

QPPSim

A Tool for Rapid Prototyping of LTE QoS, Priority and Pre-emption

Version 1.0

*Wireless Networks Division
Communications Technology Laboratory*

March 2019



U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

Contents

1	License	3
2	Quick Start	4
2.1	Requirements	4
2.2	Install	4
2.3	Creating Scenarios	4
2.4	Run	5
3	Introduction	6
3.1	Features	6
3.2	Limitations	7
4	QPP Modules	8
4.1	Introduction	8
4.2	Bearer	8
4.3	Scheduler	9
4.4	Access Control	11
4.5	Pre-emption	11
4.6	QoS Monitor	12
4.7	DES Core	12
5	Traces	14
5.1	Introduction	14
5.2	Application Trace	15
5.3	Bearer Trace	15
5.4	ARP Trace	15
5.5	QoS Trace	17
5.6	Topology Trace	18
6	Scenario Creation	19

Chapter 1

License

NIST-developed software is provided by NIST as a public service. You may use, copy and distribute copies of the software in any medium, provided that you keep intact this entire notice. You may improve, modify and create derivative works of the software or any portion of the software, and you may copy and distribute such modifications or works. Modified works should carry a notice stating that you changed the software and should note the date and nature of any such change. Please explicitly acknowledge the National Institute of Standards and Technology as the source of the software.

NIST-developed software is expressly provided "AS IS." NIST MAKES NO WARRANTY OF ANY KIND, EXPRESS, IMPLIED, IN FACT OR ARISING BY OPERATION OF LAW, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT AND DATA ACCURACY. NIST NEITHER REPRESENTS NOR WARRANTS THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT ANY DEFECTS WILL BE CORRECTED. NIST DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF THE SOFTWARE OR THE RESULTS THEREOF, INCLUDING BUT NOT LIMITED TO THE CORRECTNESS, ACCURACY, RELIABILITY, OR USEFULNESS OF THE SOFTWARE.

You are solely responsible for determining the appropriateness of using and distributing the software and you assume all risks associated with its use, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and the unavailability or interruption of operation. This software is not intended to be used in any situation where a failure could cause risk of injury or damage to property. The software developed by NIST employees is not subject to copyright protection within the United States.

Chapter 2

Quick Start

2.1 Requirements

- Python 3.4.¹
- Numpy.

2.2 Install

To install the software, clone the git repository in a directory where you have write permissions. Administrator privileges are not required.

2.3 Creating Scenarios

Scenario files are composed of six sections:

1. Heading section that imports all the required packages:

```
import qppsim.AppProfile
import qppsim.Des
import qppsim.Time
import qppsim.Ue
import qppsim.accesscontrol.AccessControlSample
import qppsim.preemption.PreemptionDummy
import qppsim.prioritypolicy.PriorityPolicySample
import qppsim.qosmonitor.QosMonitorDefault
```

2. Creation of the Discrete Event Simulator (DES), and configuration of policies, simulation time, random seed, and some parameters for the configuration of the Long Term Evolution (LTE) network:

```
des = qppsim.Des.Des(
    seed=1,
    num_rbs=50,
    stop_time=qppsim.Time.Time(seconds=35),
    rtx_threshold=0.1,
    bearer_stats_window_size=qppsim.Time.Time(seconds=10),
    access_control_policy=qppsim.accesscontrol.AccessControlSample.AccessControlSample(num_rbs=50),
    preemption_policy=qppsim.preemption.PreemptionDummy.PreemptionDummy(),
```

¹other 3.x versions may work, but the software hasn't been tested.

```

priority_policy=qppsim.prioritypolicy.PriorityPolicySample.PriorityPolicySample(),
qos_monitor=qppsim.qosmonitor.QosMonitorDefault.QosMonitorDefault(),
trace_qos=True,
preempt_qos=False
)

```

3. The definition of the application profiles:

```

cbr_profile = qppsim.AppProfile.AppProfile(
    "CBR_Application_Template",
    ["constant", [750]], // Packet size
    ["constant", [0.0005]], // Inter-packet interval
    ["constant", [10000000]], // Amount of packets in one session
    ["constant", [0]] // Inter-session interval
)

```

4. The instantiation of the terminals (known as User Equipment, or UE), specifying the International Mobile Subscriber Identity (IMSI), Modulation and Coding Scheme (MCS) and Radio Link Control (RLC) queue size:

```

ue01 = qppsim.Ue.Ue(1, "Ue_01", 18, queue_size=10000)
ue02 = qppsim.Ue.Ue(2, "Ue_02", 18, queue_size=10000)

```

5. The instantiation of applications and assignation to UEs.

```

cbr_instance01 = cbr_profile.create_app(
    "CBR_App_01",
    qppsim.Time.Time(seconds=5),
    qppsim.Time.Time(seconds=30),
    ue01
)

```

6. Running the simulation.

```

des.start_simulation()

```

2.4 Run

To run the scenarios through QPPSim, let the Python 3 interpreter process the file with the scenario:

```
$ python3 my-scenario.py
```

Chapter 3

Introduction

QPPSim is a simple discrete event simulator aimed at helping and accelerating the study of Quality of Service (QoS), Priority, and Pre-emption (QPP) mechanisms and policies in LTE. The complexities of the LTE architecture have been abstracted in order to facilitate the rapid prototyping of schedulers, priority policies, and monitoring mechanisms, and enable comparative analyses of different approaches, solutions, and proposals.

3.1 Features

The following list summarizes QPPSim's main features:

- Scenario creation from Python or XML files.
- Applications modeled as unidirectional producer-consumer systems.
- Support for multiple UEs with different types of access priority.
- Support for multiple applications per UE with different behavior and QoS requirements.
- Support for multiple bearers per UE with different QoS requirements available.
- Simple access control and pre-emption Application Programming Interfaces (APIs).
- Fixed MCS for each UE.
- Simplified Hybrid Automatic Repeat Request (HARQ) model.
- QoS monitoring (Guaranteed Bit Rate (GBR) and non-guaranteed throughput, loss, delay) in the Evolved Packet Core (EPC).
- QoS measurements (throughput, loss, delay) for each bearer, including the default bearer.
- QPP policies easily modifiable according to UE and application type.
- Outputs text traces with information about topology, bearer management, access control, pre-emption, and scheduler operation.
- Modular design that makes it easy to build and test new components and models.

3.2 Limitations

The following list summarizes QPPSim's main limitations:

- Limited mobility support: If mobility is desired, the MCS property for the UEs needs to be updated manually.
- Applications modeled as unidirectional byte streams. Bidirectional models (request-response, etc...) are not supported.
- No physical layer modeling.
- Loss is limited to a threshold for a Uniform random variable.
- No transport or network layers. The overhead of the User Datagram Protocol (UDP) and Internet Protocol version 4 (IPv4) is used for calculations.
- No real application payload.
- No LTE-Advanced support.

Chapter 4

QPP Modules

4.1 Introduction

QPPSim contains several modules that make up the core of the simulation engine, and are used to represent the topology:

- **DES Core:** Simulation code; periodically triggers the data generation by the applications, and the resource allocation and transmission by the scheduler.
- **Application:** Includes bearers and methods to collect the QoS metrics.
- **UE:** Allocates resources to get the data generated by the bearers.
- **Trace Writers:** Ensures that the bearers do not request more GBR resources than are available.

Additionally, QPPSim is made up of 6 modules that will have direct impact on the QoS, Priority, and Pre-emption:

- **Priority Policy:**
- **Bearer:** Includes bearers and methods to collect the QoS metrics.
- **Scheduler:** Allocates resources to get the data generated by the bearers.
- **Access Control:** Ensures that the bearers do not request more GBR resources than are available.
- **Pre-emption:** Deactivates bearers when needed, upon request from the Access Control module or the QoS Monitor module.
- **QoS Module:** Measures and aggregates (if needed) the QoS metrics from each bearer, triggers pre-emption if needed, and reports the status to the Scheduler module.

Additionally, there are several trace writers that generate text files with several aspects of the simulation (such as packets sent and received), a ‘Priority Policy’ module that assigns Allocation and Retention Policy (ARP) and QoS Class Identifier (QCI) values to each dedicated bearer when established, and a ‘UE’ class that represents each of the terminals in the simulation, along with the applications running in that terminal.

4.2 Bearers


These are logical representation of the Evolved Packet System (EPS) bearers. Each bearer refers to a **UE** so that the bearer is aware of the transport block sizes (TBS) available (via the MCS configured for the UE). Each bearer has a QCI, GBR, Maximum Bit Rate (MBR), Pre-emption Vulnerability Indicator (PVI), Pre-emption Capability Indicator (PCI), ARP value, and queue size. Bearers also store the values of the

different QoS metrics: Head-of-Line (HOL) delay, loss, and throughput. Each bearer also knows the port(s) used by the application(s) associated to the bearer, so it can be added in the traces for easier processing.

One bearer in each UE must be marked as ‘default bearer’ and this bearer cannot be deactivated (the creation, activation, and marking as ‘default’ is done at the UE during UE creation). The establishment and activation of dedicated (i.e. non default) bearers can be requested during the creation of **Applications**. In order to ensure that an application has a bearer ready (if accepted in the network) when the first packet is generated, the bearer activation is scheduled 100 ms before the application’s `start_time`.

Each bearer may receive traffic from one **Application** (except the default bearer, that has no applications linked at the start of the simulation, but can later receive traffic from multiple applications over the course of the simulation), which generate **Packets** of a given `packet_size` at a given `packet_rate`, containing a `packet ID`, and the `simulation time at which it was sent`. Packets are stored in a `queue`, along with the amount of bytes transmitted, the amount of bytes in transit (being retransmitted), and the time stamp at which the packet was added to the queue. As the applications connect directly to the bearer that will handle its packets, there is no need for the use of Traffic Flow Templates (TFTs).

A bearer’s QoS parameters (QCI, MBR, GBR) can be modified during the simulation by invoking the `modify_qos` method in the bearer of interest. This modification will apply if the Access Control module authorizes it according to the logic and policy in place for the simulation.

 Please note that a bearer’s traffic rates (both GBR and MBR) are values independent of the application actual data rates. It is therefore possible to define a bearer for an application with a GBR rate significantly lower than the application’s actual data rate. In this case, the behavior of the system will depend on the implementations of the QoS Monitor, Pre-emption, and Scheduler, as it will be upon them to detect this deviation and act upon it.

4.3 Scheduler

The scheduler module is a collection of several components, the main being the **Scheduler**. This is a base class (`SchedulerBase`) extended by any class that implements the `schedule()` method in order to provide a different scheduling logic. The scheduler has access to a dictionary with the Resource Blocks (RBs) that need retransmission (sorted by time), the time stamp for the last time when the bearers were queried about the QoS, and the total number of available RBs. The base class also provides the methods to handle allocations and modeling the retransmissions, so subclasses can focus only on implementing the allocation logic.

A basic scheduler needs to follow these steps:

1. Get a reference to the DES, the bearer list, and the current time:

Source Code 4.1: Custom Scheduler - Step 1

```
def schedule(self):
    des = qppsim.Des.get_des()
    current_time = qppsim.Des.get_des().now()
    bearers = qppsim.BearerList.get_bearer_list().bearers
```

2. Put in the event list the next scheduling event:

Source Code 4.2: Custom Scheduler - Step 2

```
des.add_event(qppsim.Event.Event(des.now() +
                                qppsim.Time.ONE_MILLISECOND,
                                self,
                                self.schedule, []))
```

3. Obtain the QoS metrics from the bearers, if needed:

Source Code 4.3: Custom Scheduler - Step 3

```
if current_time >= self.last_qos_check + des.qos_monitor_interval:
    bearer_qos = des.qos_monitor.get_qos()
    self.last_qos_check = current_time
```

4. Remove the amount of RBs that need to be retransmitted from the available list:

Source Code 4.4: Custom Scheduler - Step 4

```
available_rbs = super().num_rbs - self.process_retransmissions(current_time, bearers)
```

5. Perform the allocation according to the specific scheduler's logic (example shown for Round-Robin):

Source Code 4.5: Custom Scheduler - Step 5

```
try:
    ue_idx = bearers.index(self.__last_ue)
except ValueError:
    self.__last_ue = bearers.iloc[0]
    self.__last_bid = bearers[self.__last_ue].iloc[0]
    ue_idx = 0

try:
    bearer_idx = bearers[self.__last_ue].index(self.__last_bid)
except ValueError:
    self.__last_bid = bearers[self.__last_ue].iloc[0]
    bearer_idx = 0

allocations = {}
ue_tmp = bearers.iloc[ue_idx]
bid_tmp = bearers[ue_tmp].iloc[bearer_idx]
while available_rbs > 0:
    # Now we know who's next
    if ue_tmp in allocations and bid_tmp in allocations[ue_tmp]:
        b_out = qppsim.Amc.TBS_FOR_MCS[ue_tmp.mcs][allocations[ue_tmp][bid_tmp]]
    else:
        b_out = 0
    if bearers[ue_tmp][bid_tmp].pending_size() - b_out > 0:
        if not ue_tmp in allocations:
            allocations[ue_tmp] = {}
        if not bid_tmp in allocations[ue_tmp]:
            allocations[ue_tmp][bid_tmp] = 0
        allocations[ue_tmp][bid_tmp] += 1

        self.__last_ue = ue_tmp
        self.__last_bid = bid_tmp
        available_rbs -= 1

    bearer_idx = (bearer_idx + 1) % len(bearers[ue_tmp])
    if bearer_idx == 0:
        ue_idx = (ue_idx + 1) % len(bearers)
        ue_tmp = bearers.iloc[ue_idx]
```

```

bid_tmp = bearers[ue_tmp].iloc[bearer_idx]

# Check if we have gone through all the bearers
# and not being able to allocate any
if ue_tmp == self.__last_ue and bid_tmp == self.__last_bid:
    if ue_tmp in allocations and bid_tmp in allocations[ue_tmp]:
        b_out = qppsim.Amc.TBS_FOR_MCS[ue_tmp.mcs][allocations[ue_tmp][bid_tmp]]
    else:
        b_out = 0
    if bearers[ue_tmp][bid_tmp].pending_size() - b_out <= 0:
        break

```

6. Process the allocation:

Source Code 4.6: Custom Scheduler - Step 6

```

self.process_allocations(allocations, bearers)

```

The base `Scheduler` class will take care of processing the allocation list, computing which allocated transport blocks' (TB) transmissions succeed, and scheduling retransmissions for those whose transmissions fail (based on a probabilistic model).

4.4 Access Control

The Access Control module is used when the activation of a dedicated bearer is requested. Before such activation can take place, the Access Control module will check if there are enough resources to serve the new bearer's GBR, invoke pre-emption if needed, and allow or deny the activation request.

The code contains a base class ('`AccessControlBase`') that provides mechanisms to obtain the number of RBs currently used by GBR bearers, and the RBs required to fulfill a specific bearer needs (based on the UE's MCS, and the GBR for the bearer). It also provides a method to trace bearer activation and deactivation.

The actual logic for the evaluation of resource availability and pre-emption invocation will have to inherit from the base class, and implement two methods: '`check_bearer_activation`' (which evaluates the RB reservations and requests when a bearer is activated), and '`check_bearer_modification`' (which evaluates the RB reservations and requests when a bearer is modified). This method may invoke the Pre-emption module as needed. Two implementations of the access control have been provided:

- `AccessControlTraceOnly`: performs no checks and only traces the requests and success response.
- `AccessControlBasic`: provides a basic functionality expected from the module.

4.5 Pre-emption

The Pre-emption module is invoked when resources need to be freed, either to activate a new bearer, to modify the QoS parameters of an existing bearer, or due to QoS constraints. A base class ('`PreemptionBase`') provides the interface for the classes that actually implement the module. These classes have to implement the '`attempt_preemption`' and '`qos_preemption`' methods.

Three sample implementations have been provided:

- `PreemptionDummy`: performs no action.
- `PreemptionSimple`: tries to free enough resources by pre-empting all the bearers with an ARP value of lower priority than the bearer being activated or modified.
- `PreemptionBasic`: tries to pre-empt the minimum amount of bearers with an ARP value of lower priority than the bearer being activated or modified.

4.6 QoS Monitor

The QoS Monitor module has a dual function: First, it collects the QoS information from the bearers and processes it as needed for the Scheduler module. Second, it can invoke the pre-emption module if the measured QoS fails to meet the required parameters.

The base class (`QosMonitorBase`) provides the flags that indicate whether we want to trace the QoS metrics, and whether to invoke pre-emption based on QoS parameters or not. It also provides a method to trace all the QoS statistics collected from the bearers.

The classes that implement the module in order to use different strategies regarding the pre-emption invocation and QoS violations detection will need to implement the method `get_qos`, which collects the raw information from the bearers, and processes it before passing it to the scheduler. Examples of the kind of processing that may be involved include computing moving averages, reducing the time scale of the reported metrics, or normalizing the values. This `get_qos` method is invoked by the Scheduler module whenever it needs to evaluate the QoS for the bearers.

Two implementations have been provided:

- `QosMonitorDummy`: reports 'NaN' for all the values.
- `QosMonitorDefault`: reports the minimum, average, maximum, and last value for a metric during the last second.

4.7 DES Core

The **DES Core** is the class in charge of running the simulation. It stores and provides the simulation time to the rest of the system, stores 'events' generated by the applications and the scheduler, and process them in order. Events are stored with a pointer to the function to run, and a list of arguments to pass.

The DES object is a static one, and only one must be created during the simulation. Multiple simulation parameters are configured in the constructor of the class, with default values provided for all of them:

Source Code 4.7: DES Constructor - Default Values

```
def __init__(self, num_rbs=50, start_packet_id=0, default_qci=9,
             default_arp=11, default_mbr=1024*1024*1024*1024,
             rtx_threshold=0.1,
             bearer_stats_window_size=qppsim.Time.Time(seconds=60),
             seed=1, stop_time=qppsim.Time.Time(seconds=1),
             priority_policy=None,
             access_control_policy=None,
             preemption_policy=None,
             qos_monitor=None,
             scheduler=None,
             output_dir=".",
             topology_filename="topologyTrace.txt",
             app_traffic_filename="trafficTrace.txt",
             bearer_filename="bearerTrace.txt",
             arp_filename="arpTrace.txt",
             qos_filename="qosTrace.txt",
             trace_qos=False,
             preempt_qos=False,
             qos_monitor_interval=qppsim.Time.Time(seconds=1)):
```

The DES provides several methods available to all the classes in the simulation, such as:

1. `get_random_value`: generates a random value using the random distribution and arguments specified in the parameters.

2. `now`: provides the current simulation time
3. `add_event`: adds an event to the simulation event queue.

☞ Note that the events are stored in a sorted list. The list allows for multiple keys with the same values (i.e. multiple events at the same simulation time), but the order in which those events are executed is not defined. Depending on the version of the `sortedcontainer` library, `numpy` library, and `python`, the relative order may vary, so **no assumptions about the execution order of events with the same time stamp must be made!**

Chapter 5

Traces

5.1 Introduction

QppSim generates text files (traces) that record what happens in the simulation. The available traces are as follows:

- Application Trace: details the transmission and reception of packets at the application layer, marking each event with the time stamp and the packet ID.
- Bearer Trace: keeps track of dedicated bearers being activated and deactivated.
- ARP Trace: logs the ARP and pre-emption checks, results, and actions.
- QoS Trace: records the QoS metrics obtained from each bearer during the simulation.
- Topology Trace: describes the topology of the scenario.

The actual filenames used for each trace can be defined when creating the `Des` instance in the scenario, as shown in Source Code 5.1

Source Code 5.1: Defining trace filenames in the DES creation

```
des = qppsim.Des.Des(  
    seed=1,  
    num_rbs=50,  
    stop_time=qppsim.Time.Time(seconds=35),  
    rtx_threshold=0.1,  
    bearer_stats_window_size=qppsim.Time.Time(seconds=10),  
    access_control_policy=qppsim.accesscontrol.AccessControlSample.AccessControlSample(num_rbs=50),  
    preemption_policy=qppsim.preemption.PreemptionDummy.PreemptionDummy(),  
    priority_policy=qppsim.prioritypolicy.PriorityPolicySample.PriorityPolicySample(),  
    qos_monitor=qppsim.qosmonitor.QosMonitorDefault.QosMonitorDefault(),  
    app_traffic_filename="myAppTraffic.txt",  
    bearer_filename="myBearerTrace.txt",  
    arp_filename="myArpTrace.txt",  
    qos_filename="myQosTrace.txt",  
    topology_filename="myTopologyTrace.txt",  
    trace_qos=True,  
    preempt_qos=False  
)
```

5.2 Application Trace

The application trace records when an application sends or receives a packet. The default file name is `trafficTrace.txt`, and each line of the trace file is an entry, with the following fields, in order:

- **Application Name:** the name of the application that generated the trace line.
- **Time Stamp:** simulation time at which the entry was generated (in seconds).
- **Action:** the string ‘TX’ if the packet was sent, or the string ‘RX’ if the packet was received.
- **Application Packet Size:** the packet size, in Bytes, at the application, including application headers.
- **Network Packet Size:** the packet size, in Bytes, when it enters the bearer queue. This is the packet size at the application level, plus a fixed overhead of 30 Bytes (8 Bytes for UDP header + 20 Bytes for IPv4 header + 2 Bytes for PDCP header).
- **Sequence Number:** a sequence number, unique in the simulation, assigned to each packet. This sequence number can be used to compute the delay of a packet.

5.3 Bearer Trace

The bearer trace records when a bearer is activated, modified, or deactivated. The default file name is `bearerTrace.txt`, and each line of the trace file is an entry. The format of each entry is as follows:

- **Time Stamp:** simulation time at which the entry was generated (in seconds).
- **Action:** the string ‘ACTIVATION’ if the entry is about a bearer activation, ‘DEACTIVATION’ if it is about a bearer deactivation, and ‘MODIFICATION’ if it is about a bearer modification.
- **IMSI:** the UE’s IMSI, an integer uniquely identifying each UE.
- **BID:** the bearer ID.
- **QCI:** the bearer’s QCI.
- **TFT Port:** the port that would be used in this bearer’s TFT. If the entry is about a default bearer, the value will be the string ‘None’.

5.4 ARP Trace

The ARP trace records when the access control module is invoked for a check during a bearer activation or modification, and the result of such checks. Additionally, if the check triggers a bearer pre-emption, or if a bearer is deactivated, those events will also be recorded in this trace.

The default file name is `arpTrace.txt`, and each line of the file is an entry.

The format of check entries is as follows:

- **Time Stamp:** simulation time at which the entry was generated (in seconds).
- **Action:** the string ‘ARP_ACTIVATION_CHECK’ if the entry is about a bearer activation, or ‘ARP_MODIFICATION_CHECK’ if it is about a bearer modification.
- **IMSI:** the UE’s IMSI, an integer uniquely identifying each UE.
- **USED:** the number of RBs per second currently reserved by the system for other bearers.
- **REQ:** the number of RBs per second requested by this bearer.
- **QCI:** the bearer’s QCI.

- **RATE**: the GBR value for this bearer.
- **ARP**: the bearer's ARP value.
- **PCI**: the bearer's pre-emption capability indicator (i.e. whether this bearer can pre-empt others).
- **PVI**: the bearer's pre-emption vulnerability indicator (i.e. whether this bearer can be pre-empted by others).

The format of result entries is as follows:

- **Time Stamp**: simulation time at which the entry was generated (in seconds).
- **Action**: the string 'ARP_ACTIVATION_RESULT' if the entry is about a bearer activation, or 'ARP_MODIFICATION_RESULT' if it is about a bearer modification.
- **IMSI**: the UE's IMSI, an integer uniquely identifying each UE.
- **RESULT**: the string 'ACCEPT' if the request was accepted, or 'DENIED' if it was denied.
- **USED**: the number of RBs per second reserved by the system for other bearers before the request.
- **NEW_USED**: the number of RBs per second reserved by the system after the request if the request was accepted. This will be the result of adding the 'USED' and 'REQ' fields from the check entry regardless of the result of the operation. If the request was accepted, this is the new number of resources reserved by the system; otherwise, this value may provide insight on why the request was denied.
- **QCI**: the bearer's QCI.
- **RATE**: the GBR value for this bearer.
- **ARP**: the bearer's ARP value.
- **PCI**: the bearer's pre-emption capability indicator (i.e. whether this bearer can pre-empt others).
- **PVI**: the bearer's pre-emption vulnerability indicator (i.e. whether this bearer can be pre-empted by others).

The format of pre-emption entries is as follows:

- **Time Stamp**: simulation time at which the entry was generated (in seconds).
- **Action**: the string 'ARP_PRE-EMPTED'.
- **IMSI**: the UE's IMSI, an integer uniquely identifying each UE.
- **BID**: the bearer's ID for the bearer that was pre-empted.
- **USED**: the number of RBs per second reserved by the system for the bearer that was pre-empted.
- **QCI**: the bearer's QCI for the bearer that was pre-empted.
- **RATE**: the GBR value for the bearer that was pre-empted.
- **ARP**: the bearer's ARP value for the bearer that was pre-empted.
- **PCI**: the bearer's pre-emption capability indicator (i.e. whether this bearer can pre-empt others) for the bearer that was pre-empted.
- **PVI**: the bearer's pre-emption vulnerability indicator (i.e. whether this bearer can be pre-empted by others) for the bearer that was pre-empted.

Finally, the format of deactivation entries is as follows:

- **Time Stamp:** simulation time at which the entry was generated (in seconds).
- **Action:** the string ‘DEACTIVATION’.
- **IMSI:** the UE’s IMSI, an integer uniquely identifying each UE.
- **BID:** the bearer’s ID.
- **QCI:** the bearer’s QCI.
- **RATE:** the GBR value for this bearer.
- **ARP:** the bearer’s ARP value.
- **PCI:** the bearer’s pre-emption capability indicator (i.e. whether this bearer can pre-empt others).
- **PVI:** the bearer’s pre-emption vulnerability indicator (i.e. whether this bearer can be pre-empted by others).

5.5 QoS Trace

The QoS trace records the measured QoS for each bearer each time the QoS Monitor is invoked. The default file name is `qosTrace.txt` and each line of the trace file is an entry for a single bearer, with the following fields, in order:

- **Time Stamp:** simulation time at which the entry was generated (in seconds).
- **IMSI:** the UE’s IMSI, an integer uniquely identifying each UE.
- **BID:** the bearer’s ID.
- **QCI:** the bearer’s QCI.
- **Priority:** the bearer’s priority derived from the QCI value, as defined in 3GPP’s TS 23.203.
- **Throughput:** A tuple composed of:
 - The minimum throughput (in bits per second) for the reporting period.
 - The average throughput (in bits per second) for the reporting period.
 - The maximum throughput (in bits per second) for the reporting period.
 - The last observed transmission rate (in bits per second) in the reporting period.
 - A separator in the form of the string `--`.
 - The GBR value for this bearer.
- **Loss:** A tuple composed of:
 - The minimum loss (in Bytes lost) for the reporting period.
 - The average loss (in Bytes lost) for the reporting period.
 - The maximum loss (in Bytes lost) for the reporting period.
 - The last observed loss (in Bytes lost) in the reporting period.
 - A separator in the form of the string `--`.
 - The maximum allowable loss rate for this bearer, as a value between 0 and 1. Note that this value is printed here for reference, and is not directly comparable to the loss in Bytes.
- **Loss%:** A tuple composed of:

- The minimum loss rate (in percentage of Bytes lost) for the reporting period, as a value between 0 and 1.
 - The average loss rate (in percentage of Bytes lost) for the reporting period, as a value between 0 and 1.
 - The maximum loss rate (in percentage of Bytes lost) for the reporting period, as a value between 0 and 1.
 - The last observed loss rate (in percentage of Bytes lost) in the reporting period, as a value between 0 and 1.
 - A separator in the form of the string --.
 - The maximum allowable loss rate for this bearer, as a value between 0 and 1.
- **Delay:** A tuple composed of:
 - The minimum delay (in seconds) for the reporting period.
 - The average delay (in seconds) for the reporting period.
 - The maximum delay (in seconds) for the reporting period.
 - The last observed delay (in seconds) in the reporting period.
 - A separator in the form of the string --.
 - The maximum allowable delay for this bearer.

5.6 Topology Trace

The topology trace records the link between applications and the dedicated bearer that will be requested to transport its packets. The default file name is `topologyTrace.txt` and each entry is a line with the following fields, in order:

- **Application Name:** application name as defined in the scenario.
- **Start Time:** the simulation time at which this application will start sending packets.
- **Stop Time:** the simulation time at which this application will stop sending packets.
- **QCI:** the QCI of the dedicated bearer that will be requested for this application.
- **GBR:** the GBR value in bits per second of the dedicated bearer that will be requested for this application.
- **GBR:** the MBR value in bits per second of the dedicated bearer that will be requested for this application.
- **PORT:** the UDP port assigned to this application.


Chapter 6

Scenario Creation

Simulation scenarios are defined in Python 3 files that create the following objects, in this order:

1. DES.
2. Application Profiles.
3. UEs.
4. Application Instances.

Please refer to the provided example and the API documentation for further details on parameters available for each of those objects.

 Note that it is recommended that UE names and application names do not contain spaces, as to allow for easy processing of the trace files.

After creating these objects, we can invoke the `start_simulation` method on the DES object. An example is provided in the file `example_scenario.py`.