uc3m | Universidad Carlos III de Madrid

Degree in Telecommunication Technologies Engineering
2018-2019

Bachelor Thesis

Design and Implementation of a Network Slicing Aware Radio Scheduler

Sergio Fuente Pascual

Tutor: Marco Gramaglia

Leganés, 2019

ABSTRACT

In this project we will be introducing an overview of the concept of network slicing and its main implications when applied towards Long Term Evolution technologies. It starts by reviewing the main definitions and implications of LTE, regarding its protocols, physical channels and reference signals, so the reader can understand the basis of this technology, before moving to the meaning of network slicing in terms of real case scenarios.

This thesis focuses on scheduling algorithms, specifically downlink scheduling algorithms. There is a review of the Round Robin scheduling algorithm which will be later compared to the newly introduced scheduling algorithm based on slicing the network into “bandwidth slots”.

Firstly, there is a simulation on Matlab how the throughput varies from one algorithm to the other, and how we can assign an effectively a bigger throughput to a desired user. After completing this first simulation, the project moves on to a real software implementation and executes the proposed algorithm on srsLTE. Once again, there is an analysis of the results in terms of throughput on the new proposed algorithm in which a conclusion is reached on how the throughput varies depending the amount of bandwidth or slots we assign to each one of the users. The second part of the analysis shows a comparation of this throughput with respect to time.

ACKNOWLEDGEMENT

This thesis project has been conducted as a final step to achieve my bachelor's degree in Telecommunication Technologies Engineering at the Carlos III University of Madrid, under the supervision of Dr. Marco Gramaglia.

I would like to express my sincere gratitude to my advisor, Dr. Marco Gramaglia, as well as Mr. Ginés García, for their consistent support and recommendations during the realization of this project. Their guidance helped me in doing my research activity and writing this thesis, indicating me also how to move forward at particular times in which I got stuck and needed advice to continue developing the project. Sincere thanks for their motivation and complete availability to help me get this project done on a most efficient basis.

Finally, I would also like to thank my parents and friends for their unconditional support and believing in me during my undergraduate studies.

Table of Contents

1. INTRODUCTION

1.1. Network Slicing Overview

Network slicing is known as the as the “slicing” of the network by means of virtualization in order to achieve virtual dedicated network with a specific functionality, mainly used towards a certain customer or service over a common network infrastructure. Network slicing is a key technology to efficiently support services with very diverse requirements, such as the ones that should support 5G networks

This slicing technology is focused on the optimization and efficiency of the available resources and infrastructure, making these slots or “slices” customizable, where each one of them is optimized and targeted towards the needs of the specific services.

The main difference of network slicing is that it provides a means of end-to-end virtualization for a given user or service. This also allows network carriers to prioritize a selected service by giving more priority to a certain slot.

Virtualization allows for the support of many upcoming services regarding the arrival of 5G and its commercial deployment. In this case, a single physical network is sliced into a set of virtual networks focused on the support of RANs (Radio Access Networks), which can possibly be used for different services.

1.2. Background

In the past few years, the evolution of mobile broadband networks has been exponential, especially due to the high demand by user of faster and better connections to the Internet. This includes applications for multimedia and live high-quality streaming, online gaming or videoconferences among others. All these activities require a development of our networks and keep demanding for higher data rates.

The 3GPP (Third Generation Partnership Project) started working on these challenges in order to achieve higher performance, developing HSPA (High Speed Packet Access), which is used by the mentioned applications on 3G phones.

Due to the increasing demand which 3GPP could not satisfy, it evolved to a new generation of this technology, known as LTE (Long Term Evolution). LTE uses the standard of 3GPP, intending to improve the performance of HSPA. LTE includes an innovative radio access technology which will include higher spectral efficiency and data rates while reducing the RTT (round trip time). LTE also evolves to using the access network known as e-UTRAN (Evolved Universal Terrestrial Radio Access), with standards defined by 3GPP in the release 8.

However, nowadays trends in mobile networks take us towards a strong diversification of services, characterized by increasingly heterogeneous requirements in the Quality of Service (QoS), which will require different prioritization of internet traffic for different applications. This trend has taken us to the development of 5G networks, which in the near future will have to sustain and support this increasing demand of heterogeneous and high data rates.

1.3. Motivation and Goals of the Thesis

The main motivation that encourages to work on this thesis is the fast development that these technologies have experienced over the last years, and the notorious certainty that they will continue to do so in the future.

We can already appreciate how LTE is the present and future of mobile broadbands and is also nowadays experiencing the evolution to 5G technologies, which will play a key role on the development of smart cities, the IoT (Internet of Things) or augmented reality services among other near-future innovations.

Unfortunately, the current mobile networks lack the capability and flexibility needed to meet the requirements demanded by these technologies. Several solutions are being implemented to address this issue, one of them being network virtualization, which takes the hardware-based network functions into a virtual cloud architecture. This allows the deployment of several virtual instances of the whole network, known as network slices.

In this thesis, we will be focusing on network slicing for 5G technologies. This is, the capability of detaching the physical network in many virtual ones and being able of assigning data resources to every one of the network slices depending on the service we want to set.

We will be working on a specific part of the network (the radio) and, as there is no 5G radio network implementation at the moment, we will be using 4G LTE radio network slicing.

2. STATE OF THE ART. LTE OVERVIEW

LTE (Long Term Evolution) was first introduced as the successor to UMTS and HSPA, with the idea that LTE would enable much higher speeds to be achieved while also reducing the round trip time (RTT) using a much lower packet latency.

2.1. LTE Goals

LTE is set to achieve the following specifications [1, p. 9]:

Data rates

LTE requires high peak transmission rate of data up to 100 Mb/s working on a 20 MHz downlink spectrum and 50 Mb/s on a 20 MHz.

Bandwidth

LTE technology sets a bandwidth which can range a lowest of 1.25MHz up to the peak of 20 MHz, with 20MHz being saved for the peak fastest data rates. Also, LTE supports both FDD (Frequency-Division Duplex) and TFF (Time- Division Duplex).

Mobility

In LTE, the Mobility Management Entity (MME) is connected to more cells (eNodeB), which are grouped into tracking areas, therefore optimizing the mobility for low terminals speeds which range in LTE from 0 to 15 km/h whereas high UE speeds can go up to 500 km/h.

In order to fulfill these requirements, Orthogonal Frequency Division Multiplex (OFDM) was selected for the physical layer, implementing also a multiple-antenna technique such as MIMO (multiple input multiple output), which can increase channel capacity and enhance signal robustness.

OFDM

OFDM technology is based on encoding data over a multiple narrow band sub-carrier spread on a wide channel bandwidth. It is a multicarrier transmission scheme that splits up the transmitted high bit-stream signal into different sub-streams and sends these over different sub-channels. In sort, OFDM divides the bandwidth into multiple narrower sub-carriers used to transmit the data in parallel streams which results on a mitigation of ISI.

The frequencies of the sub-carrier are set orthogonally to avoid interference with each other, increasing spectrum efficiency. This way, a particular user is assigned a set of parallel subcarriers. In order to ensure the orthogonality, an explicit frequency spacing must be included to ensure orthogonality. In LTE, the carrier spacing is defined as 15kHz.[2]

Some of OFDM advantages are the following:

- High spectral efficiency
- Sever conditions on the channel can be allowed
- Low sensitivity to and offset on sample timing
- Mitigation of ISI due to the transmission of data on sub-carriers.

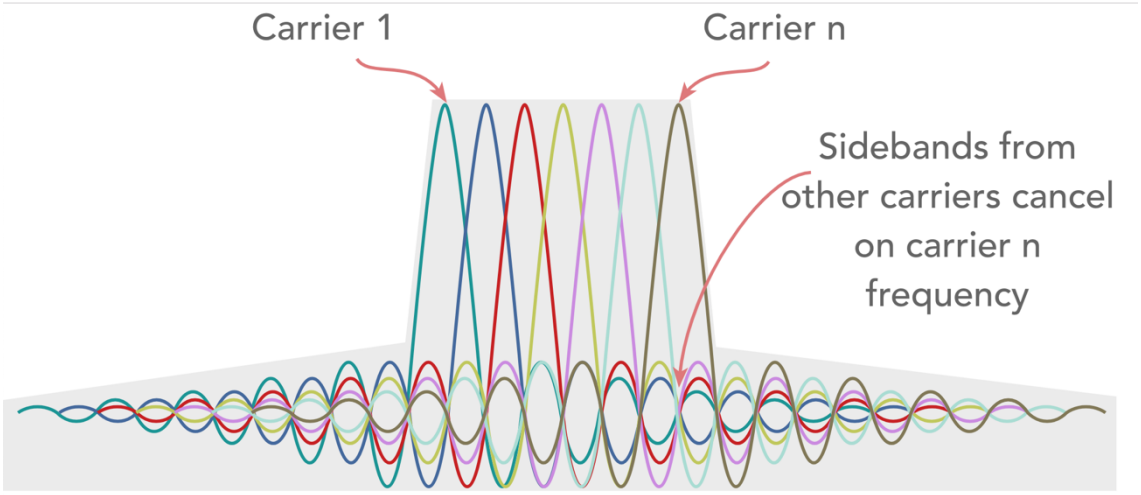


Figure: *https://www.electronics-notes.com*

MIMO

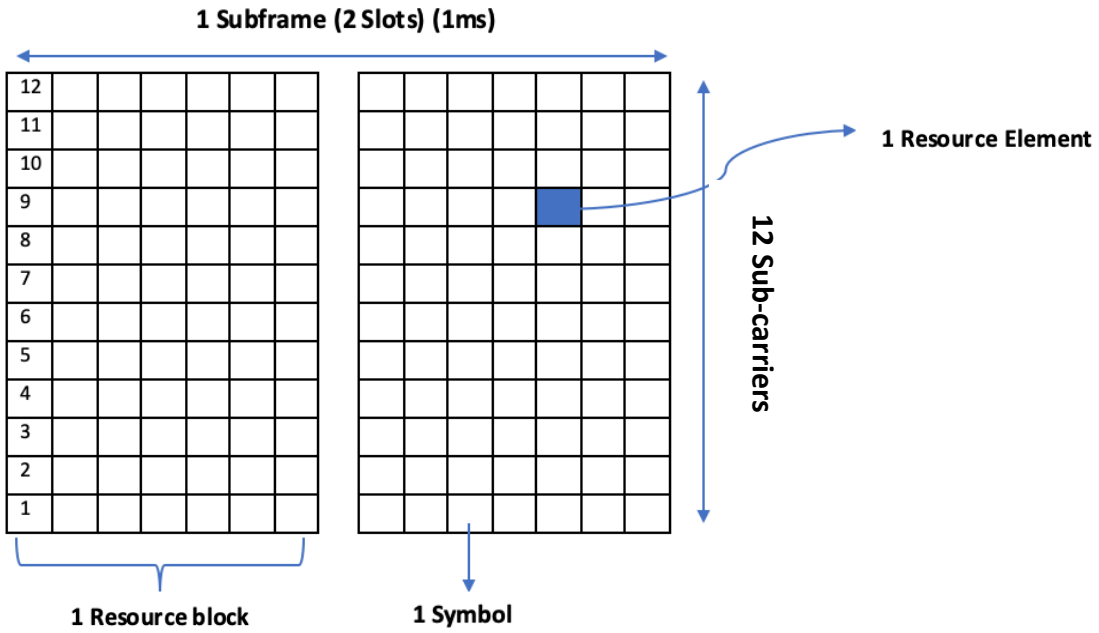
MIMO (Multiple Input Multiple Output) supports the use of multiple antennas at the receiver and the transmitter aiming to send and receive more than one signal at a time. To do so, spatial diversity and multiplexing are used.

Spatial multiplexing is used to enhance the capacity by sending independent data signals simultaneously in parallel from different antennas and spatial diversity allows to enhance the communication in fading channels by transmitting different replicas of the transmitted signal on several channels, which decreases the probability of the signal getting lost. By doing so, MIMO improves overall cell capacity.

2.2. Physical Layer Parameters

In LTE, communication is available in different frequency bands, of different sizes. Communication can take place in both paired and unpaired bands. Paired frequency bands mean that uplink and downlink transmissions use different frequency bands, whereas unpaired frequency bands would mean uplink and downlink transmissions sharing the same frequency bands.

The radio frame has a length of 10 ms, which is split into ten subframes equally sized of 1ms each in length. Both for downlink and uplink, scheduling will be done on a subframe basis. To do so, each subframe will be divided into two slots of 0.5 ms (resource blocks), and then each of these slots will contain six or seven OFDM symbols, making these elements known as Resource Elements (RE) the smallest unit of the frame, consisting of one OFDM subcarrier measured in one OFDM symbol interval. [3]



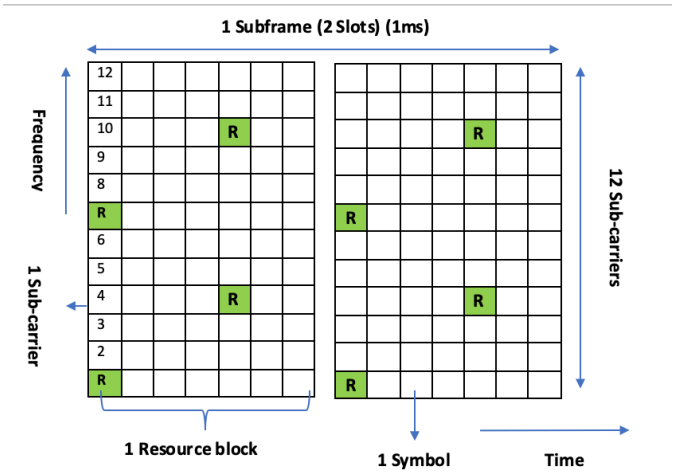
The following *resource grid* represents the number of resource blocks (RBs) available in the specified bandwidth. This number of resource blocks measured in the resource grid varies according to the bandwidth, since a bigger bandwidth will obviously mean a larger number of the available resource blocks. The OFDM subcarrier spacing is 15kHz: [1, p. 12]

Bandwidth (MHz)	1.4	3	5	10	15	20
Number of Resource Blocks	6	15	25	50	75	100
Number of occupied subcarriers	72	180	300	600	900	1200
Subcarrier spacings (KHz)	15	15	15	15	15	15

2.3. Reference Signals

Downlink Reference Signals

In order to perform the demodulation at the user equipment (UE), a channel estimation is performed by reference symbols inserted in the time-frequency grid. These reference symbols are inserted within the first and fifth OFDM symbols of each slot in the case of short CP and within the first and forth in the case of long CP.



Uplink Reference Signals

We can appreciate two different reference signals for uplink in LTE. The first one, Demodulation Reference Signals (DM-RS), are used for coherent demodulation at the eNodeB. The second one, Sounding Reference Signal (SRS) is used to allow scheduling on a channel dependent uplink. [3, p. 8]

2.4. Scheduling Algorithms

The scheduler is in charge of the allocation for time-frequency resources among users at every time instant. This scheduler can be found at the base station and is assigned both uplink and downlink resources. It allocates the different shares resources to each user equipment (UE) at every TTI (1 ms) following the specific logic of the scheduling algorithm.

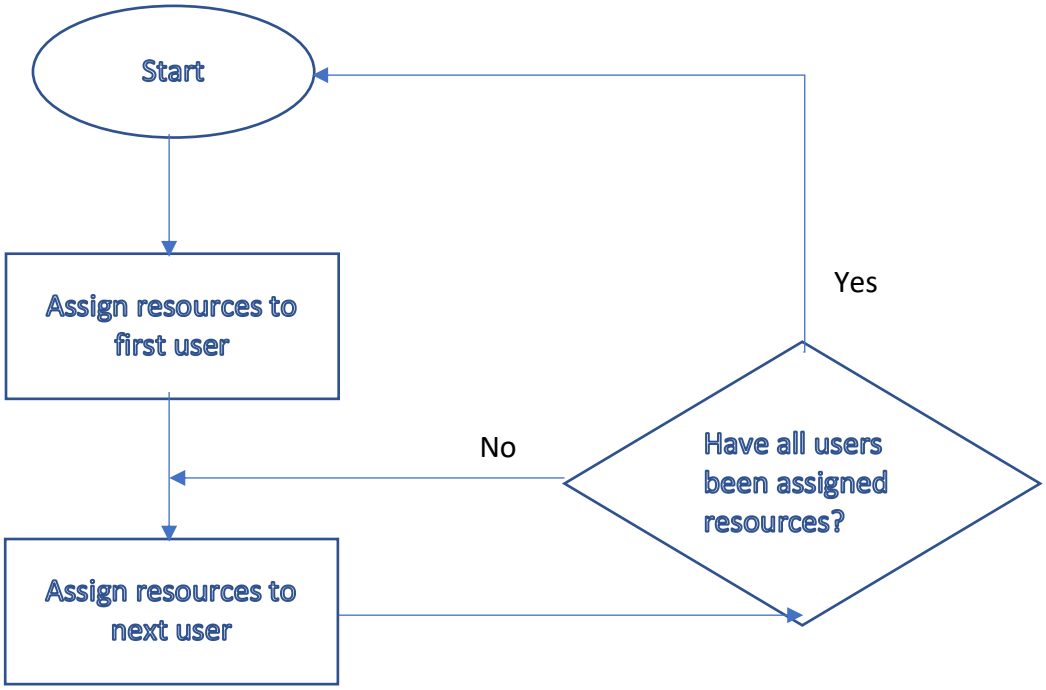
On a first view, the base station (BS) receives periodically information from each terminal acknowledging a Channel Quality Indicator (CQI). The higher this indicator goes, the better the channel is. This factor will be used to perform link adaptation.

Round Robin (RR)

Commonly used in LTE networks, terminals are assigned resources in turn sequentially. Round Robin starts to assign resources to every user starting from the first one and assigning resources from there on recursively. CQI factor is not taken into account, which simplifies the algorithm. Users can be assigned fading channels therefore throughput can be low while still appreciating a high BER. The main advantage for this scheduler would be that Round Robin is easy to be implemented, which is the reason why it is usually used by many systems.

On the downside, not taking into account the CQI factor provides a low efficiency in the management of the total amount of resources. Round Robin provides high fairness in exchange for performance, since there will only be fairness in the terms of Resource Blocks (RBs) assigned to every user.

We can appreciate how this algorithm works in the following flow chart:



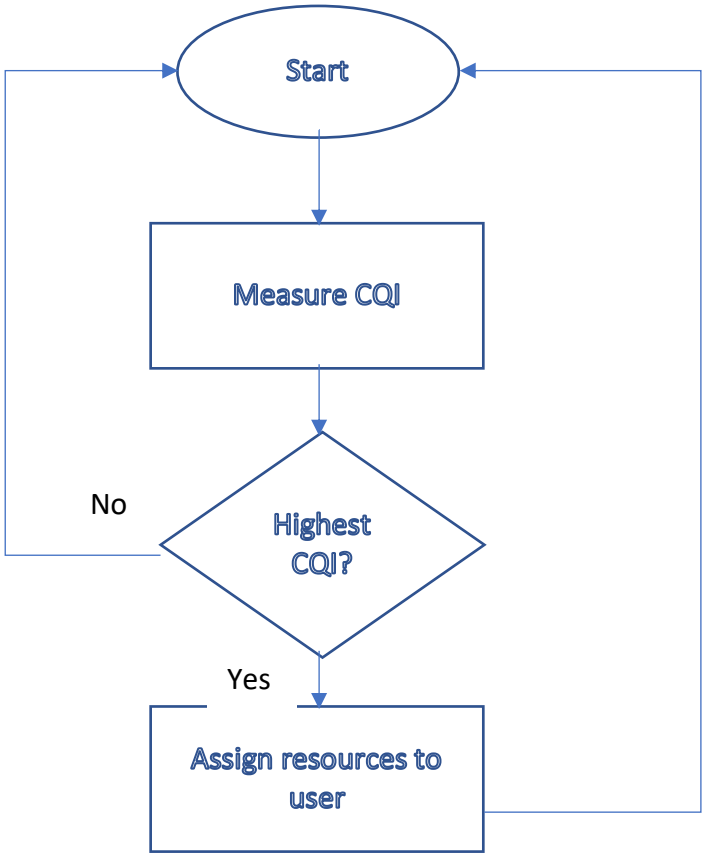
Best CQI

Best CQI assigns resource blocks to the user with the best channel quality for a particular resource block at every time interval. This algorithm can increase cell capacity at the expense of the fairness. Terminals located far from the base station (edge users) would most likely never be scheduled

Each terminal sends a Channel Quality Indicator (CQI) to the base station, which will perform the scheduling, transmitting a reference signal to the terminals. UEs will receive these reference signals which will be used to measure their specific CQI. The higher the CQI, the better the channel condition.

The downside of this algorithm is the lack of fairness from the point of view of throughput, since only users with highest CQIs will be assigned resources (these will be usually the users closest to the eNodeB).

We can appreciate how this algorithm works in the following flow chart:



MaxMin

MaxMin aims to maximize the minimum throughput for all the different users. Since it is not possible to increase throughput for a user without decreasing it for another, this will bring fairness to the system. The scheduler assigns a large amount of resources to users with low throughput to maximize their throughput, and since these users have low CQI, MaxMin will decrease overall system throughput. [4]

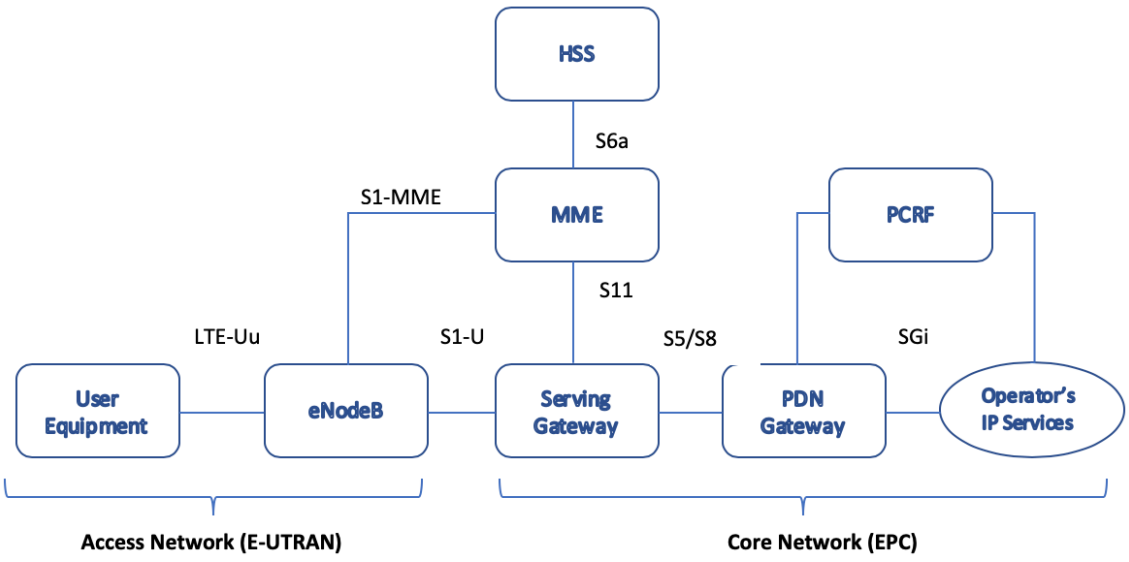
Proportional Fair

Proportional Fair provides high fairness to the system by using the channel variations to improve spectral efficiency. Resources will be assigned following an algorithm determined by the throughput at the specific TTI and the average throughput of the user.[4]

This is achieved by means of a Weighted Fair Queueing algorithm (WFQ), which sets scheduling weights for data flow i to $w_i = \frac{1}{c_i}$, where c_i represents the amount of consumed resources per data bit.

2.5. Protocol Architecture

The elements of the evolved packet system (EPS) and interface protocol designations are shown in the following figure:



This LTE radio protocol architecture consists in the user plane architecture (U-plane), the evolved UTRAN (E-UTRAN), a control plane (C-PLANE) and the evolved packet core (EPC). The user plane protocol stack between the e-Node B and UE consists of the following sub-layers: PDCP (Packet Data Convergence Protocol), RLC (radio Link Control), and Medium Access Control (MAC). We can find the Radio Resource Control layer (RRC) inside the control plane. This layer is responsible for configuring the lower layers. [5]

The packets received by a layer are named Service Data Unit (SDU) and the output of the layer is known as Protocol Data Unit (PDU).

Core Network Elements

1. The Mobility Management Entity (MME)

   Consists of a central control unit set in the core network (EPC). It is in charge of the control plane, creating a logical connection between both user and control plane and authenticates users on their first registration to the network. As a method of secure connection for the UEs, the UE's ciphering protection keys are collected and produced by a master key assigning to each UE a temporary id.

   Another task MME handles would be the handover between several eNodeBs. While MME is busy working on am UE, the network needs to decide which data packet will be allocated to the UE, so the MME stores the profile information of the user in order to retrieve this information and allocates it to the assigned packet data network connection.[2]

2. The Serving Gateway (S-GW)

   The S-GW is in charge of its own resources, acting once it gets a request from the Packet Data Network Gateway (P-GW) or the MME. These elements request the serving gateway's resources in order to establish handlers for that user. Also, the Serving Gateway is in charge of switching tunnels from two neighboring eNodeB using mobility.

3. The Home Subscription Server (HSS)

   The HSS is in charge of storing all the temporary data of all UEs associated with LTE networks. It stores essential parts of a subscriber profiles such as information required for mapping different services to the corresponding UEs. As on the MME, HSS also uses different keys for authentication in order to protect integrity and encryption.

4. The Packet Data Network Gateway (P-GW)

   The P-GW is in charge of coordinating the mobility between non-3GPP and 3GPP technologies. A different IP address is assigned to each UE by means of a P-GW so that the user is also able to access external networks.

Control Plane Protocols

The Control Plane is in charge of radio-specific functionality which depending on the state of the user equipment (UE) has two different states: idle or connected. The idle mode includes cell selection procedures, whereas in the connected mode, the UE supplies the E-UTRAN with information about the quality of the channel and also points the neighbor cell, which is used to enable the E-UTRAN to select the most suitable cell for the user. [5, p. 4]

Inside the Control Plane Protocols, the Radio Resource Control (RRC) is responsible for configuring lower layers, covering the following areas:

1 System Information → Broadcasts information of the system to set two types which will be applied to the connected mode the idle mode.
2 RRC Connection Control → Its main function is setting procedures for the establishment and modification of RRC connections.
3 Network Controlled Mobility → In charge of mobility, activating security and transferring UE information.
4 Measurement and Configuration Reporting →Supports the mobility function

User Plane Protocols

This protocol is formed by two different layers:[5, p. 5]

1. ***Packet Data Convergence Protocol Layer (PDCP)***

   This layer will be responsible for the following functions. First, it is in charge of header compression and decompression for all user plane data packets.

   PDCP is also responsible for handover management, reordering and sequencing PDUs when switching from the coverage area of one cell to another.

   Executes encryption and decryption of all data in the user and control plane, and also sets integrity and verification of data in the control plane.

2. **Radio Link Control (RRC)**

When transmitting, the RLC is in charge of reformatting PDCP PDUs to fit the required size by the MAC layer. On the receiving end, the RLC is tasked with the reconstruction of the PDUs.

The RLC also reorders packets received out of the sequence when performing the HARQ operation and can transmit on three modes depending on the delay of the traffic. These modes would be transparent mode, unacknowledged mode, and acknowledge mode.

3. **Medium Access Control Layer (MAC)**

Performs important functions that include the scheduler, which is in charge of distributing the available bandwidth to the users, depending on the scheduling algorithm. [5, p. 7]

It also performs the operation to retransmit received data blocks and generate ACK or NACK signaling in case of CRC (Cyclic Redundancy Check) failure.

The Medium Access Control Layer also maps the received RLC data to logical channels connecting the MAC with the physical layer.

2.6. Physical Channels

Physical channels are divided for downlink and downlink.

2.6.1. Downlink Physical Channels

LTE sets several downlink physical channels to carry information received from the MAC and higher layers. These channels can be divided into transport and control channels. [3, p. 10]

Transport Channels

1. **Physical Broadcast Channel (PBCH)** → Broadcasts key parameters needed for initializing the access such as bandwidth, ARQ indicator channel, or the eight bits most significant from the System Frame Number.
2. **Physical Downlink Shared Channel (PDSCH)** → Main channel to carry data which will be allocated to users on a dynamic basis.
3. **Physical Multicast Channel (PMCH)** → Definition of the structure in the physical layer carry multimedia services.

Control Channels

1. **Physical Downlink Control Channel (PDCCH)** → Carries Downlink Control Information (DCI) messages, which manages the resources assignment for the different users.
2. **Physical Control Format Indicator Channel (PCFICH)** → Carries control flame indicators (CFI) for every subframe, including the amount of OFDM symbols in use to control the transmission on the channel.
3. **Physical Hybrid ARQ Indicator Channel (PHICH)** → Indicates the user whether the eNodeB received correctly the user data sent on the uplink user.

2.6.2. Uplink Physical Channels

There are three different channel for uplink LTE transmission. [3, p. 10]

1. **Physical Uplink Shared Channel (PUSCH)** → Carries user data and control information necessary to decode the information.
2. **Physical Uplink Control Channel (PUCCH)** → Uplink data transmitted regardless of traffic data, including HARQ acknowledge messages, Channel Quality Indicator (CQI), MIMO feedback and Scheduling Requests.

3. **Physical Random Access Channel (PRACH)** → Carries the previous random access information that a UE sends to access the network in non-synchronized mode, used to allow the UE to synchronize its timing with that of the eNodeB.

3. DESIGN AND SIMULATION OF A NEW DOWNLINK SCHEDULING ALGORITHM

In this section, we will be designing a new downlink scheduling algorithm which will allow us to assign Resource Blocks (RBs) to different UEs following a RoundRobin directive, but in which we can choose how many RBs to assign to every UE. To do so, the terminals will be assigned resources in turn sequentially. When a user runs out of RBs to be assigned (as we have chosen), it will continue assigning resources to the terminals that we have chosen.

The implementation and analysis of this algorithm, and the comparison with a regular Round Robin scheduler has been done through simulations executed on the downlink link level simulator developed by the "Institute of Communications and Radio-Frequency Engineering" at the University of Vienna.

3.1. Link Level Simulator Overview

Link level simulations allow the user to perform different simulations, analyzing different behaviors between the user and the base station and the performance in terms of cell and user **throughput**. We used the LTE_Link_Level_1.7_r1089 simulator from the University of Vienna. This simulator works on the 2012 version of Matlab. Newer versions of the simulator could not be used since they were not available as free software for public download, and a specific license was required.

This simulator allows generating simulations with the following main characteristics:

- Scheduling technique: Round Robin, Best CQI, Proportional Fair
- Simulation Length: number of subframes
- LTE Bandwidth: 1.4MHz, 3MHz, 5MHz, 10MHz, 15MHz y 20MHz
- Transmission Scheme: SUSISO, MUSISO, SUMIMO, MUMIMO, etc
- Channel Type: PedB, PedA, PedB, PedBcorr, AWGN, etc

3.2. Round Robin Scheduler

In the RoundRobin scheduling implementation provided in this simulator, RBs are not assigned in turn to users. Taking as a example the case of four Mobile Stations and 6RBs, the scheduler does not assign cyclically a RB to MS1, MS2, MS3 and MS4. Using the following parameters for this simulation:

Parameter	**Value**
Number of User Equipments (UE)	4
Number of Base Stations	1
Bandwidth	1.4MHz
Channel Type	PedB
Simulation Length	100 subframes
Scheduling Algorithm	Round Robin, CQI=7
Transmission Scheme	MUSISO

The variable UE_mapping_all_UEs stores the mapping of RBs to users. In the example above, four users have been considered with a bandwidth of 1.4MHz, which corresponds to 6 RBs. After executing the simulation, we can appreciate that the first three RBs have been assigned to the first user and the next three RBs to the second user, whereas in the second time slot, the first three RBs have been assigned to the third user and the following three to the fourth user.

1	3
1	3
1	4
1	4
2	4
2	4

We can easily appreciate these results are clearly wrong, since in a Round Robin scheduler, the RBs should be assigned cyclically to the different UEs, being the proper Round Robin simulator results the following:

1	3
2	4
3	1
4	2
1	3
2	4

The provided simulation file roundRobinScheduler.m follows the following algorithm:

```
classdef roundRobinScheduler < network_elements.lteScheduler
% A round robin scheduler (equally schedule every user). Please note the
% following:
%  - For a static scheduler: since all the scheduling info is generated at
%    object creation, the UE allocation will be time invariant. Thus, if
%    the number of RBs is not an integer multiple of the number of UEs,
%    some RBs will be left unused.
%  - For a dynamic scheduler: still to be implemented :P
% Josep Colom Ikuno, jcolom@nt.tuwien.ac.at
% (c) 2009 by INTHFT
% www.nt.tuwien.ac.at

  properties
  end

  methods
     function obj =
roundRobinScheduler(RB_grid_size,Ns_RB,UEs_to_be_scheduled,scheduler_params,CQI_params)

        % Fill in basic parameters (handled by the superclass constructor)
        obj =
obj@network_elements.lteScheduler(RB_grid_size,Ns_RB,UEs_to_be_scheduled,scheduler_params,CQI_
params);

        switch scheduler_params.assignment
           case 'static'
              obj.static_scheduler = true;
              number_of_RBs_per_UE = floor(RB_grid_size*2 / obj.nUEs);

              % Get a vector of scheduling params (one for each UE)
              % initialized to the values that we want
              obj.UE_static_params = obj.get_initialized_UE_params(scheduler_params,CQI_params);

              % Fill in the RB allocation grid for each user (and codeword)
              %UE_mapping_all_UEs = zeros(RB_grid_size,2,obj.maxCodewords);
              UE_mapping_all_UEs = zeros(RB_grid_size,2);
              for u_=1:obj.nUEs
                 % NOTE: Same RB assignment for both codewords.
                 UE_RBs = 1 + number_of_RBs_per_UE*(u_-1) : number_of_RBs_per_UE +
number_of_RBs_per_UE*(u_-1);
                 cw_RB_grid = UE_mapping_all_UEs;
                 cw_RB_grid(UE_RBs) = u_;
                 UE_mapping_all_UEs = cw_RB_grid;
%                  for cw_=2:scheduler_params.nCodewords(u_)
%                     UE_mapping_all_UEs(:,cw_) = UE_mapping_all_UEs(:,1);
%                  end
              end
```

We can appreciate how the variable *UE_RBs* assigns UEs sequentially instead of cyclically, therefore the slots are not being assigned properly.

The variable *UE_mapping_all_UEs* stores a matrix of the corresponding RB slots, in which each position is filled with the correspondently assigned UE. This is the variable we have previously represented.

3.3. Proposed Downlink Scheduler

In order to assign a previously designated number of RBs to every UE, first of all we have created a new vector named *LTE_params.UEvals* of length the number of UEs in the system (*LTE_params.nUEs*), in which we will be storing how many RBs we want to assign to the UE in that position of the array.

```
LTE_params.nUE = 4;      % number of user equipments to simulate
LTE_params.UEvals = [4 2 2 5];
```

In the scheduler file, we created a new array *guardo* which will be storing how many RBs have already been assigned to each of the UEs. This way, we can compare this increasing array with the previously defined *UEreal* array, and know whose UE turn is it to be assigned resources.

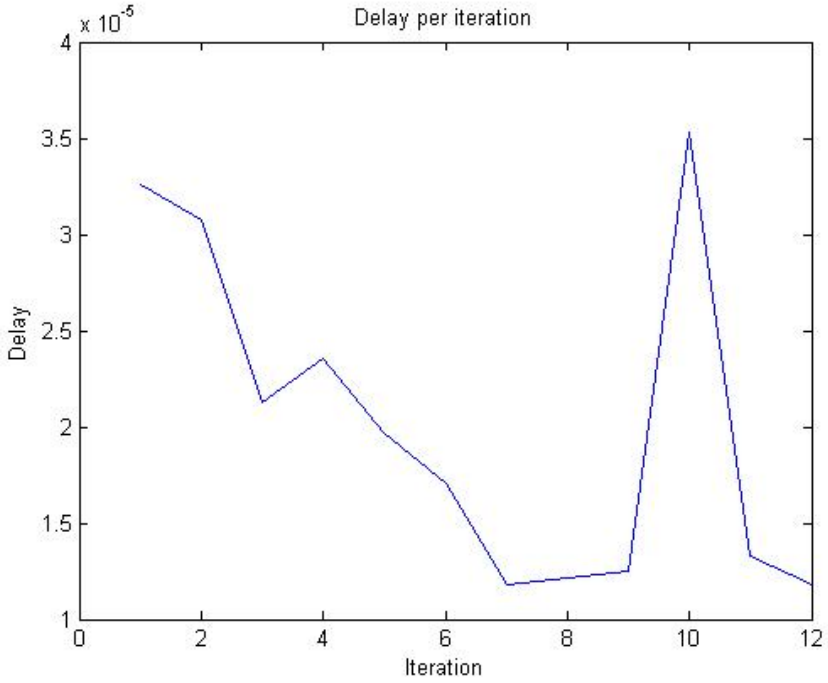
```
classdef roundRobinScheduler < network_elements.lteScheduler

  properties
  end

  methods
    function obj =
roundRobinScheduler(RB_grid_size,Ns_RB,UEs_to_be_scheduled,scheduler_params,CQI_params, UEreal)

        %UEs_to_be_scheduled = nUEs, linea 254 LTE_load_parameters_generate elements
        % Fill in basic parameters (handled by the superclass constructor)
        obj =
obj@network_elements.lteScheduler(RB_grid_size,Ns_RB,UEs_to_be_scheduled,scheduler_params,CQI_
params, UEreal);

        switch scheduler_params.assignment
          case 'static'
            obj.static_scheduler = true;

            % Get a vector of scheduling params (one for each UE)
            % initialized to the values that we want
            obj.UE_static_params = obj.get_initialized_UE_params(scheduler_params,CQI_params);

            % Fill in the RB allocation grid for each user (and codeword)

            UE_mapping_all_UEs = zeros(RB_grid_size,2);
            disp(UE_mapping_all_UEs);

            %for u_=1:obj.nUEs
            cw_RB_grid = UE_mapping_all_UEs;

            guardo = zeros(obj.nUEs,1);
            delay = zeros(1,obj.nUEs);

            for i_=1:2*length(UE_mapping_all_UEs)
              tic;
              counter = 0;
              for a=1:obj.nUEs
                if UEreal(a) ==0
                  counter = counter +1;
                end
              end

              [val,idx] = min(guardo);

              if counter ~= obj.nUEs
              while UEreal(idx) ==0
                guardo(idx) = 99;
                [val,idx] = min(guardo);
              end
```

```
            cw_RB_grid(i_) = idx;
                UEreal(idx)= UEreal(idx)-1;
                guardo(idx) = guardo(idx) +1;
              end %end if counter


              UE_mapping_all_UEs = cw_RB_grid;
              delay(i_) = toc;
              disp(UE_mapping_all_UEs);
            end
            plot(delay)
            title('Delay per iteration')
            xlabel('Iteration')
            ylabel('Delay')


            % Assign the static scheduling parameters for each user
            for u_=1:obj.nUEs
              obj.UE_static_params(u_).UE_mapping = (UE_mapping_all_UEs==u_);
              obj.UE_static_params(u_).assigned_RBs =
squeeze(sum(sum(obj.UE_static_params(u_).UE_mapping,1),2));
            end
```

On each iteration, we will be decreasing the number of resources to be assigned on *UEreal* and increasing the number on *guardo.* The variable *counter* will indicate how many of the UEs have already been provided the required resources, so when it reaches the total number of terminals, we can stop searching for a UE missing resources.

From there on, the algorithm assigns cyclically one RB to each terminal, assigning the turn to be given resources to the terminal with minimum value on the *guardo* array (terminal that has been assigned the least resources up to that iteration).

The performance of the algorithm has also been measured, so we can check how the scheduler performs assigning RBs on each iteration. For the previous example, in which twelve RBs are being assigned, the performance is the following:



3.4. Simulation Results

On this section, we will be performing several simulations and comparing results and performance between both algorithms in terms of throughput. To do so, we will be investigating different scenarios (number of users, transmission scheme, number of users, etc), and different assignation of RBs to the UEs.

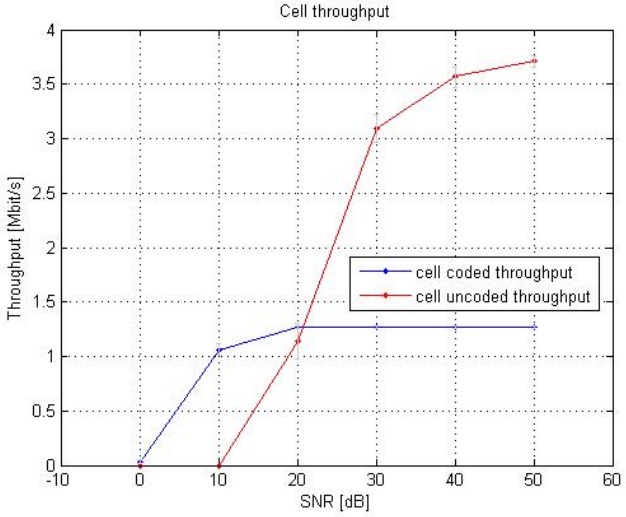
The schedulers to be compared (on a 1.4MHz bandwidth) are:

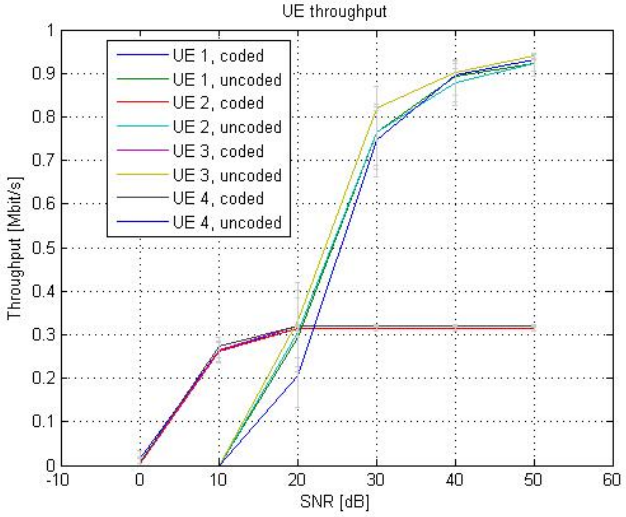
1. Using four users:
   - Initial RoundRobin scheduler (RB assignation of [3 3 3 3]
   - New proposed scheduler with RB assignation of [4 2 2 4]
   - New proposed scheduler with RB assignation of [5 1 1 5]
2. Using six users:
   - Initial RoundRobin scheduler
   - New proposed scheduler with RB assignation of [3 2 1 1 2 3]
   - New proposed scheduler with RB assignation of [6 0 0 0 0 6]

<u>Case 1</u>. 4 users, RB assignation of [3 3 3 3]

RB assignation in time slots:

1	3
2	4
3	1
4	2
1	3
2	4

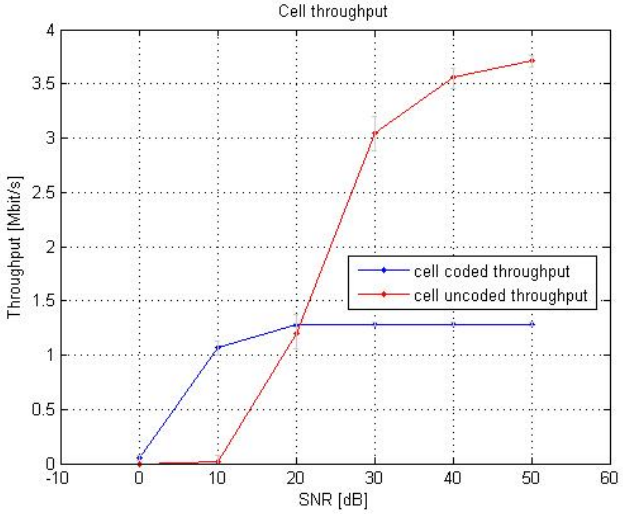


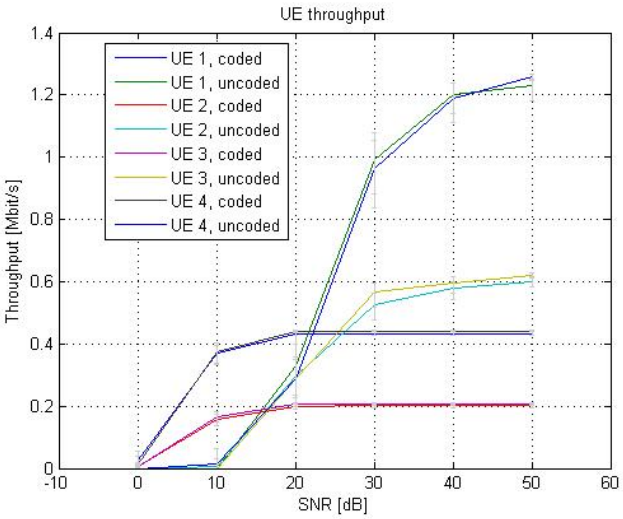


<u>Case 2</u>. 4 users, RB assignation of [4 2 2 4]

RB assignation in time slots:

1	3
2	4
3	1
4	4
1	1
2	4

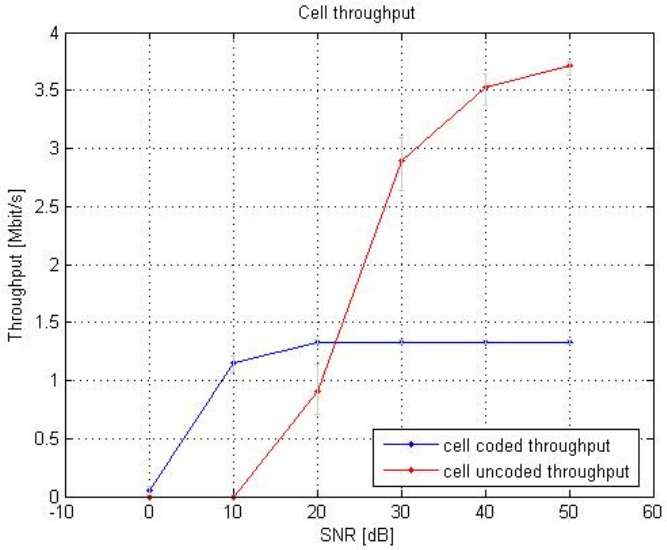


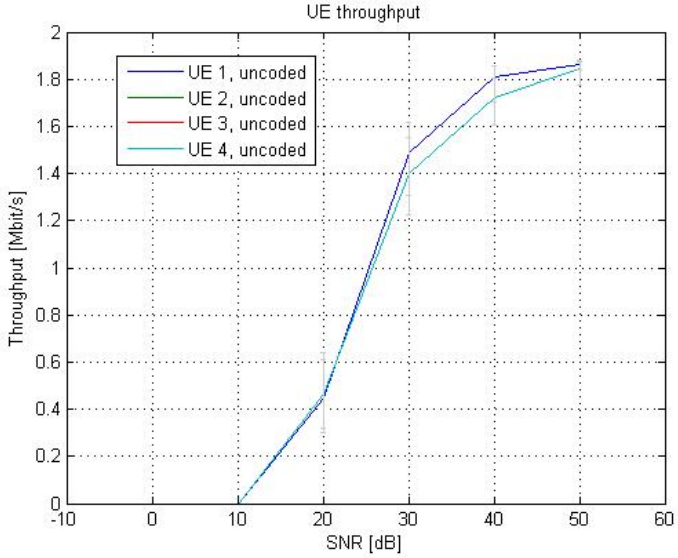


<u>Case 3</u>. 4 users, RB assignation of [5 1 1 5]

RB assignation in time slots:

1	1
2	4
3	1
4	4
1	1
4	4

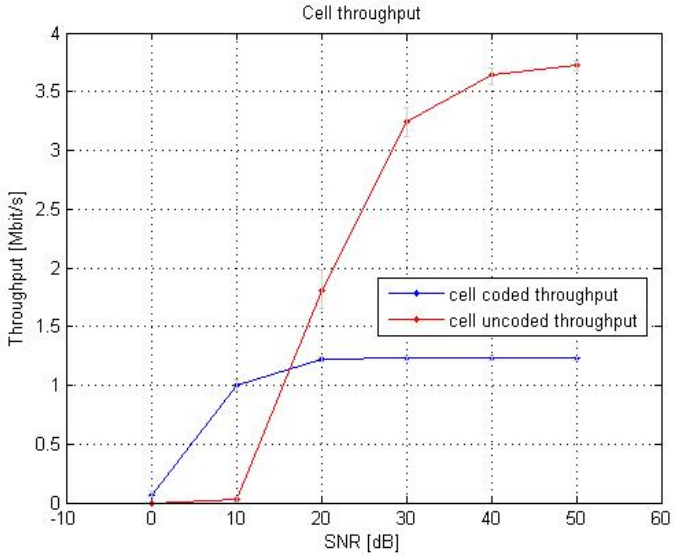


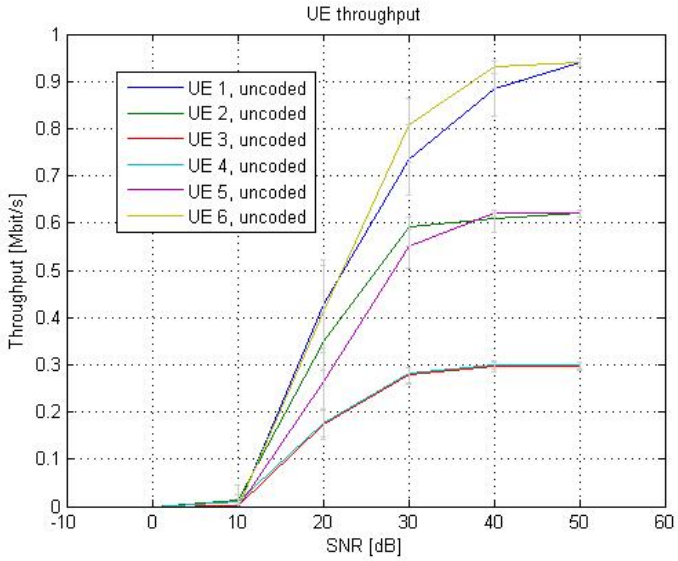


<u>Case 4</u>. 6 users, RB assignation of [3 2 1 1 2 3]

RB assignation in time slots:

1	1
2	2
3	5
4	6
5	1
6	6

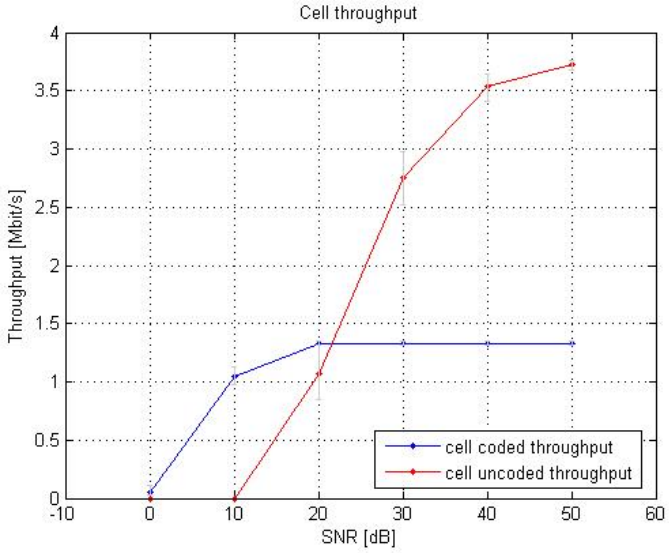


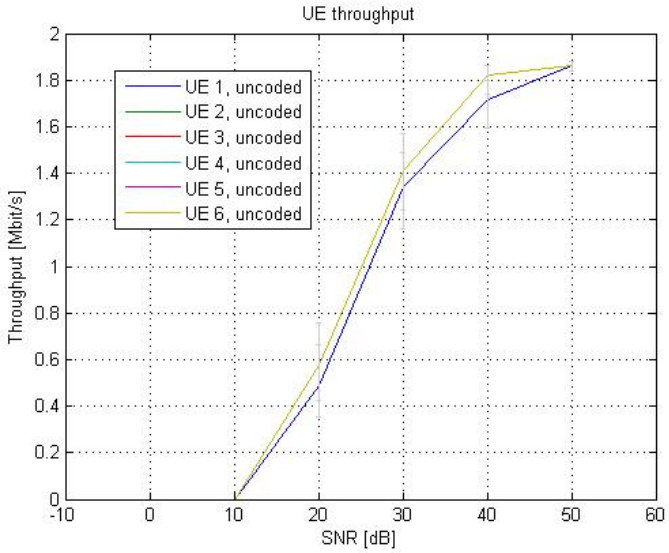


<u>Case 5</u>. 6 users, RB assignation of [6 0 0 0 0 6]

RB assignation in time slots:

1	1
6	6
1	1
6	6
1	1
6	6





3.5. Simulation conclusions

Comparing the results obtained in the previous scenarios, we can easily appreciate how assigning more resource blocks to a specific user, increases significantly its throughput in exchange of lowering that of one of the other users. This can be helpful if for a particular reason we want a specific user or users to have a better connection than the others, and using the designed algorithm, we can provide them with a higher throughput.

It is also worth noting how the cell throughput for uncoded channels gives higher values in low SNR for the simulations with RBs assigned more equally among users. For example, for the case 4, the uncoded channel cell throughput for a SNR of 20dB, the throughput reaches around 1.8Mbits/s whereas for a more unequally assigned case such as case 3, for the same value of SNR of 20dB, the throughput reaches just around 1.3Mbits/s.

4. IMPLEMENTATION OF PROPOSED SOLUTION ON SRSLTE

In this section, we will be implementing the downlink scheduling algorithm designed in the previous simulation and comparing it to Round Robin on srsLTE, which is a real experimental evaluation of an LTE base station developed by SRS (Software Radio System) and designed for fully compliant with LTE Release 8.

This testbed proves to be an essential platform for experimental research and developing a new prototype, allowing developers to validate and assess the performance of LTE.

A testbed can usually include in LTE context from one up to several User Equipments (UEs), one base station (eNodeB) and an Evolved Packet Core (EPC). These components are provided as commercial packaged solutions adapted to satisfy the needs of the the purchasing organization. The performance on this kind of solutions is splendid and its functionality has been substantially validated. [6]

Software-Defined Radio (SDR) is an approved notion used for carrying out radio equipment in software, by means of commercially available, cheap computers and radio frontends. Over the last few years, SDR technologies have been acquiring approval as a tool to build close-to-reality testbeds that can be used for experimentation and research. If the user of the platform has access to the source code, as it is with our case implemented on open source, the testbed can be modified quite simply which can be used for instrumenting the stack for the desired functionality becomes easy. [6]

srsLTE has been designed as an open source library dedicated to the physical layer of LTE in Release 8. It is built for highest modularity and allows to reuse the needed code keeping a low external dependency. The software is coded in ANSI C and has been optimized for maximum performance. [6]

This software includes: **[7]**

- srsUE: a complete SDR LTE UE application featuring all layers from PHY to IP
- srsENB: a complete SDR LTE eNodeB application
- srsEPC: a light-weight LTE core network implementation with MME, HSS and S/P-GW
- a highly modular set of common libraries for PHY, MAC, RLC, PDCP, RRC, NAS, S1AP and GW layers.

Common Features:

- LTE Release 8 compliant (with selected features of Release 9)
- FDD configuration
- Tested bandwidths: 1.4, 3, 5, 10, 15 and 20 MHz
- Transmission mode 1 (single antenna), 2 (transmit diversity), 3 (CCD) and 4 (closed-loop spatial multiplexing)
- Frequency-based ZF and MMSE equalizer
- Evolved multimedia broadcast and multicast service (eMBMS)
- Highly optimized Turbo Decoder available in Intel SSE4.1/AVX2 (+100 Mbps) and standard C (+25 Mbps)
- MAC, RLC, PDCP, RRC, NAS, S1AP and GW layers
- Detailed log system with per-layer log levels and hex dumps
- MAC layer wireshark packet capture
- Command-line trace metrics
- Detailed input configuration files

srsUE Features:

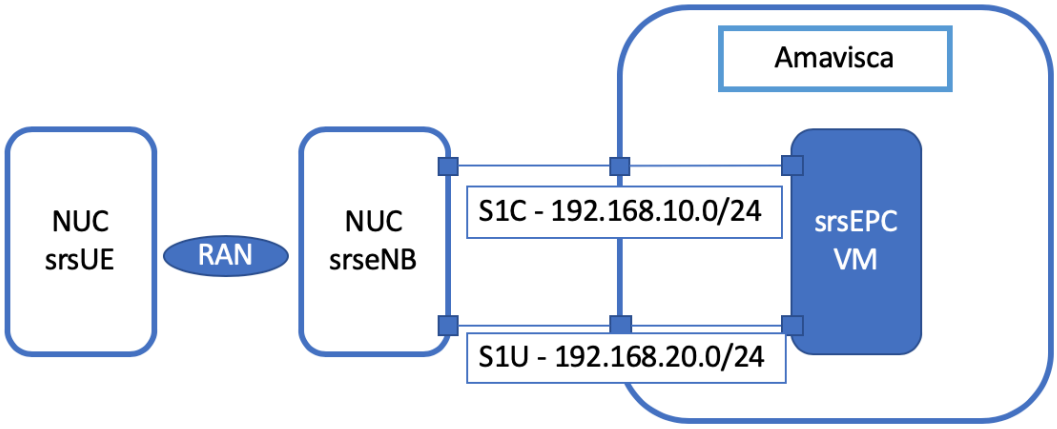
- Cell search and synchronization procedure for the UE
- Soft USIM supporting Milenage and XOR authentication
- Hard USIM support using PCSC framework
- Virtual network interface *tun_srsue* created upon network attach
- 150 Mbps DL in 20 MHz MIMO TM3/TM4 configuration in i7 Quad-Core CPU.
- 75 Mbps DL in 20 MHz SISO configuration in i7 Quad-Core CPU.
- 36 Mbps DL in 10 MHz SISO configuration in i5 Dual-Core CPU.

srsENB Features:

- Round Robin MAC scheduler with FAPI-like C++ API
- SR support
- Periodic and Aperiodic CQI feedback support
- Standard S1AP and GTP-U interfaces to the Core Network
- 150 Mbps DL in 20 MHz MIMO TM3/TM4 with commercial UEs
- 75 Mbps DL in SISO configuration with commercial UEs
- 50 Mbps UL in 20 MHz with commercial UEs

This software has been implemented using the following Control Plane topology:

We can appreciate the following Data Plane topology:



Where srsEPC Virtual Machine is located inside Amavisca

4.1. Round Robin Scheduling

As we have previously indicated in the features of the different elements of srsLTE, the default implemented scheduler is the Round Robin scheduler. This scheduler is set in the eNodeB, under the path *srsLTE/srsenb/src/mac,* on the file scheduler_metric.cc. Specifically, the function named new_tti will be the one in charge of assigning the resources to the different UEs according to the scheduling algorithm that we design.

Therefore, in this scheduler, the default downlink scheduling algorithm (Round Robin), will be assigning resources to each of the users on an iterative basis.

```
void dl_metric_rr::new_tti(std::map<uint16_t,sched_ue> &ue_db, uint32_t start_rbg, uint32_t
nof_rbg, uint32_t nof_ctrl_symbols_, uint32_t tti)
{
 total_rbg = start_rbg+nof_rbg;
 for (uint32_t i=0;i<total_rbg;i++) {
  if (i<start_rbg) {
   used_rbg[i] = true;
  } else {
   used_rbg[i] = false;
  }
 }
 available_rbg = nof_rbg;
 used_rbg_mask = calc_rbg_mask(used_rbg);
 current_tti = tti;
 nof_ctrl_symbols = nof_ctrl_symbols_;

 if(ue_db.size()==0)
  return;

 // give priority in a time-domain RR basis
 uint32_t priority_idx = current_tti % ue_db.size();
 std::map<uint16_t, sched_ue>::iterator iter = ue_db.begin();
 std::advance(iter,priority_idx);
 for(uint32_t ue_count = 0 ; ue_count < ue_db.size() ; ++iter, ++ue_count) {
  if(iter==ue_db.end()) {
   iter = ue_db.begin(); // wrap around
  }
  sched_ue *user = (sched_ue*) &iter->second;
  user->dl_next_alloc = apply_user_allocation(user);
 }
}
```

We can easily appreciate how an iterator is created, which starts pointing at the beginning of the ue_db vector, which stores each of the different UEs in use.

A loop is created which will be iterating each of the UEs, for the length of ue_db.size() (which indicates the total number of UEs). In this loop, a user is created, pointing to the iterator -> second value (in an iterator for c++, iterator->first points to the key member of the iterator, and iterator -> second, points to the value of the mapping).

Once we have created this user pointing to the value we are mapping, the function apply_user_allocation(user) is called in order to assign resources for the user pointed by the iterator. This way, in each tti, we are assigning the resources to all the available user in an iterative manner.

4.1.1. RR Evaluation Results

First of all, we open three different terminal windows, in which we will be connecting to the virtual machine containing the ePC, the eNodeB, and an UE respectively. In order to connect to the virtual machine and the ePC, first we need to start a remote session to Amavisca, in which the virtual machine with the ePC is located.

Once we have initiated the connections and made the proper binding connections between the mentioned elements, we can start the testbed.

In the following step, we initiate the ePC, which will show the following screen:

```
user@ubuntu:~/srsLTE/build/srsepc/src$ sudo ./srsepc epc.conf
[sudo] password for user:

Built in Release mode using commit 9def82d on branch master.


---  Software Radio Systems EPC  ---

Reading configuration file epc.conf...
HSS Initialized.
MME GTP-C Initialized
MME Initialized.
SP-GW Initialized.
```

We can appreciate how the different software radio systems are initialized according to the configuration file included in the EPC.

Once the ePC is started, we can initiate the eNodeB, which will connect to the ePC and they show the following information of the screen reflecting the correct connection between both elements.

We can check how the ePC reads the connection:

```
user@ubuntu:~/srsLTE/build/srsepc/src$ sudo ./srsepc epc.conf
[sudo] password for user:

Built in Release mode using commit 9def82d on branch master.


---  Software Radio Systems EPC  ---

Reading configuration file epc.conf...
HSS Initialized.
MME GTP-C Initialized
MME Initialized.
SP-GW Initialized.
Received S1 Setup Request.
S1 Setup Request - eNB Name: srsenb01, eNB id: 0x19b
S1 Setup Request - MCC:208, MNC:93, PLMN: 194617
S1 Setup Request - TAC 7, B-PLMN 0
S1 Setup Request - Paging DRX 2
Sending S1 Setup Response
```

This image shows how the ePC is receiving a setup request from the eNodeB (srsenb01), this eNB is given the id 0x19b and different parameters are set up.

And the eNodeB reflects the following:

```
sergio_fuente@ue-oai-nuc:~/srsLTE/build/srsenb/src$ sudo ./srsenb enb.conf.example
[sudo] password for sergio_fuente:

Built in Release mode using commit 5c088d7 on branch master.

---  Software Radio Systems LTE eNodeB  ---

Reading configuration file enb.conf.example...
[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_3.13.1.0-release
Opening USRP with args: type=b200,master_clock_rate=30.72e6
[INFO] [B200] Detected Device: B210
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 30.720000 MHz...
[INFO] [B200] Actually got clock rate 30.720000 MHz.
Setting frequency: DL=2685.0 Mhz, UL=2565.0 MHz
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Setting Sampling frequency 5.76 MHz

==== eNodeB started ===
Type <t> to view trace
```

We can appreciate how the eNodeB reads the configuration from the ePC and registers itself before being fully started.

Once the connection between ePC and eNodeB has been fully established, we can try initiating an UE to show the assignment of resources provided by the initial RoundRobin scheduling algorithm.

On the eNodeB side, once we start one UE (unfortunately we only have one UE available for testing), we can read the following:

```
==== eNodeB started ===
Type <t> to view trace
t
Enter t to stop trace.
RACH:  tti=6261, preamble=0, offset=1, temp_crnti=0x46
User 0x46 connected

------DL-----------------------------UL-----------------------------------
rnti  cqi    ri   mcs  brate   bler   snr   phr   mcs  brate   bler    bsr
46   12.9     0  5.50  3.74k     0%  32.3  40.0  10.5  46.3k     0%    0.0
46   15.0     0   0.0   0.0      0%  31.4  40.0  15.0  12.8k     0%    0.0
46   15.0     0   0.0   0.0      0%  32.1  40.0  15.0  12.8k     0%    0.0
46   15.0     0   0.0   0.0      0%  32.5  40.0  15.0  12.8k     0%    0.0
46   15.0     0   0.0   0.0      0%  32.7  40.0  15.0  12.8k     0%    0.0
46   15.0     0   0.0   0.0      0%  32.8  40.0  15.0  12.8k     0%    0.0
```

Here we can appreciate how the RACH (Random Access Procedure) is used in order to synchronize the UE, in which a TTI is established, with an offset and a crnti that shows the id of the UE connecting to the network. Then, packets start being assigned to the UE with rnti 46 (the only UE we have).

On the UE side, we can see the following:

```
sergio_fuente@f5gw-nuc-2:~/srsLTE/build/srsue/src$ sudo ./srsue ue.conf.example
[sudo] password for sergio_fuente:
Reading configuration file ue.conf.example...

Built in Release mode using commit 9def82d on branch master.

---  Software Radio Systems LTE UE  ---

Opening RF device with 1 RX antennas...
[INFO] [UHD] linux; GNU C++ version 4.8.4; Boost_105400; UHD_3.13.1.0-release
tOpening USRP with args: type=b200,master_clock_rate=30.72e6
[INFO] [B200] Detected Device: B210

[INFO] [B200] Operating over USB 3.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 30.720000 MHz...
[INFO] [B200] Actually got clock rate 30.720000 MHz.
Waiting PHY to initialize...
...
Attaching UE...
Enter t to stop trace.
Searching cell in DL EARFCN=3400, f_dl=2685.0 MHz, f_ul=2565.0 MHz
.
Found Cell:  PCI=1, PRB=25, Ports=1, CFO=0.6 KHz
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Found PLMN:  Id=20893, TAC=7
Could not find Home PLMN Id=00101, trying to connect to PLMN Id=20893
--- disconnected ---
Random Access Transmission: seq=0, ra-rnti=0x2
Random Access Complete.     c-rnti=0x46, ta=1
RRC Connected
Network attach successful. IP: 192.168.200.2
Software Radio Systems LTE (srsLTE)

--Signal-------------DL-----------------------------UL----------------------
  rsrp    pl    cfo   mcs   snr turbo  brate   bler   ta_us  mcs   buff  brate   bler
   -45    45    681   5.3    30  0.91   3.7k     0%   0.0    11    0.0    51k     0%
   -43    43    683   6.5    37   1.0    0.0     0%  0.52    15    0.0    13k     0%
   -72    72    682   6.5    38   1.0    0.0     0%  0.52    15    0.0    13k     0%
   -72    72    680   6.5    37   1.0    0.0     0%  0.52    15    0.0    13k     0%
   -72    72    679   6.5    38   1.0    0.0     0%  0.52    15    0.0    13k     0%
   -72    72    679   6.5    37   1.0    0.0     0%  0.52    15    0.0    13k     0%
   -72    72    680   6.5    37   1.0    0.0     0%  0.52    15    0.0   8.5k     0%
   -72    72    680   6.5    38   1.0    0.0     0%  0.52    15    0.0    13k     0%
   -72    72    680   6.5    37   1.0    0.0     0%  0.52    15    0.0    13k     0%
```

We can check how the UE is configured and attached to the network, being assigned the IP 192.168.200.2 and the crnti 0x46. After that, we can check how the different resource packets are assigned, depending on how many resources this UE is requesting.

4.2. Proposed Downlink Scheduler

We will be validating the designed network slicing solution on srsLTE, which works as a real experimental evaluation of an LTE base station, in order to implement the algorithm on ta testbed which would be the previous step before moving to the real world software implementation. Specifically, in this solution we will be focusing on downlink scheduling.

In order to implement a different scheduler in srsLTE, first of all we need to locate the file that includes the scheduler in the testbed. This file is *scheduler_metric.cc* on the path *srsLTE/srsenb/src/mac/* inside the eNodeB.

In this file, as we have mentioned previously, the function new_tti(…) will be in charge of assigning resources to the different UEs. Therefore, in this section we will be modifying the provided Round Robin algorithm and aiming to create a new network slicing aware radio scheduler.

After changes to the indicated function, we have the following:

```
void dl_metric_rr::new_tti(std::map<uint16_t,sched_ue> &ue_db, uint32_t start_rbg, uint32_t nof_rbg,
uint32_t nof_ctrl_symbols_, uint32_t tti)
{
   total_rbg = start_rbg+nof_rbg;
   for (uint32_t i=0;i<total_rbg;i++) {
      if (i<start_rbg) {
         used_rbg[i] = true;
      } else {
         used_rbg[i] = false;
      }
   }
   available_rbg = nof_rbg;
   used_rbg_mask = calc_rbg_mask(used_rbg);
   current_tti = tti;
   nof_ctrl_symbols = nof_ctrl_symbols_;

   if(ue_db.size()==0)
      return;


   int resources_to_assign [3] = { 5, 2, 3};
   uint32_t priority_idx =0;
   int a = sizeof(resources_to_assign)/sizeof(int);
   for(int i = 0 ; i< a; i++){
   priority_idx += resources_to_assign[i];
   }

//la clave es el TTI que va subiendo en cada iteracion
   priority_idx = current_tti % priority_idx; //LO PONEMOS EN MODULO EL NUMERO DE SLOTS TOTAL

   //printf("Vamos a ver el valor del index: %d \n", priority_idx);

   std::map<uint16_t, sched_ue>::iterator iter = ue_db.begin();
   int vector_suma[a];

   for (int i=0; i< a; i++){
      if(i==0){
         vector_suma[i]=resources_to_assign[i];
      }else{
         vector_suma[i]=resources_to_assign[i]+vector_suma[i-1];
      }
      //printf("%d \n", vector_suma[i]);

   `
```

```
  }

  for(int i=0; i<a; i++){
    if(priority_idx < vector_suma[i]){
      //printf("Asignamos recursos al UE numero %d \n", i);
      if(i ==0){
          sched_ue *user = (sched_ue*) &iter->second;    // iter indica el UE en el que estamos.
          user->dl_next_alloc = apply_user_allocation(user);
      }else{
        std::advance(iter,i);
        //sched_ue *user = (sched_ue*) &iter->second;    // iter indica el UE en el que estamos.
        //user->dl_next_alloc = apply_user_allocation(user);
      }
      priority_idx =9999;
      break;
    }
  }

}
```

In this algorithm, we are assigning slots according to the priorities we want to set. We have created a new vector that will be storing the different priorities we want to assign (same way we were doing in the Matlab simulation). This vector will be storing the number of slots we want to assign to each of the available UEs. The length of the vector is that of the number of UEs available.

The *priority_idx* variable has also been modified, so that it stores the TTI number inside the modulus of the total number of slots we will be assigning (the sum of all the values inside the vector).

Then a new vector has been created, named *vector_suma*, which will be storing in each position the sum of the total number of slots assigned in the previous array up until that same position.

Once we have this vector created, we can compare the priority_idx in a way that if it is lesser than the position of this new array, it means it is the turn of that UE to be assigned resources. In case it were bigger, we would just skip to the next number of the array, which will be increasing so that the condition will always be satisfied.

4.2.1. Results on Proposed Scheduler

In order to prove the performance of the new scheduler, we will be using the tool *iperf*, which allows to measure the bandwidth of a specific network. Therefore, we will be measuring the dedicated bandwidth between the ePC and the UE, once a connection has been established between ePC and eNodeB, and the UE has been assigned an IP from which it can reach the SPGW inside the virtual machine.

Since we are measuring a downlink scheduling algorithm, we will set iperf listening on the UE, and check the performance of the bandwidth from the ePC.

Knowing that the total available bandwidth on the experimental evaluation is set to around 16Mb/s, we will be taking different measurements by means of variating the vector which stores the number of slots we will be assigning to every UE.

Since we only have one UE available in the lab for testing, every measurement will be taken with respect to the first UE set in the *resources_to_assign* vector.

Overall, we can appreciate the following throughput on a particular node, respect to the assigned bandwidth we set on the algorithm:

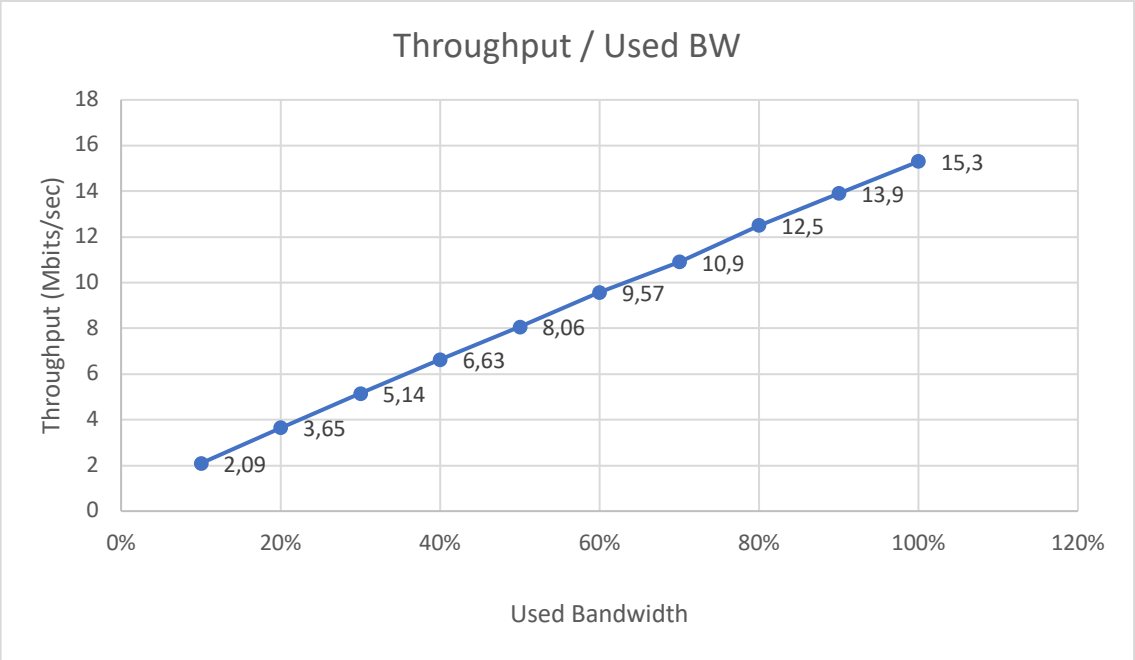


Figure 1. Values measured using iperf with the respective priority vector to achieve the desired % of BW.

It is noticeable how it starts at a throughput of 2,09 Mbits/sec for an assignment of 10% of the total bandwidth, and from there on it increases on around 1,5 Mbits/sec for every new 10% we assign, until we reach a total bandwidth of 15,3 Mbits/sec.

This graph has been measured using two users, since when we have taken the same measurements assigning the same % of bandwidth using a different number of users, the results in throughput have been the same as the measured case.

4.2.2. Measurements of Throughput with respect to time

In this section, using the new proposed scheduler, we will be measuring the throughput with respect to the time on different scenarios. Namely, we will be measuring the throughput over one minute (divided in five second time slots), on a UE that has been assigned 1/n of the total bandwidth, where n equals 2, 3, 4, 5 and 6 respectively on each case.

Case 1. n=2, 2 users assigned the same amount of bandwidth

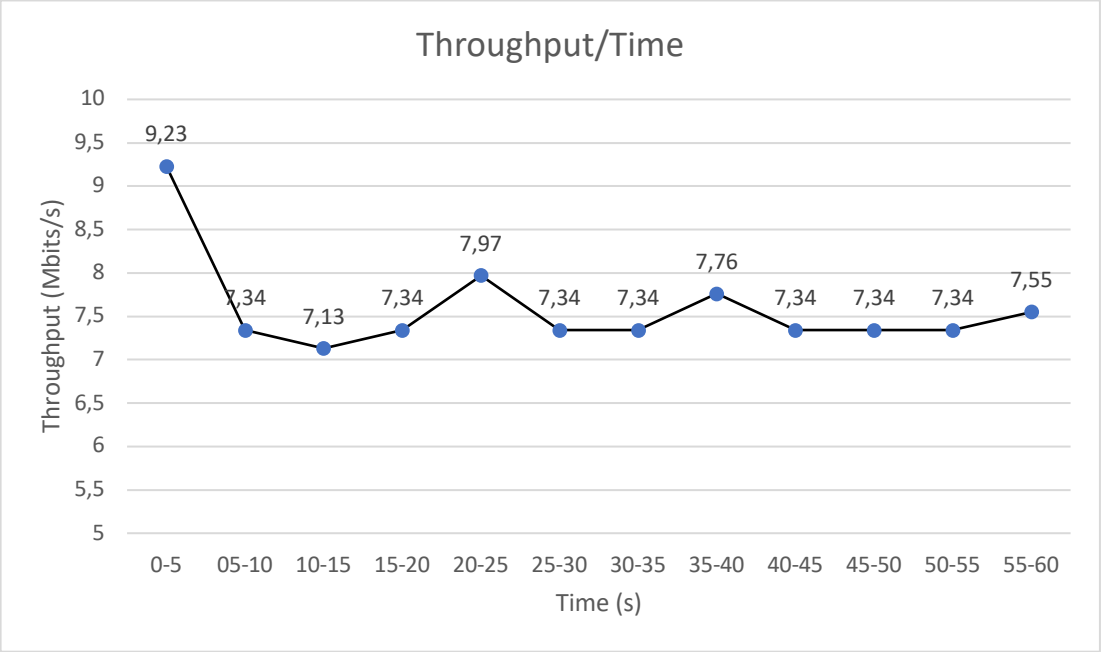


Figure 2.

Total average bandwidth over one minute = 7,55 Mbits/s

Case 2. n=3, 3 users assigned the same amount of bandwidth

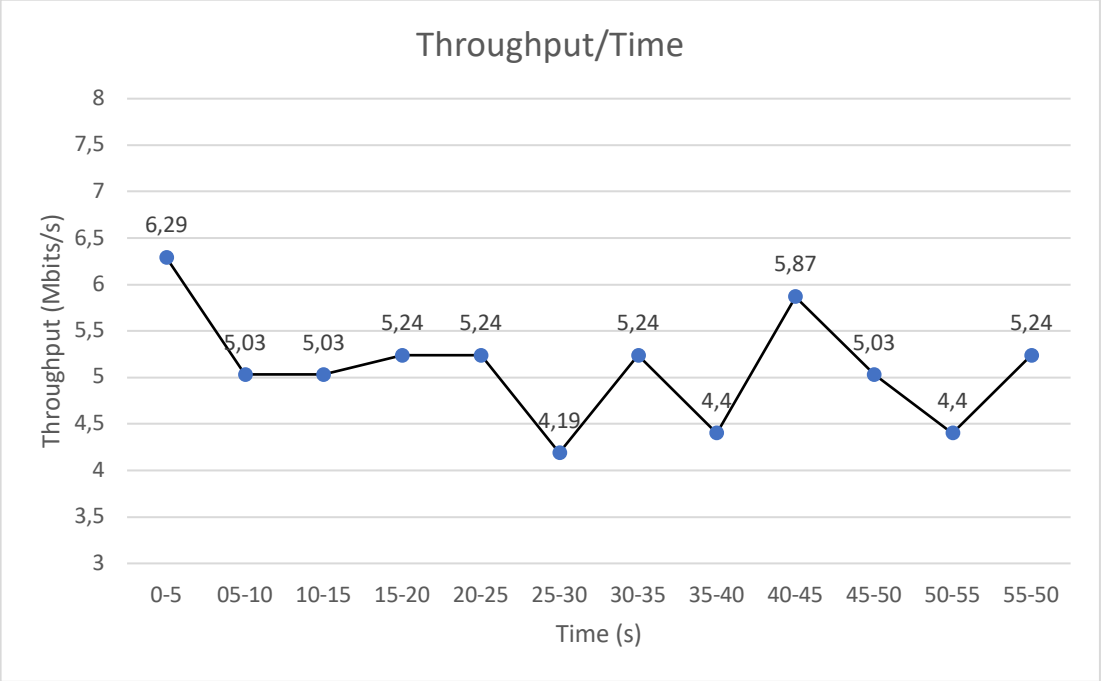


Figure 3.

Total average bandwidth over one minute = 5,1Mbits/s

Case 3. n=4, 4 users assigned the same amount of bandwidth

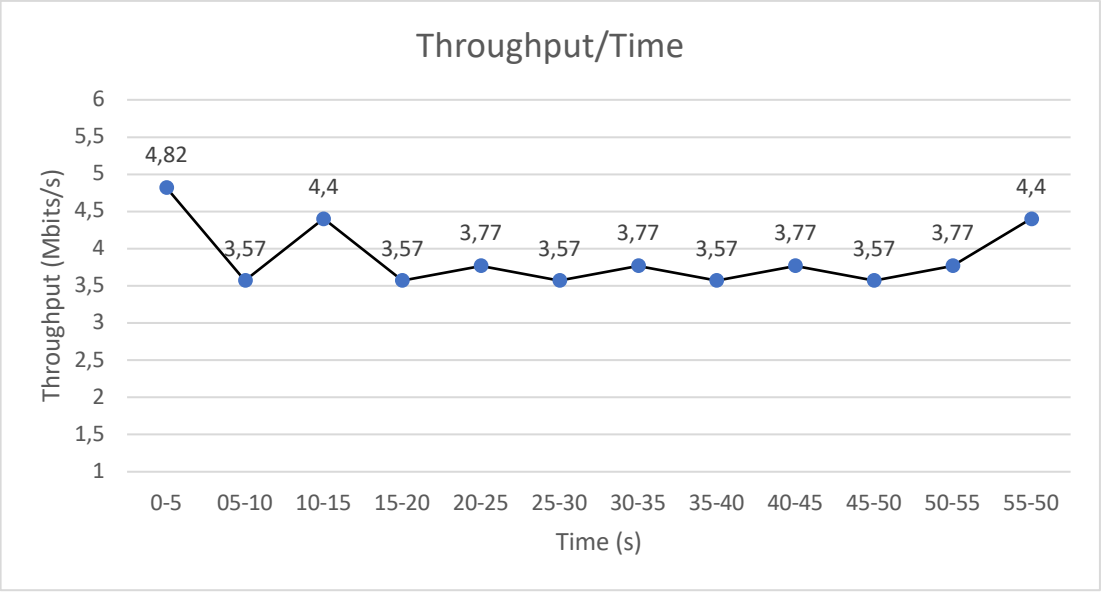


Figure 4.

Total average bandwidth over one minute = 3,84 Mbits/s

Case 4. n=5, 5 users assigned the same amount of bandwidth

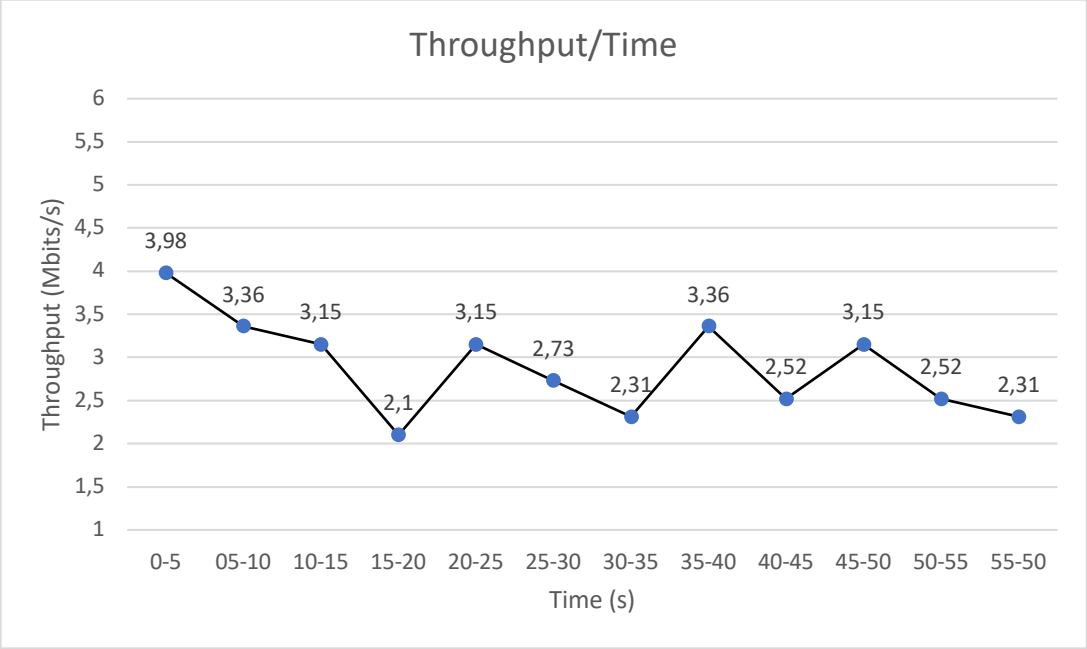


Figure 5.

Total average bandwidth over one minute = 2,89 Mbits/s

Case 5. n=6, 6 users assigned the same amount of bandwidth

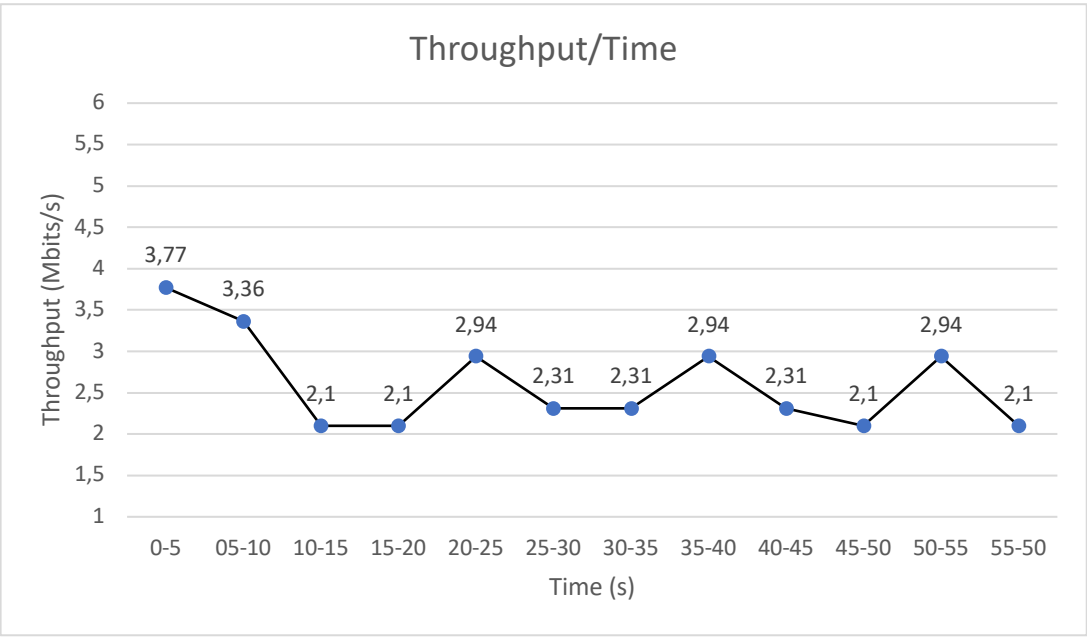


Figure 1

Total average bandwidth over one minute = 2,6 Mbits/s

It is worth noting how different throughput rates are assigned on every time slot, where the scheduler aims to assign a proportional amount of the total bandwidth over the specified time for each one of the numbers of UEs.

4.2.3. Conclusions

The previous graphs showed above show the performance and implementation of this algorithm. Both of these are taken into account on a new main factor, which is the priority vector we have created in order to manage the assignment of slots to each user.

This allows us assigning to every user the percentage of bandwidth we desire, no matter the number of users in the system, nor the quality of the channel (measuring on the srsLTE simulator, real life measurements may introduce delays due to the performance of the bandwidth on every slot for every user).

The algorithm detects the desired assignation and will proceed to assign slots to each of the users according to the priority vector, getting more precise on the exact assignation during time. Therefore, ideally if we were to run the algorithm for a longer time, it will get us an even more precise bandwidth assignment on srsLTE.

5. SOCIO-ECONOMIC ENVIRONMENT

The implementation on the real world of this thesis could cause an impact on different environments. Mainly, we can highlight:

Social environment

The new designed algorithm allows for network slicing and therefore, assigning slots of the available bandwidth to the desired user. This means that on a crowded event or situation, such as a sports event or main touristic areas, we would have the ability of prioritizing resources to whichever user we decide.

As an example, in a situation where two users (A and B), attend together to a football game, the implementation of this scheduler would allow user A to be assigned more slots than to user B, and consequentially this would translate to a better data connection for user A due to a higher bandwidth assignment.

It is also worth noting that, for user A to have a better connection, the newly assigned bandwidth has to be taken from the overall bandwidth. Since we only have a limited bandwidth to spare, assigning more slots to user A would mean taking them from the total of the rest of the users.

Economic-Ethical environment

From the economic point of view, this new implementation could mean a new source of revenue for telecommunications companies. Users demanding a faster data connection or a prioritization of its connection with respect to the other users, would be required a higher disbursement.

The application of this measure would be ethically controversial, since it could mean an interference with the European Union laws of net neutrality, in which all end-users must be treated equally by the network operator, prohibiting the prioritization of a user over the other. The implementation of this thesis would implicate that users with higher income or financial capability would be able to afford a better and more effective connection, whereas the users that cannot afford for a stable assignation of bandwidth would only see their internet connection worsen in favor of these other premium users after the implementation.

6. REGULATORY FRAMEWORK

Implementing this algorithm over the public network would clearly interfere with the latest regulations applied by the European Commission on net neutrality.

The European Union regulation on its article 3 of EU Regulation 2015/2120 states that ISP (Internet Service Providers) should treat all data transmitted over the Internet equally. ISPs are forbidden from blocking or slowing Internet traffic unless it is specifically necessary. [8]

This regulation also states that all users are free to access and distribute information and content, run applications and use services of their choice, and are entitled to the same treatment and provision by the service providers, independently of who are the sender or receiver of this traffic. [8]

The non-compliance with this regulation can lead up to two million euros in fines for the ISPs breaking it.

Taking all of this into account and knowing that this thesis is based on the design of a network slicing algorithm for the downlink scheduler, implementing this solution inside the European Union by an Internet Service Operator would mean breaking the current legislation on net neutrality.

The design and implementation of this scheduler has been created, as we have previously explained, using free software tools and simulators. Specifically, the first simulation has been designed on the free open version of the "LTE_Link_Level_1.7_r1089" Matlab simulator implemented by the University of Vienna, and the posterior implementation has been designed on the real software simulator "srsLTE", which is an open source SDR LTE software suite for Software Radio Systems (SRS).

Therefore, no specific license or permission has been required on the implementation of this thesis.

Since the results and implementation of this thesis is still a field to be explored and be developed specially upon the arrival of 5G, there is still much work left to do with this technology, which will imply more regulation and standardizations to arrive in the upcoming years.

7. CONCLUSIONS AND FUTURE WORK

In this thesis I have introduced an overview of the concept of network slicing and its main implications when applied towards Long Term Evolution technologies. We have reviewed the main definitions and implications of LTE, regarding its protocols, physical channels and reference signals, so the reader can understand the basis of this technology, before moving to its meaning in terms of real case scenarios.

The focus of this thesis has been set on scheduling algorithms, specifically working on a new downlink scheduling algorithm. We have appreciated how a Round Robin scheduling algorithm divides the bandwidth in equal slots assigning resources cyclically among all users connected to the base station. I have introduced a new scheduling algorithm based on slicing the network into "bandwidth slots". We can afterwards assign as many of these slots as we want to each of the users.

We have first appreciated on a simulation on Matlab how the throughput varies from one algorithm to the other, and how we can assign an effectively bigger throughput to a desired user. After completing the simulation successfully, we moved on to a real software implementation and simulated the proposed algorithm on srsLTE. Once again, we compared using the iperf tool the results in terms of throughput on the new proposed algorithm and checked how the throughput varies depending the amount of bandwidth or slots we assign to each one of the users, and also the comparation of throughput with respect to time. Therefore, the initial hypothesis set on the simulation has been confirmed and bandwidth slotting achieved.

Network slicing is still a technology to be fully developed focusing on 5G, therefore plenty of research can be made on this field. Network virtualization can allow operators to place different network requirements in terms of functionality, for different cases such as the use of a smart home, the arrival of the Internet of Things, connected cars, etc.

Following the implementation of this thesis, there is still research to be made focusing on uplink connections between the users and the base stations.

We have also focused on this thesis on developing a downlink scheduling algorithm not taking into account channel quality, consequently the logical following step for this thesis could be improving this algorithm so that instead of assigning "slots" of the available bandwidth, we could assign effective bandwidth to each user taking into account channel quality. This could imply assigning a bigger number of slots to a user that doesn't meet the bandwidth assignment set by the operator.

The results and conclusions of this thesis have been partly published under the paper *Experimenting with open source tools to deploy a multi-service and multi-slice mobile network* by Elsevier, a global information analytics business and one of the world's most prestigious and major providers of scientific and technical information.

REFERENCES

[1] “Downlink Scheduling in 3GPP Long Term Evolution (LTE).pdf.” .
[2] S. Alotaibi, “3GPP Long Term Evolution LTE Scheduling,” p. 80.
[3] “LTE in a Nutshell - Physical Layer.pdf.” .
[4] C. Deniz, O. G. Uyan, and V. C. Gungor, “On the performance of LTE downlink scheduling algorithms: A case study on edge throughput,” *Comput. Stand. Interfaces*, vol. 59, pp. 96–108, Aug. 2018.
[5] “LTE-in-a-Nutshell-Protocol-Architecture.pdf.” .
[6] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, “srsLTE: an open-source platform for LTE evolution and experimentation,” in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization - WiNTECH ’16*, New York City, New York, 2016, pp. 25–32.
[7] “GitHub - srsLTEsrsLTE Open source SDR LTE software suite from Software Radio Systems (SRS).pdf.” .
[8] "https://berec.europa.eu/eng/netneutrality/"