caused by the difference of data rate allowed in the WTP, as when a low data rate is used, it processes fewer packets while new packets are arriving to the system, which results in an increase of the queue. Therefore, we use the relation of times of the maximum data rate and the minimum data rate in the system to avoid that packets accumulate in the queue.

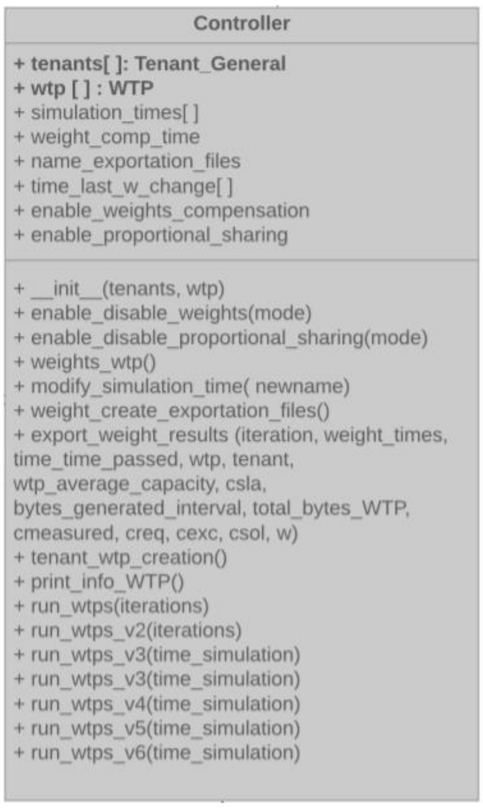
$$C_{average} = f_{comp} \cdot \sum_{i=0}^{N-1} \frac{Packet_{length_{average}}(bytes) \cdot 8\,\frac{bits}{Byte}}{tp_{average_{Rbi}}} \cdot prob(Rb_i) \qquad \textbf{(101)}$$

$$tp_{average_{Rb_i}} = tp(Rb_i)\cdot prob(packet_{ok}) + 2\cdot tp(Rb_i)\cdot prob(packet_{ok})\cdot prob(packet_{Nok}) + 3 \cdot tp(Rb_i)\cdot prob(packet_{ok})\cdot prob(packet_{Nok})^2 \qquad (102)$$

$$f_{comp} = 1 - \left(\frac{tp\left(MAX\ Rb, packet\ lenght_{average}\right)}{tp\left(MIN\ Rb, packet\ lenght_{average}\right)}\right) \qquad (103)$$

5. Controller

The Controller is the element in charge of managing the different WTP and assuring the SLA of the different Tenants in the network. Figure 86 shows the variables and the methods of class Controller.



```
Controller
+ tenants[ ]: Tenant_General
+ wtp [ ] : WTP
+ simulation_times[ ]
+ weight_comp_time
+ name_exportation_files
+ time_last_w_change[ ]
+ enable_weights_compensation
+ enable_proportional_sharing

+ __init__(tenants, wtp)
+ enable_disable_weights(mode)
+ enable_disable_proportional_sharing(mode)
+ weights_wtp()
+ modify_simulation_time( newname)
+ weight_create_exportation_files()
+ export_weight_results (iteration, weight_times,
time_time_passed, wtp, tenant,
wtp_average_capacity, csla,
bytes_generated_interval, total_bytes_WTP,
cmeasured, creq, cexc, csol, w)
+ tenant_wtp_creation()
+ print_info_WTP()
+ run_wtps(iterations)
+ run_wtps_v2(iterations)
+ run_wtps_v3(time_simulation)
+ run_wtps_v3(time_simulation)
+ run_wtps_v4(time_simulation)
+ run_wtps_v5(time_simulation)
+ run_wtps_v6(time_simulation)
```

Figure 86. Controller class variables and methods.

Giving that the controller manages all Tenants and WTPs, to create a Controller it is needed to introduce a list with all the Tenant General objects (*tenants*[ ]) and a list with the WTP objects (*wtp*[ ]) in the system. The controller has three main responsibilities: the tenant_wtp creation, the weight compensation algorithm and the run of all the WTP operations.

The first responsibility is the creation of the tenant_wtp. The Controller class has the method *tenant_wtp_creation*(), which adds a tenant_wtp per each Tenant_General and per each WTP. It starts creating the tenant_wtp in the Tenant_General through the function *add_tenant_wtp*() in Tenant_General, and then the tenant_wtp is added at the

WTP tenant list (*tenants*[ ]). This function has to run before starting the simulation, for all the tenant_wtp to be initialized.

Secondly, the Controller class is in charge of performing the weight compensation algorithm, which has been previously explained. The function in charge of this is called *weights_wtp*(). In relation to the weight compensation algorithm there are two functions related to the exportation of results: the first to create the exportation files for the weight compensation algorithm, which is called *weight_create_exportation_files*() and the second to export the results provided by the *weights_wtp*() function, which is called *export_weight_results*().

Moreover, the Controller has a variable called *enable_weights_compensation*, which is used to enable or disable the weights compensation, and the variable *enable_proportional_sharing*, to enable or disable the proportional sharing. The state of the variables can be modified through the functions *enable_disable_weights*(), for *enable_weights_compensation*, and *enable_disable_proportional_sharing*(), for the *enable_proportional_sharing*. Both variables can take two values: 'on' or 'off'. There is still another two-variable called *weight_comp_time*, which corresponds to the period in which weights are compensated, and the variable *time_last_w_change*, which counts how much time since the last weight compensation has passed.

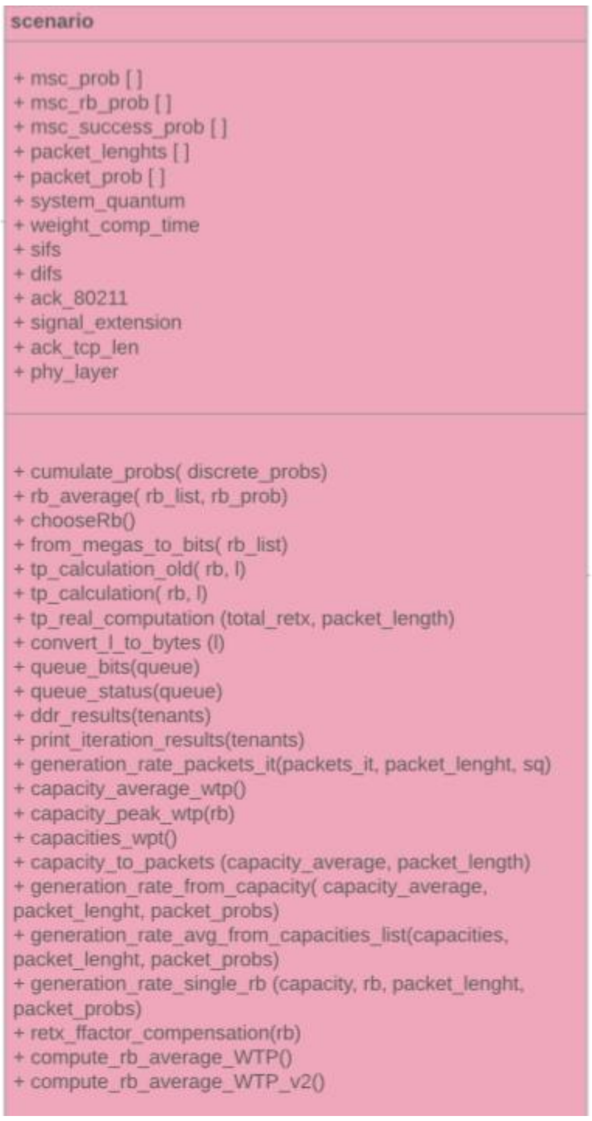
Finally, the Controller is the element responsible of running all the WTPs operation through functions *wtp_operation*() or *wtp_operation2()* in WTP class.

There are different *run_wtps*() versions. The first version is based on iterations and the wtp_operation(). When the function wtp_operation2() was designed it was integrated in the second version. After deciding that there was the need for the simulations to be based on a simulation time instead of iterations, the versions 3 and 4 were also designed for both wtp_operation() and wtp_operation2(). In version 5 it was included the weight compensation algorithm. In that version, the operation of different WTPs did not work as desired so it was designed v6, which is capable to run the multi-WTP solution properly.

The most upgraded version, *run_wtps_v6*(),works as explained in the following. In order to run the different WTP and change the tenant's weights, each WTP is run every *weight_comp_time*. The function *wtp_operation2*() of the different WTPs in the system is called sequentially and once all WTP have finished, the weights are changed through *weights_wtp*(). Then, the procedure is repeated until the simulation time is reached. It has to be pointed out that, the WTP may run different times but really close to *weight_comp_time*, as the WTP are not exactly synchronized. This will be discussed in the section of 'Time synchronization'.

6. <u>Scenario</u>

The scenario is a python script, not a class, that contains different global system variables and generic functions that may be used by different classes of the system. The values of variables in scenario are not modified during all the simulation.



scenario

+ msc_prob [ ]
+ msc_rb_prob [ ]
+ msc_success_prob [ ]
+ packet_lenghts [ ]
+ packet_prob [ ]
+ system_quantum
+ weight_comp_time
+ sifs
+ difs
+ ack_80211
+ signal_extension
+ ack_tcp_len
+ phy_layer

+ cumulate_probs(discrete_probs)
+ rb_average(rb_list, rb_prob)
+ chooseRb()
+ from_megas_to_bits(rb_list)
+ tp_calculation_old(rb, l)
+ tp_calculation(rb, l)
+ tp_real_computation (total_retx, packet_length)
+ convert_l_to_bytes (l)
+ queue_bits(queue)
+ queue_status(queue)
+ ddr_results(tenants)
+ print_iteration_results(tenants)
+ generation_rate_packets_it(packets_it, packet_lenght, sq)
+ capacity_average_wtp()
+ capacity_peak_wtp(rb)
+ capacities_wpt()
+ capacity_to_packets (capacity_average, packet_length)
+ generation_rate_from_capacity(capacity_average, packet_lenght, packet_probs)
+ generation_rate_avg_from_capacities_list(capacities, packet_lenght, packet_probs)
+ generation_rate_single_rb (capacity, rb, packet_lenght, packet_probs)
+ retx_ffactor_compensation(rb)
+ compute_rb_average_WTP()
+ compute_rb_average_WTP_v2()

Figure 87. Scenario variables and functions

Some important system parameters are configured through this script before starting the simulation. This is the case of the different modulation schemes data rates (*msc_rb*), its probabilities (*msc_prob*) and its success packet transmission probability (*msc_success_prob*). Moreover, the packets lengths (*packet_length*) and their probability (*packet_prob*) can be set in this script. Another important variables are set in this scenario script: the system quantum (*sytem_quantum*), the time in which weights are changed (*weight_comp_time*) and the 802.11g delays (*difs, sifs, ack_80211, signal_extension* and *phy_layer*).

As mentioned, the script not only includes global variables but also generic functions, accessible by all the elements in the system. Different groups of functions have been programmed in this script and can be classified as shown in Table 43.

Function type	**Description**	**Functions**

Conversion units	Functions that allow the conversion of magnitudes.	*from_megas_to_bits()*
		convert_l_to_bytes()
		capacity_to_packets()

Time	Functions related to the time computation.	*tp_calculation_old()*
		tp_calculation()
		tp_real_calculation()

Data rate related	Functions that work with the data rate	*rb_average()*
		chooseRb()

Status	These functions return the status of a certain element. The status is return by using a print	*Queue_status()*
		ddr_results()
		queue_bits()
		print_iteration_results()

Traffic generation rate	Functions related to the traffic generation computation and the capacity of a WTP.	*generation_rate_packets_it()*
		generation_rate_from_capacity()
		generation_rate_avg_from_capacities_list()
		generation_rate_single_rb()
		retx_factor_compensation()

WTP Capacity	Compute the WTP capacity considering different assumptions	*capacity_average_wtp()*
		capacity_peak_wtp()
		capacities_wtp()
		compute_rb_average_WTP()
		compute_rb_average_WTP_v2()

Statistical	Provide statistical computations.	*Cumulate_probs()*

Table 43. Scenario functions classification

It was decided to use a separated script for these auxiliary functions, as all the system classes can access them, so they can be shared. Moreover, some of these functions are used for checking the expected values of other functions in a faster way. This is the case of some traffic generation rate functions.

Annex 2. Studies Detail

In this annex is provided the extended version of some of the studies already explained in the Studies section of the thesis. As before, these studies have been slit into single WTP and Multi-WTP.

1. Single WTP

In this section, the extended version of some single-WTP studies is provided.

1.1. Iterations Convergence Study

In this first study, it is found the convergence study done in order to determine which is the adequate number of iterations needed to obtain valid results and minimize the size of result simulation files. It deserves to be pointed out that this is the first study performed with the ADRR and it is being considered that the input to run the WTP operation is the number of iterations, as it is the first implementation. This study also wants to have a first approach of the behavior of the ADRR and the simulator performance.

It is also important to point out that the DC adjustment in this section is performed by reducing the real packet transmission time from the DC after the transmission, as in equation **(31)**.

$$DC_{Tenant\ i} = DC_{Tenant\ i} - tp_{real} \qquad (104)$$

The scenario used for the simulations in this section consists of two tenants in one WTP that share the available resources based on given weights. Moreover, the modulation scheme and the packet length are chosen randomly. The function used to generate traffic is traffic_generation(). The probabilities for selecting the packet length and the modulation coding schemes (MCS) are shown in Table 44 and Table 45, respectively.

Packet Length (Bytes)	Packet probability
1514	**0.7**
512	**0.3**

Table 44. Packet lengths and probabilities

MCS transmission rate	MCS probability	MCS success probability
54 Mbps	0.8	0.9
48 Mbps	0.1	0.95
24 Mbps	0.05	0.98
12 Mbps	0.03	0.99
6 Mbps	0.02	0.999

Table 45. Data rate, probability and success probability of each modulation scheme

The study has been developed focusing on ADRR and it has been analyzed for both cases of using and not using delays from the 802.11g protocol (SIFS, DIFS…), in order to prove that the use of these delays does not affect the convergence. Moreover, it has also been contrasted the convergence when WDRR is used.

In order to determine which is the minimum number of iterations, for each of the cases, different simulations of different number of iterations have been run. For each of the simulations, the average time and transmitted bytes over all the iterations has been computed. Afterwards, it has been obtained the percentage of time and transmitted bytes over the total time and bytes of each of the tenants in order to obtain how both parameters are shared. As ideally, both tenants should equally share the resources, it has been computed the dispersion in relation to the ideal value that is the weight initially specified for each of the tenants. For the first simulations it is 50% for both tenants.

This document wants to analyze which is the adequate number of iterations needed for the algorithm to converge. This is why the dispersion threshold has been set to 1E-4 for the parameter so, dispersion greater than 1E-4 will not be acceptable for the optimized parameter.

1.1.1. ADRR without 802.11g delays

As a first step, the convergence has been analyzed for the case of ADRR without having into account 802.11g delays (SIFS, DIFS…) for the packet transmission. The analysis has been performed for both time and transmitted bytes. For ADRR the quantum has been set to 2ms, which takes into account the maximum packet length (1514 bytes) and the slower data rate (6Mbps).

In this case, the time considered for a packet transmission is computed according to the following equation:

$$time_{packet\_tx}[s] = \frac{Packet_{length}\ [bytes]}{R_b[bps]} \quad (105)$$

1.1.1.1. Time convergence

As a first step, we will look at the results in terms of percentage of time used by each of the tenants over the total time per iteration. This can be found in the first two columns of Table 46. The expected or ideal percentage is the 50%. From the average values, it has been computed the dispersion towards the theoretical value 50%. The values used are in parts out of one (0.5 instead of 50%) per each tenant according to the following expression:

$$dispersion = |\%time_{tenant_X} - 0.5| \quad (106)$$

The dispersion results are shown in the third and fourth columns of Table 46. The values in the table are in parts per unit.

% TOTAL TIME	% Time Tenant 1	% Time Tenant 2	Dispersion Tenant 1	Dispersion Tenant 2
100 Iterations	0.500213	0.499787	2.1339E-04	2.1339E-04
500 Iterations	0.499943	0.500057	5.6794E-05	5.6794E-05
600 Iterations	0.499968	0.500032	3.1920E-05	3.1920E-05
750 Iterations	0.499980	0.500020	2.0350E-05	2.0350E-05
1000 Iterations	0.500005	0.499995	4.5657E-06	4.5657E-06
2000 Iterations	0.499993	0.500007	7.3106E-06	7.3106E-06
3000 Iterations	0.499992	0.500008	7.9232E-06	7.9232E-06
4000 Iterations	0.500002	0.499998	1.7524E-06	1.7524E-06
5000 Iteracions	0.500006	0.499994	5.7905E-06	5.7905E-06
10000 Iterations	0.500001	0.499999	1.1519E-06	1.1519E-06
Theoretical	0.500000	0.500000	0	0

Table 46. Percentages of average transmitted bytes over the total and dispersion around 0.5 for both Tenants

The dispersion for both tenants is the same because it is computed as the absolute value of the difference between the obtained percentage and the theoretical one (0.5). In Figure 88 it is plotted the dispersion results of the table, which will be the same for both tenants.

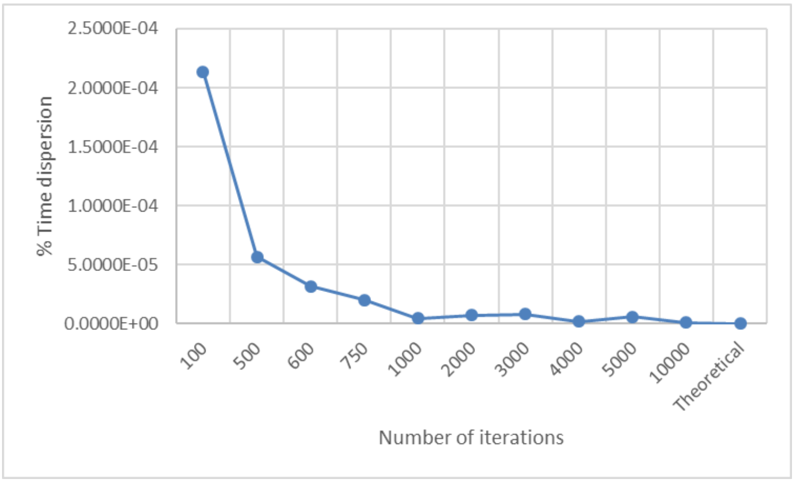


Figure 88. Time dispersion over 0.5 for different number of iterations. Case without 802.11g delays

In the represented dispersion graphic in Figure 88, it can be observed that the difference between the ideal value and the obtained through simulations decreases drastically at 500 iterations. At 600 iterations, the difference between the ideal and simulated values is of order 1E-5.

Giving this result, the minimum number of iterations to converge are 500 iterations considering that our criteria is that the dispersion must be lower than 1E-4.

1.1.1.2. Transmitted bytes convergence

Secondly, it has been analyzed the convergence when focusing on the percentage of transmitted bytes per iteration in average. The statistics analyzed are the same as previous ones. The table has not been included as the dispersion results can be observed in Figure 89.

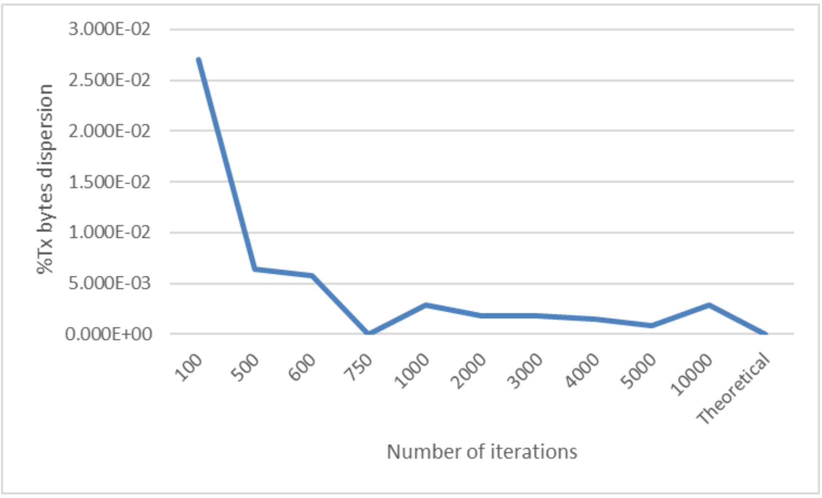


Figure 89. Transmitted bytes dispersion over 0.5 for different number of iterations. Case without 802.11g delays.

When looking at the results obtained with the number of transmitted bytes, it can be observed that the dispersion values are greater than the values when focusing on the time in Figure 88. This difference is due to the fact that the ADRR quantum is set in time units, so it adjusts the time used by each tenant. However, the algorithm does not limit the number of bytes transmitted directly but through the time quantum. If the time considered did not consider retransmissions, the transmitted bytes would coincide to the time one, as they would be directly connected. However, when retransmissions are considered, this is not the case.

Even though the dispersion observed in Figure 89 is greater than 1E-4, the convergence is reached at 500 as for ADRR the parameter that is optimized is the time, not the transmitted bytes.

1.1.2. ADRR with 802.11g delays

In order to discard differences in the convergence when 802.11g protocol delays are taken into account, it has been proven that the results obtained are quite similar.

1.1.2.1. Time convergence

The same parameters have been obtained and analyzed for the case of enabling 802.11g protocol delays. In Figure 90 it is plotted the time dispersion around 0.5.

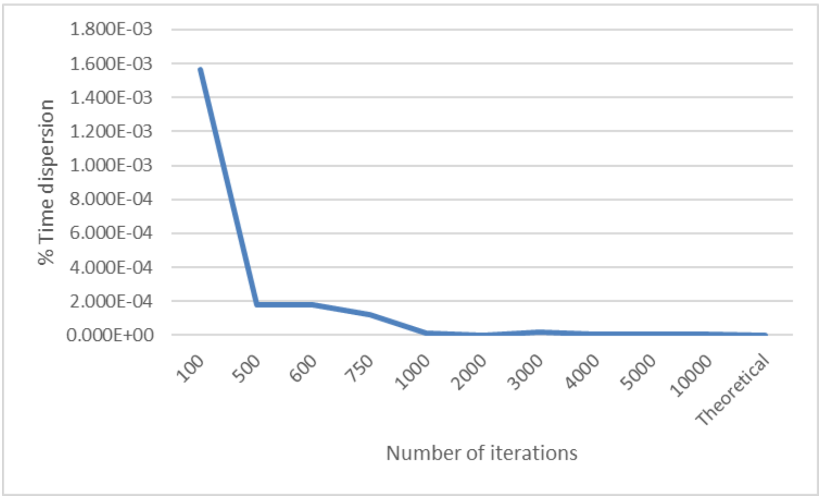


Figure 90. Time dispersion over 0.5 for different number of iterations. Case considering 802.11g delays

Comparing these results to the results in Figure 88, where not delays are considered, it can be seen that the dispersion obtained for each number of iterations have the same order. In addition, the dispersion decreases drastically when using 500 iterations as before. So, the use of 802.11g delays does not affect the convergence in time. In this case, it can be also stated that the convergence is reached at 500 iterations, as the dispersion is of the order of 1E-4.

1.1.2.2. Transmitted bytes convergence

It has also been checked the effect of 802.11g delays in the transmitted bytes convergence. Once again, the dispersion evolution depending on the number of iteration has been obtained.

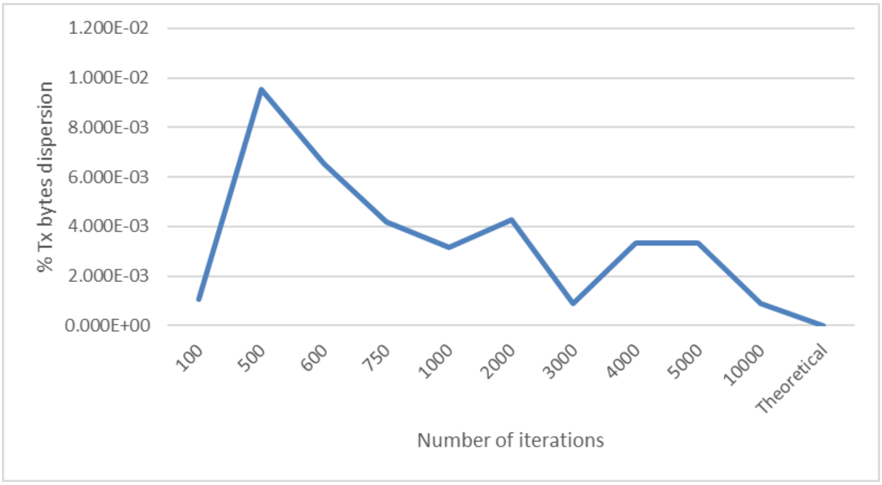


Figure 91. Transmitted bytes dispersion over 0.5 for different number of iterations. Case considering 802.11g delays

When Figure 89 and Figure 91 are compared, it can be seen that the results are very similar so the delays added by the 802.11g protocol do not affect the convergence study. In this case it is also observed that the dispersion values are greater than in the time analysis.

1.1.3. WDRR without 802.11g delays

As the simulator can perform the scheduling by using different algorithms, it is relevant to check the convergence when using these algorithms. In this section, the convergence is analyzed for WDRR algorithm using a system quantum of 1514 bytes. For ADRR, the time converged faster than for the transmitted bytes so, for this case, it is expected the contrary because DRR controls how many bytes are transmitted per iteration.

For this case, the delays added by 802.11g are not taken into account but, as also shown in the previous section, they do not affect the convergence study.

1.1.3.1. Time convergence

This time it has been studied the time convergence for WDRR using the same parameters as in the previous sections. The sharing percentage time dispersion around 0.5 is plotted in Figure 92.

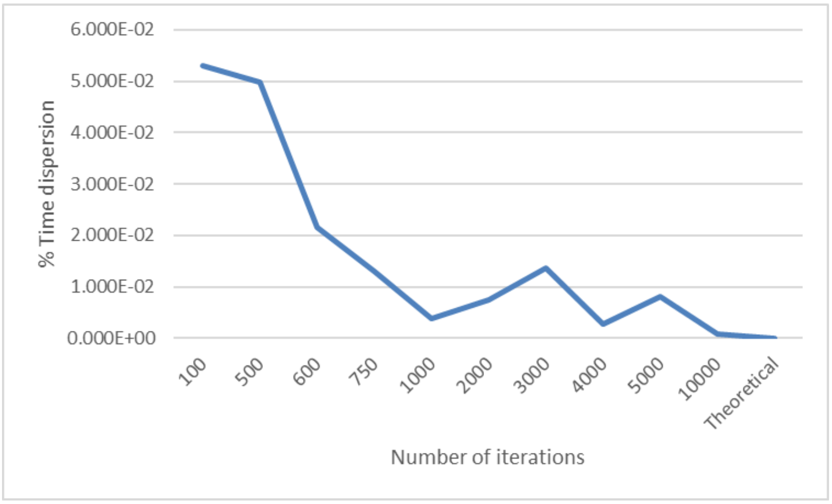


Figure 92. Time dispersion over 0.5 for different number of iterations. Case of WDRR without 802.11g delays

As expected, the convergence results for the WDRR algorithm for the case of the time are similar to the ones obtained for the transmitted bytes convergence of the ADRR. This is because in WDRR the quantum is set in bytes so what is controlled in each iteration is the number of bytes to be transmitted not the time for each tenant. As the transmission time is related to the number of bytes transmitted, it is indirectly controlled. The differences obtained are because for a packet transmission we only count how many bytes has the packet sent but for the time we consider all the time needed to successfully transmit the packet, including retransmissions.

1.1.3.2. Transmitted bytes convergence

Moreover, the convergence has been studied for the transmitted bytes for WDRR. The dispersion around 0.5 for the percentage of transmitted bytes by each tenant is showed in Figure 93.

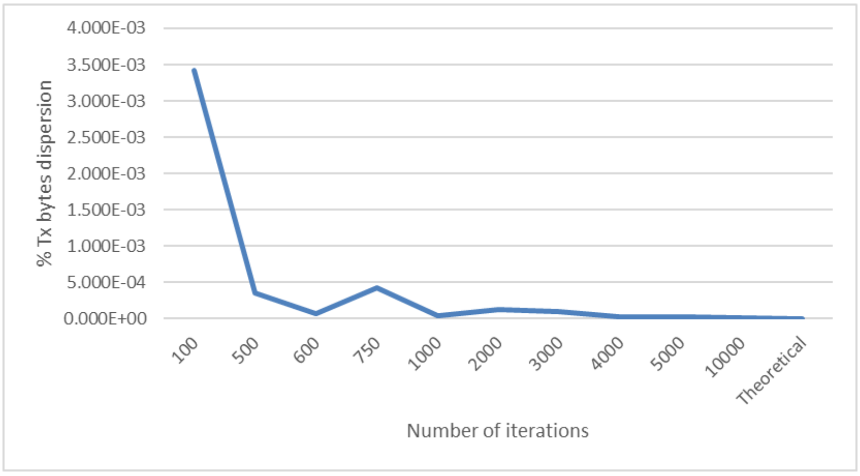


Figure 93. Transmitted bytes dispersion over 0.5 for different number of iterations. Case of WDRR without 802.11g delays

For this last case, the convergence behavior for the transmitted bytes is similar to the behavior obtained for the time convergence in ADRR in Figure 90. When comparing both figures, it is shown a high similarity, as in both cases the dispersion decreases really fast at 500 iterations. In this case the dispersion at 500 iterations is around 1E-4 so, according to our criteria, with that number of iterations is enough to converge.

Finally, it can be concluded that the convergence for WDRR and ADRR behaves equally for the quantum units (time for ADRR and transmitted bytes for WDRR).

1.1.4. ADRR without 802.11g delays and weights 80%-20%

In this section, it has been analyzed the convergence of ADRR when the resources are not shared equally between the both Tenants. The weights of Tenant 1 and Tenant 2 have been set to 80% and 20%, respectively. In this case, no 802.11g delays have been taken into account. The study has just been performed for the time convergence, as during this convergence study, it has been seen that ADRR adjusts the time.

1.1.4.1. Time convergence

It has been studied the time convergence using the same parameters as in the previous sections.

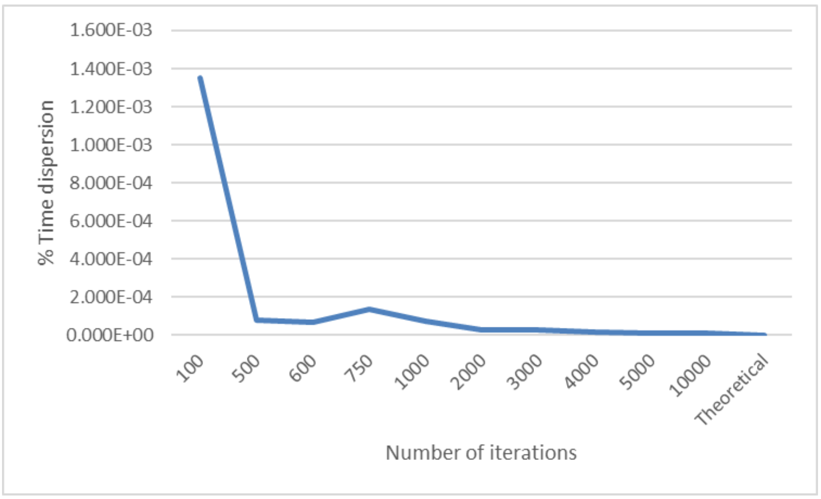


Figure 94. Time dispersion over 0.5 for different number of iterations. Case of WDRR without 802.11g delays

In the simulation results, it can be observed that the convergence of time when different weights are applied to each Tenant is the same as in the equal weights case. In Figure 94, it is shown that the dispersion decreases drastically to 1E-5 at 500 iterations like in Figure 88, where the weights were set to 50%-50%. As our dispersion threshold is 1E-4, the minimum number of iterations needed would be 500 as in the previous cases.

With this last analysis, it can be stated that the convergence is not affected by the weights used for each of the Tenants.

1.2. Deficit counter adjustment: theoretical vs real packet time.

In this study, it is discussed the difference between adjusting the deficit counter (DC) using the real time in which packets are transmitted or using a theoretical time in the ADRR algorithm.

The first approach of the ADRR algorithm for the simulator assumed that the DC was reduced using the real time in which the packet was transmitted, as in equation (**107**). In this way, the exact time that the tenant had spent transmitting the packet was taken into account, considering the data rate in which was transmitted and the retransmissions.

$$DC_{Tenant\ i} = DC_{Tenant\ i} - tp_{real} \tag{107}$$

Although this solution is the fairer, when thinking of the real implementation of hypervisor it is not a feasible option. The reason of this is that in the real hypervisor computing the real time in which a packet is transmitted involves having to read ACK packets. This would increase the complexity of the algorithm, introducing latency to the system when serving packets. Because of this, it was decided to change the initial assumption and use the initial expected time as in equation(**108**).

$$DC_{Tenant\ i} = DC_{Tenant\ i} - tp_{theoretical} \tag{108}$$

Moreover, in the first approach the expected time was always computed using the most probable data rate without considering retransmissions so the expected time was always the same. In our case, the expected time was always considering the 54Mbps rate. After deciding to use the expected time, it was opted to upgrade the computation of the expected theoretical time, to include the possibility of different data rates in its computation. This is performed by using the function chooseRb().

With the upgrade, the expected time calculation uses the average packet length and a randomly chosen data rate, which considers the probability of the different data rates. In chooseRb(), it is generated a random number between 0 and 1 and this is related to a certain data rate according to the probabilities of the different data rates. Therefore, different data rates are considered but retransmissions are not taken into account, as its probability is really low. In this way, the DC reduction, in average, is nearer the first approach, as the expected packet transmission time is not always the same and has similar statistical behavior to the real packet transmission time.

Considering this, it is required to check the differences between the two approaches. The main difference between them is reflected in the convergence study, as their behavior differs. In order to compare both approaches, it has been considered the simulation conditions in Table 47. The justification of the field empty queue is provided later in this ANNEX.

Number of WTP	1
Tenants	2
Tenant 1 SLA	0.7
Tenant 2 SLA	0.3
Packet lengths	1514 Bytes (both)
System quantum	2ms
802.11g delays	ON
Empty queue time	100us
Traffic generation mode	Fixed
Simulation time	60 s

Table 47. Simulation conditions

The first simulations have been performed fixing a single data transmission rate to 54Mbps. In this way, the theoretical time and the real time will just differ because of retransmissions. From the simulation results, it has been computed the average percentage of used time and the average percentage of transmitted bytes of each tenant when using the real packet transmission time (tp real) and expected packet transmission time (tp theoretical). After this it has been computed dispersion around the desired percentage (0.7 for tenant 1 and 0.3 for tenant 2) and it has been plotted. The dispersion graphics are equal for both tenants, as the dispersion is computed taking the absolute value as in previous studies.

Figure 95 and Figure 97 show the dispersion focusing on the used time percentage, which are similar. It must be pointed out that in the convergence study, when using real transmission time, the simulations were performed focusing on iterations, so long time dispersion like in this case was not performed. Comparing both graphics, it can be seen how when using tp real, the value of dispersion is stabilized a little later than in the case of tp theoretical. In Figure 96 and Figure 98 it is plotted the dispersion when paying attention to the transmitted bytes. Once again, the graphics behaviors are extremely similar, being more homogeneous in the case when tp theoretical is used. In all figures, a logarithmical y axis has been used, to clearly show the results.

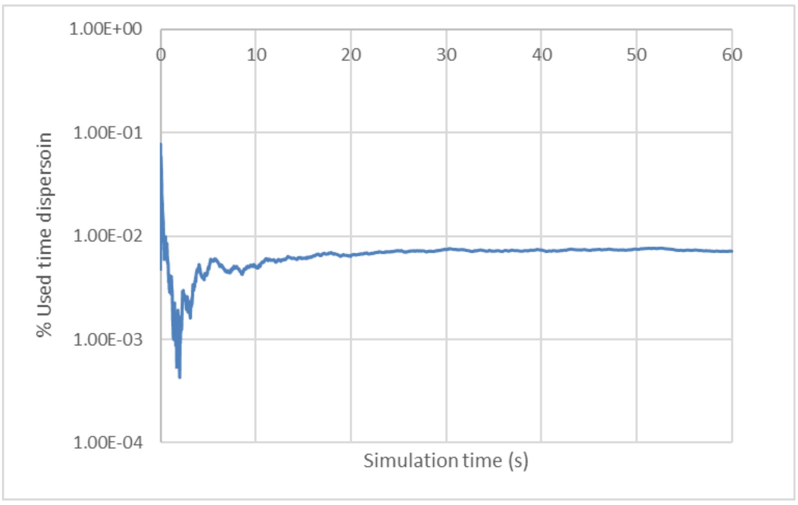


Figure 95. Used time dispersion around expected weight for the case of 54Mbps and tp real

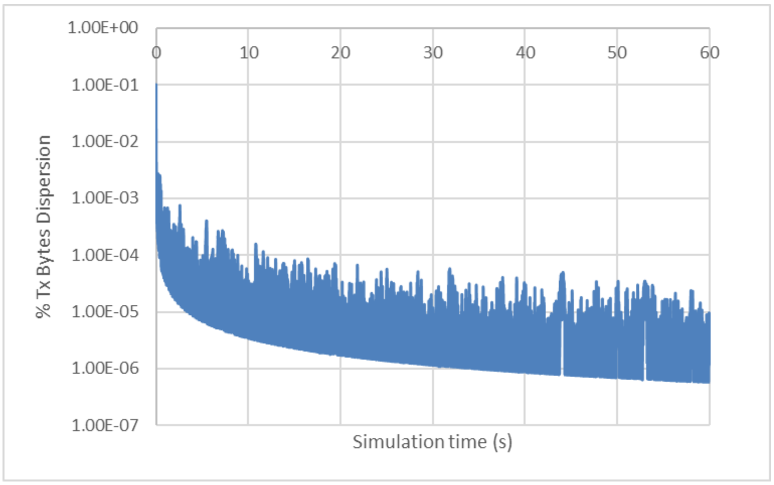


Figure 96 Transmitted bytes dispersion around expected weight for the case of 54Mbps and tp real

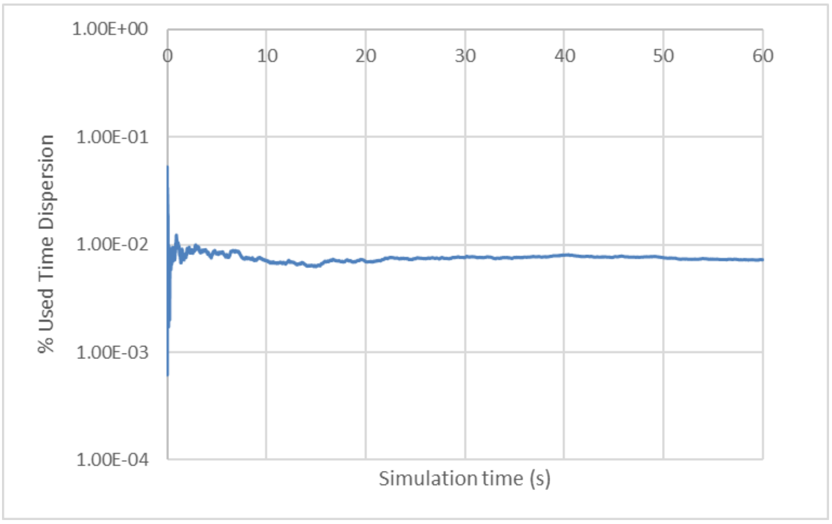


Figure 97. Used time dispersion around expected weight for the case of 54Mbps and tp theoretical

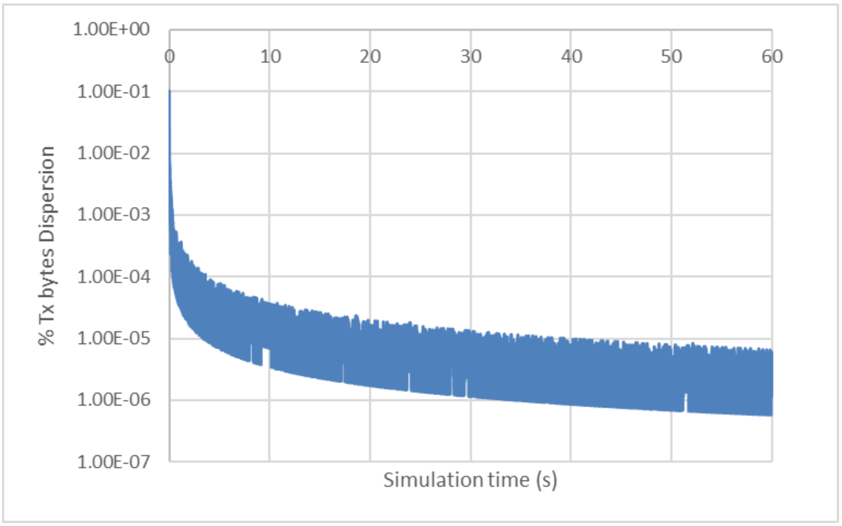


Figure 98. Transmitted bytes dispersion around expected weight for the case of 54Mbps and tp theoretical

Secondly, it has been performed the same simulations but allowing the WTP to transmit at different data rates, specified at Table 45.

In this simulation, big differences are found when comparing the performances in terms on convergence. In Figure 99 and, Figure 101 it is represented the dispersion in the percentage used time by a tenant. It can be observed how in the case of using the tp real, the dispersion reaches smaller values than when using tp theoretical. The same happens in Figure 100 and Figure 102 for the transmitted bytes dispersion. This is caused by the differences between the tp theoretical and tp real as different data rates are possible. When considering the real time, the algorithm is being more accurate in the time sharing between the different tenants. However, when using the theoretical time, the accuracy is lost in both the time and transmitted bytes sharing.

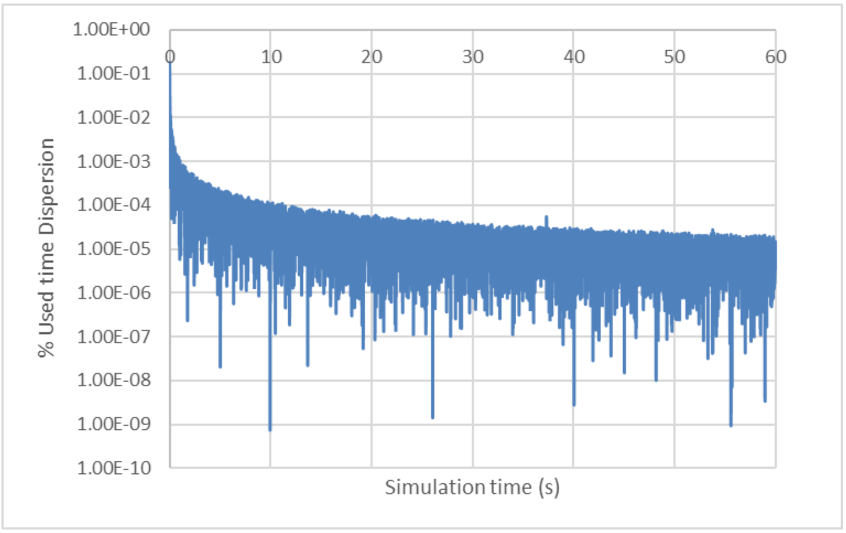


Figure 99. Used time dispersion around expected weight for the case of random data rate and tp real

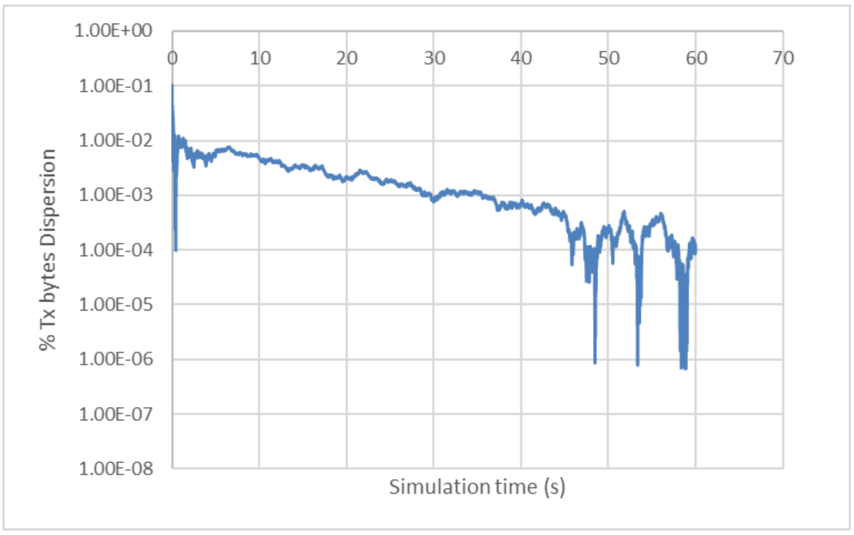


Figure 100. Transmitted bytes dispersion around expected weight for the case of random data rate and tp real

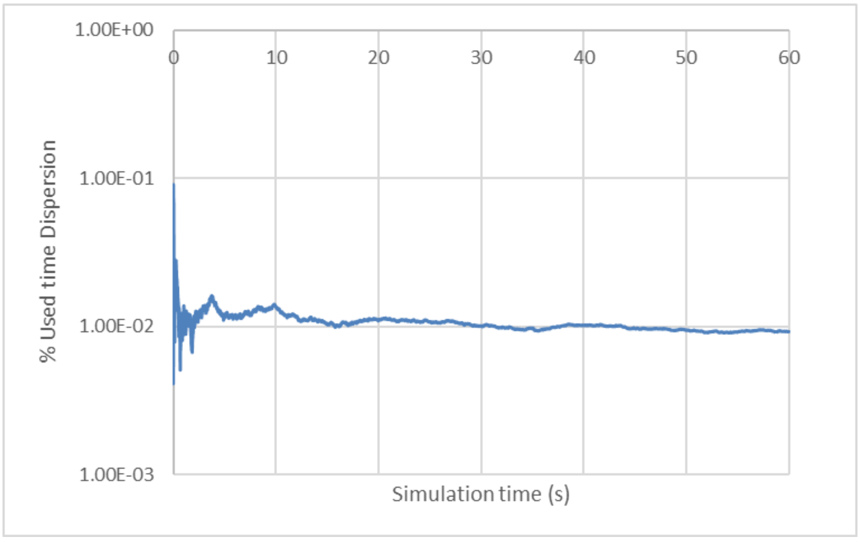


Figure 101. Used time dispersion around expected weight for the case of random data rate and tp theoretical

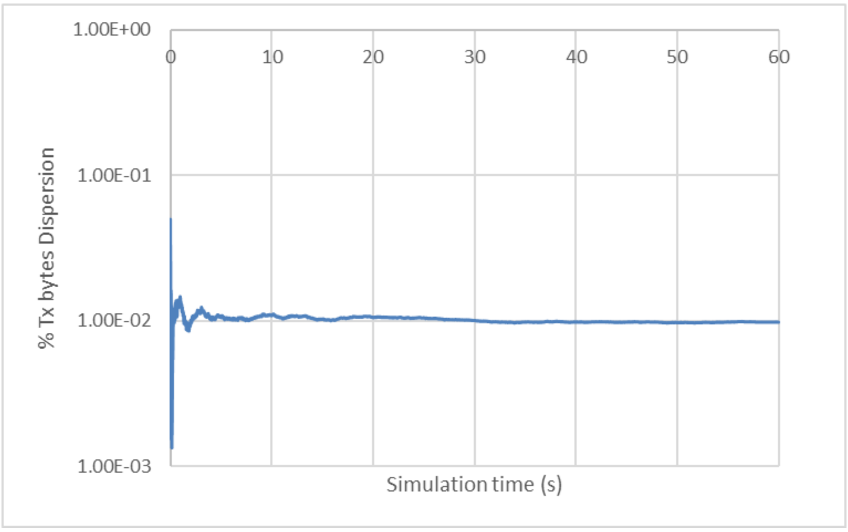


Figure 102. Transmitted bytes dispersion around expected weight for the case of random data rate and tp theoretical

In conclusion, when using different data rates in a WTP, the resource sharing is more accurate and fairer when using the real transmission time in the DC adjustment. Although this, it is needed to use the expected or theoretical packet transmission time because of implementation issues for the real hypervisor so a dispersion of 1E-2 will be achieved focusing in both used time and transmitted bytes. Giving this, the time convergence has been analyzed in the following section.

1.3. Time Convergence Study

A key factor when studying the performance of ViRANsim simulator is to determine the amount of time needed to obtain reliable results, what means to study the convergence of the system.

This convergence study focuses on time convergence while in the previous studies the focus was on the number of iterations needed to converge. The reason of this is that for the EmPOWER testbed is would be much useful to know the convergence in time units, to set a coherent simulation time. Moreover, in the study of this document, the expected theoretical packet transmission time is subtracted from the DC instead of the real time, as in equation (**107**).

For all the tests performed, the simulation scenario considers two tenants with a SLA of the 60% for Tenant 1 and 40% for Tenant 2 and a single WTP. Both tenants just transmit packets of 1514 Bytes and the system quantum is set to 2ms. Moreover, the queues are initially empty. Different generators and transmission rates have been used in the following sections to study how the convergence is affected.

1.3.1. Fixed traffic generation rate and single transmission rate

The first convergence test has been performed when the WTP just works at a single nominal transmission rate of 54Mbps and tenants generate traffic in a fixed mode. The traffic generation rate of each of the tenants is set considering the weight of the tenant, the transmission rate, the 802.11g delays and the effect of retransmissions.

To study the convergence, it has been computed the average percentage of used time and the percentage of transmitted bytes for both tenants. Afterwards, it has been calculated the deviation around the expected percentages (60% for tenant 1 and 40% for tenant 2).

In Figure 103 it is plotted the time deviation during all the simulation, that last 60s. As we are working with small values, it has been generated Figure 104, which represents the same values but using a logarithmical scale in the vertical axis. In that figure, it is possible to observe in more detail that the deviation converges to a value around 0.5E-3, which is achieved in 10 seconds.

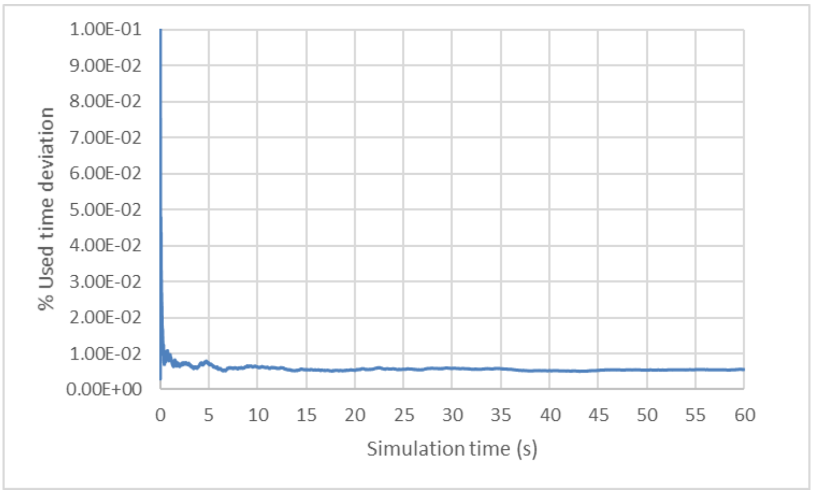


Figure 103. Time deviation during the simulation time.

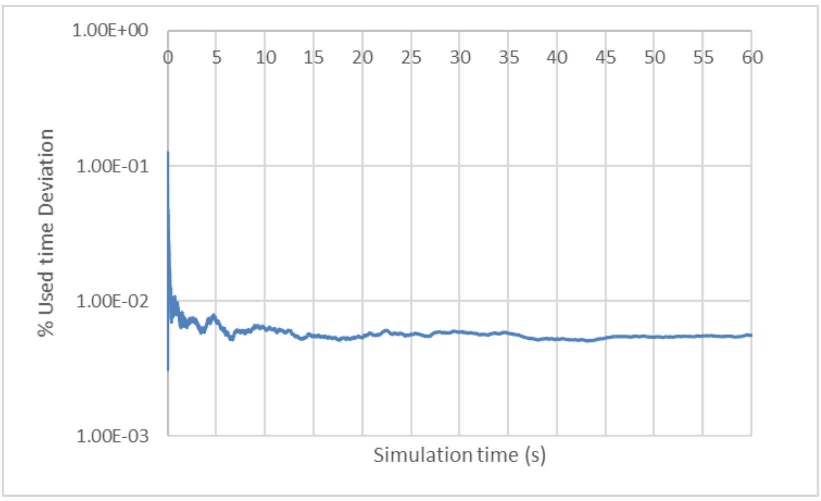


Figure 104. Time deviation with vertical axis in logarithmical scale

Figure 104 shows the deviation in the transmitted bytes using a logarithmical scale in the vertical axis again. It can be observed that the deviation in the transmitted bytes reaches smaller values than the deviation in time and it keeps decreasing during time.

It must be pointed out that in the, previous convergence study in section 1.1, when the real time was discounted from the DC instead of the theoretical, it was obtained the contrary; the time deviation was always decreasing while the transmitted bytes didn't. So, what makes that the time deviation does not keep decreasing is related to the fact that the algorithm is not exactly subtracting the time used by each tenant from the DC. This is because the theoretical time used does not take into account retransmissions, so the theoretical and the real used time can be different. However, as we are just working with a single transmission rate and a single packet length, the number of bytes expected to be transmitted into a certain theoretical time will coincide with the transmitted bytes into a real time, as we are just counting correctly transmitted packets. That means that the sent bytes in retransmissions are not considered.

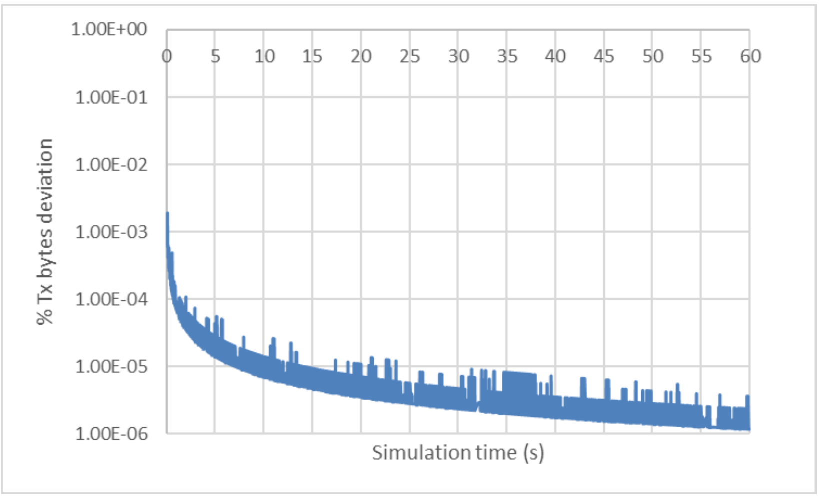


Figure 105. Transmitted bytes deviation with vertical axis in logarithmical scale.

From this section, it can be concluded that the time needed to converge is approximately 10 seconds and that with the theoretical time modification in the algorithm, what the algorithm adjusts is the percentage of transmitted bytes.

1.3.2. Fixed traffic generation rate and random transmission rate

In this section, the convergence is studied when the tenants are using a fixed generator as before but the WTP can transmit at different transmission rates. The nominal transmission rates (without considering 802.11g delays) with its related probabilities can be found in Table 45.

The fixed generation rate is computed as concluded in the traffic generation section, where it is considered an average of the different rates considering the delays and the related data rates probabilities. Once again, the same graphics have been generated.

In Figure 106, it can be observed how the deviation is slightly greater than in Figure 103, as in this case the deviation converges to 0.6E-3. However, in Figure 107 it is observed that the deviation in the transmitted bytes is no longer decreasing during all the simulation but it converges to 0.6E-3. This change is due to the possibility of a packet being transmitted in different rates. To compute the theoretical time, it is selected a data rate randomly according to the Rb probabilities. Nevertheless, the real data rate can be totally different which can introduce big differences in the number of packets sent during an iteration.

When looking at the time deviation, it can be observed that the deviation is stabilized at 20 seconds but looking at the transmitted bytes, the deviation stabilizes at 5 seconds.

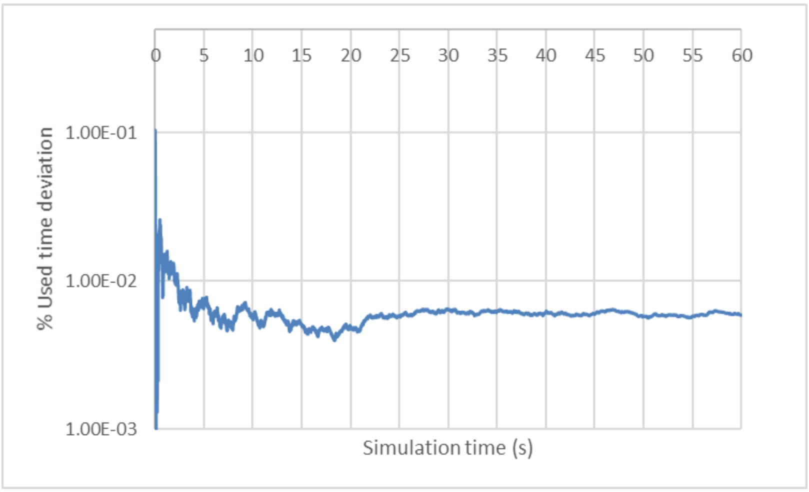


Figure 106. Time deviation with logarithmical vertical axis for the case of Rb random and fixed generation rate.

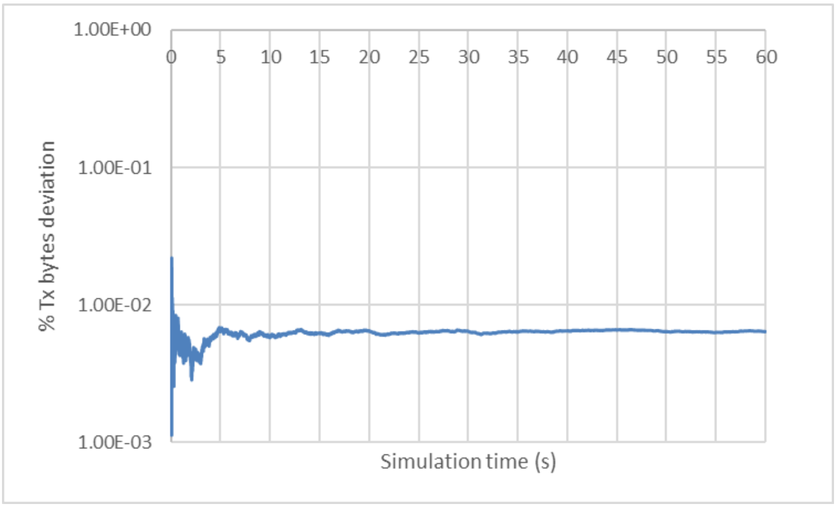


Figure 107. Transmitted bytes deviation with logarithmical vertical axis for the case of Rb random and fixed generation rate.

1.3.3. Gaussian traffic generation rate and random transmission rate

In this section, the transmission rate is random as in the previous section and the traffic generation rate is changed following a Gaussian distribution every 1 second. The Gaussian distribution has the following parameters as a mean and standard deviation, where $C_{average}$ is the average capacity that the WTP can provide, considering 802.11g delays. The function selected to generate the traffic is traffic_generation_rate_Gaussian_Shifted(), previously explained.

$$\mu = w_{SLA} \cdot C_{average} \tag{109}$$

$$\sigma = 0.15 \cdot w_{SLA} \cdot C_{average} \tag{110}$$

This distribution has been selected as it approaches well a real scenario, where different traffic generation rates can happen around the capacity average. In this case the standard deviation of the Gaussian distribution has been set to 0.15 which could be modified in the case of wanting to emulate more critical or relaxed distributions. This distribution is the one that will be used in future studies of other parts of the algorithm, which makes it interesting to be analyzed.

The same convergence than in previous sections have been performed.

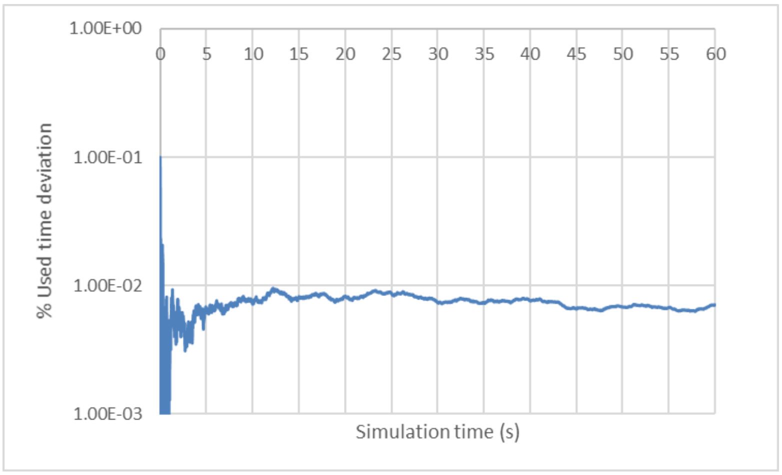


Figure 108. Time deviation with logarithmical vertical axis for the case of Rb random and Gaussian generation rate.

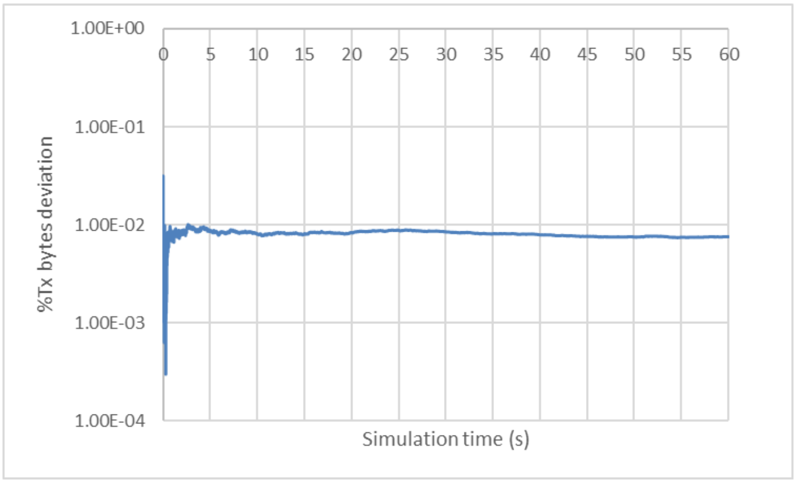


Figure 109. Transmitted bytes deviation with logarithmical vertical axis for the case of Rb random and gaussian generation rate.

In Figure 108, it can be observed how the time deviation is higher than in Figure 104 but similar to Figure 106. This is because the only thing that changes in comparison to Figure 104 is the traffic generation mode so, once the queue has packets, the performance is the same in both cases. For the transmitted bytes deviation, similar results to Figure 107 are found.

Looking at Figure 108 it can be stated that the convergence is reached approximately at 10 seconds, where the deviation is stabilized.

1.3.4. Full Queue

Another test performed is the same as in the previous section but initializing the system with the queue full, to isolate the algorithm convergence analysis. Once again, the same graphics have been generated.

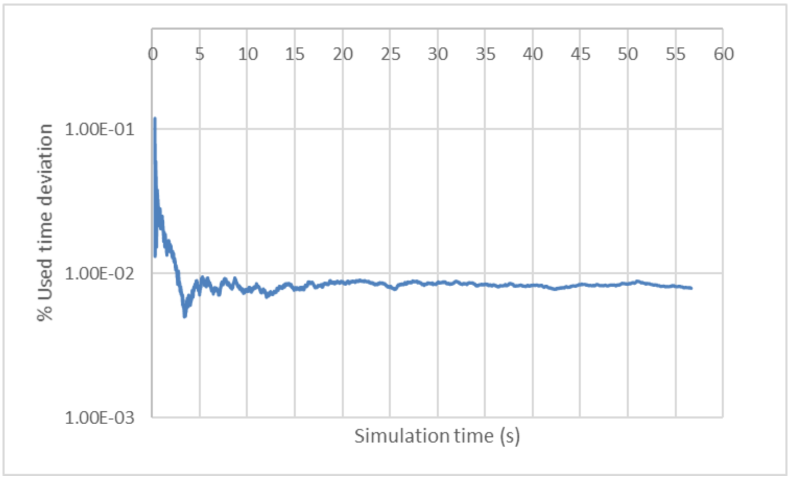


Figure 110. Time deviation with logarithmical vertical axis for the case of Rb random, fixed generation rate and queue initially full.

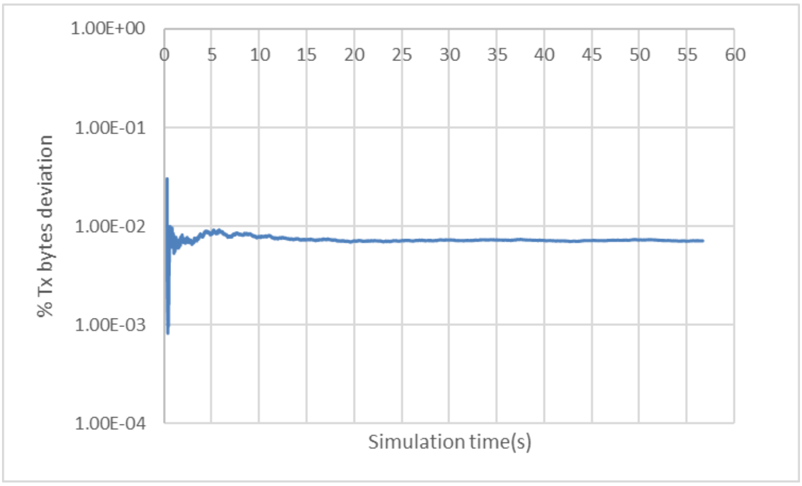


Figure 111. Transmitted bytes deviation with logarithmical vertical axis for the case of Rb random, fixed generation rate and queue initially full.

As it can be observed in both Figure 110 and Figure 111, in comparison to Figure 108 and Figure 109 the algorithm converges faster. For both transmitted bytes and time parameters, the system converges in 5s. Moreover, in both cases the dispersion converges to approximately the same value.

1.4. Traffic Generation Analysis

In this study, a description of how the traffic is generated in the hypervisor simulator is presented. One requirement for the simulator was to be able to control the traffic generation of the different tenants but also to simulate the real traffic expected from them. In order to do so, different traffic generators have been designed with this purpose.

All the designed simulators work using the same principle of operation, which will be explained in the first part of this section. Moreover, different issues corresponding to the generators design as well as its results will be discussed.

1.4.1. Fixed Traffic Generator Rate

The fixed traffic generator rate sets a rate at the beginning of the simulation and that rate is not modified again during the simulation.

An important issue related to this generator is how to fix the traffic rate. The traffic generation rate of each of the tenants is related to the traffic that a given WTP can manage. This is why for each WTP it is computed which is its capacity in bits. As a first approach, it was supposed that a WTP was just serving at a single Rb, so the wtp capacity in a system quantum (Q_S) has the expression in equation (**111**) :

$$Capacity_{WTP}[bits] = Rb\ [bps] \cdot Q_s[s] \tag{111}$$

According to this, each of the tenants in a WTP could transmit the following amount of bits, considering its weight in the WTP, as it is specified in equation (**112**):

$$Capacity_{Tenant\ i}[bits] = w_i \cdot Capacity_{WTP}[bits] \tag{112}$$

So the traffic generation rate (G_T) for a given tenant in a WTP follows equation (**113**), considering that we are working in a system quantum interval of time.

$$G_{T\,Tenant\ i}[bps] = \frac{Capacity_{Tenant\ i}[bits]}{Q_s[s]} \tag{113}$$

If expressions the previous equations are written together, the traffic generation rate (G_T) of a given tenant in a WTP could also be written as in equation (**114**).

$$G_{T\,Tenant\ i}[bps] = \frac{Capacity_{Tenant\ i}[bits]}{Q_s[s]} = \frac{w_i \cdot Capacity_{WTP}[bits]}{Q_s[s]} = \frac{w_i \cdot Rb\ [bps] \cdot SQ[s]}{Q_s[s]} = w_i \cdot Rb\ [bps] \tag{114}$$

In the simulator, it is more convenient to consider the capacity of the WTP so we will work with equations (**111**), (**112**) and (**113**) but for checking it will be also useful equation (**114**).

In order to evaluate the performance of the proposed algorithm a simulation was performed. The conditions for the simulation are specified in Table 48.

Simulation Conditions	
Rb	54Mbps
Probability of error	0.1
Generation rate	Fixed mode
802.11g delays	ON
System Quantum	2 ms
Retransmissions	ON
Simulation time	60s
Number of WTP	1
Number of tenants	2
Weight Tenant 1	0.6
Weight Tenant 2	0.4
Packet length (Both tenants)	1514 Bytes
Time empty queue	100us

Table 48. Simulation conditions for fixed generator

With the specified conditions, the traffic generation parameters expected are showed in **Error! Reference source not found.** and the equations above explained are used to o btain its values.

Traffic generation parameters	
Rb	54Mbps
System Quantum	2 ms
WTP capacity	108000 bits
Tenant 1 Capacity (w=0.6)	64800 bits
Tenant 2 Capacity (w=0.4)	43200 bits
Tenant 1 Gt	32.4 Mbps
Tenant 2 Gt	21.6 Mbps

Table 49. Expected traffic generation parameters

The results of the simulations can be observed in Table 50.

	Tenant 1	Tenant 2
Total Time (us)	59998936	59998936
Utilized Time (us)	36021614	23978659.48
Transmitted bytes	155106272	103403172
Number of packets generated	160499	106999
Number of packets transmitted	102448	68298
BW (Mbps)	20.68	13.78
Gt (Mbps)	32.399	21.599
Number of packets in queue	58051	38701

Table 50. Simulation results with fixed generator and single Rb

As it can be seen in the simulation results, the generation rates (Gt) are really near the expected ones in Table 49. However, the effective throughput[7] (referred as BW) provided

[7] *The effective bandwidth is the real data rate provided to users considering retransmissions, packet transmission delays, the total simulation time and the bytes transmitted.*

to users is smaller than the traffic generation rate. Moreover, the number of packets in the queue at the end of the simulation is really high. All this informs us that the system is not capable of managing the traffic generated, as the traffic generation rates are too high. Figure 112 shows how the packets in queue grow as a function of the simulation time. It can be observed that the packets in queue for Tenant 1 are greater than the packets in queue for tenant 2 as the traffic generation rate is greater for Tenant 1.

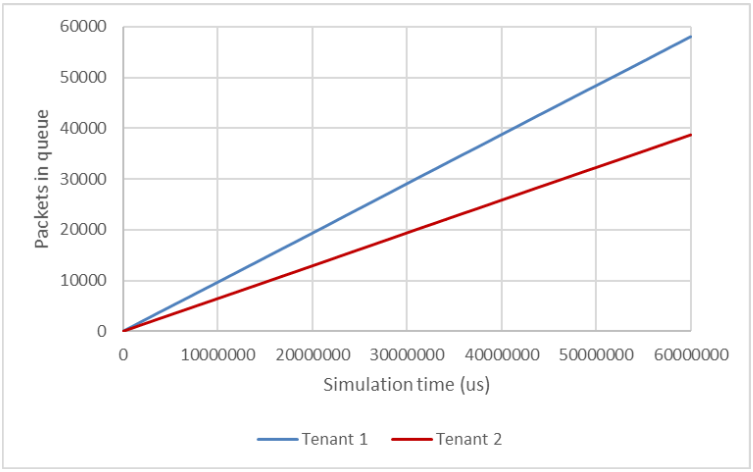


Figure 112. Packets in queue during time

One of the reasons of the growth of the queue is that the system does not transmit packets at a single nominal data rate (54Mbps in the simulation) and that 802.11g Wi-Fi protocol introduces delays that affects to the data rate. This is why it has been designed a mechanism to incorporate the effect of these delays to the computation of the traffic generation rate in which packets have to be sent.

For each tenant in each WTP it will be necessary to compute a **delay compensation factor** considering the average packet length and all the nominal data rates. The delay compensation factor (r_i) is the relation of the delays in front of the time required to send a packet, at a certain nominal rate i. This factor will be computed for each of the tenants in each WTP as it depends on the packet length as well as on the effective data rate. Equation (**115**) shows how to compute it:

$$r_i = \frac{time_{delays_{Rb_i}}[s]}{time_{total\ packet\ transmission_{Rb_i}}[s]} \tag{115}$$

Considering a general case, where in a WTP N different modulations schemes are possible and, in consequence, different nominal data rates (Rb_i) exist with different probabilities (p_i). So, for the N nominal data rates it will be obtained its system capacity following in equation (**116**).

$$Capacity_{WTP_{Rbi}}[bits] = Rb_i\ [bps] \cdot SQ[s] \quad for\ i\ in\ 1..N \tag{116}$$

For each WTP and tenant equation (**117**) computes the tenant capacity in the WTP:

$$Capacity_{Tenant\ k_{Rb_i}}[bits] = w_k \cdot Capacity_{WTP_{Rbi}}[bits] \cdot (1 - r_i) \tag{117}$$

After this it is possible to compute the traffic generation rate of tenant k through equation.

$$G_{T\,Tenant\ k} = \frac{\sum_{i=0}^{N} Capacity_{Tenant\ K_{Rb_i}} \cdot p_i\ [bits]}{SQ[s]} \qquad (118)$$

Working with equation (**118**), it can be obtained equation.(**119**).

$$G_{T\,Tenant\ k} = \frac{\sum_{i=0}^{N} Capacity_{Tenant\ K_{Rb_i}} \cdot p_i}{SQ} = \frac{\sum_{i=0}^{N} w_k \cdot Capacity_{WTP_{Rbi}} \cdot p_i \cdot (1-r_i)}{SQ}$$
$$= \frac{w_k \cdot \sum_{i=0}^{N} Rb_i\ [bps] \cdot SQ \cdot p_i \cdot (1-r_i)}{SQ} = w_k \cdot \sum_{i=0}^{N} Rb_i\ [bps] \cdot p_i \cdot (1-r_i) = w_k \cdot Rb_{effective_{average}} \qquad (119)$$

Where the average effective data rate can be defined like in equation (**120**).

$$Rb_{effective_{average}} = \sum_{i=0}^{N} Rb_i\ [bps] \cdot p_i \cdot (1-r_i) \qquad (120)$$

The delay compensation has been included in the simulator and tested initially with a single Rb. In order to do so, the same conditions as before were taken into account. Considering the conditions of Table 48 and the traffic compensation factor, the expected parameters are the ones in Table 51.

Traffic generation parameters	
Rb	54Mbps
System Quantum	2 ms
Packet Length	1514 Bytes
WTP capacity	108000 bits
r delay compensation factor	0.291
Tenant 1 Capacity (w=0.6)	45943.2 bits
Tenant 2 Capacity (w=0.4)	30628.8 bits
Tenant 1 Gt	22.97 Mbps
Tenant 2 Gt	15.31 Mbps

Table 51. Expected traffic generation parameters with compensation of delays

The obtained results are resumed inTable 52.

	Tenant 1	**Tenant 2**
Total Time (us)	59999682.56	59999682.56
Utilized Time (us)	35987129.63	24013660.22
Transmitted bytes (us)	155086590	103391060
Number of packets generated	113790	75860
Number of packets transmitted	102435	68290
BW (Mbps)	20.6783214	13.7855476
Gt (Mbps)	22.97052953	15.31368635
Number of packets in queue	11355	7570

Table 52. Simulation results with fixed generator and single Rb and delays compensation.

With the last results, it is observed that the traffic generation rate is the expected so the system is generating the packets correctly. However, the effective bandwidth in which the packets are sent is still lower than the traffic generation rates generated. In fact, the BW has hardly changed. This produces that packets still accumulate in the queue. In Figure 113 it is represented how the number packets in the queue for Tenants 1 and 2 grows but with a smaller slope in comparison to Figure 112.

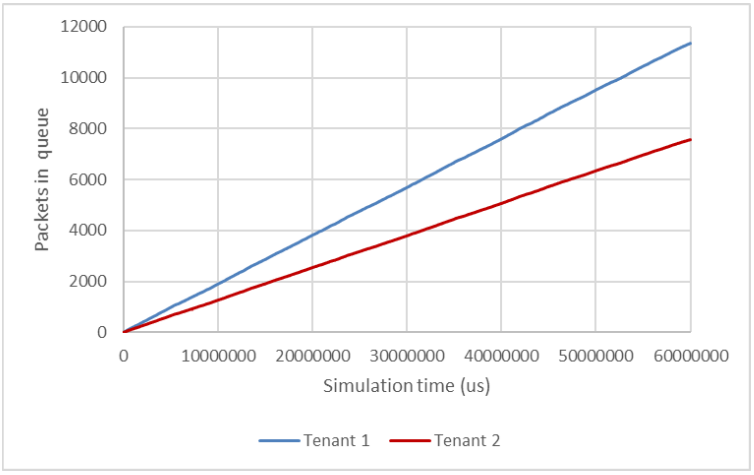


Figure 113. Packets in queue during time with 802.11g delays compensation

The reason why the queue continues growing is that retransmissions of packets are possible. As we are considering a situation where the 10% of the packet transmitted are erroneous, many retransmissions are possible. If the probability of error is reduced to the 0.1%, the number of packets in queue decreases drastically but the system is still not capable of managing them so the queue grows during the simulation. This can be observed in Figure 114, which is the result of a simulation performed with the same conditions as before but with a probability of error of 0.001.

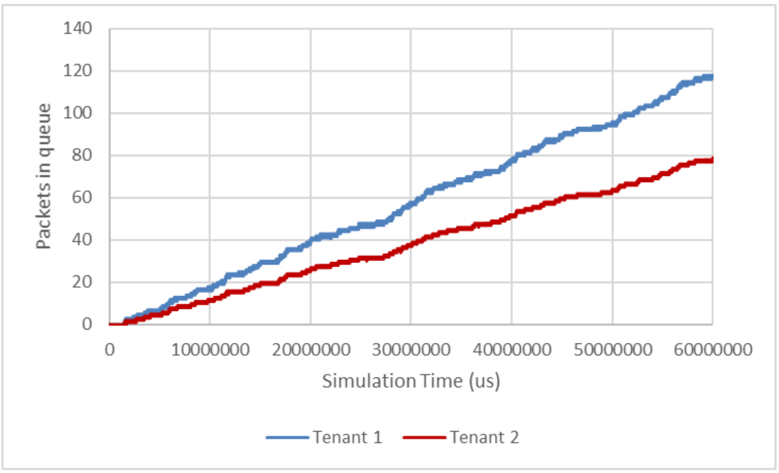


Figure 114. Packets in queue during time with 802.11g delays compensation and probability of error 0.001

In order to solve this issue, it is proposed to incorporate a **compensation factor for retransmissions** ($f_{retx,\ rbi}$), that will be different for each data rate. After doing some tests, it has been concluded that the best way to include the compensation factor is to multiply the generation rate by a factor slightly smaller than the probability to transmit a packet correctly. As it will be proven, it is important to set the compensation factor slightly smaller than the probability of success, since giving that a retransmission occurs, another retransmission could occur. The expressions used to obtain the generation rate of tenant k are equation (**121**), (**122**), (**123**), (**124**) and (**125**) which are similar to equations (**115**), (**116**), (**117**), (**118**), (**119**) and (**120**) but considering $f_{retx,\ rbi}$.

$$Capacity_{Tenant\ k_{Rb_i}}[bits] = w_k \cdot Capacity_{WTP_{Rbi}}[bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}} \quad (121)$$

$$G_{T_{Tenant\ k}} = \frac{\sum_{i=0}^{N} Capacity_{Tenant\ K_{Rb_i}} \cdot p_i\ [bits]}{SQ[s]} \quad (122)$$

$$G_{T_{Tenant\ k}} = \frac{\sum_{i=0}^{N} w_k \cdot Rb_i[bps] \cdot SQ[s] \cdot p_i\ [bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}}}{SQ[s]} = w_k \cdot \sum_{i=0}^{N} Rb_i[bps] \cdot p_i\ [bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}} \quad (123)$$

$$G_{T_{Tenant\ k}} = w_k \cdot Rb_{effective_{average}} \quad (124)$$

In this case the average effective data rate can be expressed like in equation **(21)**.

$$Rb_{effective_{average}} = \sum_{i=0}^{N} Rb_i\ [bps] \cdot p_i \cdot (1 - r_i) \cdot\ f_{retx_{rbi}} \quad (125)$$

In the following figures, it is shown the queue evolution when different retransmission compensation factors are tested for Tenant 1. For all the next figures, the probability of error has been set to 0.1, so the probability that a packet transmission is correct is 0.9.

The first approach was to set the retransmission compensation factor to 0.9 and the result can be seen inFigure 115. It can be observed how the packets in queue oscillate more than in previous simulations and the number of packets in queue is reduced in comparison to previous results. However, the queue keeps growing. In this case the traffic generation rate is 20.67 Mbps while the bandwidth is 20.66 Mbps. As the traffic generation rate is slightly greater than the bandwidth that the system can provide to Tenant 1, the queue grows. This is why it is necessary to adjust the retransmission compensation factor in the way that the generation traffic rate is a little bit smaller than 20.66 Mbps.

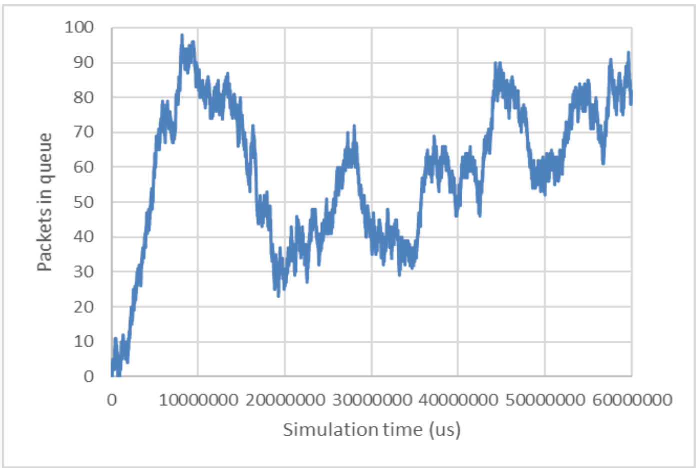


Figure 115. Packets in queue during time with retransmission and delays compensation. Retransmission compensation set to 0.9

The second approach was to set the compensation factor a little lower than 0.9, so a value of 0.89 was chosen. Given a data rate Rb_i with probability of retransmission p_{retx}, the retransmission compensation factor f_{retx} has been defined in equation(**126**).

$$f_{retx_{rbi}} = p_{retx_{rb_i}} - 0.01 \qquad (126)$$

The packets in queue for Tenant 1 are represented in Figure 116. It can be observed how the system is capable of processing the packets without accumulating them in the queue. Moreover, the traffic generation rate in this case is 20.44Mbps, which is really close to the bandwidth that Tenant 1 was managing when the queue was full.

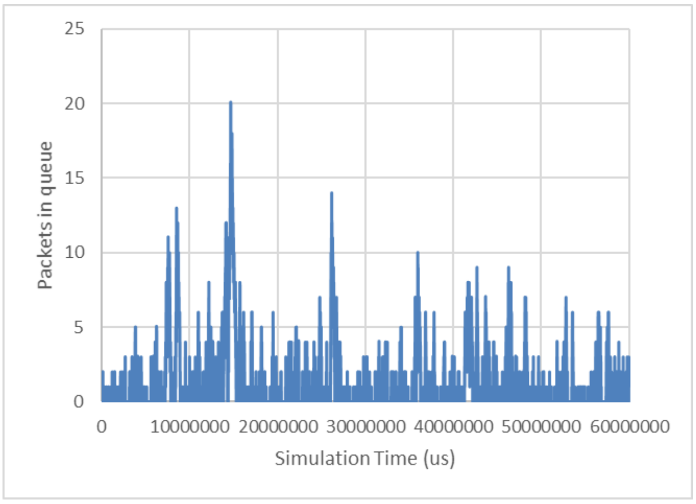


Figure 116. Packets in queue during time with retransmission and delays compensation. Retransmission compensation set to 0.89

The third approach was to set the compensation factor to 0.8. In Figure 117, it is possible to observe that the queue is nearly always empty. Setting the compensation factor to 0.8 do not maximize the traffic that we are introducing to the system. The traffic generation rate in this case is 18.38 Mbps while when no compensation factor the bandwidth to Tenant 1 was 20.68Mbps, which is the maximum generation rate that could afford the system.

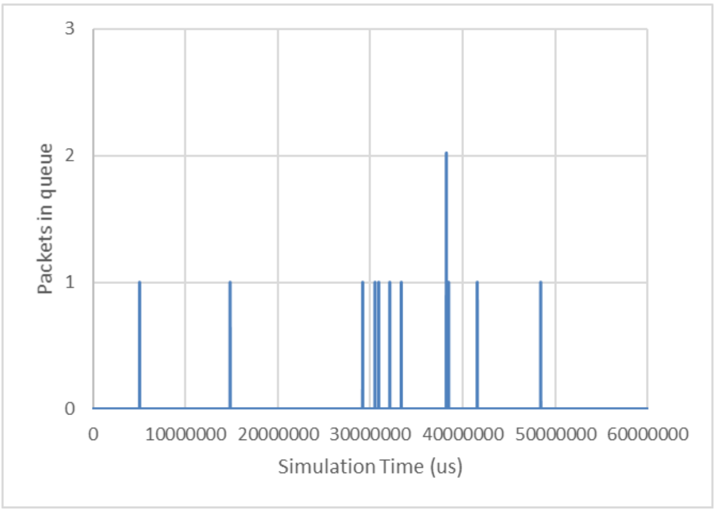


Figure 117. Packets in queue during time with retransmission and delays compensation. Retransmission compensation set to 0.8

In conclusion, it is necessary to set the compensation factor a little bit smaller than the probability of sending correctly the packet but without setting it too small so that it is possible to maximize the traffic we are sending to the system. So, from the tested retransmission compensation factors, the most appropriate would be to set the factor to 0.89.

Taking into account the retransmission and delays compensation factors, it has also been tested the performance of the system when different data rates are available in the WTP. Each data rate has a certain probability and a probability of success, which can be found in Table 45.

The simulation conditions when testing the performance of the system with different data rates available in the WTP can be found in Table 53.

Simulation Conditions	
Rb	54Mbps, 48 Mbps, 24 Mbps, 12 Mbps, 6 Mbps
Generation rate	Fixed mode
802.11g delays	ON
System Quantum	2ms
Retransmissions	ON
Simulation time	60s
Number of WTP	1
Number of tenants	2
Weight Tenant 1	0.6
Weight Tenant 2	0.4
Packet length (Both tenants)	1514 Bytes
Time empty queue	100us

Table 53. Simulation conditions for multi-data rate test with fixed generator

Table 54 contains the parameters needed to compute the traffic generation rate of Tenants 1 use, when using the values in Table 45, related to the data rates and its probabilities.

Transmission Rate	System Capacity (bits)	r_i	$f_{rtx,\,i}$	Capacity Tenant 1 (bits)	Capacity Tenant 2 (bits)
54 Mbps	108000	0.291	0.89	40889.448	27259.632
48 Mbps	96000	0.268	0.94	39633.408	26422.272
24 Mbps	48000	0.158	0.97	23522.112	15681.408
12 Mbps	24000	0.089	0.98	12856.032	8570.688
6 Mbps	12000	0.051	0.989	6757.6392	4505.0928

Table 54. Traffic generation rate parameters for multi-data rate simulation

The capacities of each of the tenants in the last two columns have been computed through the following equation:

$$Capacity_{Tenant\ k_{Rb_i}}[bits] = w_k \cdot Capacity_{WTP_{Rbi}}[bits] \cdot (1 - r_i) \cdot f_{retx_{rbi}} \qquad (127)$$

By using the values in Table 54, it is possible to compute the expected generation rate for Tenant 1 and 2.

$$G_{T\,Tenant\,1} = \frac{\sum_{i=0}^{N} Capacity_{Tenant\,1_{Rb_i}} \cdot prob(Rb_i)\ [bits]}{SQ[s]} = 19.19\ Mbps \qquad (128)$$

$$G_{T\,Tenant\,2} = \frac{\sum_{i=0}^{N} Capacity_{Tenant\,2_{Rb_i}} \cdot prob(Rb_i)\ [bits]}{SQ[s]} = 12.79 Mbps \qquad (129)$$

The results obtained through simulation are the ones in the following table.

	Tenant 1	**Tenant 2**
Total Time (us)	59998610.1	59998610.1
Utilized Time (us)	35556893.3	24443612.1
Transmitted bytes (us)	126898938	87226082
Number of packets generated	95037	63358
Number of packets transmitted	83817	57613
BW (Mbps)	16.9202504	11.6304137
Gt (Mbps)	19.1852468	12.7901646
Number of packets in queue	11220	5745

Table 55. Simulation results for multi-data rate test with fixed generator

In this case, it can be observed how the average bandwidth given to users is lower than the traffic generation rate. This growth in the queue size can be also observed in Figure 118. The queue grows because the fact that there exist data rates lower than the traffic generation rate, which makes the queue to grow. This is because the time to transmit a packet will be greater when sending it with a lower data rate, which will also produce that the generator generates more packets in the following iteration.

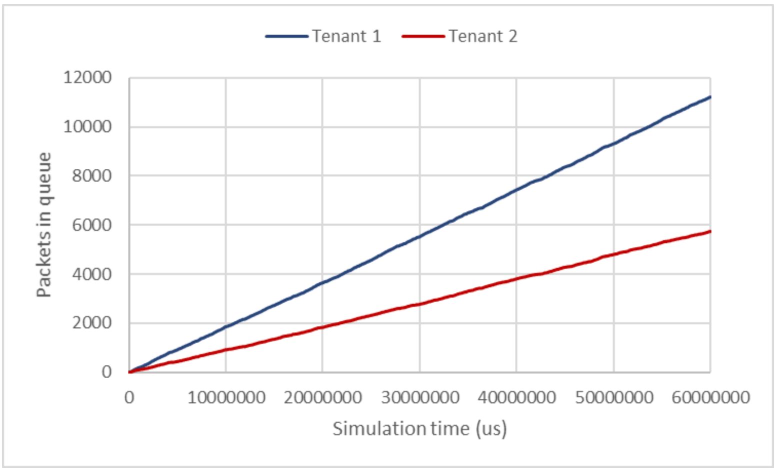


Figure 118. Packets in queue for fixed generator and multi-data rates in WTP.

Finally, it could be concluded that when more than a single Rb value are considered in a WTP, it is more difficult to control the traffic generation so it will be more likely the queue to grow. It has to be taken into consideration that in a real scenario, a tenant would not be generating traffic constantly, so this generator is just to study the performance of the algorithm as an experiment.

1.4.2. Uniform generator

Another requirement for the simulator was to test its performance when different traffic generator rates are taken into account and the traffic is not constant. This is why a generator that chooses the traffic rates using a uniform distribution has been designed.

This generator has been tested doing a simulation where the traffic generation rate is modified in every iteration. After this is computed the time to generate a packet and, if possible, packets are generated. The conditions of the simulation are the ones in Table 56. In this case, different modulation schemes are possible in the WTP, so different data rates are available with its success probabilities, which are the ones specified in Table 45.

Simulation Conditions	
Rb	54Mbps, 48 Mbps, 24 Mbps, 12 Mbps, 6 Mbps
Generation rate	Uniform mode
802.11g delays	ON
System Quantum	2ms
Retransmissions	ON
Simulation time	500ms
Number of WTP	1
Number of tenants	2
Weight Tenant 1	0.6
Weight Tenant 2	0.4
Packet length (Both tenants)	1514 Bytes
Time empty queue	100us

Table 56. Simulation conditions for uniform generation test

The traffic generation rates of the first 35ms of the simulation for Tenant 1 have been represented in Figure 119. It has not been represented all the simulation as it was clearer to represent less time. It has to be considered that the maximum generation rate for this tenant is computed in the following equation:

$$Gt_{max} = 54\ Mbps \cdot (1 - 0.291) \cdot 0.6 = 22.97Mbps \tag{130}$$

In the simulation, the values range from 0 to 22.97Mbps, so the uniform generator is working properly.

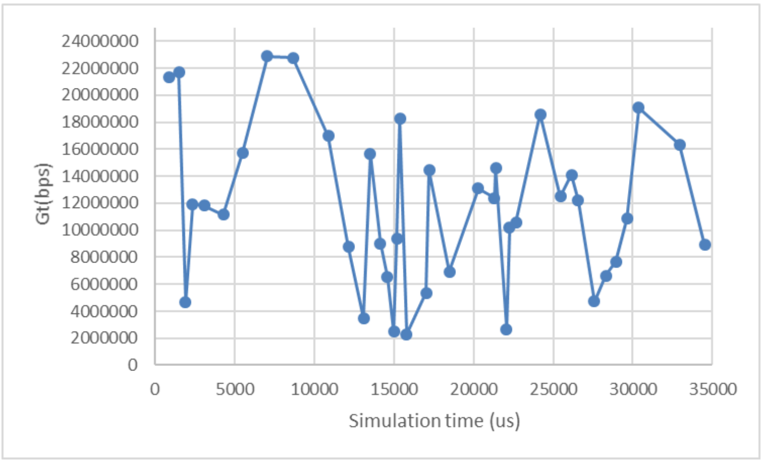


Figure 119.Traffic generation rate (Gt) using a uniform generator.

It has also been analysed the consequences of using a uniform generator in the performance of the system. The results obtained with this simulator are the ones in Table 57.

	Tenant 1	Tenant 2
Total Time (us)	499931.678	499931.678
Utilized Time (us)	292560.963	208922.444
Transmitted bytes (us)	964418	654048
Number of packets generated	637	432
Number of packets transmitted	637	432
BW (Mbps) Average	15.4327968	10.4661981
Gt (Mbps) Average	15.4327968	10.4661981
Number of packets in queue	0	0

Table 57. Results from simulator with uniform distribution

From the results, it is observed how the generation rate is smaller than the one obtained with the fixed traffic generation. If we consider the average traffic generation rates for Tenant 1 and 2, it has a middle value between the maximum generation rate and 0. It is also interesting to see the evolution of the queue with this traffic generator, which is represented in Figure 120. It is observed that the number of packets in the queue is really low and that the system is capable of managing the queue when it grows.

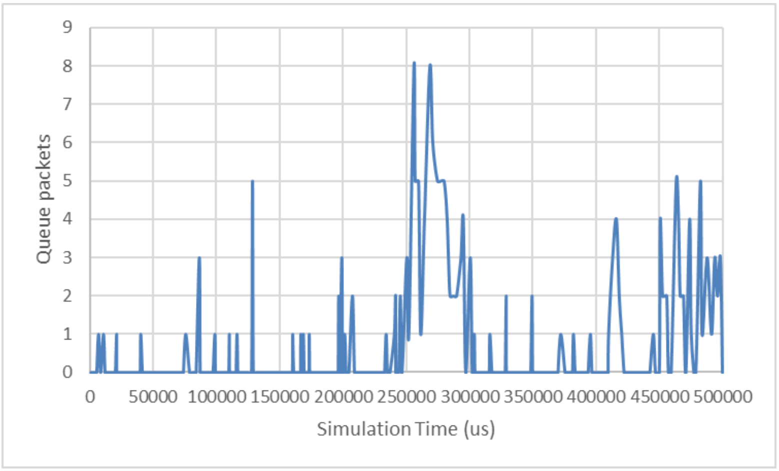


Figure 120. Packets in queue for the uniform generator simulation

1.4.3. Gaussian Generator

Moreover, it has been designed a Gaussian generator, as it is expected to generate traffic similarly to the reality.

After designing the Gaussian generator and implementing it, it has been tested with the following conditions. The nominal data rate probabilities are the ones in Table 45.

Simulation Conditions	
Rb	54Mbps, 48 Mbps, 24 Mbps, 12 Mbps, 6 Mbps
Generation rate	Gaussian mode
802.11g delays	ON
System Quantum	2 ms
Retransmissions	ON
Simulation time	500ms
Number of WTP	1
Number of tenants	2
Weight Tenant 1	0.6
Weight Tenant 2	0.4
Packet length (Both tenants)	1514 Bytes
Time empty queue	100us

Table 58. Simulation conditions for Gaussian generator

From the simulation results, it has been represented the traffic generation rates obtained by the Gaussian generator during the first 35ms for Tenant 1 (Figure 121), which has a maximum traffic generation rate of 22.97Mbps. As represented, the Gaussian generator sets traffic generation rates in its range of possible values (0, 22.97Mbps), generating more traffic generation rates around the mean.

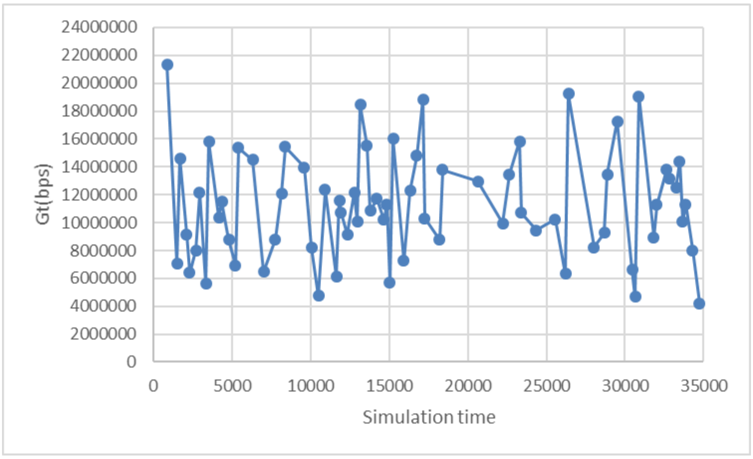


Figure 121. Traffic generation rate (Gt) using a Gaussian generator.

If we compare Figure 121 (Gaussian Generator) and Figure 119 (Uniform Generator), it can be observed that with the Gaussian Generator more values around the mean are used while with the Uniform Generator the traffic generator rates take more dispersed values.

Like in the uniform generator, it has been analysed the consequences of using a Gaussian Generator. Table 59 summarizes the obtained results when using Gaussian generator.

	Tenant 1	Tenant 2
Total Time (us)	498993.9	498993.9
Utilized Time (us)	285417.778	215122.741
Transmitted bytes (us)	823616	563208
Number of packets generated	544	372
Number of packets transmitted	544	372
BW (Mbps) Average	13.204	9.029
Gt (Mbps) Average	13.204	9.029
Number of packets in queue	0	0

Table 59. Results from simulator with Gaussian distribution

When we use a Gaussian generator the average generation rate obtained is a bit lower than when a uniform generator is used (Table 57). Moreover, it has been analysed the evolution of packets in the queue for Tenant 1, which is represented in Figure 122. As less packets are generated, there are less packets in the queue and they do not accumulate.

Finally, it could be concluded that the Gaussian generator is the one that is thought to be nearer a real scenario, so problems in queue would not be a problematic issue.

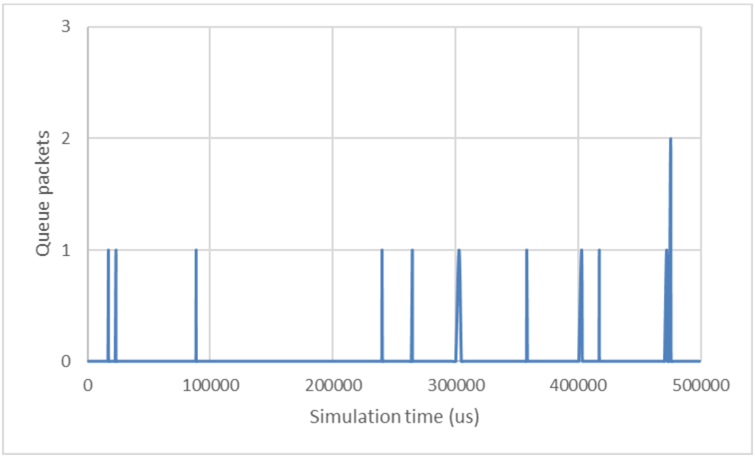


Figure 122. Packets in queue for the Gaussian generator simulation

1.5. Study time for empty queue

In this section, it is discussed the performance of the hypervisor simulator when the queue is empty. As a first approach, no additional time was taken into account when the queue was empty, which is not realistic because when the queue is empty the time keeps running. As the simulator tries to be as closer as possible to the reality, the management of the time when the queue is empty has been studied and designed.

Two different approaches were proposed. The first approach was to define a time per iteration, so all the system could be synchronized, so if during a certain iteration the queue was empty the time does not stop. The problem of this approach is that it adds complexity to the simulator, having to incorporate time compensation variables for the case when the transmission of a packet is greater than the time per iteration. Moreover, it makes it difficult to synchronize the times of different WTP.

The second approach, involves the definition of a fixed time to be added only when the queue is empty. Moreover, with this approach, instead of fixing a number of iterations per simulation, what is fixed is a time for the whole simulation because, if the fixed time for empty queue is too small, it could happen that the number of iterations is not enough to obtain packets in the queue.

The chosen approach is the last one. As explained, this option implies having to fix a time when the queue is empty. In order to do so, different simulations have been performed to study the behavior of the system when different values for empty queue are fixed and choose an appropriate value for this parameter.

In the following sections, the results obtained fixing the value to 10us, 100us and 500us considering different traffic generators are showed. In all simulations, the queue has been initialized empty.

1.5.1. Fixed Generator without retransmissions compensation factor

The first simulation uses a generator that produces traffic at a fixed rate. This rate has into account the effect of the delays introduced by the 802.11g Wi-Fi protocols so that the traffic generation rate and the serving rate are similar. The simulation conditions are shown in Table 60.

Simulation Conditions	
Rb	54Mbps
Success probability	0.9
Generation rate	Fixed mode
802.11g delays	ON
System Quantum	2 ms
Retransmissions	ON
Simulation time	30s
Number of tenants	2
Weight Tenant 1	0.6
Weight Tenant 2	0.4
Packet length (Both tenants)	154 Bytes

Table 60. Simulation Conditions for fixed generator without retransmissions compensation factor

Taking into account that the 802.11g delays represent the 29.1% of the time required to send a packet for our simulation conditions, the generation rate (Gt) for each of the tenants is the following one:

$$Gt_{T1} = 0.6 \cdot 54\ Mbps\ (1 - 0.291)\ =\ 22.97\text{Mbps} \tag{131}$$

$$Gt_{T2} = 0.4 \cdot 54\ Mbps\ (1 - 0.291)\ =\ 15.31\text{Mbps} \tag{132}$$

As it has been mentioned before, three different simulations have been run with the following times per empty queue: 10us, 100us and 500us. The results obtained are showed in the following tables.

SIMULATION 10 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	29997959.41	17989465.63	77433530	56891	51145	5746	34.4350550 9	330
Tenant 2	29997959.41	12010708.37	51621344	37927	34096	3831	34.3835466 9	340

Table 61. Results with fixed generator and 10 us as empty queue time

SIMULATION 100 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	**29999454.89**	**18004088.67**	**77671228**	**56894**	**51302**	**5592**	**34.51270628**	**400**
Tenant 2	**29999454.89**	**11997580.81**	**51780314**	**37929**	**34201**	**3728**	**34.52716997**	**500**

Table 62. Results with fixed generator and 100 us as empty queue time

SIMULATION 500 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	**29997656.67**	**18013047.04**	**77475922**	**56890**	**51173**	**5717**	**34.40880239**	**500**
Tenant 2	**29997656.67**	**11987140.59**	**51650110**	**37927**	**34115**	**3812**	**34.47034568**	**500**

Table 63. Results with fixed generator and 500 us as empty queue time

For each simulation, it has been obtained the total time of the simulation, the used time by each tenant, the number of bytes sent during the simulation, the number of packets generated and transmitted as well as the number of packets in the queue at the end of the simulation. Moreover, the effective bandwidth for each of the tenants and the time added by the empty queue has been computed.

With these results, it is shown that the time added by the empty queue is really small in comparison to the total time of the simulation (approximately 30s). This is due to the fact that with the generator mode used, the queue never empties because the traffic generation rate is greater than the serving rate. It is shown in all cases, as the number of packets generated is greater than the packets transmitted, and this difference is found in the number of packets in the queue for each of the tenants. The reason of this is that in the computation of the generation rate, the effect of retransmissions is not taken into account. Because of all this, the effect of the empty queue in this case cannot be evaluated, and the times added for all simulations correspond to the added time to generate the first packet.

Moreover, another important fact to point out is that the generation rate for the three cases is the same one as the number of packets generated in the three simulations is really similar. The small differences between the values are because the simulations do not exactly long 30s, just a little bit less.

1.5.2. Fixed Generator with retransmissions compensation factor

In this case, the effect of the retransmissions is taken into account in the computation of the generation traffic rate. Considering that the probability of transmitting a packet correctly is 0.9, it has been determined that if the traffic generation rate is multiplied by a factor smaller than 0.9, the queue is stabilized. In this case the retransmission compensation factor has been set to 0.89 and with this value the system is capable of managing the queue, clearing it when it grows. The generation rates obtained according to what has been said are the following ones:

$$Gt_{T1} = 0.6 \cdot 54\,Mbps \cdot\ (1 - 0.291) \cdot 0.89\ =\ 20.44\text{Mbps} \tag{133}$$

$$Gt_{T2} = 0.4 \cdot 54\,Mbps \cdot\ (1 - 0.291) \cdot 0.89\ =\ 13.63Mbps \tag{134}$$

The conditions for this case are the same as in the previous section, shown in Table 60 but including the retransmissions compensation factor. The results obtained are shown in the following tables.

SIMULATION 10 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	29998343.9	17951836.6	76653820	50634	50630	4	34.1597673	131010
Tenant 2	29998343.9	12048721.9	51102042	33756	33753	3	33.9302659	173760

Table 64. Results with fixed generator and 10 us as empty queue time with retransmission compensation factor

SIMULATION 100 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	29998767.1	17830188.1	76659876	50634	50634	0	34.3955433	21700
Tenant 2	29998767.1	12169844.4	51106584	33756	33756	0	33.5955545	258500

Table 65. Results with fixed generator and 100 us as empty queue time with retransmission compensation factor

SIMULATION 500 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	29998408.3	17756470.7	76656848	50634	50632	2	34.536975	500
Tenant 2	29998408.3	12244152.2	51103556	33756	33754	2	33.3896901	394500

Table 66. Results with fixed generator and 500 us as empty queue time with retransmission compensation factor

In this simulation, the parameters obtained are the same ones as in the previous section. With the retransmission compensation factor the number of packets generated and transmitted are balanced, so the packets do not accumulate in the queue and sometimes the queue is empty. In this case we will be able to analyze the effect of the time added when the queue is empty.

When setting the empty queue time to 10 us the simulation takes a lot of time. This is because when the queue clears, a lot of iterations are required until the generator is capable of generating a packet. Considering the generation rates previously computed, the time required to generate a packet for each of the tenants are the following ones:

$$t_{packet_{T1}} = \frac{1514\,Bytes \cdot 8\,\frac{bits}{Byte}}{20.44\text{Mbps}} = 592.45\,us \tag{135}$$

$$t_{packet_{T2}} = \frac{1514\,Bytes \cdot 8\,\frac{bits}{Byte}}{13.63\text{Mbps}} = 888.68\,us \tag{136}$$

Considering these times, a lot of iterations are required to generate a packet when 10us are chosen as time to add when the queue is empty. It can also be observed that the time added by the empty queue is greater than the obtained in the previous section because the queue is empty many times. It is observed that the time added by empty queue for tenant 2 is greater since the time needed to create a packet is greater than for tenant 1. According to this, the tenant 2 queue will be empty more times than tenant 1. However, the time added for both tenants has the same order of magnitude.

In the case of setting the empty queue time to 100us, the time required to obtain the simulation results is reduced significantly in comparison to the 10us case. This is because the number of iterations to generate a packet when the queue is empty is much smaller. Another issue to point out is that the difference in time added by empty queue between both tenants is much greater than in the case of 10us. This is because, when the queue is empty and we add time, that time is added to the simulation time, so it affects both tenants. In this way, when tenant 2 queue is empty and 100us are added, in the following iteration, tenant 1 will be more capable of producing packets. As the generation rate of tenant 2 is lower, it will be empty more times, so it will add more empty time, that will make tenant 1 add more packets, which also avoids that tenant 1 empties. This effect is accentuated when the time added by empty queue is of the same magnitude as the time to generate a packet.

In the third simulation, it is used 500us as empty queue time. The time required to obtain the simulations results is similar to the case of 100us. The difference in time added by empty queue between both tenants is extremely big. Tenant 1 just adds 500us in the first iteration, so the queue never empties. 500us is nearly the time that Tenant 1 needs to generate a packet (592.45 us), so when Tenant 2 is empty and adds 500us, in the following iteration, Tenant 1 nearly always will produce a packet. Considering that when a packet is correctly sent without retransmissions, the duration of the transmission is 316.37 us, adding 500us when the queue is empty is not really appropriate so a lower value would have more sense. Using a time per empty queue of 100us would be appropriate.

1.5.3. Gaussian Generator

The last group of simulations uses a Gaussian generator. The Gaussian generator updates the traffic generation rate every iteration using a Gaussian distribution function with mean of half the maximum possible generation rate and a variance of one third of the mean. The results obtained are shown in the following tables.

SIMULATION 10 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	29999804.5	17034410.9	58684154	38761	38761	0	27.5602858	3368160
Tenant 2	29999804.5	12965720	40828038	26967	26967	0	25.1913742	3484100

Table 67. Results with Gaussian generator and 10 us as empty queue time

SIMULATION 100 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	29999897.8	16361509.5	50887054	33611	33611	0	24.8813493	4555200
Tenant 2	29999897.8	13638588.3	35099062	23183	23183	0	20.5880909	5475600

Table 68. Results with Gaussian generator and 100 us as empty queue time

SIMULATION 500 us	Total time(us)	Used time (us)	Transmitted bytes (bytes)	Packets generated	Packets transmitted	Packets in the queue	BW(bps)	Time added by empty queue (us)
Tenant 1	29999543.1	15400821.9	47029382	31063	31063	0	24.4295439	4462000
Tenant 2	29999543.1	14599537.6	32290592	21328	21328	0	17.6940355	7112000

Table 69. Results with Gaussian generator and 500 us as empty queue time

In this case, the times added by empty queue are greater for all cases. This is because lower generation traffic rates are possible, so the time required to generate packets are greater and the probability of having the queue empty is higher. The Gaussian generator effect is shown in the number of packets generated, which is approximately half the number of packets generated with the fixed generator. The time added by empty queue is one order of magnitude greater than in section 2, which is produced due to the lower generation rates.

Once again, the time added by Tenant 1 is lower than the time added by Tenant 2 as the generation rates will be greater for Tenant 1 than for Tenant 2. The times added increase when the time added by empty queue is greater as expected.

2. Multi-WTP

The studies of this section are focused on using more than a single WTP. As a result of this, the different studies included are related to the compensation weight algorithm. Apart from running different WTPs satisfactorily, the purpose of these studies is to study the performance of the weight compensation algorithm as well as to determine which are the appropriate input values, to perform the simulations, considering different scenarios: different traffic deviation, slow and fast traffic change.

For all the following simulations, the scenario chosen consists of two WTP and three Tenants with the weights shown in Table 31.

Number of WTP	2		
Tenants SLA	Tenant 1	50%	
	Tenant 2	30%	
	Tenant 3	20%	
WTP Data rates (All WTP)	MCS RB (Mbps)	MCS Probability	MCS success probability
	54	0.8	0.9
	48	0.1	0.95
	24	0.05	0.98
	12	0.03	0.99
	6	0.02	0.999
802.11g Delays	ON		
Packet lengths	1514 Bytes for all tenants		
Simulation time	180.1 s		
Scheduling algorithm	ADRR		
System Quantum	2 ms		
Time Empty queue	0.01ms		
Traffic generation mode	Gaussian Shifted		

Table 70 Simulation conditions

2.1. Traffic generation deviation

The first study performed consists of varying the deviation of the Gaussian shifted traffic generation. Here there is an extended explanation of the study presented in section 4.2.1.For this analysis, the traffic generation rate changes every 1second for each of the tenants and WTP. The proportional sharing is enabled. Moreover, the weights are compensated every 20 seconds.

The deviations for the Gaussian shifted generator tested are 15% and 40%. The capacity average for the WTP is 28.16Mbps, which is computed from the effective data rates of the WTP (including 802.11g delays), its probabilities and a compensation factor, as in equation (137).

Equation (138) shows how the average time to transmit, given a certain data rate Rb$_{i,}$ is computed. Notice that it is an approximation, as it just considers three retransmissions while the programmed algorithm decreases the data rate if a packet is sent incorrectly three times, which will have a low probability with the values given. Equation (139) computes the compensation factor, needed to avoid long queues. This was caused by the difference of data rate allowed in the WTP, as when a low data rate (6Mbps) is used, it processes fewer packets while new packets are arriving to the system, which results in an increase of the queue. Therefore, we use the relation of times of the maximum data rate and the minimum data rate in the system, to avoid packets accumulate in the queue.

$$C_{average} = f_{compensator} \cdot \sum_{i=0}^{N-1} \frac{Packet_{length_{average}}(bytes) \cdot 8\frac{bits}{Byte}}{tp_{average_{Rbi}}} \cdot prob(Rb_i) \qquad (137)$$

$$tp_{average_{Rb_i}} = tp(Rb_i) \cdot prob(packet_{ok}) + 2 \cdot tp(Rb_i) \cdot prob(packet_{ok}) \cdot prob(packet_{Nok}) + 3 \cdot tp(Rb_i) \cdot prob(packet_{ok}) \cdot prob(packet_{Nok})^2 \qquad (138)$$

$$f_{compensator} = 1 - \left(\frac{tp(54Mbps, 1514Bytes)}{tp(6Mbps, 1514\,Bytes)}\right) = 0.85 \qquad (139)$$

The values of capacity average for each of the tenants and deviations for the case of 15% and 40% are specified the following table.

Tenant	Capacity average (bps)	Deviation (15%) (bps)	Deviation (40%) (bps)
Tenant 1 (50%)	14079943.8	2111991.57	5631977.521
Tenant 2 (30%)	8447966.281	1267194.942	3379186.513
Tenant 3 (20%)	5631977.521	844796.6281	2252791.008

Table 71. Capacity average and deviations for each of the tenants

The capacity average and the deviation are computed by using equations (140) and (141).

$$C_{average}(Tenant_i) = w_{SLA} \cdot Capacity_{WTP} \qquad (140)$$

$$Deviation(Tenant_i, Deviation(\%)) = C_{average}(Tenant_i) \cdot \frac{Deviation(\%)}{100} \qquad (141)$$

The traffic generated by the tenants in WTP for the deviation of 15% is represented in the following figures. In the figures it is also checked the performance of the Gaussian Shifted generator.

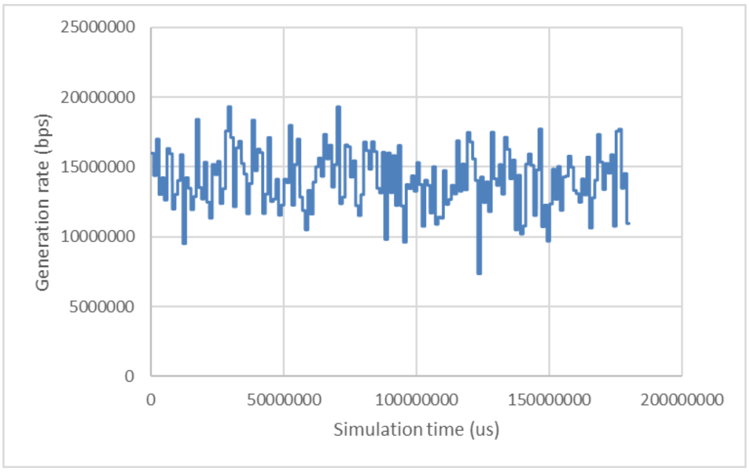


Figure 123. Traffic generation evolution for tenant 1 in WTP 1 with Gaussian deviation of 15%

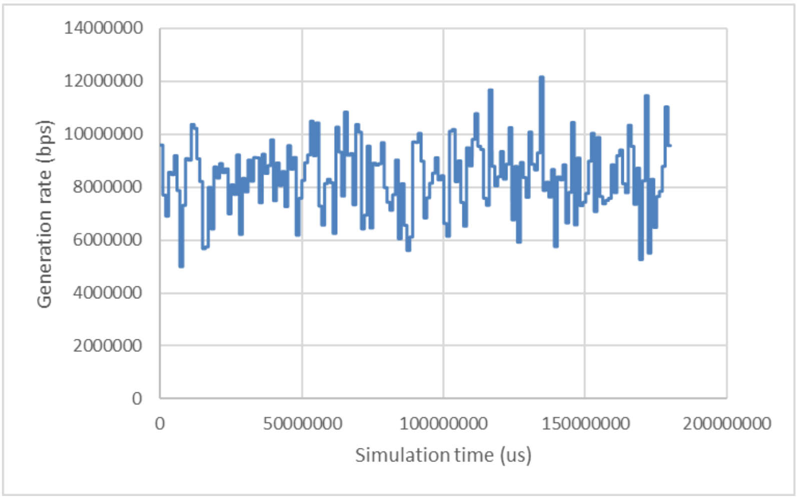


Figure 124. Traffic generation evolution for tenant 2 in WTP 2 with Gaussian deviation of 15%

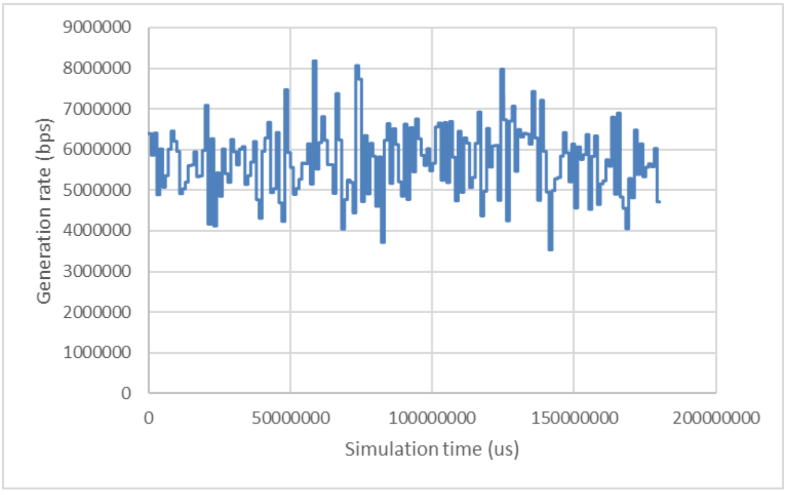


Figure 125. Traffic generation evolution for tenant 3 in WTP 2 with Gaussian deviation of 15%

Looking at the graphics, it can be checked that the capacity average computed in Table 71 and the deviation are accomplished correctly. The same figures have been generated when the Gaussian deviation is modified to 40%, which can be observed in the next figures. They show how the traffic generation rates reach more dispersed values around the capacity average.

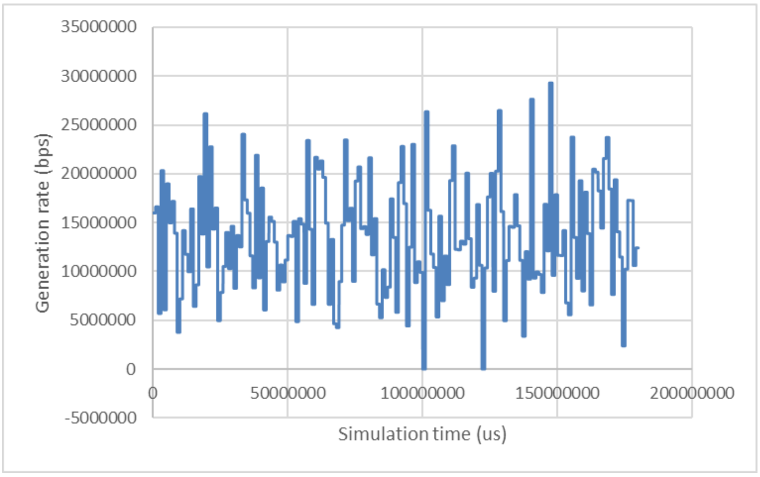


Figure 126. Traffic generation evolution for tenant 1 in WTP 1 with Gaussian deviation of 40%

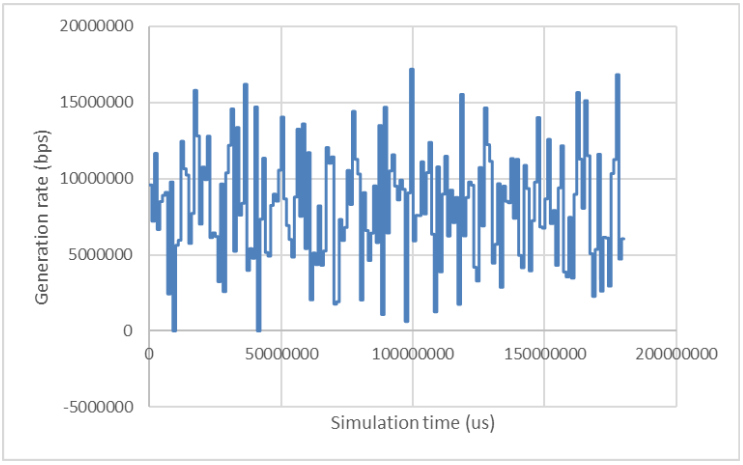


Figure 127. Traffic generation evolution for tenant 2 in WTP 1 with Gaussian deviation of 40%

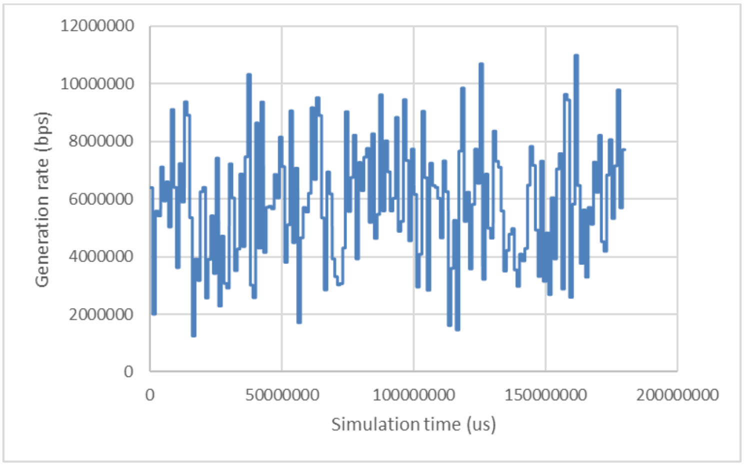


Figure 128. Traffic generation evolution for tenant 3 in WTP 1 with Gaussian deviation of 40%

After analyzing the generation rates, it has been analyzed how the weights are compensated during time. In order to do so, it has been represented the evolution of the weights during the simulation for WTP1, as it is represented in Figure 129 and Figure 130.

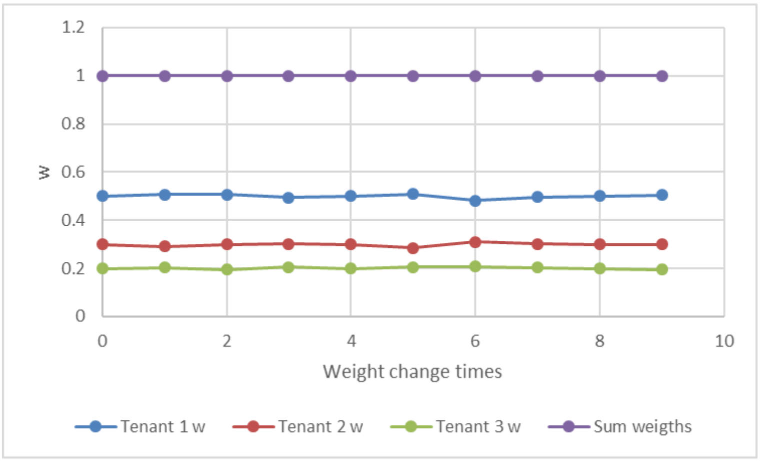


Figure 129. Weights evolution for the case of deviation of 15%

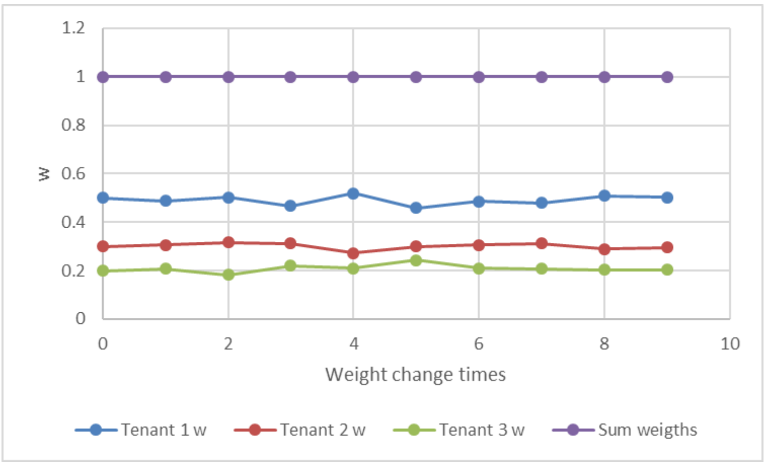


Figure 130. Weights evolution for the case of deviation of 40%

In Figure 129 and Figure 130, it is observed that when the weight of a tenant is increased, the weight of the other tenants is decreased. Moreover, it is showed how the sum of the weights of the system during all the simulations is always equal to 1, so we are assigning the total of the capacity in the WTP. In these figures, it can be slightly observed that the changes of weight are larger when a deviation of the 40% is used than when the deviation is set to 15%.

In Figure 131, Figure 132 and Figure 133, it can be observed the variation of the weights for each of the tenants in the case of deviation of 15% and 40%. It can be stated that when a deviation of the Gaussian Generator is set to 40%, the weights change to more dispersed values. This difference is more visible for tenant 1 than for tenant 2 and 3, as the dispersion values are greater in proportion.

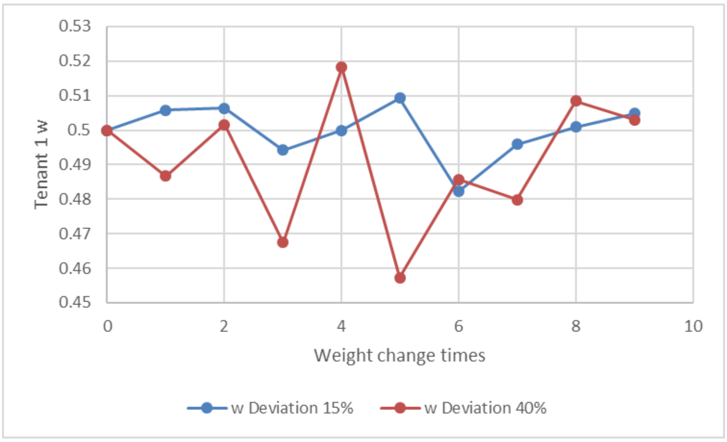


Figure 131. Comparison of the weights of tenant 1 when using a deviation of 15% and 40%

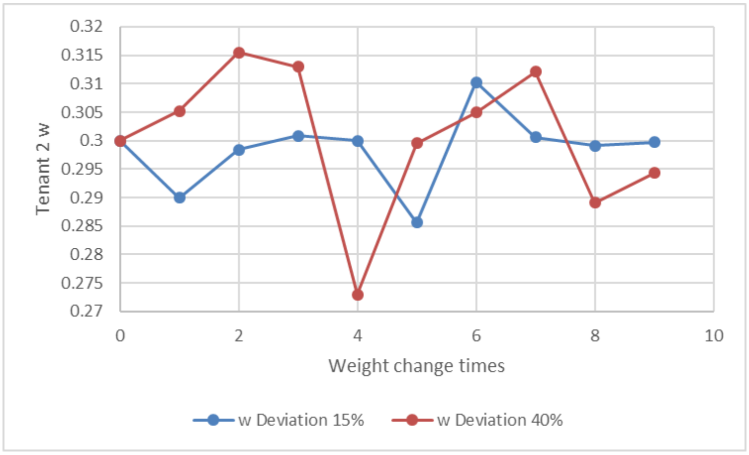


Figure 132. Comparison of the weights of tenant 2 when using a deviation of 15% and 40%

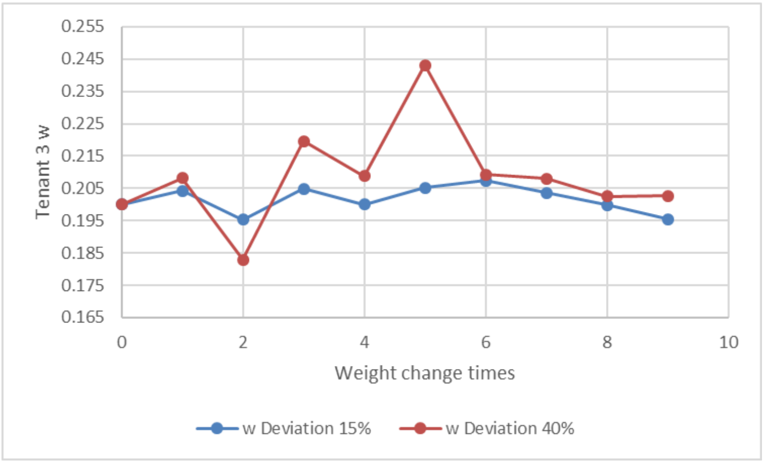


Figure 133. Comparison of the weights of tenant 3 when using a deviation of 15% and 40%

Finally, it has been analyzed if the SLA is accomplished in average for the whole network. Table 72 shows the performance of each tenant separated per WTP. It can be observed how the SLA is accomplished looking at the % of Used Time and the % of Transmitted Bytes. Moreover, In Table 73 , it can be found the average of the used time and transmitted bytes in the whole network, which also accomplishes the SLA.

WTP	Tenant	Used Time (us)	Transmitted bytes	Packet transmitted	Packets generated	%Used Time	%Transmitted bytes
WTP 1	**Tenant 1**	89158156	317243560	209540	209646	0.4950	0.5006
	Tenant 2	53657359.56	187893456	124104	124343	0.2979	0.2965
	Tenant 3	37284325.11	128564338	84917	84917	0.2070	0.2029
	TOTAL	**180099840.7**	**633701354**	**418561**	**418906**	**1.0000**	**1.0000**
WTP 2	**Tenant 1**	89417256.07	317450978	209677	209691	0.4965	0.5016
	Tenant 2	53653286.22	188567186	124549	124765	0.2979	0.2980
	Tenant 3	37028556.74	126814154	83761	83762	0.2056	0.2004
	TOTAL	**180099099**	**632832318**	**417987**	**418218**	**1.0000**	**1.0000**

Table 72. Performance results for the case of 15% of Gaussian deviation

Tenant	Network % Used Time	Network %Transmitted bytes
Tenant 1	0.4958	0.5011
Tenant 2	0.2979	0.2972
Tenant 3	0.2063	0.2016

Table 73. Performance of tenants over the whole network for the case of 15% of Gaussian deviation

The same results have been generated for the case of 40% of deviation. Looking at Table 74, Table 75, it can be seen that similar results are obtained for this case.

WTP	Tenant	Used Time (us)	Transmitted bytes	Packet transmitted	Packets generated	%Used Time	%Transmitted bytes
WTP 1	Tenant 1	86626618.89	302279184	199656	200418	0.4810	0.4906
	Tenant 2	54056154	183902552	121468	122146	0.3001	0.2985
	Tenant 3	39417752.96	130007180	85870	86465	0.2189	0.2110
	TOTAL	180100525.9	616188916	406994	409029	1.0000	1.0000
WTP 2	Tenant 1	88418829.41	310056602	204793	207037	0.4916	0.4956
	Tenant 2	54505631.41	189254542	125003	125003	0.3030	0.3025
	Tenant 3	36950252.07	126267600	83400	83402	0.2054	0.2018
	TOTAL	179874712.9	625578744	413196	415442	1.0000	1.0000

Table 74. Performance results for the case of 40% of Gaussian deviation

Tenant	Network % Used Time	Network %Transmitted bytes
Tenant 1	**0.4863**	**0.4931**
Tenant 2	**0.3016**	**0.3005**
Tenant 3	**0.2121**	**0.2064**

Table 75. Performance of tenants over the whole network for the case of 40% of Gaussian deviation

2.2. Proportional sharing

This is an extended version of the study presented in section 4.2.3. Another study developed is the evaluation of the performance of the proportional sharing algorithm. The weight compensation algorithm was initially defined in the way that if during the previous period, the different tenants were not requesting the 100% of the WTP capacity, not all the resources were assigned. That means that the sum of the weights assigned to the different tenants in a WTP could be less than 1.

Regarding this issue, it was decided to add a mechanism that allow the proportional sharing of the non-assigned resources between the different tenants, having into account its SLA. In this section, the whole results comparing the usage and non-usage of this mechanism are provided.

The simulation conditions used consist of a weight compensation period of 20s, a traffic generator with a Gaussian dispersion of 40% with a variation period of 1s. The simulations have been run with proportional sharing and without proportional sharing.

Figure 134 shows the weight evolution and the sum of weights when proportional sharing is enabled while in Figure 135 the proportional sharing is disabled. It can be observed that in the first case the addition of all the weights is always equal to 1 while in the second case sometimes the addition of weights is lower than 1. In both cases the sum of weights never exceeds 1.

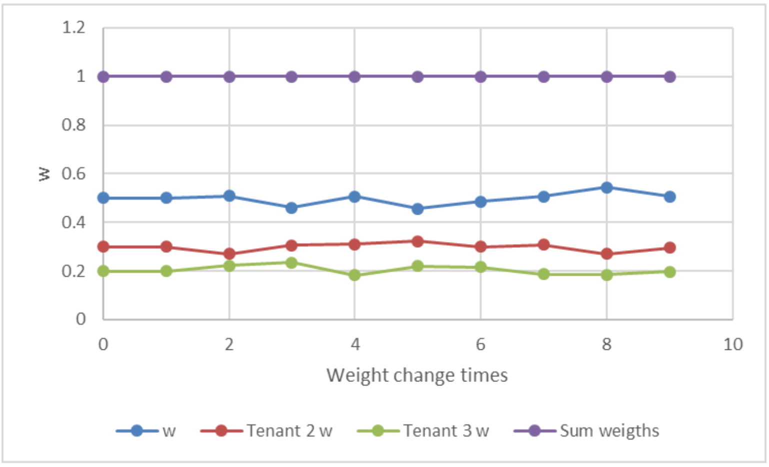


Figure 134. Weight evolution and sum of weights with proportional sharing

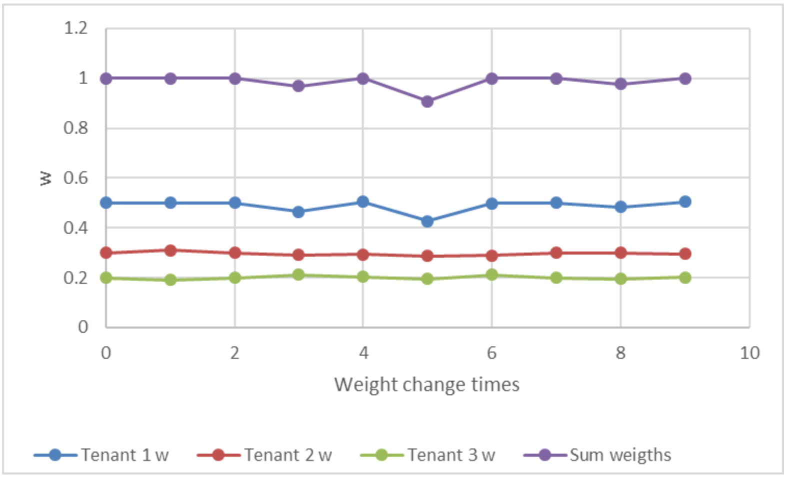


Figure 135. Weight evolution and sum of weights without proportional sharing

The following tables show the performance results with and without proportional sharing. It can be observed how the results in both cases are similar.

WTP	Tenant	Used Time (us)	Transmitted bytes	Packet transmitted	Packets generated	%Used Time	%Transmitted bytes
WTP 1	**Tenant 1**	89106288.07	316515326	209059	209841	0.4948	0.4977
	Tenant 2	54462038.52	192797302	127343	128070	0.3024	0.3032
	Tenant 3	36530745.41	126653670	83655	84611	0.2028	0.1992
	TOTAL	**180099072**	**635966298**	**420057**	**422522**	**1.0000**	**1.0000**
WTP 2	**Tenant 1**	93364591.56	333102710	220015	224429	0.5184	0.5217
	Tenant 2	50690241.7	178873044	118146	120215	0.2815	0.2801
	Tenant 3	36044057.41	126517410	83565	83611	0.2001	0.1981
	TOTAL	**180098890.7**	**638493164**	**421726**	**428255**	**1.0000**	**1.0000**

Table 76. Performance results for the case of proportional sharing

Tenant	Network % Used Time	Network %Transmitted bytes
Tenant 1	**0.5066**	**0.5097**
Tenant 2	**0.2919**	**0.2917**
Tenant 3	**0.2015**	**0.1987**

Table 77. Performance of tenants over the whole network for the case of proportional sharing

WTP	Tenant	Used Time (us)	Transmitted bytes	Packet transmitted	Packets generated	%Used Time	%Transmitted bytes
WTP 1	Tenant 1	90408199.56	322680334	213131	215218	0.5020	0.5063
	Tenant 2	53785658.07	189617902	125243	126211	0.2986	0.2975
	Tenant 3	35904870.81	125083652	82618	83322	0.1994	0.1962
	TOTAL	**180098728.4**	**637381888**	**420992**	**424751**	**1.0000**	**1.0000**
WTP 2	Tenant 1	86072065.33	301753826	199309	199309	0.4779	0.4858
	Tenant 2	56230642.07	193602750	127875	127875	0.3122	0.3117
	Tenant 3	37795620.37	125814914	83101	83102	0.2099	0.2025
	TOTAL	**180098327.8**	**621171490**	**410285**	**410286**	**1.0000**	**1.0000**

Table 78. Performance results without proportional sharing

Tenant	Network % Used Time	Network %Transmitted bytes
Tenant 1	0.4900	0.4960
Tenant 2	0.3054	0.3046
Tenant 3	0.2046	0.1994

Table 79. Performance of tenants over the whole network without proportional sharing

Figure 136, Figure 137, Figure 138 show the queue evolution for each of the tenants without proportional sharing. If those figures are compared to Figure 49, Figure 50 and Figure 51, of the weight compensation period study, it can be observed how the queue lengths are much higher in this case. This is caused by the fact that not all the available resources are considered.

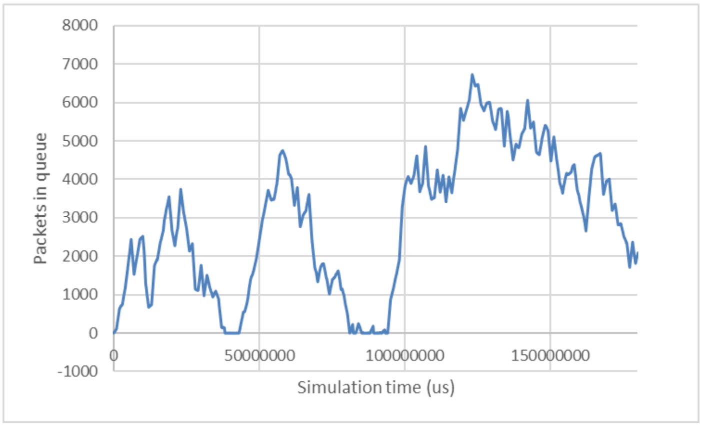


Figure 136. Queue evolution for tenant 1 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 20s without proportional sharing.

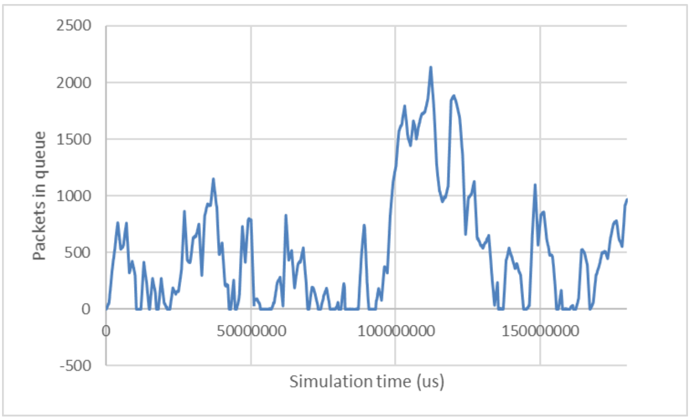


Figure 137. Queue evolution for tenant 2 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 20s without proportional sharing.

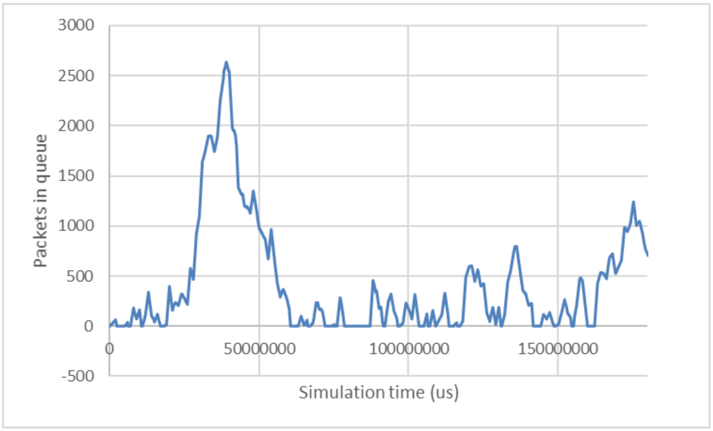
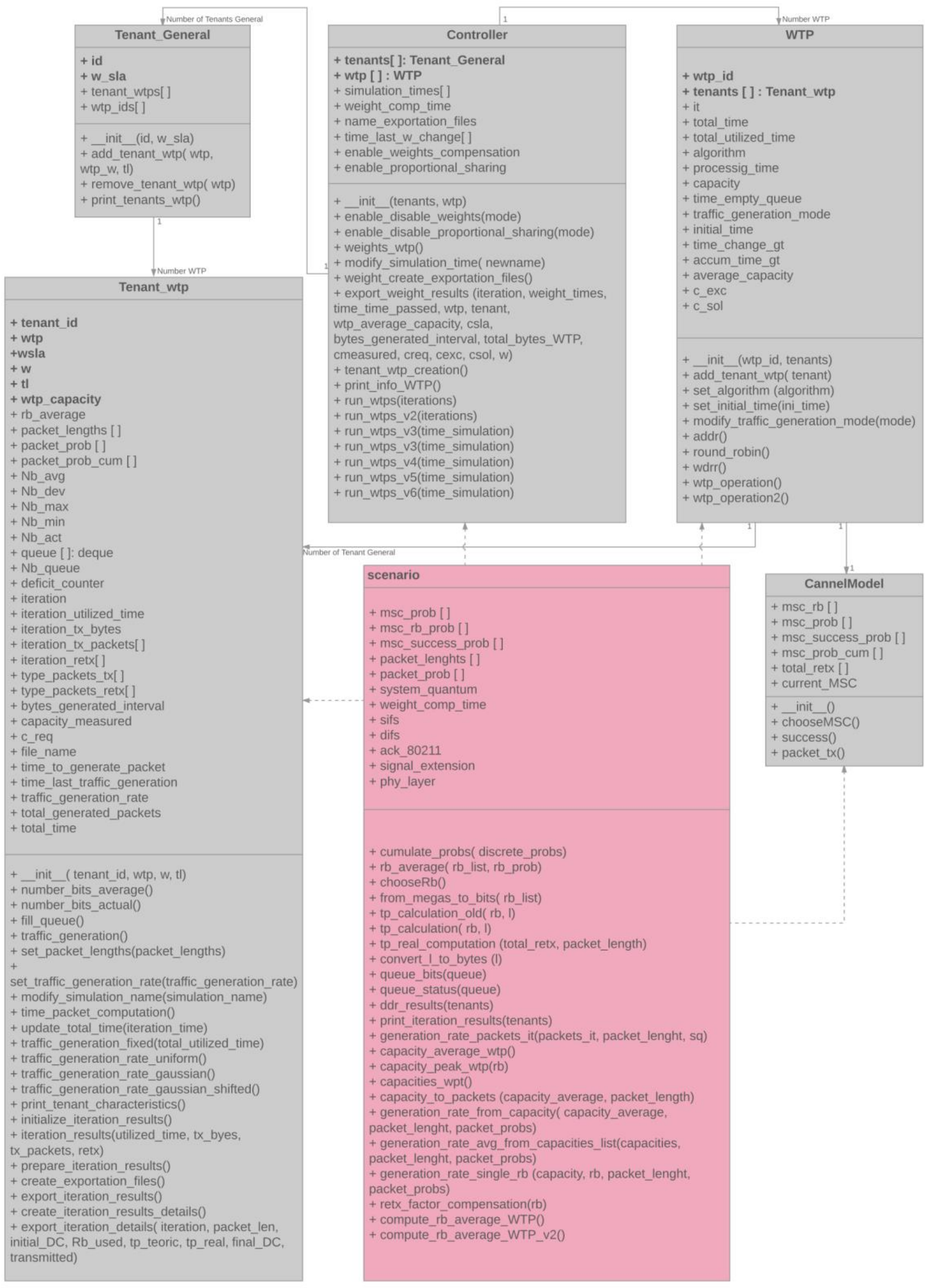


Figure 138. Queue evolution for tenant 3 in WTP 1. Case of period traffic change to 1 s and weights compensation period to 20s without proportional sharing.

In a scenario where the capacity requested is low during a period, it could happen that the computed weights without considering proportional sharing are smaller. So, if during a period of low activity, a peak of traffic arrives, the fact that no proportional sharing is used could result in a bad management of the resources: larger queues, latency… Proportional sharing can provide a softer transition between periods of high and low traffic levels and a reduction of the queue lengths.

ANNEX 3. ViRANsim simulator class and functions scheme

In this section it is included a global scheme of the entire ViRANsim simulator.

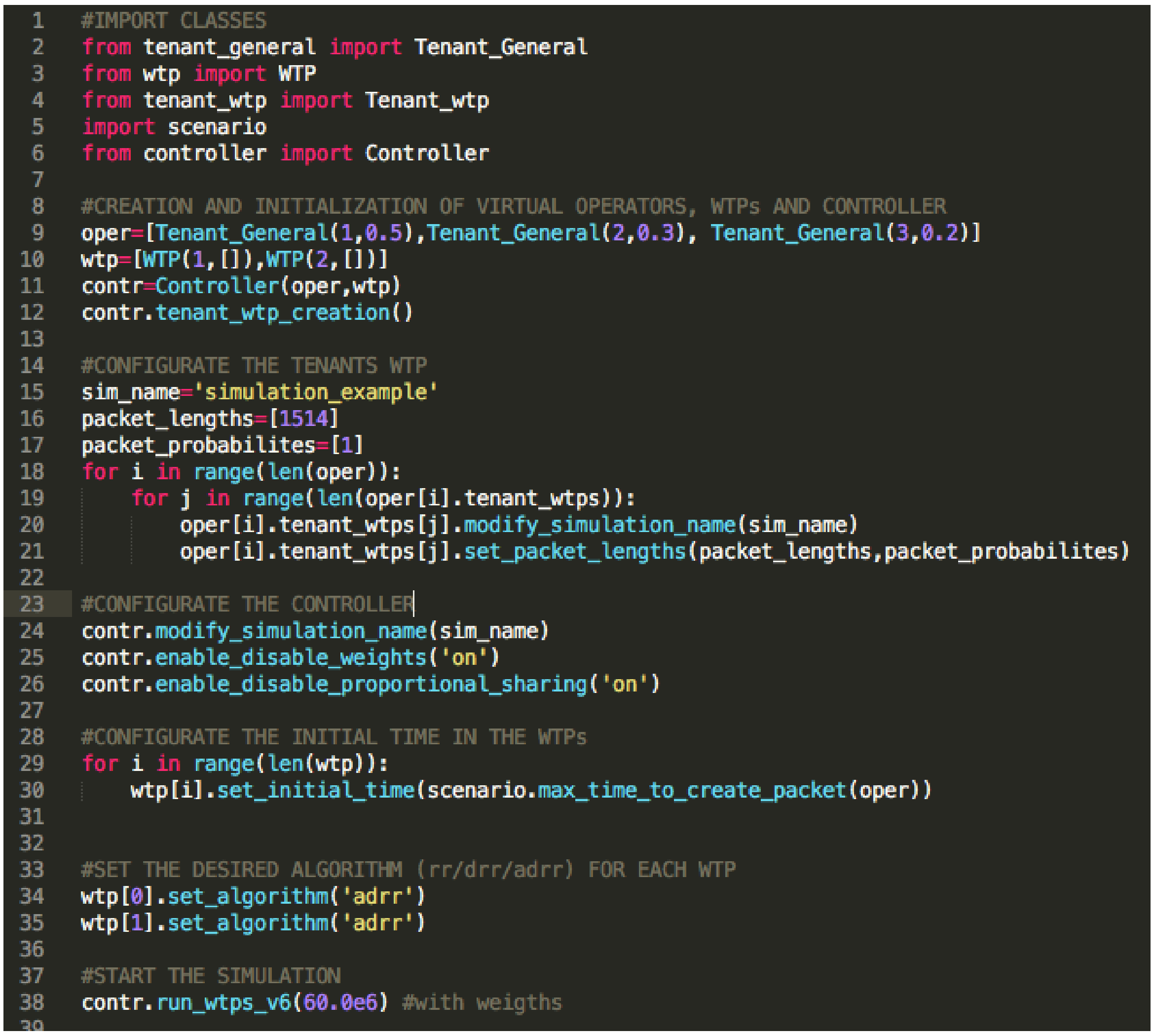


Number of Tenants General
1
Number WTP
Tenant_General
+ id
+ w_sla
+ tenant_wtps[ ]
+ wtp_ids[ ]
+ __init__(id, w_sla)
+ add_tenant_wtp(wtp, wtp_w, tl)
+ remove_tenant_wtp(wtp)
+ print_tenants_wtp()
Controller
+ tenants[ ]: Tenant_General
+ wtp [ ] : WTP
+ simulation_times[ ]
+ weight_comp_time
+ name_exportation_files
+ time_last_w_change[ ]
+ enable_weights_compensation
+ enable_proportional_sharing
+ __init__(tenants, wtp)
+ enable_disable_weights(mode)
+ enable_disable_proportional_sharing(mode)
+ weights_wtp()
+ modify_simulation_time(newname)
+ weight_create_exportation_files()
+ export_weight_results (iteration, weight_times, time_time_passed, wtp, tenant, wtp_average_capacity, csla, bytes_generated_interval, total_bytes_WTP, cmeasured, creq, cexc, csol, w)
+ tenant_wtp_creation()
+ print_info_WTP()
+ run_wtps(iterations)
+ run_wtps_v2(iterations)
+ run_wtps_v3(time_simulation)
+ run_wtps_v3(time_simulation)
+ run_wtps_v4(time_simulation)
+ run_wtps_v5(time_simulation)
+ run_wtps_v6(time_simulation)
WTP
+ wtp_id
+ tenants [ ] : Tenant_wtp
+ it
+ total_time
+ total_utilized_time
+ algorithm
+ processig_time
+ capacity
+ time_empty_queue
+ traffic_generation_mode
+ initial_time
+ time_change_gt
+ accum_time_gt
+ average_capacity
+ c_exc
+ c_sol
+ __init__(wtp_id, tenants)
+ add_tenant_wtp(tenant)
+ set_algorithm (algorithm)
+ set_initial_time(ini_time)
+ modify_traffic_generation_mode(mode)
+ addr()
+ round_robin()
+ wdrr()
+ wtp_operation()
+ wtp_operation2()
1
Number WTP
1
Tenant_wtp
+ tenant_id
+ wtp
+wsla
+ w
+ tl
+ wtp_capacity
+ rb_average
+ packet_lengths [ ]
+ packet_prob [ ]
+ packet_prob_cum [ ]
+ Nb_avg
+ Nb_dev
+ Nb_max
+ Nb_min
+ Nb_act
+ queue [ ]: deque
+ Nb_queue
+ deficit_counter
+ iteration
+ iteration_utilized_time
+ iteration_tx_bytes
+ iteration_tx_packets[ ]
+ iteration_retx[ ]
+ type_packets_tx[ ]
+ type_packets_retx[ ]
+ bytes_generated_interval
+ capacity_measured
+ c_req
+ file_name
+ time_to_generate_packet
+ time_last_traffic_generation
+ traffic_generation_rate
+ total_generated_packets
+ total_time
+ __init__(tenant_id, wtp, w, tl)
+ number_bits_average()
+ number_bits_actual()
+ fill_queue()
+ traffic_generation()
+ set_packet_lengths(packet_lengths)
+ set_traffic_generation_rate(traffic_generation_rate)
+ modify_simulation_name(simulation_name)
+ time_packet_computation()
+ update_total_time(iteration_time)
+ traffic_generation_fixed(total_utilized_time)
+ traffic_generation_rate_uniform()
+ traffic_generation_rate_gaussian()
+ traffic_generation_rate_gaussian_shifted()
+ print_tenant_characteristics()
+ initialize_iteration_results()
+ iteration_results(utilized_time, tx_byes, tx_packets, retx)
+ prepare_iteration_results()
+ create_exportation_files()
+ export_iteration_results()
+ create_iteration_results_details()
+ export_iteration_details(iteration, packet_len, initial_DC, Rb_used, tp_teoric, tp_real, final_DC, transmitted)
Number of Tenant General
1
1
1
scenario
+ msc_prob [ ]
+ msc_rb_prob [ ]
+ msc_success_prob [ ]
+ packet_lenghts [ ]
+ packet_prob [ ]
+ system_quantum
+ weight_comp_time
+ sifs
+ difs
+ ack_80211
+ signal_extension
+ phy_layer
+ cumulate_probs(discrete_probs)
+ rb_average(rb_list, rb_prob)
+ chooseRb()
+ from_megas_to_bits(rb_list)
+ tp_calculation_old(rb, l)
+ tp_calculation(rb, l)
+ tp_real_computation (total_retx, packet_length)
+ convert_l_to_bytes (l)
+ queue_bits(queue)
+ queue_status(queue)
+ ddr_results(tenants)
+ print_iteration_results(tenants)
+ generation_rate_packets_it(packets_it, packet_lenght, sq)
+ capacity_average_wtp()
+ capacity_peak_wtp(rb)
+ capacities_wpt()
+ capacity_to_packets (capacity_average, packet_length)
+ generation_rate_from_capacity(capacity_average, packet_lenght, packet_probs)
+ generation_rate_avg_from_capacities_list(capacities, packet_lenght, packet_probs)
+ generation_rate_single_rb (capacity, rb, packet_lenght, packet_probs)
+ retx_factor_compensation(rb)
+ compute_rb_average_WTP()
+ compute_rb_average_WTP_v2()
CannelModel
+ msc_rb [ ]
+ msc_prob [ ]
+ msc_success_prob [ ]
+ msc_prob_cum [ ]
+ total_retx [ ]
+ current_MSC
+ __init__()
+ chooseMSC()
+ success()
+ packet_tx()

Figure 139. Overall ViRANsim simulator classes and variables

ANNEX 4. ViRANsim simulation script example

This annex consists of an example of how to run a simulation using the simulator designed.



```
#IMPORT CLASSES
from tenant_general import Tenant_General
from wtp import WTP
from tenant_wtp import Tenant_wtp
import scenario
from controller import Controller

#CREATION AND INITIALIZATION OF VIRTUAL OPERATORS, WTPs AND CONTROLLER
oper=[Tenant_General(1,0.5),Tenant_General(2,0.3), Tenant_General(3,0.2)]
wtp=[WTP(1,[]),WTP(2,[])]
contr=Controller(oper,wtp)
contr.tenant_wtp_creation()

#CONFIGURATE THE TENANTS WTP
sim_name='simulation_example'
packet_lengths=[1514]
packet_probabilites=[1]
for i in range(len(oper)):
    for j in range(len(oper[i].tenant_wtps)):
        oper[i].tenant_wtps[j].modify_simulation_name(sim_name)
        oper[i].tenant_wtps[j].set_packet_lengths(packet_lengths,packet_probabilites)

#CONFIGURATE THE CONTROLLER
contr.modify_simulation_name(sim_name)
contr.enable_disable_weights('on')
contr.enable_disable_proportional_sharing('on')

#CONFIGURATE THE INITIAL TIME IN THE WTPs
for i in range(len(wtp)):
    wtp[i].set_initial_time(scenario.max_time_to_create_packet(oper))


#SET THE DESIRED ALGORITHM (rr/drr/adrr) FOR EACH WTP
wtp[0].set_algorithm('adrr')
wtp[1].set_algorithm('adrr')

#START THE SIMULATION
contr.run_wtps_v6(60.0e6) #with weigths
```

It has to be mentioned that it has been used Sublime Text editor for the thesis development.

Glossary

SDN:	Software Defined Networking.
NFV:	Network Function Virtualization
RAN:	Radio Access Network
MVNO:	Mobile Virtual Network Operator
AP:	Access Points
GRCM:	Mobile Communications Research Group
CREATE-NET:	Center for REsearch And Telecommunication Experimentation for NETworked communities
SLA:	Service Level Agreement
ADRR:	Air-Time Deficit Round Robin
RR:	Round Robin
WDRR:	Weighted Deficit Round Robin
NOS:	Network Operating System
IT:	Information Technology
VNF:	Virtualized Network Functions
COTS:	Commercial Off-The-Shelf
CAPEX:	CAPital EXpenditures
OPEX:	OPerating EXpense
CPU:	Central Processing Unit
RAM:	Random Access Memory
VM:	Virtual Machines
IaaS:	Infrastructure as a Service
CPP:	Click Packet Processors
LVAP:	Light Virtual Access Point
WTP:	Wireless Termination Point
LVNF:	Light Virtual Network Function
TCP:	Transmission Control Protocol
SDK:	Software Development Kit
LAN:	Local Area Network
MCS:	Modulation Coding Scheme
FIFO:	First In First Out
CSV:	Comma-Separated-Values
DC:	Deficit Counter