Python LTE API documentation

Jump to bottom

Roberto Riggio edited this page on Aug 18, 2019 · 1 revision

Table of Contensts

Overview

In this section we shall first introduce some fundamental concepts like *VBSes*, *UEs* and *Cell Pools*, then we shall describe all the LTE primitives available to applications developers. Notice how all the methods described in the section are class methods of the `EmpowerApp` class and as such can be invoked only from applications extending this base class.

Virtual Base Stations

Are the LTE eNodeB connected to an instance of the EmPOWER Runtime.

The list of VBSes available to the application can be accessed using:

```
self.vbses()
```

User Equipment Abstraction

The UE abstraction provides a high–level interface for UE state management. A UE attemping to join a certain PLMN will trigger the creation of a UE instance object at the controller. Notice how the UE object will be created only after all the attach procedure have been completed by the eNodeB.

The list of UEs available to the application can be accessed using:

```
self.ues()
```

Handovers can be performed in different ways. The easiest is to assign a new VBS instance to the `vbs` property of an UE instance:

```
ue.vbs = new_vbs
```

Cell Abstraction

The *Cell* is essentially an LTE interface at a given VBS. A cell is essentialy a full LTE stack and is identified by the Physical Cell Id (PCI). Each cell moreover has a downlink frequency (dl_earfcn) and an uplink frequnecy (ul_earfcn), Moreover each cell has a certain number of downlink Physical Resource Blocks (dl_prbs) and uplink Physical Resource Blocks (ul_prbs)

Each VBS instance has a `cells` property which contains the list of cells supported by the VBS. For example:

```
for vbs in self.vbses():
    print(vbs.cellls)
```

This will print the block supported by each VBS available to the *Network App*:

```
{1: vbs 00:00:00:00:00:01 pci 1 dl_earfcn 1750 dl_earfcn 19750}
```

A shortcut to get the list of all *Cells* available to an applications can be accessed using the following method:

```
cells = self.cells()
```

Cells can also be sorted by RSRP/RSRQ with:

```
self.cells().sort_by_rsrp(ue.ue_id).first()
self.cells().sort_by_rsrq(ue.ue_id).first()
```

The code above returns the cells which are within range of the specified UE sorted from the one with better RSRP/RSRQ to the worst. Finally, the *first* returns the first cell in the list.

The primitives described above can be used to implement a simple handover routine (in the EmpowerApp's loop method):

```
def loop(self):
    for ue in self.ues():
        ue.cell = self.cells().sort_by_rsrp(ue.ue_id).first()
```

Events

The following table lists the EmPOWER LTE events.

Primitive	Parameters	Description
ue_join	ue	Callback when a UE joins a tenant.
ue_leave	ue	Callback when a UE leaves a tenant.
vbs_up	vbs	Callback when a VBS comes online.
vbs_down	vbs	Callback when a VBS goes offline.

Give a look at this Network App for some examples about how to use events.

Primitives

The following table lists the EmPOWER LTE primitives. All primitives are non–blocking and, as such, they immediately return control to the calling Network App.

An optional callback method can be specified for Poller/Trigger primitives. The specified callback method will be called when the primitive returns.

All polling primitives are executed periodically with the period set by the *every* parameter (in ms). Specifying `every = -1` will result in a single query being issued.

Primitive	Parameters	Mode	Description
rrc_measurements	ue, measurements	Trigger	Instructs the specifed UE to perform and report certain measurements.

RRC Measurements

The *rrc_measurements* primitive allow programmers to ask UEs to perform one or more measurements. For example:

```
measurements = \
    [{"earfcn": ue.cell.dl_earfcn,
      "interval": 2000,
```

```
        "max_cells": 2,
        "max_meas": 2}]

    self.rrc_measurements(ue=ue, measurements=measurements, callback=self.callback)
```

The code above asks the specified UE to perform RSRP/RSRQ measurements on the same downlink band on which the UE is operating. The UE shall report up to two cells and up to 2 measurements. Measurements shall me reported every 2 seconds.

Notice how the syntax above allows to request the UE to perform multiple measurements at once.

When the measurements are available the specified callback is invoked:

```
def callback(self, measurements):
    print(measurements.to_dict()
```

The measurement object contains something like this:

```
{
    "id": 1,
    "module_type": "rrc_measurements",
    "tenant_id": "757b2620-0246-4af6-88c3-963c3a2c54e6",
    "every": 5000,
    "callback": "callback",
    "ue": {
        "ue_id": "f5edf0a1-e12e-47be-bc81-2226422c6812",
        "rnti": 1,
        "cell": {
            "addr": "00:00:00:00:00:01",
            "pci": 1,
            "cap": 0,
            "dl_earfcn": 1750,
            "dl_prbs": 25,
            "ul_earfcn": 19750,
            "ul_prbs": 25,
            "mac_reports": {},
            "rrc_measurements": {
                "f5edf0a1-e12e-47be-bc81-2226422c6812": {
                    "rsrp": -110,
                    "rsrq": -11
                }
            }
        },
        "vbs": {
            "addr": "00:00:00:00:00:01",
            "last_seen": 7,
            "last_seen_ts": "2018-07-14T18:58:34.688580Z",
            "period": 2000,
            "label": "Generic VBS Node",
```

```
            "datapath": null,
            "state": "online",
            "connection": [
                "127.0.0.1",
                33312
            ],
            "cells": {
                "1": {
                    "addr": "00:00:00:00:00:01",
                    "pci": 1,
                    "cap": 0,
                    "dl_earfcn": 1750,
                    "dl_prbs": 25,
                    "ul_earfcn": 19750,
                    "ul_prbs": 25,
                    "mac_reports": {},
                    "rrc_measurements": {
                        "f5edf0a1-e12e-47be-bc81-2226422c6812": {
                            "rsrp": -110,
                            "rsrq": -11
                        }
                    }
                }
            }
        },
        "state": "running",
        "rrc_measurements": {}
    },
    "measurements": {
        "0": {
            "meas_id": 0,
            "earfcn": 1750,
            "interval": 2000,
            "max_cells": 2,
            "max_meas": 2
        }
    },
    "results": {
        "0": {
            "1": {
                "meas_id": 0,
                "pci": 1,
                "rsrp": -110,
                "rsrq": -11
            }
        }
    }
}
```


▸ Pages 25

Getting Started

- Introduction
- Terminology
- Network Setup
- Setting up the WTP
- Setting up the CPP
- Setting up the VBS
- Setting up the EmPOWER Controller
- Setting up the Backhaul Controller

Using EmPOWER

- Publications

Intent Based Networking

- Introduction

Downloads

- Pre-built WTP Firmwares

Developers

- REST API documentation
- Python API documentation
- Python API (WiFi/LVAP)
- Python API (LTE)
- Python API (Click/LVNF)

Tutorials

- Mobility Manager (WiFi)
- Mobility Manager (LTE)
- Service Function Chaining

Support

- Mailing List

Acknowledgements

- Acknowledgements

Clone this wiki locally

```
https://github.com/clicknf/clicknf.github.io.wiki.git
```