MOPS Power Control

Developer’s Documentation

Contents

1 Introduction

MOPS Power Control is an application that adjusts the transmitting power of Base Transceiver Station (BTS) and Mobile Station (MS) based on measurement reports of BTS and MS.

1.1 Algorithm

Basic algorithm for MOPS Power Control covers:

- sending decrease power instruction if measured value is above target
- sending increase power instruction if measured value is below target
- no action required id measured value is within expected range:
  target - 1dB <= signal <= target + 1dB

Default values configuration for the algorithm is as follows:

- target signal = -75bBm
- hysteresis = 3dB
- maximum power increase = 8dB
- minimum power decrease = 4dB

The value of measurement matched with target is calculated of 1 to 8 last measurement reports - by default the application uses 8 last reports for calculations. The application does not send any commands at the beginning of communication when number of measurements collected is less than 4.

In case of missing measurements the algorithm proceeds as follows:

- With first missing measurement algorithm replaces it with value of previous one
- Number of allowed missing measurements recovered as previous value is configurable from 1 to 3 - by default it's 3
- Next consecutive missing measurements are replaced with value -95dBm
- Weight of each next measurement (first in calculations is last received) is dropping, if last measurement has weight 1, then one before last ½ , next ¼, etc.
- When received (average) signal is within hysteresis difference from target, algorithm does not stop, but maximum step is 1 dB, algorithm sends NCH command when measurement to target is less than 1dB

In an input data there is quality parameter which impacts on algorithm behaviour. Quality is processed similar way as power measurement:

- Average is calculated over (1..8) window with weights - by default it's 8

- All missing quality measurements are considered = 5
- If quality is less than 2, algorithm works normally
- If quality is in range of < 2,4>, algorithm is not allowed to decrease power. If that would be decision from power measurements no change is used instead, increase is executed normally
- If quality is 4 or higher, decision is always to increase power, by minimum 2 dB or bigger if such calculated from power

2 Input and output of the application

2.1 Input

Algorithm takes input through standard input. Required input line is structured this way:

```
XL   XX   XXXX num1 num2
```

Where:
XL - indicates direction of transmition (UL or DL)
XX - BTS number (S0 for current cell, N1-6 for neighbour)
XXXX - Mobile station name (any string without special signs)
Num1 - power level
Num2 - quality level

2.1.1 Configuration

Algorithm has default configuration parameters:

```
# target:-75    --> Setting target power in dBm
# hister:3      --> histeresy threshold
# maxInc:8      --> Maximum power increase
# maxIncHist:1  --> Maximum power increase inside histeresy area
# maxDec:4      --> Maximum power decrease
# maxDecHist:1  --> Maximum power decrease inside histeresy area
# changeThresh:1--> Threshold of change
# maxMissing:3  --> Maximum number of missing signals before\ launching
MaxPower mode
# window:8      --> Number of measurements included in calculations
# offset:3      --> Minimum difference between current cell power\ and
neighbour
# minAmount:4   --> Minimum amount of measurements to start PC
```

These settings can be customized. Customization process is described in section 9. "Additional Features".

2.2 Output

There are few options for getting an output. Script output is printed by default in standard output in a terminal:

```
>>> cat input.txt | python3 pc.py
Loading default configuration
DL    S0      MS222    NCH
UL    S0      MS222    NCH
```

2.2.1 Printing output to a file

Commands are always printed in terminal, they can be also redirected to a file by using **`> file_name`** at the end of a terminal command:

```
>>> cat input.txt | python pc.py > output.txt
```

2.2.2 Database

Measurement data is stored in database directly, or through http server if one is set up. See section 9.3. “HTTP communication” for details.

2.2.3 Graphic output

There is a possibility to visualize measurement data for specified mobile station, See section 9.5. “Plotter” for details.

3 A link to our code

https://bitbucket.org/pat049b/mops_power_control

4 Installation instructions

Download all the files from link provided in previous section. Unzip them all into one directory, shell command for unzipping:

```
unzip <file name>.zip -d <directory>
```

If unzip library is not installed on your system, run:

`sudo apt-get install unzip`.

Files being in another directory may cause necessity to provide absolute path to them.
No installation is required, script is interpreted by shell built-in interpreter. Program can be run from terminal.

4.1 Required libraries

In order to install libraries with pip command first you have to instal pip:

```
sudo apt-get install python3-pip
```

- **`Matplotlib`** - library is required for grapher.py to draw charts. Installation can be run by:
  `pip-python3 matplotlib`
  or
  `apt-get install python3-matplotlib`

- **`Requests`** - required for http server:

```
pip3 install requests
```

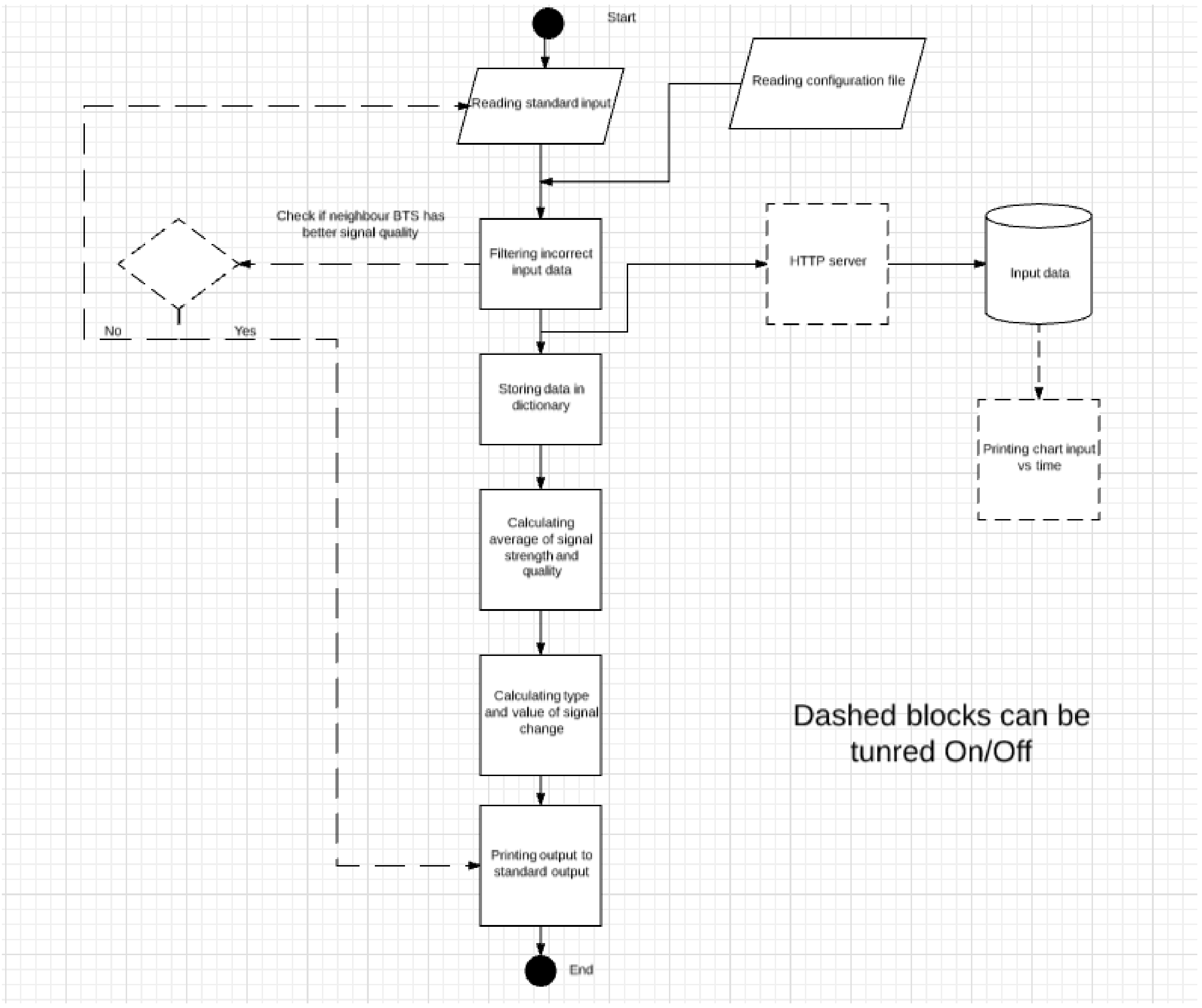
5 Project's license

MOPS Power Control is under free license that grants the recipient of a piece of software extensive rights to modify and redistribute that software.

6 Development language and coding standards

The application was written in Python 3.6.1 using PEP8 style guidelines.

7 Program's architecture


Start
Reading standard input
Reading configuration file
Check if neighbour BTS has better signal quality
Filtering incorrect input data
HTTP server
Input data
No
Yes
Storing data in dictionary
Printing chart input vs time
Calculating average of signal strength and quality
Calculating type and value of signal change
Dashed blocks can be tunred On/Off
Printing output to standard output
End

8 Functions description

8.1 **Function** avg(power, quality):

Function calculates average of n last measurements of power and quality. It has two parameters:
`power` - a list of every power measurement in n power window

quality - list of every quality measurements in n quality window

Function returns a two element tuple with average power and quality.

Function uses math operation to calculate weight average if following weights: 1, ½, ¼...1/n
Of n elements. Although for the empty array function is going to return (0, 0) to avoid division by 0.

8.2 Function missing_power(unresolved, m, count)

This function interpolates missing value from measurements of power with convert to integer. Function takes three parameters:
unresolved - a list with n data from measurements
m - maximum number of missing elements to interpolate
count - number of missing signals in previous list

Function returns a resolved list with n data with interpolated missing data.

First function initializes a resolved list with the same length as unresolved list:

```
resolved = [0 for _ in range(len(unresolved))]
```

If the first element is missing from the input data, we cannot interpolate so the function assigns max power value to the missing element:

```
if unresolved[0] == "missing" or unresolved[0] == "":
  resolved[0] = "-95"
```

If we have enough data for interpolation and if the number of interpolates is not bigger than parameter **m,** function interpolates last measurement as a value of previous one. If number is bigger than parameter **m** function replaces missing value with maximum value:

```
if unresolved[-1] == "missing" or unresolved[-1] == "":
  if 0 <= count < m:
      resolved[-1] = int(unresolved[-2])
  if count >= m:
      resolved[-1] = int(-95)
  count += 1
```

If last element is not missing then function reassigns a value from the unresolved:

```
else:
  resolved[-1] = int(unresolved[-1])
  count = 0
```

Lastly, function reassigns other elements of the list:

```
for i in range(len(unresolved)-1):
      resolved[i] = int(unresolved[i])
```

8.3 Function read_file(input_data)

This function has one parameter - single string read from standard input. The role of this function is to check data validity and to filter it according to rules dictated by main algorithm.

The function returns a single list of five strings, e.g.:

`parameter_list = ['DL', 'S0', 'MS12', '-78', '2']`

where:
`parameter_list[0] -` indicates either DL (Downlink) or UL (Uplink) signal
`parameter_list[1] -` indicates BTS (Base Transceiver Station); it should be either **`S0 -`** serving cell, or **`N1, N2, N3, N4, N5, N6`** - neighbour cell
`parameter_list[2] - i`ndicates MS (Mobile Station) identity, represented by a single string
`parameter_list[3] -` indicates strength of received signal in dBm
`parameter_list[4] -` indicates signal quality - represented by an integer ranging from 0 to 5

When called, the function first creates an empty list of five strings (**`parameter_list`**). Then it splits a string received as a parameter (**`input_data`**) into five elements which are put into **`parameter_list`**:

```
for _ in range(len(parameter)):
  parameter_list[_] = parameter[_]


dl_ul = parameter_list[0]
bts = parameter_list[1]
ms = parameter_list[2]
signal_strength = parameter_list[3]
signal_quality = parameter_list[4]
```

After that the function validates each element of received list of strings.

For the first element **`parameter_list[0]`** function checks if the format of received data is correct - if the first element is neither **`DL`** nor **`UL`** the function raises an error, otherwise function proceeds to the next conditional:

```
# checking if DL/UL
  dl_ul_list = ['DL', 'UL']
  if dl_ul not in dl_ul_list:
      raise ValueError("Invalid data")
```

For the second element **`parameter_list[1]`** function checks its validity - to see if this element fits definition of a BTS:

```
# checking BTS validity
  bts_list = ['S0', 'N1', 'N2', 'N3', 'N4', 'N5', 'N6']
  if bts not in bts_list:
      raise ValueError("Invalid BTS")
```

For the third element **`parameter_list[2]`** function checks its validity - MS should be a single string of alphanumeric characters:

```
# checking MS validity
  for _ in ms:
      if                 _                 not               in                 \
"1234567890qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM":
          raise ValueError("Invalid MS")
```

For the fourth element **`parameter_list[3]`** function checks its validity and filters it according to some preset rules:

- if the value is “missing” the function passes to the next conditional
- else, if the element in the list is empty, function raises an error
- finally function checks if the value falls into predefined range; if not, an error is raisen

```
# signal strength filtering
  if signal_strength == 'missing':
      pass
  elif signal_strength == '':
      raise ValueError("Invalid signal value")
  elif int(signal_strength) not in range(-95, -44):
      raise ValueError("Signal strength out of range")
```

For the last element **`parameter_list[4]`** function checks if the element is present:

- if the string is empty, function replaces the empty string with a string **`'5'`**

- else, if the value falls into predefined range function passes the value, otherwise function raises an error

```
# signal quality filtering
  if signal_quality == '':
      parameter_list[4] = '5'
  elif int(signal_quality) in range(0, 6):
      pass
  else:
      raise ValueError("Signal quality value out of range")
```

8.4 Function pc()

Function takes an input file containing measurement data of signal transmission and sends back an information about power change. Possible commands are:
`INC` - Increasing power by specified amount
`DEC` - Decreasing power by specified amount
`NCH` - Power level is satisfying customer's needs and no action is required
`HOBC` - Possibility of handover
Function returns a string with BTS, MS data and corresponding command.

Functions control flow of data in whole PC algorithm

- *All the debug statements are only executed when -d flag is active*

First, line is read from standard input and sanitized in read_file function. If line is not matching input patterns, loop iteration is not executing the rest of code:

```
for line in sys.stdin:


  try:
      count += 1
      debug("Current input line \t" + line)
      debug("Line number %s\n" % count)
      file_line = read_file(line)
  except ValueError:
      debug("Line incorrect\n")
      continue
```

Measurements are packed into 50 element lists and sent to database directly or through http server:

```
if len(pack)<50:
  pack.append(file_line)
else:
  try:
      for item in pack:
          request(item)
      pack = []
  except:
      dbsender(pack)       # Sending measurements to database
      pack = []
```

If -h flag is active, handover algorithm is initialized

```
if (phone in Phones) and ('-h' in sys.argv):
  do_hand      =      handover_a(file_line,      avg(Phones[phone][direction][0],
Phones[phone][direction][1]), conf['offset'], conf['target'])
  if do_hand == 1:
      pass
  elif do_hand == 2:
      continue
  elif do_hand == 3:
      print('%s\t%s\t%s\tHOBC' % (file_line[0], file_line[1], file_line[2]))
      continue


else:
  if file_line[1].startswith('N'):
      continue
```

Phone dictionary is updated and missing statements are evaluated. Every new mobile station connected to BTS is separate key in this dictionary:

```
if phone in Phones:
  Phones[phone][direction][0].append(file_line[3])
  Phones[phone][direction][1].append(file_line[4])
  if len(Phones[phone][direction][0]) > conf["window"]:
      Phones[phone][direction][0].pop(0)
      Phones[phone][direction][1].pop(0)


else:
  Phones[file_line[2]] = {"UL": [[], [], 0], "DL": [[], [], 0]}
  Phones[phone][direction][0].append(file_line[3])
  Phones[phone][direction][1].append(file_line[4])
# Updating Phones dictionary with new line


change_list = missing_power(Phones[phone][direction][0], conf['maxMissing'],
                          Phones[phone][direction][2])


debug("Missing power result:  %s %s \n" % (change_list[0], change_list[1]))
Phones[phone][direction][0] = change_list[0]
Phones[phone][direction][2] = change_list[1]
```

Algorithm is taking action only if minimum amount of data is reached:

```
if len(Phones[phone][direction][0]) >= conf['minAmount']:
```

When enough data amount is reached, avg function is evaluating average power and quality to provide worker function with arguments necessary to work out command for BTS:

```
averages = avg(Phones[phone][direction][0], Phones[phone][direction][1])
debug("Average power: %s\nAverage quality:%s\n" % (averages[0], averages[1]))
m_data = worker(averages, conf)
```

Finally command is sent to terminal:

```
print("%s\t%s\t%s\t%s\t%s" % (file_line[0], file_line[1], file_line[2],
                            m_data[0], m_data[1]))
```

8.5 Function worker((signal,quality),conf)

Function **`worker()`** as an input takes tuple of two values and a config file (conf.cfg). Values in tuple are:
`signal` - average value of last power measurements (number of last measurements taken into calculating average is configurable)
`quality` - average value of last quality measurements (number of last measurements taken into calculating average is configurable)
Both values are of float type.

Argument **`conf`** is a link to a dictionary with configurable variables.

```
def worker(data, conf):
       signal = data[0]
       quality = data[1]
       x = conf['target'] - signal
```

At the beginning, difference between target and signal strength is calculated and assigned to a variable x. Later on it will be used as a value of increasement or decreasement for situation, in which maximum steps of change are too big to match the target

First thing that worker() does is to check quality of signal and based on that follow one of the three scenarios:

- If quality is less than 2, algorithm works normally (it will be explained in the next paragraph)
- For quality in range from 2 (inclusive) to 4 (exclusive) algorithm is not allowed to decrease power. If that would be decision from power measurement no change is used instead, increase is executed normally.
- For quality equal or larger than 4, algorithm always increases power, minimum 2dB or higher if such calculated from power measurement

Once quality is checked, algorithm checks what is the signal power and compare it to desired target (below is described how algorithm works for quality value less than 2. Different values of quality apply constraints to algorithm. Those constraints can be found in previous paragraph).

- For signal power less than sum of target and hysteresis (both values are configurable), algorithm returns command to increase power and calculates value that is required to match the target, however calculated value cannot exceed maximum step of increasement (configurable value)
- For signal power higher than sum of target and hysteresis (both values are configurable) algorithm returns command to decrease power and calculates value that is required to match the target, however calculated value cannot exceed maximum step of decreasement (configurable value)
- If the signal strength falls into interval of target +- hysteresis, maximum decreasement and increasement steps are changed (configurable values)
- If the signal strength falls into interval of target +- change threshold (configurable value), algorithm returns command to not change signal strength and value of change becomes an empty string

Last part of the function checks if value of change is an empty string or float. For empty string, function pass it to the result. For float, rounding is applied so the result is an integer:

```
if value != '':
  value = round(float(value))
result = (action, value)
```

As an output, function returns tuple of two elements:

- Action to be taken of string type
- Value of change:
    - of string type if no change is applied
    - of integer type if change is applied

8.6 Function dbsender(line)

Flag: ‘-db’

Function is creating a table in "measure_hist.db" database, located in the same directory. If table exists, function adds next pack of row of measurements and appends the line with current processor time.

```
conn = sqlite3.connect("measure_hist.db")
c = conn.cursor()
c.execute('''CREATE TABLE IF NOT EXISTS HISTORY
      (Direction text, Transmiter text, Mobile text, Power text, Quality text, Time
date)''')
for line in pack:
  line.append(str(datetime.now()))
  c.execute("INSERT INTO HISTORY VALUES (?, ?, ?, ?, ?, ?)", line)
conn.commit()
```

9 Additional features

9.1 Configuration

Default configuration can be overridden. To do so, a flag has to be used when invoking a program in a command line:

Flag: **-c**

Invoke example:

```
cat input.txt | python3 pc.py -h -c
cat input.txt | python3 pc.py -c
```

Feature description: Overrides default configuration with one specified in conf.cfg file. Detailed information related to changing configurable parameters are included inside this file.

If -c flag is active, algorithm overrides default configuration parameters with ones specified in conf.cfg file:

```
if '-c' in sys.argv:
  conf = default.copy()
  print("Loading customized configuration")
  with open('conf.cfg', 'r') as f:
      for line in f:
          if line.startswith('#'):
              continue
          else:
              conf[line.split(':')[0]] = int(line.split(':')[1])
```

Moreover algorithm sanitizes inputed data with aforementioned conditions:

```
if conf['target'] not in range(-95, -44):
  print("Invalid target parameter, setting target to %s" % (default['target']))
  conf['target'] = default['target']


if conf['minAmount'] > conf['window']:
  print("Window must be bigger than minimum amount of measurements")
  print("Setting window to %s" % (conf['minAmount']))
  conf['window'] = conf['minAmount']


for item in conf.keys():
  if item != 'target' and conf[item] < 0:
      print("Invalid value of %s, setting to %s" % (item, default[item]))
      conf[item] = default[item]
```

9.2 Handover algorithm

Handover algorithm:

Flag: **-h**

Invoke example:

```
cat input.txt | python3 pc.py -h -c
cat input.txt | python3 pc.py -h
```

Feature description: When neighbour cell measurement received, algorithm is comparing signal power of S0 cell with signal power of neighbour. Sends HOBC (Handover Better Cell) signal if handover is profitable.

```
def handover_a(f_line, avg_power, hister):
  """
  Function which does the following things:
  - compares power measurement of current cell to another cell
  - recommends handover
  :param f_line: list of strings from input
  :param avg_power: avg power of current cell
  :param hister: minimum offset before handover
  :param target: power target
  :return: string
  """
  cell = f_line[1]
  if cell == 'S0':
      return 1
  elif f_line[3] == "missing" or f_line == "":
      return 2
  else:
      dist_s0 = fabs(avg_power[0]) - fabs(hister)
      dist_nx = fabs(float(f_line[3]))
      if fabs(dist_nx) < fabs(dist_s0):
          return 3
      else:
          return 2
```

9.3 HTTP communication

To establish HTTP server user need to launch it before launching whole application.
Steps to do that:

1. Go to project directory
2. Launch **`http_server.py`** in python3 interpreter in separate console by command:
   `python3 http_server.py`

3) Now your server is working in separate console.
4) Run your program in another console and communication with HTTP in server is established now. If you skip steps 1-3 your program is sending data to database directly.

It may happen that your environment **doesn't have requests lib** which is necessary for properly working HTTP communication. **To get requests lib** please find attached producent website: http://docs.python-requests.org/en/master/

Client side:

Function needs to convert list to dictionary and send is as params with requests.get.
If the code response is not equal 200, we raise ConnectionError which are handled in pc.py function where request is invoked.

```
def request(line):
  """
  Function using requests library to send data(line) via get
  to local server on port 8000.
  :param line: data to be send
  """
  buffer = {1: 0, 2: 0, 3: 0, 4: 0, 5: 0}
  for i in range(5):
      buffer[i + 1] = line[i]
  r = requests.get('http://127.0.0.1:8000', params=buffer)
  if r.status_code != 200:
      raise ConnectionError
```

Server side:

As a server class SimpleHTTPRequestHandler1 was defined which inherited from http.server.SimpleHTTPRequestHandler and overwrites do_Get method to receive and convert data which was send by client. It is converting this data to list and invoke **`dbsenderhttp`** which is working the same as **`dbsender(line)`** (see section 8.6. “Function dbsender(line)”**)** but sends only one line to database at the time.

```
class SimpleHTTPRequestHandler1(http.server.SimpleHTTPRequestHandler):
  def do_GET(self):
      """
      Overwriting class method to send data to database through
      get request. After receiving data as query we need to convert
      them to list with single data inside.
      """
      bits = urllib.parse.urlparse(self.path)
      print(self.command, bits)
      buff = bits.query.split('&')
      buffer = []
      for i in buff:
          a = i.split('=')
          buffer.append(a[1])
      dbsenderhttp(buffer)
      return super(SimpleHTTPRequestHandler1, self).do_GET()
```

This function has only one task to run HTTP server forever on a specific port in loopback interface.

```
def run(server_class=http.server.HTTPServer,
      handler_class=SimpleHTTPRequestHandler1):
  """
  Function running server forever on port 8000
  :param server_class: server
  :param handler_class: specific type of server
  """
  server_address = ('', 8000)
  httpd = server_class(server_address, handler_class)
  httpd.serve_forever()
```

9.4 Debugger

Flag: **`-d`**

Invoke example:

```
cat input.txt | python3 pc.py -d -h
cat input.txt | python3 pc.py -d
```

Feature description: Creates log for debugging purposes in `logdeb.txt`. Debugger appends information to this file. In order to wipe logs out, manual deletion of file is required. Example log from one input line:

```
Current input line  UL S0 MS776 -78 2
Line correct
Current power history [-78, -75, -70, -70, -78]
Current quality history ['2', '2', '1', '1', '2']
```

```
Consecutive missings 0
Enough data to take an action
Average power: -74.71
Average quality:1.61
Command sent: UL   S0 MS776  NCH
```

Debug function initialized during loop execution appends to specified file any string provided as argument to a function:

```
def debug(something):
  """Function sends information to logdeb.txt file"""


  if '-d' in sys.argv:
      with open('logdeb.txt', 'a') as f:
          f.write(something)
```

9.5 Plotter

Flag: none - plotter is a separate application, using data in database to visualize history of power levels downlink and uplink for certain mobile station in specified time window.

Invoke example:
`python3 grapher.py ‘2017-07-20 20:00’ ‘2017-07-21 22:00’ ‘MS111’`

After name of script (**`grapher.py`**), two following dates are beginning and end of time window.
Date format: ‘**`YYYY-MM-DD HH:MiMi:SS.SsSsSs`**’ where:

`Y` - year, **`M`** - month, **`D`** - day
`H` - hour, **`Mi`** - minute, **`S`** - seconds, **`Ss`** - parts of second

Provided dates accuracy does not matter.
The last parameter is a name of mobile station to be displayed.

Plotter uses **`matplotlib`** library, which must be downloaded previously. Installation on Ubuntu/Debian systems can be executed by command:
`sudo apt-get install python3-matplotlib` or
`pip-python3 matplotlib`

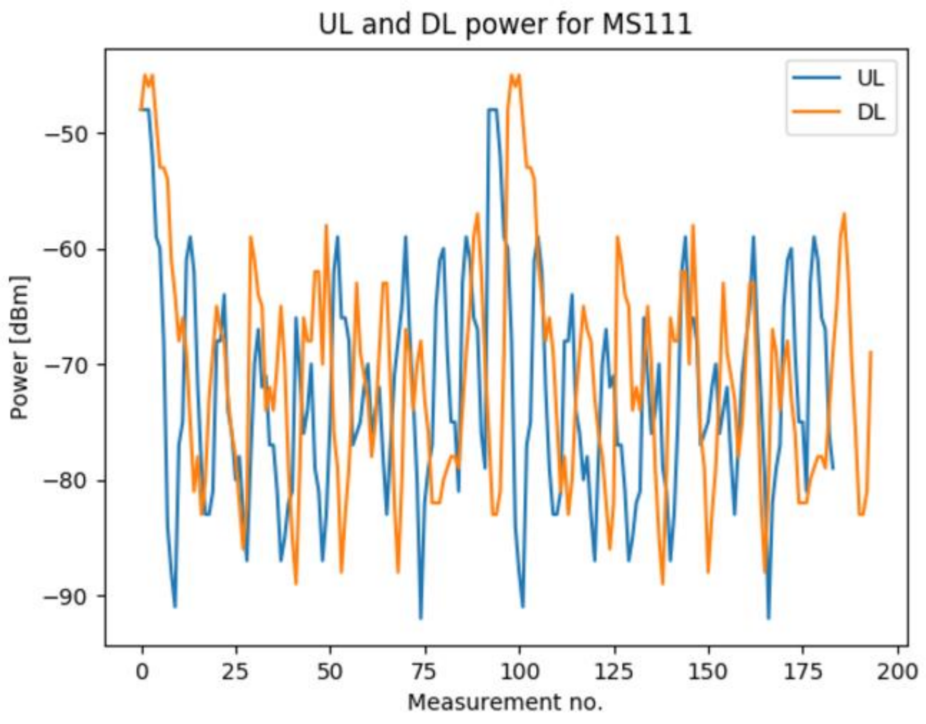
Function connects to measure_hist.db database and fetches all data within for selected MS:

```
def db_read(start, end, ms):
  """Function connects to database with measurements history and fetch data from
it,
  data is initially filtered by SQL query and then filtered again, due to python3
  well implemented string comparison
  Input:
  start - beginning of time window
  end - end of time window
  ms - mobile station alias

  Output = graphical representation of measurements"""
  conn = sqlite3.connect("measure_hist.db")
  c = conn.cursor()
  c.execute("SELECT Direction, Power, Time FROM HISTORY "
            "WHERE (Mobile = '%s') AND (Transmiter = 'S0') " % ms)
  data = c.fetchall()
```

Then, sorting algorithm is dividing data into 2 separate plotting list and creates plot figure:

```
for item in data:
  if start < item[2] < end:
      if item[0] == 'UL' and item[1] != 'missing':
          UL.append(item[1])


      elif item[0] == 'DL' and item[1] != 'missing':
          DL.append(item[1])
```

Example figure:



9.6 Parser to analyze TCPDUMP

Flag: none - parser is a separate application, using data from tcpdump aplication to visualize history of communication with HTTP server.

Steps to start:

1. Start HTTP server - see 9.6 HTTP communication

2. Start a new terminal:

   Invoke example:
   1. **`tcpdump -A -v -i <interface> > <file_name>`**

   2. **`python3 ana.py <file_name>`**

```
def tcpfilter(file):
    """
    Function filtering tcpdump file to allow us to debug
    communication between our program and server. In this function
    we are filtering line with HTTP GET method and response 200 OK
    :param file: file from tcpdump
    as a result we get print of lines interesting for us.
    """
    ok200 = 0
    gethttp = 0
    with open(file, 'r') as string:
        for line in string:
            if ('GET' in str(line)) and ('HTTP' in str(line)):
                gethttp += 1
            elif '200 OK' in str(line):
                ok200 += 1

    print("GET sent %s times" % gethttp)
    print("200 OK response received %s times" % ok200)
```

10 Reporting problems

If you encounter any issues with MOPS Power Control, please report them to our project manager at: **patryk.bogusz@ust-global.com**