Master Thesis

Design, Development and Implementation of an Energy Consumption Simulator for the CORE Network Emulator

Submitted by: Nazowa Tanim
First examiner: Dr.Prof. Ulrich Trick
Second examiner: Dr.Prof. Armin Lehmann
Date of start: 14.01.2019
Date of submission: 27.06.2019

Statement

I confirm that I have written this thesis on my own. No other sources were used except those referenced. Content which is taken literally or analogously from published or unpublished sources is identified as such. The drawings or figures of this work have been created by myself or are provided with an appropriate reference. This work has not been submitted in the same or similar form or to any other examination board.

_____

Date, signature of the student

Content

Table of Contents

1 Introduction

A model is used to define description of simulated subject and represents its key behaviour. A simulation of a system is the operation of a model of the system. These techniques help to analyse the behaviour of a system before it has been built. These techniques help to reduce expenses, time and to have visualisation before the real system has been built.

Energy consumption simulator is a tool which can simulate the energy of a system depending on its parameters. These parameters can be related to hardware components or softwares. Simulator is designed with a defined model and should be analysed with different scenarios.

CORE is a tool which can build virtual networks and can be connected to physical networks. It is based on Linux virtualization namespace.

The goal of this research is to design, develop and evaluation of an Energy Consumption Simulator for CORE Network Emulator. This research focuses on simulating energy selecting four metrics related to the main system resources (CPU, Disk, Memory and Network).

The name of the main chapters of this thesis document are named as following:

1. Introduction
2. Theoretical Background
3. Requirements Analysis
4. Realisation
5. Usage of Application
6. Summary and Future Scope
7. Abbreviations
8. References

In the theoretical part the document holds the description of background of simulation, Linux OS management and performance metrics, CORE network emulator and python modules. In realisation chapter detail of development and process have been covered. In chapter 5 there will be some scenarios using the developed application and some usage of the application will be visualised. And in chapter 6 there will be a discussion of summary of research work and some possibilities of this topic in future.

2 Theoretical Background

This chapter will cover the theoretical discussion of all technologies which have been used in this work. It will be helpful to understand the related technologies to evaluate the research work. Different technologies have been used to reach the goal of this thesis task. Therefore, it will be useful to understand these technologies, methods and softwares that have been used to reach the goal. This chapter consists of different sections and some sections includes multiple subsections.

Major sections of this chapter are:

- Models and Simulation
- Linux OS Management
- Energy Consumption in a System
- Battery Management
- CORE Network Emulator
- Python Modules

Above mentioned sections will be covered in this chapter to understand the task and make it easier to relate with the implementation part.

2.1 Models and Simulation

Model and Simulation are known as powerful method for designing and evaluating complex systems and processes. The technique is particularly useful in solving problems of complex systems where easier solutions do not present themselves. Modeling and simulation are used to study systems. A system is defined as the collection of entities that make up the facility or process of interest. (Menner,1995).

2.1.1 Model

A model is an entity that is used to represent some other entity for some defined purpose. Models are used when investigation of the actual system is difficult, and direct investigation is expensive, slow, unsafe or even illegal. Modeling is used to create and analyse a digital prototype of a physical model. Indeed, models can be used to study systems that exist only in concept (White and Ingalls,2009).

2.1.2 Simulation

Simulation is experimentation with a model. Simulation is the process of using the model and to imitate the behaviour of the system. Simulation is used before an existing system is altered or a new system built. It reduces the chances of failure to meet specifications, to ease the way of development, to ensure perfect utilization of resources, and to optimize system performance (Maria,1997).

For example, computer simulations have been responsible for many advancements in many fields such as biology, meteorology, cosmology, population dynamics, and military effectiveness. Without simulation, study of these subjects can be difficult by the lack of accessibility to the real system, the need to study the system over long time periods, the difficulty of recruiting human manpower for experiments, or all these factors. Because the technique simulation offers solutions to these problems, it has become a tremendously powerful tool for examining complexity of the today's world (Menner,1995).

Figure 2. 1 shows a schematic design of Simulation. It follows an iterative process where system under study becomes the altered system and vice versa with a repeating cycle. In a simulation study, human decision making is required at all stages, such as: model development, experiment design, output analysis, conclusion formulation, and making decisions to alter the system under study. The only stage where human intervention is not required is the running of the simulations, which most simulation software packages perform efficiently (Maria,1997).

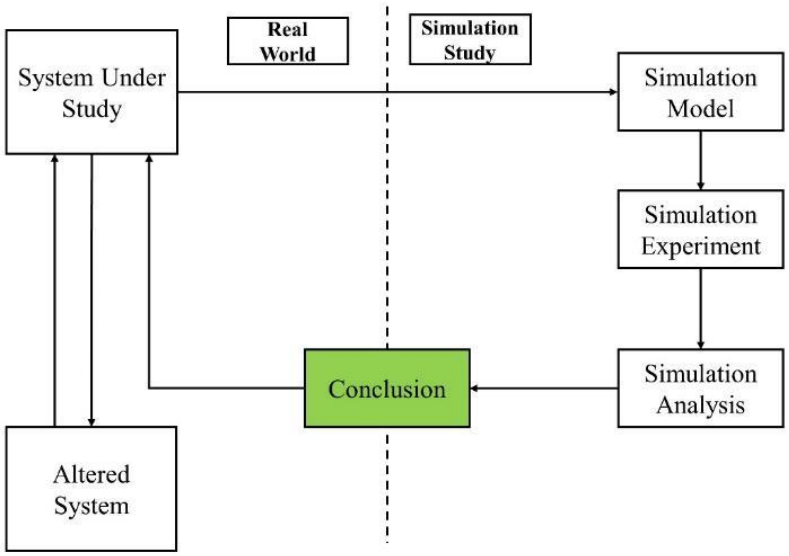


Figure 2. 1 Simulation Study Schematic (Maria,1997)

2.1.3 Benefits of Simulation, Modeling And Analysis

According to practitioners, simulation modeling and analysis is one of the most frequently used operations research techniques. Following some advantages of using simulation technique are given (Maria,1997).

- Obtain a better understanding of the system by developing a mathematical model of a system of interest and observing the system's operation in detail over long periods of time.
- Test hypotheses about the system for feasibility.
- Experiment with new or unknown situations about which only weak information is available.
- Developing a well-designed system and reducing time.
- Identifying the "driving" variables and analysis its performance and behaviour.

2.2 Linux OS Management

OS is affected by multiple factors (Ciliendo and Kunimasa,2007). Such as:

- Hardware
- Firmware
- Drivers
- Kernel
- Libraries
- Applications

To understand the Linux OS, it is important to understand some key features. Following discussion will be on Linux different management systems.

2.2.1 Linux Process Management

Process management is one of the most important roles of any operating system. Effective process management enables an application to operate steadily and effectively. A process is an instance of execution that runs on a processor. The process uses any resources that the Linux kernel can handle to complete its task. All processes running on Linux operating system are managed by a process descriptor. A process descriptor contains all the information necessary for a single process to run such as process identification, attributes of the process, and resources which construct the process (Ciliendo and Kunimasa,2007).

2.2.2 Life Cycle of a Process

Every process has its own life cycle such as creation, execution, termination, and removal. These phases will be repeated literally millions of times as long as the system is up and running. Therefore, the process life cycle is very important from the performance perspective (Ciliendo and Kunimasa,2007). A typical lifecycle of a processes is a s Figure 2.2:

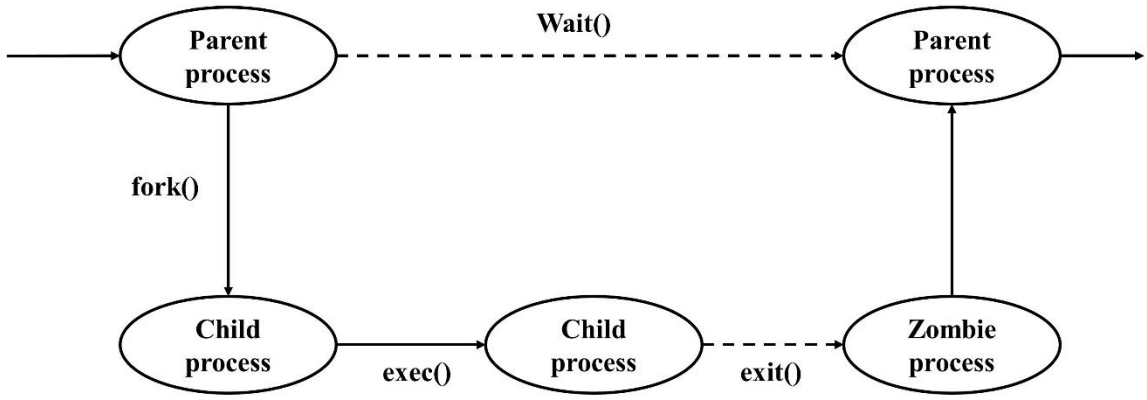


Figure 2. 2 Lifecycle of processes (Ciliendo and Kunimasa,2007)

When a process creates a new process, the creating process (parent process) issues a *fork ()* system call. When a *fork ()* system call is issued, it gets a process descriptor for the newly created process (child process) and sets a new process id. It copies the values of the *parent process* descriptor to the *child process*. At this time the entire address space of the parent process is not copied; both processes share the same address space (Ciliendo and Kunimasa,2007).

The *exec ()* system call copies the new program to the address space of the child process. Because both processes share the same address space, writing new program data causes a page fault exception. At this point, the kernel assigns the new physical page to the child process (Ciliendo and Kunimasa,2007).

This deferred operation is called the *Copy on Write.* The child process usually executes their own program rather than the same execution as its parent does. This operation avoids unnecessary overhead because copying an entire address space is a very slow and inefficient operation which uses a lot of processor time and resources (Ciliendo and Kunimasa,2007).

When program execution has completed, the child process terminates with an *exit ()* system call. The *exit ()* system call releases most of the data structure of the process and notifies the parent process of the termination sending a signal. At this time, the process is called a *zombie process* (Ciliendo and Kunimasa,2007).

The child process will not be completely removed until the parent process knows of the termination of its child process by the *wait ()* system call. As soon as the parent process is notified of the child process termination, it removes all the data structure of the child process and release the process descriptor (Ciliendo and Kunimasa,2007).

2.2.3 Linux Performance Metrics

Linux performance is measured based on different metrics available on Linux system. System performance of Linux are largely dependent on this metrics. Following there will be discussion about some metrics which are responsible to impact on Linux performance (Ciliendo and Kunimasa,2007).

Network Subsystem
The network subsystem is another important subsystem in the performance perspective. Networking operations interact with many components other than Linux such as switches, routers, gateways, PC clients, and so on.The performance metrics Network Interface are given below(Ciliendo and Kunimasa,2007):

Packets received and sent: This metric informs of the quantity of packets received and sent by a given network interface. RX means receive bytes and TX means tranmission bytes.

Bytes received and sent (RX/TX): This value depicts the number of bytes received and sent by a given network interface.

Collisions per second: This value provides an indication of the number of collisions that occur on the network that the respective interface is connected to. Sustained values of collisions often concern a bottleneck in the network infrastructure, not the server. On most properly configured networks, collisions are very rare unless the network infrastructure consists of hubs.

Packets dropped: This is a count of packets that have been dropped by the kernel, either due to a firewall configuration or due to a lack of network buffers.

Overruns: Overruns represent the number of times that the network interface ran out of buffer space. This metric should be used in conjunction with the packets dropped value to identify a possible bottleneck in network buffers or the network queue length.

Errors: The number of frames marked as faulty. This is often caused by a network mismatch or a partially broken network cable. Partially broken network cables can be a significant performance issue for copper-based gigabit networks.

Processor Metrics
The processor metrics is likely mostly dependent on CPU, there are some other metrics which are given below (Ciliendo and Kunimasa,2007):

CPU utilization: It describes the overall utilization per processor.

User time: Depicts the CPU percentage spent on user processes, including nice time. High values in user time are generally desirable because, in this case, the system performs actual work.

System time: Depicts the CPU percentage spent on kernel operations

Waiting: Total amount of CPU time spent waiting for an I/O operation to occur.

Idle time: Depicts the CPU percentage the system was idle waiting for tasks

Nice time: Depicts the CPU percentage spent on re-nicing processes that change the execution order and priority of processes.

Memory Metrics
To execute a process, the Linux kernel allocates a portion of the memory area to the requesting process (Ciliendo and Kunimasa,2007).

Memory Utilization: How much physical memory is used by a process.

Buffer and cache: Cache allocated as file system and block device cache

Free memory: How much memory is free at that block of time.

Disk I/O Subsystem Metrics:
Disk I/O subsystems are measured depending on following metrics (Caliendo and Kunimasa,2007).

Iowait Time: The CPU spends waiting for an I/O operation to occur.

Average queue length: Amount of outstanding I/O requests. In general, a disk queue of 2 to 3 is optimal; higher values might point toward a disk I/O bottleneck.

Average wait: A measurement of the average time in ms it takes for an I/O request to be serviced. The wait time consists of the actual I/O operation and the time it waited in the I/O queue.

Blocks read/write per second: This metric depicts the reads and writes per second expressed in blocks of 1024 bytes

Kilobytes per second read/write: Reads and writes from/to the block device in kilobytes represent the amount of actual data transferred to and from the block device.

2.3 Energy Consumption in a System

The management of system components can be done either in hardware or software. When a device is bought it cannot be programmed to limit its energy. The designers have already made choices in terms of selection of components and in terms of resource management. On the other hand, if a system can be programmed, choices made by developers will affect the management of energy the device consumes. Looking at embedded systems, all the responsibilities in terms of management of energy resources are dependent on the hardware management and on the firmware. Firmware has immediate effects on energy consuming of device, the hardware and the operating system have an important role in energy management, but it is not the only one. On this type of device is it possible to install a multiple of programs that will impact on the management of energy resources (Procaccianti et al, 2011).

For example, a software can increase the energy consumption. The underlying idea is that energy consumption depends not only on hardware features, but also (and probably mostly) on software usage and software internal characteristics. For instance, a more complex software will require more CPU cycles, or a single long write operation on disk can be less energy consuming rather than several small write operations (Procaccianti et al, 2011).

Energy consumption of a system depends on several factor like capacity, temperature, hardware and software systems. These factors impact CPU, RAM, Network block subsystem. Energy consumption can be analysed with four system metrics: Network usage, CPU usage, RAM usage, Disk Usage (Procaccianti et al, 2012).

A system or device can be in idle or active mode. Device is in the idle state if it is fully awake (not suspended) but no applications are active. This also causes a consumption in energy. If the CPU has completed all tasks it is called as an idle mode (Carroll,2010). Although, when the system is in idle mode, it is normal to have CPU utilization of 0-10% during idle period (Bleeping Computer LLC,2019).

A device is called active when some processes are running actively and consuming continuous energy of the system. At this period CPU is active largely and this causes energy consumption of the system (Carroll,2010).

2.4 Battery Management

Battery capacity, battery life of a system depends on the designed battery capacity and load charge that's being drained while the system is on. It depends on what system capacity is and how much is the load consuming.

Battery Capacity and Battery Life
Battery capacity (Ah) is defined as a product of the current that is drawn from the battery while the battery is able to supply the load until its voltage is dropped to lower than a certain value for each cell. The battery capacity corresponds to the quantity of the electric charge which can be accumulated during the charge, stored during the open circuit stay, and released during the discharge in a reversible manner. The required battery capacity is calculated by determining the load which the battery will be expected to supply. (ScienceDirect,2019).

Battery capacities measured in watthour [Wh], mili watt hour[mWh] or milliampere hour[mAh].

It can be changed one to the other using below formulas according to TechConsumer(TechConsumer,2018).

- Changing Wh to mAh
  Assuming, a laptop has a battery capacity of 42 Wh and need to change it to mAh. The formula is:
  (Wh×1000) ÷V where V is the voltage of the laptop battery. As an example, a laptop battery has 15.7V and capacity is=100 W.
  Then calculation in mAh would be:
  (100×1000)÷15.7=636.9 mAh
- Changing mAh to Wh
  Using the formula **(mAh×V) ÷1000.As** an example: battery capacity= 2500mAh, laptop battery voltage=15.7 then,
  Wh=(2500×15.7)÷1000=39.25 Wh
- Changing mWh to Wh
  Dividing mWh by 1000 will give Wh. Multiplying Wh by 1000 will give mWh.

Battery life is a measure of battery performance and longevity. For example, if a battery has 250 mAh capacity and provides 2.5 mA average current to a load, in theory, the battery will last 100 hours. In pc or laptop battery life is shown as percentage value. It depends on the full battery capacity and discharging load capacity. With time the full charge capacity decreases.

As an example, Remaining percentage of battery in a laptop or PC can be derived from below formula (Quora,2015):

Battery Percentage remaining= (remaining capacity÷total capacity) ×100

It is determined by a power meter integrated in a laptop.

2.5 CORE Network Emulator

The Common Open Research Emulator (CORE) is a framework for emulating networks on one or more PCs. CORE emulates routers, PCs, and other hosts and simulates the network links between them. Because it is a live-running emulation, these networks can be connected in real-time to physical networks and routers. (CORE,2015).

Real applications can be run on CORE using the virtualization technique of Linux operation system.

Some key features of CORE are:

- Efficient and scalable
- Runs applications and protocols without modification.
- Drag and drop GUI.
- Highly customizable.

CORE is typically used for network and protocol research, demonstrations, application and platform testing, evaluating networking scenarios, security studies, and increasing the size of physical test networks. (CORE,2015).

2.5.1 CORE Architecture

Two main components of CORE are: CORE Daemon and CORE GUI.

CORE daemon is responsible to manage emulation sessions and to build the emulated networks using kernel virtualization for nodes and some form of bridging and packet manipulation for virtual networks. The nodes and networks come together via interfaces installed on nodes. It is written on python and can be controlled through customized script (CORE,2015).

The daemon is controlled via the graphical user interface, the CORE GUI. The GUI and the daemon communicate using a custom, asynchronous, sockets-based API, known as the CORE API. Nodes and network interfaces can be dragged and dropped. Terminal can be launched during running sessions. It is managed by TCL/TK program (CORE,2015). The following Figure 2.2 depicts the architecture:

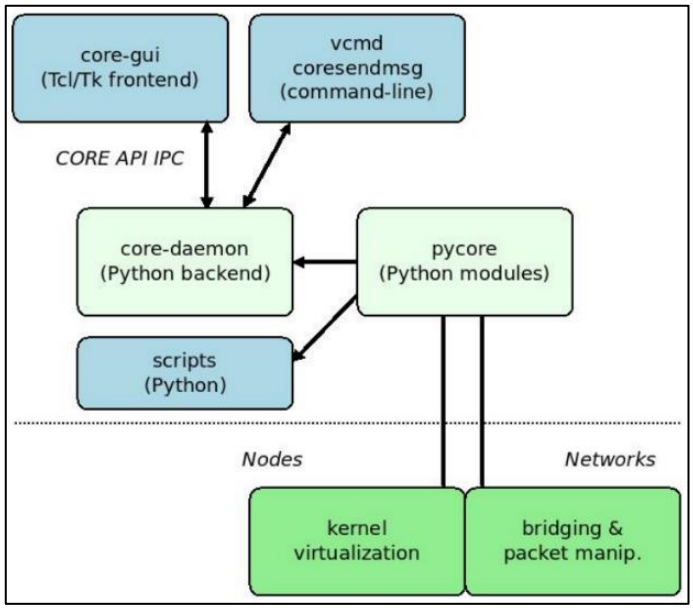


Figure 2. 3 CORE Architecture (CORE,2015)

2.5.2 CORE with Linux

A CORE node is a lightweight virtual machine. The CORE framework runs on Linux and FreeBSD systems. The primary platform used for development is Linux. CORE uses Linux network namespace virtualization to build virtual nodes and ties them together with virtual networks using Linux Ethernet bridging.

Linux network namespaces (also known as netns, LXC, or Linux containers) is the primary virtualization technique used by CORE. A namespace is created using the `clone ()` system call. Similar to the BSD, each namespace has its own process environment and private network stack. Network namespaces share the same filesystem in the CORE. Wireless network can also be emulated in the CORE(CORE,2015)

2.5.3 Toolbar

The toolbar is a row of buttons that runs vertically along the left side of the CORE GUI window. The toolbar changes depending on the mode of operation.

Editing Toolbar:
When CORE is in Edit mode (the default), the vertical Editing Toolbar exists on the left side of the CORE window. Below are brief descriptions for each toolbar item, starting from the top. Most of the tools are grouped into related sub-menus, which appear when user click on their group icon (CORE,2015).

-  Selection Tool - tool for selecting, moving, configuring nodes
-  Start button - starts Execute mode, instantiates the emulation
-  Link - the Link Tool allows network links to be drawn between two nodes by clicking and dragging the mouse (CORE,2015).

Network-layer virtual nodes:

-  Router - runs Quagga OSPFv2 and OSPFv3 routing to forward packets
-  Host - emulated server machine having a default route, runs SSH server
-  PC - basic emulated machine having a default route, runs no processes by default
- 

MDR - runs Quagga OSPFv3 MDR routing for MANET-optimized routing
-  PRouter - physical router represents a real testbed machine, physical.
-  Edit - edit node types button invokes the CORE Node Types dialog. New types of nodes may be created having different icons and names. The default services that are started with each node type can be changed here.

Link-layer nodes:

-  Hub - the Ethernet hub forwards incoming packets to every connected node
-  Switch - the Ethernet switch intelligently forwards incoming packets to attached hosts using an Ethernet address hash table
-  Wireless LAN - when routers are connected to this WLAN node, they join a wireless network and an antenna is drawn instead of a connecting line; the WLAN node typically controls connectivity between attached wireless nodes based on the distance between them
-  RJ45 - with the RJ45 Physical Interface Tool, emulated nodes can be linked to real physical interfaces on the Linux or FreeBSD machine; using this tool, real networks and devices can be physically connected to the live-running emulation (RJ45 Tool)
-  Tunnel - the Tunnel Tool allows connecting together more than one CORE emulation using GRE tunnels (Tunnel Tool).

Annotation Tools:

-  Marker - for drawing marks on the canvas
-  Oval - for drawing circles on the canvas that appear in the background
-  Rectangle - for drawing rectangles on the canvas that appear in the background
- 

Text - for placing text captions on the canvas (CORE,2015).

Execution Toolbar
When the Start button is pressed, CORE switches to Execute mode, and the Edit toolbar on the left of the CORE window is replaced with the Execution toolbar. Below are the items on this toolbar, starting from the top (CORE,2015).

-  Selection Tool - in Execute mode, the Selection Tool can be used for moving nodes around the canvas, and double-clicking on a node will open a shell window for that node; right-clicking on a node invokes a pop-up menu of run-time options for that node
-  Stop button - stops Execute mode, terminates the emulation, returns CORE to edit mode.
-  Observer Widgets Tool - clicking on this magnifying glass icon invokes a menu for easily selecting an Observer Widget. The icon has a darker gray background when an Observer Widget is active, during which time moving the mouse over a node will pop up an information display for that node (Observer Widgets).
-  Plot Tool - with this tool enabled, clicking on any link will activate the Throughput Widget and draw a small, scrolling throughput plot on the canvas. The plot shows the real-time kbps traffic for that link. The plots may be dragged around the canvas; right-click on a plot to remove it.
-  Marker - for drawing freehand lines on the canvas, useful during demonstrations; markings are not saved
- 

Two-node Tool - click to choose a starting and ending node and run a one-time traceroute between those nodes or a continuous ping -R between nodes. The output is displayed in real time in a results box, while the IP addresses are parsed, and the complete network path is highlighted on the CORE display.
- 

Run Tool - this tool allows easily running a command on all or a subset of all nodes. A list box allows selecting any of the nodes. A text entry box allows entering any command. The command should return immediately, otherwise, the display will block awaiting a response. The ping command, for example, with no parameters, is not a good idea. The result of each command is displayed in a results box. The first occurrence of the special text "NODE" will be replaced with the node name. The command will not be attempted to run on nodes that are not routers, PCs, or hosts, even if they are selected (CORE,2015).

2.6 Python Modules

To develop the application different modules have been used in this task. Python has some built in function which are easy to use and implement. Below some of the python modules have been discussed

Psutil
Psutil (process and system utilities) is a cross-platform library of python for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors) in python. It implements many functionalities offered by UNIX command line tools such as: ps, top, lsof, netstat, ifconfig, who, df and so on(Python Software Foundation,2019a).

It can be installed in Linux with below terminal command:

```
$ pip install psutil
```

Below is an example to extract network information with psutil module:

```
psutil.net_io_counters(pernic=True)
```

output:

```
{'eth0':netio(bytes_sent=485291293,bytes_recv=6004858642,pack-
ets_sent=3251564, packets_recv=4787798, errin=0, errout=0, dropin=0, drop-
out=0),
 'lo':netio(bytes_sent=2838627,bytes_recv=2838627, packets_sent=30567, pack-
ets_recv=30567, errin=0, errout=0, dropin=0, dropout=0)}
```

Pandas
Pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc (TutorialsPoint,2019).

Some key features of Pandas are (TutorialsPoint,2019):

- Fast and efficient DataFrame object with default and customised indexing.
- Tools for loading data into in-memory data objects from different file format.
- Reshaping and pivoting of data sets.
- Label-based slicing indexing and subsetting of large data sets.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.

It can be installed in Linux with below command:

```
$ pip install pandas
```

Threading
The threading module used to be the primary way of accomplishing concurrency. Some key features of threading are (Carbajal,2013).

- A program can remain responsive to input and this happens both on single and on multiple CPUs.
- Allows to run multiple threads while one thread is waiting for a task to complete.
- Some programs are easy to express using concurrency which leads to elegant solutions, easier to maintain and debug.

In python threads only a single thread is allowed to execute in the python interpreter at once. (Carbajal,2013).

OS
This module provides a portable way of using operating system dependent functionality. It allows the user to interact with the terminal and it is enriched with built in functions. As an example to make a test directory in Linux a simple one line code is used: `os.mkdir("test").` It can be used to implement Linux commands and same output can be retrived as Linux commands do. (Python Software Foundation,2019b).

TinyDB
TinyDB is a simple database that just works without lots of configuration. Any document can be stored and it is represented as JSON format. It works with both Python2 and Python3. It is extensible and it's great when only simple data types are involved but it cannot handle more complex data types like custom classes. On Python 2 it also converts strings to Unicode strings upon reading (Siemens,2016).

It can be installed easily with:

```
$ pip install tinydb
```

And DB can be created with one line code as following:

```
from tinydb import TinyDB, Query
```

```
db = TinyDB('db.json')#to store in db.json Database
```

The `db.json` DB will be created after running the script and then data can be stored like following code:

```
db.insert({'type': 'apple', 'count': 7})
```

Flask

Flask is a lightweight web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It has become a popular python web application framework. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy (Python Software Foundation,2019c).

To install Flask in Linux the command is:

```
$ pip install Flask
```

Flask is very easy, and it has default port 5000. A simple hello world program with Flask would be:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')

def hello():

    return 'Hello, World!'
```

output

```
$ FLASK_APP=hello.py flask run

 * Serving Flask app "hello"

 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

ConfigParser

It is used as a basic configuration file parser. Through this file can be customised easily. ConfigParser consists of sections and in these sections different values can be included. As an example, a ConfigParser section looks like following (Python Software Foundation,2019c)

[section1]

value_1=n1

value_2=n2

[section2]

value_3=n1

value_4=n2

3 Requirements Analysis

The objective of this thesis work to design, development and implementation of an energy consumption simulator for the CORE network emulator and analyse the performance based on use cases.

3.1 General Objectives

In this task an application will be developed which is able to simulate energy consumption of individual nodes in CORE network emulator. This application will consist of a configuration file to specify the parameters. The parameters are selected relative to the main system resources (CPU, Disk, Memory and Network). Therefore, based on these specifications developed application will be able to emulate the energy of CORE nodes. User will be able to change configuration parameters while server will be running on CORE nodes and can read new battery level. Through http requests user will be able to get response of current battery, total consumption and history. Throughout the work python will be used as development language.

To implement the task python and CORE network emulator will be used in the Virtual Machine. The whole implementation and performance analysis will be based on following points

- Design of the application.
- Evaluation of Linux terminal commands.
- Implementation of energy consumption simulator.
    - Implementation of data aggregator, data processor, battery emulator, consumption history.
    - Running script at different battery level.
    - Implementation of Consumption History.
- Usage of the application

3.2 Clarifying the Requirements

In this part there will be a detail discussion to reach the target state of this thesis. To realise the development python 2.7 version will be used and Energy Consumption Simulator will be developed for CORE network emulator.

3.2.1 Comparison with Linux Terminal Command

Parameters that will be used in the application to emulate battery can also be read by the Linux terminal commands. There can be multiple commands to read data of the specific parameters RX/TX, CPU/RAM, DISK. In this task these commands will be evaluated and will be compared to the result of the developed application.

3.2.2 Development of the Application

In this thesis task an application has to be developed which will be responsible to act as an energy simulator.

Energy consumption parameter metrics to develop the simulator are:

- Network Usage (Receive transmission bytes of ethernet-RX/TX)
- CPU usage(percentage)
- Memory Usage(percentage)
- Disk Usage(read/Writes)
- Consumed Power (percentage)

This application will consist of a Configuration file, *Data Aggregator, Data Processor, Battery Emulator, Client Script Executor, Consumption History*. The components which will be realised in this task are given below. A conceptual design of these components is given below in Figure 3.1.

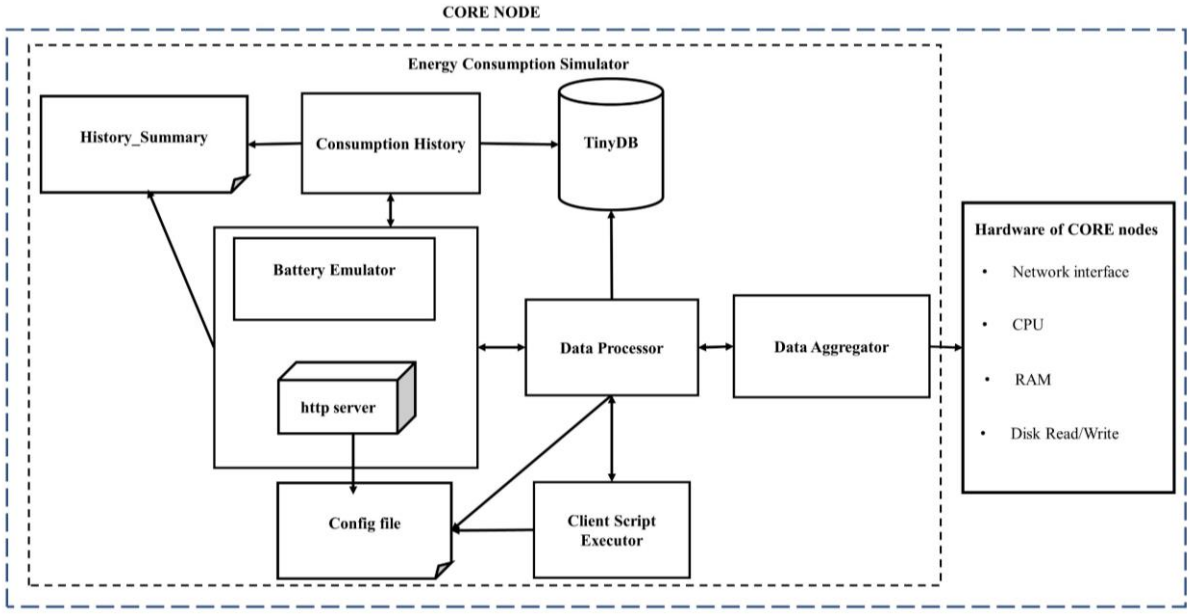


Figure 3.1: Conceptual design of the application

Data Aggregator
It will be responsible to read data from CORE nodes. In total six parameters will be used which will act as load in the energy consumption. They are:

1. RX byte (receiving byte of an interface).
2. TX byte (transmitting byte of an interface).
3. CPU usage.
4. RAM usage.
5. Disc read byte.
6. Disc write byte.

Data aggregator will read these data every second. These parameters will be taken to act as load for the consumption and will be calculated in total to get the total consumption. Loopback interface will be ignored here as this does not consume energy. The calculation will be done as following:

- Total RX byte without loopback RX.
- Total TX byte without loopback TX.
- Total CPU usage of all process.
- Total RAM usage of all process.
- Total bytes for read operation of disk.
- Total bytes for write operation of disk.

Configuration
In Configuration file user can define parameters to emulate battery consumption. These parameters will be processed by data processor to get total consumption. It will be possible to change the configuration as user needs. Mentioned six parameters above in data aggregator section will be specified in this file. An example to define these parameters can be given as below:

- N% battery consumption for 1000 RX bytes.

- N% battery consumption for 1000 TX bytes.
- N% battery consumption for 1% CPU usage.
- N% battery consumption for 1% RAM usage.
- N% battery consumption for 1000 disk read bytes.
- N% battery consumption for 1000 disk write bytes.
- Interval to get consumption history.

As an example, in configuration file if it is given as: rx=n, that means 1000 bytes consume n% battery.

Data Processor
Based on the parameter specification defined in configuration file data processor will process the data to get the total consumption. After reading the parameters, those parameters who are acting as load will be processed by data processor and individual consumption will be calculated consumed by the six parameter load. Therefore, in total there will be six load consumption and finally there will be a total consumption which will be an addition of these loads. An example can be given as below:

- In configuration file: rx=n, that means 1000-bytes RX consumes n% battery. Total RX bytes read by data aggregator from node n1 of CORE is N bytes. Therefore, consumption by RX is (n/1000) ×N

Similarly, all other parameter will be calculated, and total consumption will be taken by adding all those. Based on CPU usage the system mode will be defined as idle or active.

Consumption History
To store the history a small database will be used in this development. User can define in the configuration if consumption history should be activated. If history is activated history will be stored in the database and user can read this by requesting through http request. This will be including average consumption history and average usage of RX, TX, CPU, RAM, Disk Read Write.

Client Script Executor
This component will be used to run shell script at different battery level. The shell scripts will be user definable and it will be defined in configuration file.

Battery Emulator
It will be responsible to get the total consumption and subtract it and update the battery level. It will also consist of a http server to serve user with http requests.

This http server will be responsible to and facilitate user with current battery, consumption history and configuration parameters can be changed upon http request. General workflow will be handled by three CORE parts: Battery Emulator, Data Processor, Data Aggregator. These three will be responsible to show current battery and consumption.

3.2.3 Integration of Application in CORE as a Service

As CORE runs on Linux machine, it will be installed in virtual machine. This application has to be integrated as service-script in CORE service feature. It is already told that this application will consist of a webserver. And the server will keep running when this service is checked from CORE service feature. User can retrieve the battery level, consumption history through the HTTP request.

3.3 Time Frames

Milestones:

14.01.2019 to 26.01.2019:	Analysis of the requirements
27.01.2019:	Submission of the requirements analysis
27.01.2019 to 27.02.2019:	Literature review, installing CORE, set up necessary environment.
27.01.2019 to 27.03.2019:	Writing up the thesis and the documentation
27.03.2019 to 30.04.2019:	Elaborating of the concept and planning of the prototype
27.03.2019 to 10.05.2019:	Implementation of the prototype
15.06.2019:	Submission of the Thesis draft to supervisor
15.06.2019 to 25.06.2019:	Revise the thesis based on the proposals from the supervisor
27.06.2019:	Submitting the Thesis to the examination office.

3.4 Target State

For the implementation of the prototype, the final state shown in Figure 3.1 is expected. It is already mentioned that battery emulator will be including of a http server. It will be deployed in CORE as a service. After completion of the development this application will be able allow the user to know current battery, consumption history and can change the configuration parameters. These three responses can be retrieved by http request sent by the user.

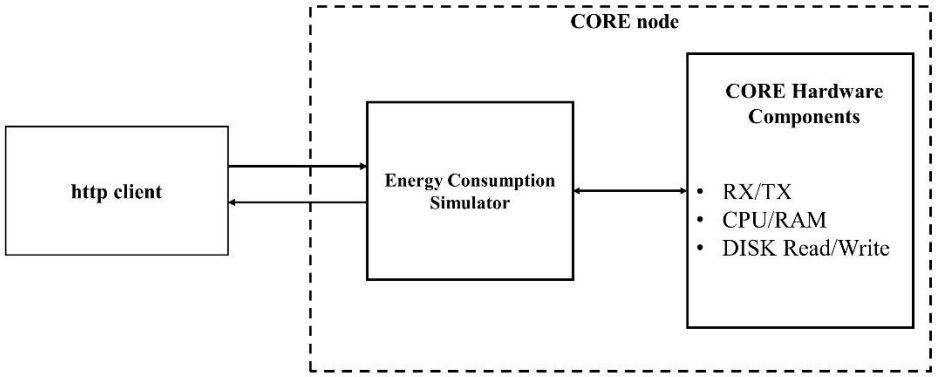


Figure 3.1: Target state

3.5 Software Requirements

Plan of configuring virtual machine and choice of tools is given below:

- Oracle VirtualBox 6.0.8(Oracle,2016).
- Virtual Machine: OS-ubuntu 18.04 LTS.
- CORE emulator: Version-5.2.
- Script language: Python-2.7.

3.6 Use Cases for the Prototype

To demonstrate and analysing the performance of the Application of *Energy Consumption Simulator* following approaches will be implemented:

Testing of the parameters
To calculate the total consumption six parameters will be acting as load. These will be evaluated and compared with the existing commands of Linux. It will be tested checking the parameters every second. Some testing commands of Linux like `stress -c` (stressing CPU with load) `or ping -f` (continuous flow of data: flood ping) will be used to verify the value.

Running script at different battery level
Shell scripts defined by the user will be run at different battery level which will be specified in configuration file. Regardless of what script is used these scripts will be running at different battery levels. As an example, if user specify `script1.sh` at battery level 10, this `script2.sh` will be run when battery is 10% or below 10% level. This will be visualised with a network scenario and changing the route with running user definable different shell scripts.

GET consumption and current battery through http request
It will be possible to get response from http server implemented in *Battery Emulator* from CORE corresponding node, outside of the CORE node (Linux Terminal), from browser and also from Widget. For continuous data receiving and sending `ping -f` command of Linux will be used, and changing battery percentage can be shown with Widget.

Specifying maximum CPU/RAM consuming process
It will be possible to get highest CPU RAM consuming process while getting current battery level and consumption. It will be possible to get three PID and USER which are consuming maximum CPU RAM at that particular time.

Changing configuration through http request
It will be possible to change the configuration file as user needs. When this application will be running in the CORE as a service user can send POST request to change the energy consumption parameters. The battery percentage will be recalculated with the new configuration parameters.

Read history from database
Consumption history and parameter usage will be stored in the database and therefore user will be able to read it through http request. Average usage of all the parameters and average consumption will be retrieved using user definable interval. User will be able to get this response through http GET request.

Simulation with parameter load
Simulation with the application will be tested with loaded network, CPU, RAM and Disk. Nodes in the network will be activated with the Energy Consumption service and changes of consumption in different nodes will be simulated. Changes in consumption and battery percentage with the changing parameter usage will be shown.

4 Realisation

This chapter will describe the implementation and analysis of this thesis task. This chapter will be including the following sections:

- Environment preparation.
- Configuration of Ubuntu host.
- Evaluation of Linux terminal commands.
- Implementation of energy consumption simulator.
- Summary of implementation.
- Deployment of the Application in CORE as a service.

4.1 Environment Preparation

To achieve the goal of the thesis work at first it is necessary to prepare the environment. It is mentioned in section 3.5 that CORE version 5.2 will be used in this task and to work with CORE it is necessary to have a Linux environment. And in this task VirtualBox will be used to install Ubuntu 18.04 and all other software tools will be installed in this Ubuntu virtual machine.

To set up necessary environment following steps have been followed.

4.1.1 Software Tools Installation

Some software tools should be installed to complete the thesis task. To complete the task Ubuntu 18.04 OS system should be installed in VirtualBox and also CORE will be installed in this virtual machine. This section describes the installation procedure of necessary software tools.

Installing VirtualBox
Oracle VM VirtualBox is a free and open source hosted hypervisor for virtualization technique which allows the user to run multiple operating systems inside the Virtual Machine (VM). It is available for multiple OS like Windows, Linux, MAC and Solaris operating system. Users of VirtualBox can load multiple guest OSes under a single host operating-system (host OS). Each guest can be started, paused and stopped independently within its own virtual machine (VM). That means, the guest operating system is independent of host operating system. (Oracle,2016).

In this thesis task ubuntu 18.04 is installed using virtual machine. The latest version of VirtualBox v 6.0.8 can be downloaded from the official website (Oracle,2016). After launching the downloaded file installation prompt will be appeared as following Figure 4.1 and user will be guided to complete the installation successfully.

An example of creating virtual machine and installing Ubuntu 18.04 on that virtual machine is illustrated from Figure 4.1 to 4.9.

Figure 4. 1 VirtualBox Installation Wizard

4.1.2 Creating Virtual Machine

In this task ubuntu 18.04 will be installed in the virtual machine. First new virtual machine must be created by clicking New button from the top left corner of the home screen of VirtualBox. A window will appear like Figure 4. 2 asking for the name of the virtual machine and type of operating system to be installed on that machine.

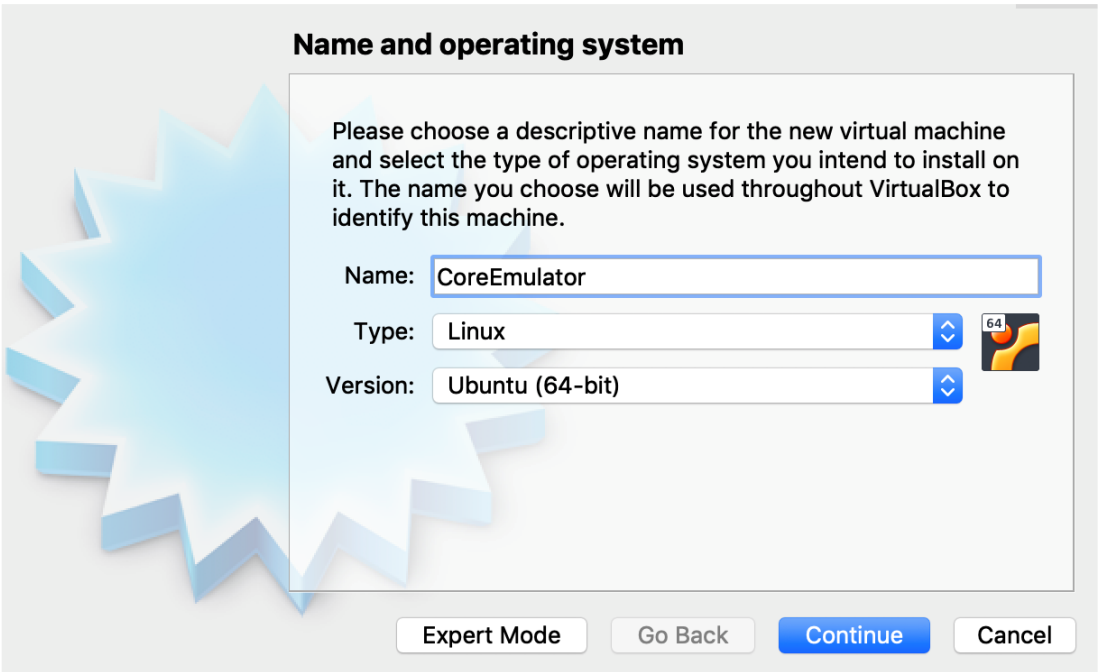


Figure 4. 2 Assigning name and OS type of the VM

In the next step it will ask for the memory (RAM) size that will be allocated for the new virtual machine. Figure 4.3 illustrates that below:

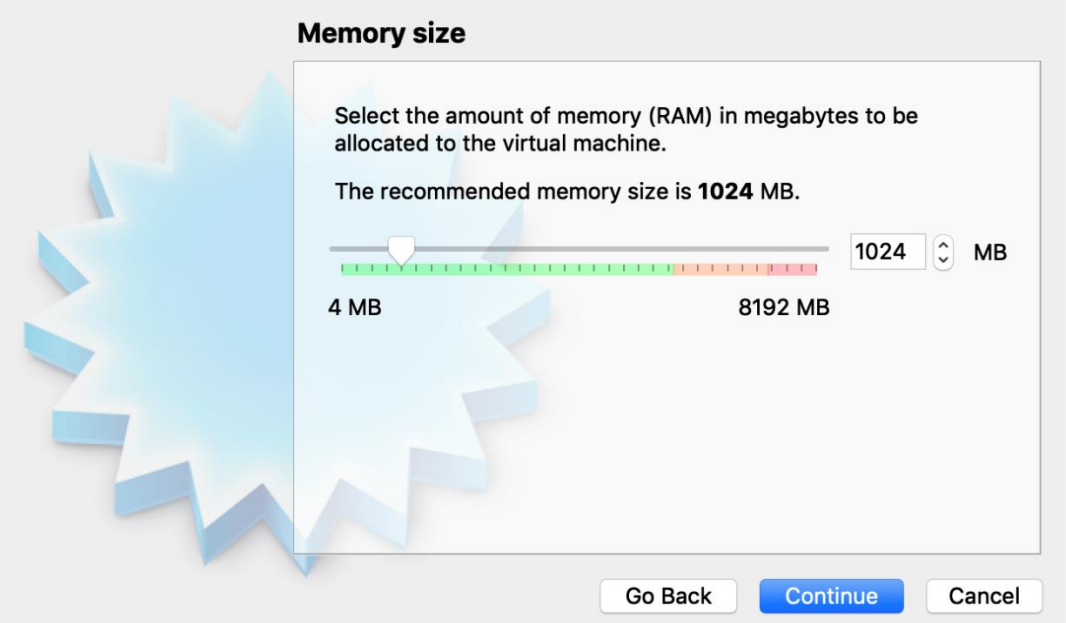


Figure 4. 3 Defining RAM size

The newly created virtual machine will need at least one hard disk to operate and as this will be a new virtual machine, therefore it is better to create a new virtual disk instead of using the existing virtual hard disk like Figure 4.4:

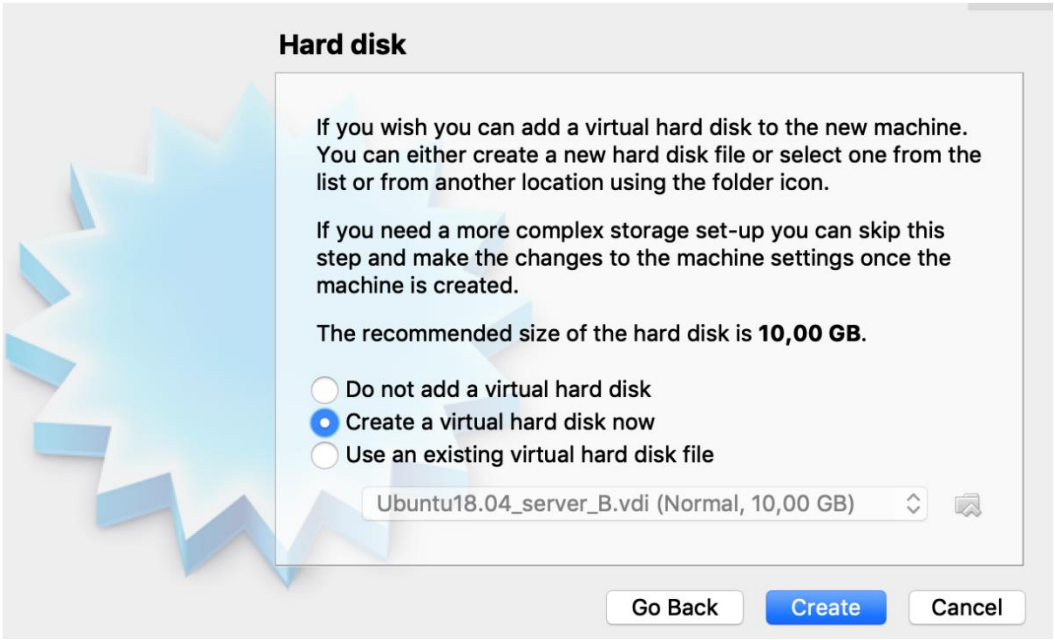


Figure 4. 4: creating new virtual hard disk

Then it will ask to select virtual hard disk type. In this thesis task VirtualBox Disk Image has been used as following Figure 4.5

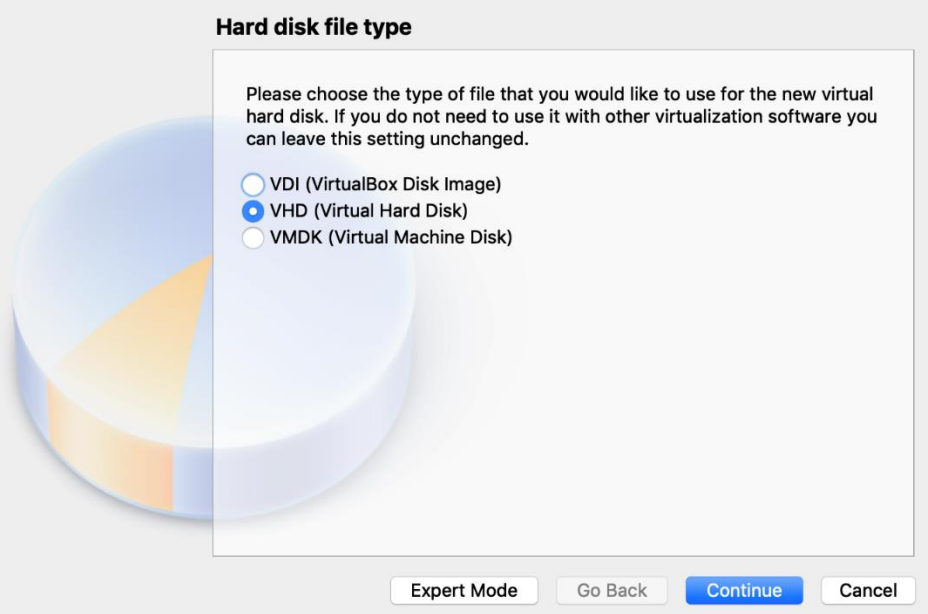


Figure 4. 5: Selecting virtual disk type

In next section it will ask how virtual hard disk file should grow with the usage. It should be selected Dynamically allocated as it allows user to use the space on physical hard disk as it fills up rather taking a static size. Figure 4.6 illustrates the option:

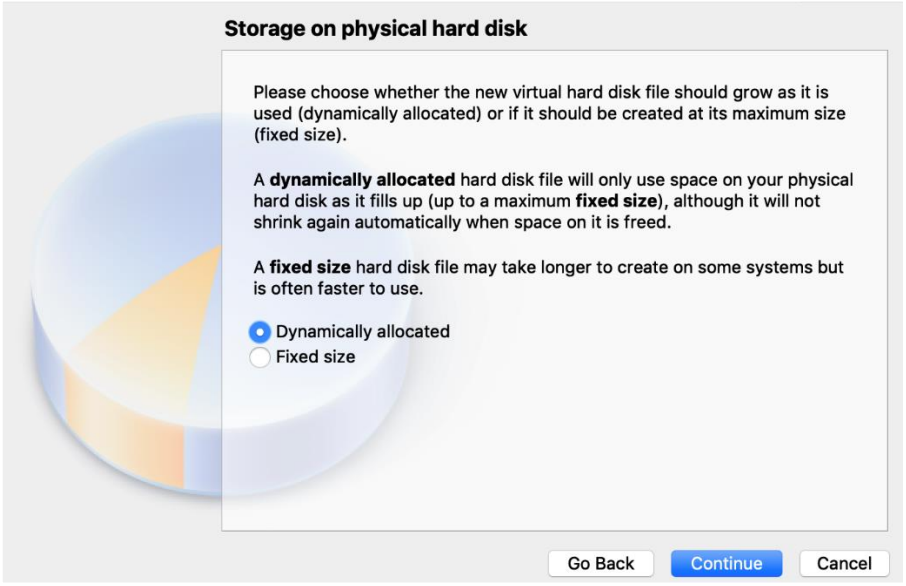


Figure 4. 6 Selecting the storage type

Finally, it will ask for the name of the virtual hard disk, in which directory to store it and the maximum size of the newly created virtual hard disk. It can be done like Figure 4.7:

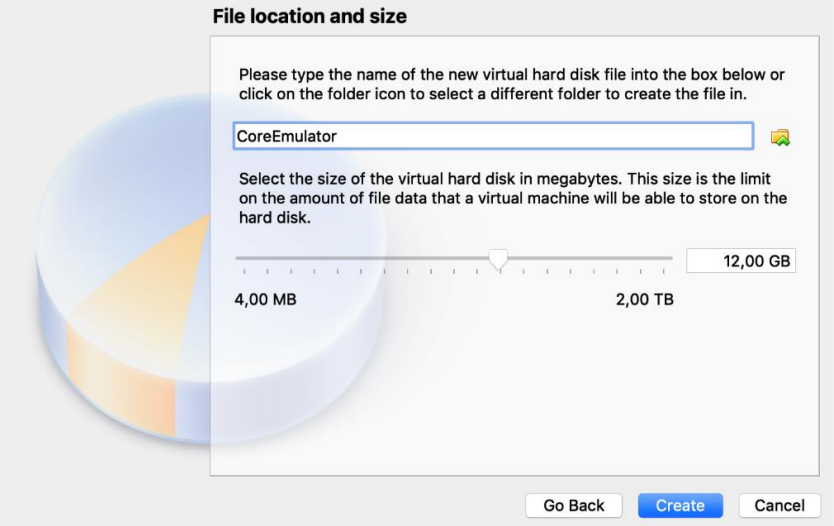


Figure 4. 7:Network configuration of the VM

After following the above-mentioned steps, virtual machine will be created. Now to install ubuntu 18.04 in the newly created VM an ISO file of the operating system installation media has to be loaded as an optical drive of the VM. This option can be found on the settings of the VM as shown below figure 4.9. The ISO file for ubuntu can be downloaded from official website of Ubuntu (Canonical LTD, 2019).

After loading the ISO file network adapter needs to be selected. Following Figures 4.8 and 4.9 depict the installation procedure.

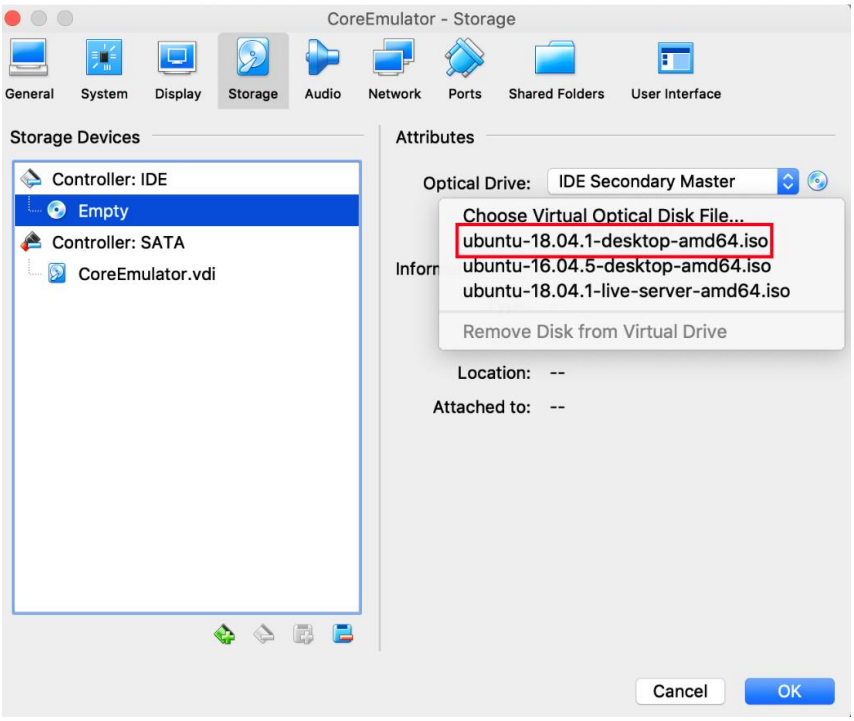


Figure 4. 8: Loading OS installation media

Network configuration can be set Bridged Adapter selecting network configuration from network tab.

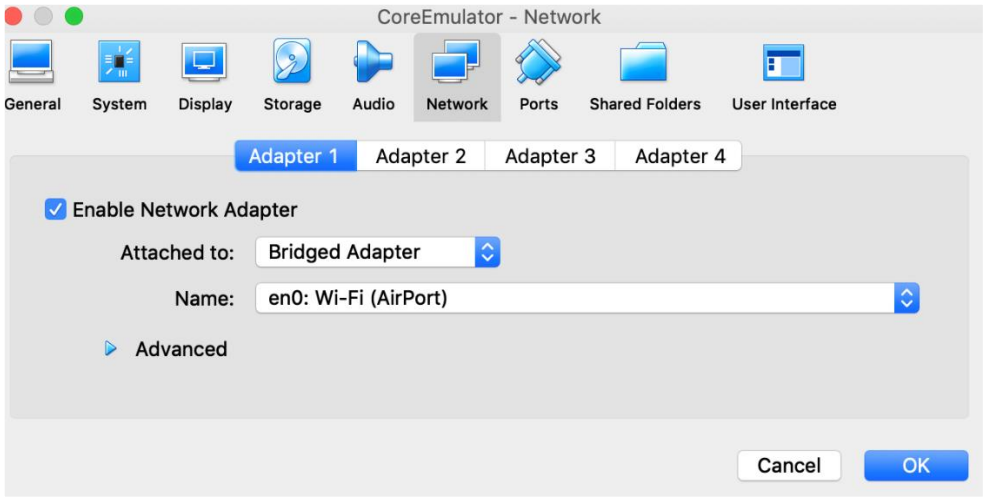


Figure 4. 9: Network configuration of the VM

After these steps ubuntu can be installed in this VM. While installing OS username and password of this VM should be specified. In this thesis task VirtualBox name is servercore2 and credentials are: username: *server* and password *1234*.

4.2 Configuration of Ubuntu Host

In this task python 2.7 is used for the implementation and in Ubuntu 18.04 it is already available, only CORE network emulator has to be installed on ubuntu host. Before installing the CORE, some configurations can be done as following:

Prerequisites by Ubuntu Version Ubuntu 18.04 (LTS)

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Uninstalling Unnecessary Softwares

```
$ sudo apt -y remove --purge avahi-daemon
$ sudo apt -y remove --purge ufw
```

4.2.1 Installing CORE Network Emulator on Ubuntu Host

For this thesis CORE 5.2 has been used and following steps can be followed to install CORE. To install CORE first some prerequisite packages, need to be installed that allow to build the CORE system (Linkletter,2017).

Prerequisites for installing CORE emulator

```
$ sudo apt-get update
$ sudo apt-get install git
$ sudo apt-get install bash bridge-utils ebtables
$ sudo apt-get install iproute libev-dev python tcl8.5 tk8.5 libtk-img
$ sudo apt-get install autoconf automake gcc libev-dev make python-dev
```

```
$ sudo apt-get install libreadline-dev pkg-config imagemagick help2man
```

Then Quagga needs to be installed to allow the routing:

```
$ sudo apt-get install quagga
```

Configuring Wireshark:

```
$ sudo rm /usr/bin/wireshark
$ sudo ln -s /usr/bin/wireshark-gtk /usr/bin/wireshark
$ sudo dpkg-reconfigure wireshark-common
$ sudo adduser $USER wireshark
```

Downloading CORE

To download the specific version of CORE network emulator, desired version can be chosen from Branch tab. In following figure, it is shown that CORE release 5.2 is selected from the Branch tab (CORE,2019). And it can be downloaded or cloned with Clone or download tab as depicted in Figure 4.10:

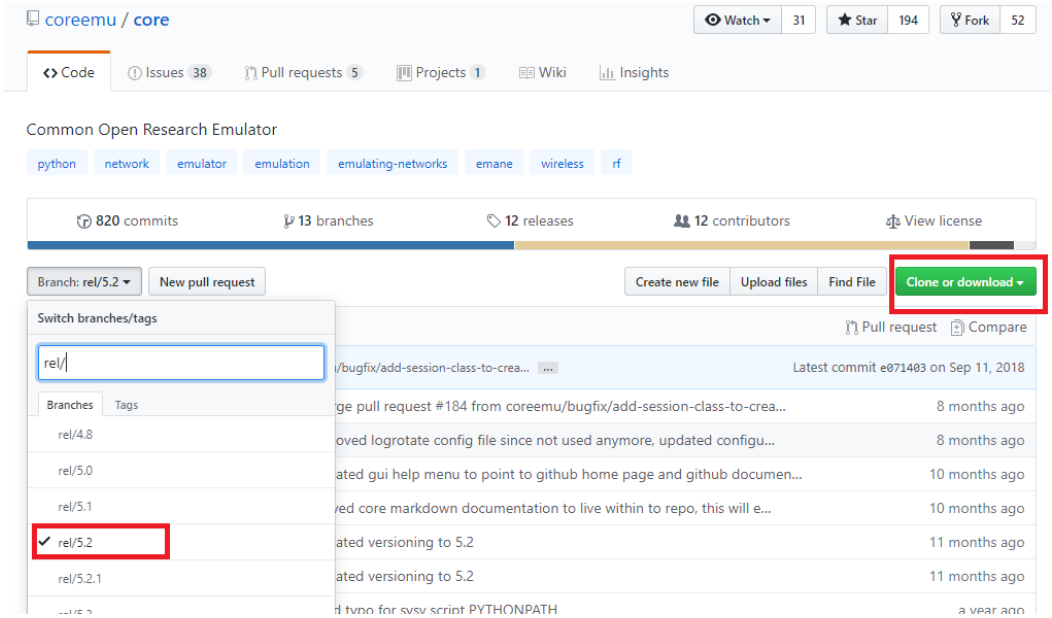


Figure 4. 10: Selecting and Downloading CORE v5.2 from GitHub

Unzip and Install CORE

By using below commands CORE has been unzipped

```
$ cd Downloads
$ unzip core-rel-5.2.zip
```

After unzipping, CORE has been installed by using below commands.

```
$ cd core-rel-5.2
$ ./bootstrap.sh
$ ./configure
$ make
$ sudo make install
```

Setting up Quagga:

```
$ sudo touch /etc/quagga/zebra.conf
$ sudo touch /etc/quagga/ospfd.conf
$ sudo touch /etc/quagga/ospf6d.conf
$ sudo touch /etc/quagga/ripd.conf
$ sudo touch /etc/quagga/ripngd.conf
$ sudo touch /etc/quagga/isisd.conf
$ sudo touch /etc/quagga/pimd.conf
$ sudo touch /etc/quagga/vtysh.conf
$ sudo chown quagga.quaggavty /etc/quagga/*.conf
$ sudo chown quagga.quaggavty /etc/quagga/*.conf
$ sudo chmod 666 /etc/quagga/*.conf
```

Configuring Quagga daemons:

To configure Quagga daemons daemon file should be edited from `/etc/quagga/daemons` and started `zebra` and `ospfd` daemons by using below command which is shown in Figure 4.11

```
$ sudo nano /etc/quagga/daemons
```

Figure 4. 11: Configuring Quagga daemons

h) Setting up environment variables

To avoid the Quagga **`vtysh END`** problem in Ubuntu Linux following variables have to be set.

```
$ sudo bash -c 'echo "export VTYSH_PAGER=more" >>/etc/bash.bashrc'
$ sudo bash -c 'echo "VTYSH_PAGER=more" >>/etc/environment'
```

Start CORE from terminal

```
$ sudo /etc/init.d/core-daemon restart
[ ok ] Restarting core-daemon (via systemctl): core-daemon.service.
$ core-gui
```

Then CORE GUI will appear like below Figure 4. 12:

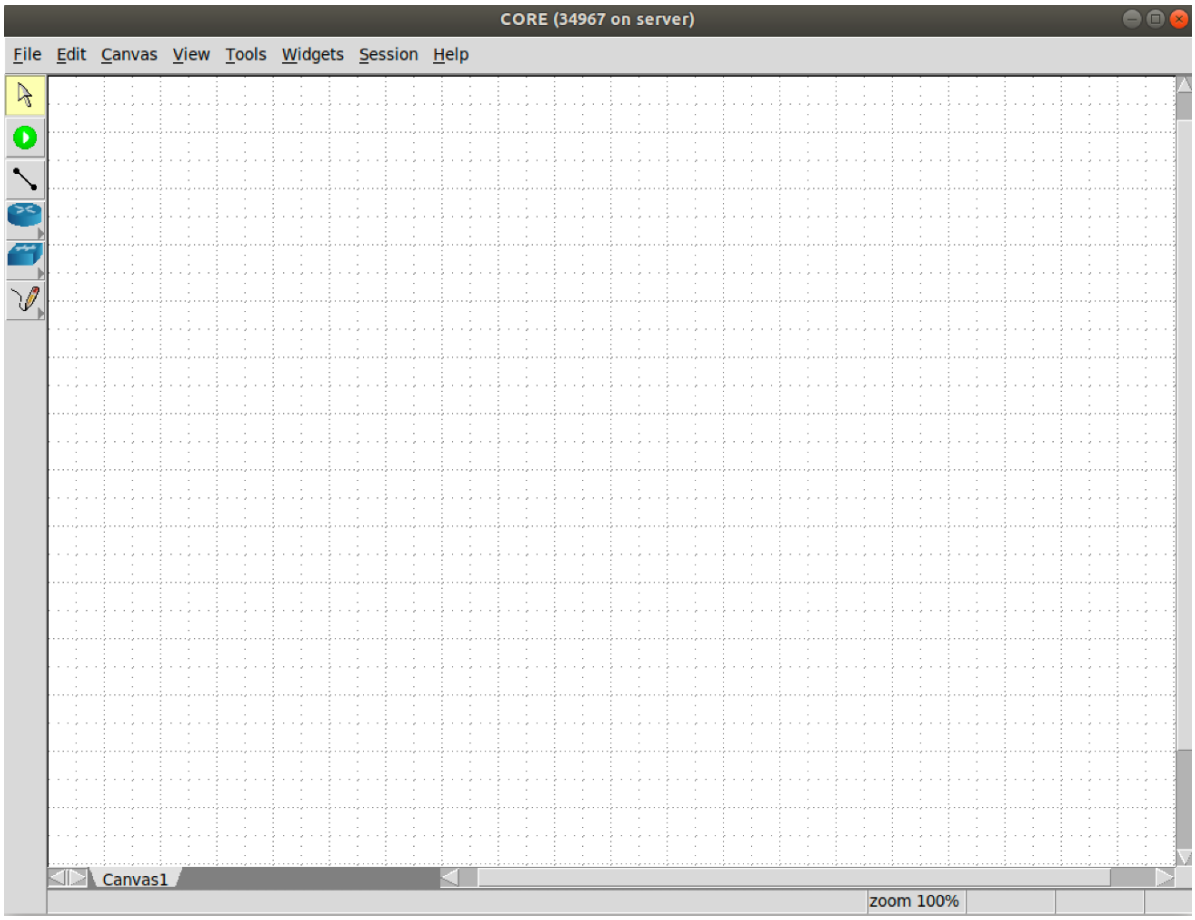


Figure 4. 12: CORE GUI

4.3 Evaluation of Linux Terminal Commands

Before developing Energy Consumption Simulator some Linux commands have been evaluated to take those commands as a reference for development task. It is necessary to evaluate those commands which are able to get data related to our chosen parameters. They are:

- Network receive and transmission (RX/TX) bytes of all interfaces.
- CPU/RAM usage of all processes.
- Disk Read/Write of all processes.

.

4.3.1 Available Linux command for Rx/Tx

To get desired parameters several commands are available in Linux. In this section those commands will be discussed and also which commands have been chosen it will be discussed in section 4.3.5.Some popular commands which are used frequently by Linux users are given below:

ifconfig

This is the most used commands to extract receive and transmission bytes (RX/TX) of ethernet interface. It is a command line interface tool for network interface configuration (Canonical LTD,2019a). Figure 4. 13 depicts ifconfig terminal command. It can show RX/TX bytes of all ethernets and also other information like RX/TX packets, IP address, MAC address and so on.

```
root@n1:/home/servercore2/Desktop# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 2001::1  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::200:ff:feaa:0  prefixlen 64  scopeid 0x20<link>
        ether 00:00:00:aa:00:00  txqueuelen 1000  (Ethernet)
        RX packets 333  bytes 33556 (33.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 270  bytes 24232 (24.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 4. 13 ifconfig to extract RX/TX bytes

ip -s link

This command is also used to extract RX/TX bytes. It also shows some other information like packets errors/dropped/overrun/mcast information (Canonical LTD,2019b). It can be depicted in Figure 4. 14

```
root@n1:/home/servercore2/Desktop# ip -s link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    RX: bytes  packets  errors  dropped overrun mcast
    0          0        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    0          0        0       0       0       0
124: eth0@if125: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    link/ether 00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    RX: bytes  packets  errors  dropped overrun mcast
    20796      190      0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    12088      132      0       0       0       0
```

Figure 4. 14:ip -s link to extract RX/TX bytes

cat /proc/net/dev

The *proc* filesystem is a pseudo-filesystem which provides an interface to kernel data. With this command user can read information about network from proc filesystem (Canonical LTD,2019c). Scenario can be depicted in Figure 4. 15

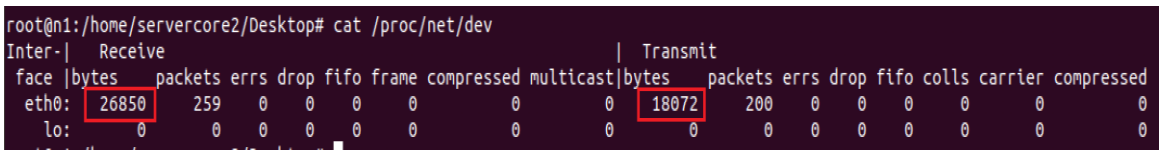


Figure 4. 15: cat/proc/net/dev to extract RX/TX bytes

4.3.2 Available Linux Commands for CPU usage

In this section some frequently used commands to extract CPU will be described. There are several commands which can be used to extract desired parameter CPU usage percentage. And for calculation of energy consumption CPU usage every process should be listed. Some commands are mentioned below:

ps aux

ps displays information about a selection of the active processes. *aux* is used to define flags like *a*=show processes for all users, *u*=display the process's user/owner, *x*=also show process not attached to a terminal Canonical LTD (Canonical LTD,2019d). That means it is able to show CPU usage percentage of all processes. Following Figure 4. 16 shows CPU usage of all processes.

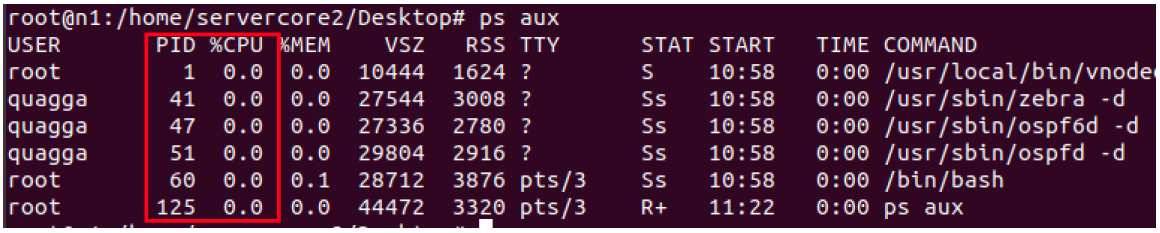


Figure 4. 16:ps aux command for CPU usage

iostat

iostat reports information about CPU activity. It can show the collective CPU utilization at user level (Computer Hope,2019). It is depicted in Figure 4. 17.

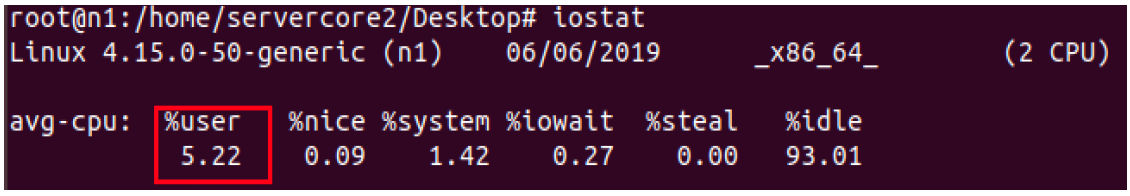


Figure 4. 17: iostat command for CPU usage

Sar

sar is able to show the collective CPU usage with defining interval to monitor CPU. After this interval it can show the total CPU usage. In Figure 4. 18 an example is shown where every 20 seconds it gets the collective CPU usage as interval is given 20 with command: `sar 20`(Canonical LTD,2019e).

Red color marked in the Figure 4. 18 shows first 20 seconds usage and then blue marked content shows next 20 second usage.

```
root@n1:/home/servercore2/Desktop# sar 20
Linux 4.15.0-50-generic (n1)    06/06/2019      _x86_64_        (2 CPU)

11:24:53 AM     CPU     %user     %nice   %system   %iowait    %steal     %idle
11:25:13 AM     all      0.78      0.00      0.28      0.05      0.00     98.89
11:25:33 AM     all      1.49      0.00      0.48      0.05      0.00     97.98
```

Figure 4. 18: sar command for CPU usage

4.3.3 Available Linux Commands for RAM usage

This section describes some commands to get desired output of RAM or memory usage percentage for all processes. Below some of the frequently used commands are given:

ps aux
In section 4.3.2 and Figure 4. 19 it can be shown that *ps aux* can extract RAM usage of all processes like CPU usage.

```
root@n2:/tmp/pycore.45397/n2.conf# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME
root          1  0.0  0.0  10572  1788 ?        S    10:45   0:00
root         28  1.4  0.5  77248 21312 ?        Ss   10:45   0:00
quagga       52  0.0  0.0  27544  3144 ?        Ss   10:45   0:00
quagga       58  0.0  0.0  27340  2896 ?        Ss   10:45   0:00
quagga       62  0.0  0.0  29804  2872 ?        Ss   10:45   0:00
root         70  0.0  0.1  28580  3860 pts/5    Ss   10:45   0:00
root         80  0.0  0.0  28580  3680 pts/7    Ss+  10:45   0:00
root         90  0.0  0.0  28580  3680 pts/9    Ss+  10:45   0:00
root        101  0.0  0.0  44472  3300 pts/5    R+   10:45   0:00
```

Figure 4. 19:ps aux for RAM usage

vmstat
This command is able to show the amount of used bytes fo

r virtual memory(swpd), idle memory(free) used as buffers, memory used as cache and so on (Canonical LTD,2019f). Figure 4. 20 illustrates an example.

```
root@n1:/tmp/pycore.45869/n1.conf# vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0      0 1451480  79276 958904    0    0    67    41  183  531  5  1 93  0  0
```

Figure 4. 20: vmstat to show memory usage in bytes

free -m
With this command it is possible to show total amount of used, free memory and also amount of used swap or virtual memory (Code Yarns,2016). It can be depicted in following Figure 4. 21:

```
root@n1:/tmp/pycore.45869/n1.conf# free -m
              total        used        free      shared  buff/cache   available
Mem:           3611        1180        1417          31        1013        2173
Swap:           567           0         567
```

Figure 4. 21: free -m to show memory usage in bytes

4.3.4 Available Linux Commands for Disk Read/Write

For this task it is essential to retrieve disk read write parameters in bytes for all processes. That means all processes that are reading writing at present time need to be identified for the calculation. Some frequently used command are given below.

pidstat -dl

The *pidstat* command is used for monitoring individual tasks currently being managed by the Linux kernel. A specific task activities can be selected by using flags with *pidstat*. As example, flag *-dl*:can display PID, KB_rd/s,KB_wr/s(The Linux Foundation,2019). Command *pidstat -dl* can be used to retrieve disk read/write bytes of all current processes as Figure 4. 22:

```
root@n2:/home/servercore2/Desktop# pidstat -dl
Linux 4.15.0-50-generic (n2)    06/23/2019      _x86_64_        (2 CPU)

04:35:04 PM   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
04:35:04 PM     0         1      0.00      0.00      0.00       0  /usr/local/bin/vnoded
/tmp/pycore.44261/n2.conf
04:35:04 PM   122        50      0.00      0.00      0.00       0  /usr/sbin/zebra -d
04:35:04 PM   122        56      0.00      0.00      0.00       0  /usr/sbin/ospf6d -d
04:35:04 PM   122        60      0.00      0.00      0.00       0  /usr/sbin/ospfd -d
```

Figure 4. 22: pidstat -d showing Disk Read &Write in KB for individual process

iostat

iostat can show the information of total hard disk device but it does not show the read/write amount of processes. In Figure 4. 23 it can be shown that *sda* or hard disk read write can be retrieved by this command.

And it also shows the loop devices which are responsible to making plain files with no relation with the RAM or memory (Computer Hope,2019).

```
root@n1:/tmp/pycore.45869/n1.conf# iostat
Linux 4.15.0-50-generic (n1)    06/06/2019      _x86_64_        (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           4.81    0.08    1.31    0.25    0.00   93.55

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
loop0             0.01         0.17         0.00       1062          0
loop1             0.01         0.05         0.00        330          0
loop2             0.01         0.06         0.00        342          0
loop3             0.01         0.17         0.00       1068          0
loop4             0.01         0.02         0.00        116          0
loop5             0.01         0.02         0.00        110          0
loop6             0.01         0.05         0.00        336          0
loop7             0.01         0.05         0.00        330          0
sda               8.62       125.15        78.55     765837     480684
loop8             0.01         0.02         0.00        111          0
```

Figure 4. 23: iostat showing disk read/write usage of hard disk

Dstat

This command works in real-time. It can show total number of read (read) and write (writ) operations on disks. It is a tool which is used for continuous monitoring (Tecmint2019).

```
root@n1:/tmp/pycore.45869/n1.conf# dstat
You did not select any stats, using -cdngy by default.
Color support is disabled as curses does not find terminal "unknown
--total-cpu-usage-- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai stl|  read  writ|  recv  send|  in   out | int   csw
  4   1  94   0   0|  120k   78k|   0     0 |   0     0 | 306   877
  6   1  93   0   0|    0     0 |   0     0 |   0     0 | 399   947
  4   1  96   0   0|    0     0 | 254B  274B|   0     0 | 215  1332
  5   0  95   0   0|    0     0 |   0     0 |   0     0 | 336   755
```

Figure 4. 24: dstat command showing Disk read/write every second

4.3.5 Chosen Commands and Module for Implementation

Choosing psutil over Linux commands for RX/TX
Command mentioned in section 4.3.1 can be used to retrieve RX/TX bytes. Every command shown in the specified section provides desired output RX/TX bytes with some additional information. As in this thesis task only RX/TX bytes of every ethernet is needed and so it should be extracted from above commands. But it will be complicated to extract the particular feature from commands explained in 4.3.1 as these command include many information than needed. That's why it has been avoided in this task. And Python has a built-in function which can exactly extract the particular information which is needed.

Psutil module of Python is used to extract network information and it is much easier to get the particular feature. Moreover, when it is needed to extract multiple ethernets RX/TX bytes it is easier to handle the calculation with *psutil* module (Rodola,2019).

Figure 4. 25 shows the *psutil* and *ifconfig* output. It can be shown that the output or result is same.

```
root@n1:/home/servercore2/Desktop# python data_aggregator.py && ifconfig
{"total_bytes_rx": 16378.0}
{"total_bytes_tx": 8216.0}
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 2001::1  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::200:ff:feaa:0  prefixlen 64  scopeid 0x20<link>
        ether 00:00:00:aa:00:00  txqueuelen 1000  (Ethernet)
        RX packets 142  bytes 16378 (16.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 88  bytes 8216 (8.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 4. 25: psutil to extract RX/TX bytes

Choosing ps aux for CPU/RAM
For this task *ps aux* command has been used. Our target is to extract CPU/RAM usage for all individual process to calculate the consumption. And *ps aux* fulfils this requirement.

Two more commands *iostat* and *sar* in section 4.3.2 have been mentioned for getting CPU usage. *iostat* command cannot show the CPU usage for individual process but showing collective usage. *sar* command is able to get CPU usage with intervals but not able to get the individual usage for every process. It also shows the collective usage like *iostat*. Therefore, *ps aux* command proves to be the best fitted command for retrieving CPU usage.

In section 4.3.3 two more commands along with *ps aux* have been mentioned. *vmstat* command can show the collective memory but not for individual process. *free -m* is also able to read total used, free memory in bytes but not for process. Therefore, *ps aux* is also taken as a command to get the RAM usage.

Choosing pidstat -dl for Disk Read/Write
To carry out the implementation it needs to be dealt with usage of every process. So, to get Disk parameter also it is necessary to get the usage for every single process. In section 4.3.4 it is mentioned how *pidstat -dl* command works. In Figure 4. 22 it is visible that *pidstat -dl* command is able to show the disk read/write amount for every process.

And other commands in section 4.3.4 are *iostat* and *dstat*. *iostat* command shows total hard disk usage rather than showing individual usage of a process (Figure 4. 23). So, this cannot be used in our task. Another command called *dstat* mentioned in section 4.3.4 which can be used for getting continuous usage of Disk read/write of collective usage. It also does not fulfil the requirements of extracting usage for every process. Therefore, for Disk read/write *pidstat -dl* has been chosen for retrieving disk parameters.

4.4 Implementation of Energy Consumption Simulator

In this chapter detail explanation of the implementation process will be covered. In chapter 3 section 3.2.2 it is already stated that to develop *Energy Consumption Simulator* there will be four components or python scripts which will be responsible to calculate energy consumption of every node in CORE. And to fulfil the requirement of section 0 *'Running script at different battery level'* another component called *Client script Executor* needed to be developed. Therefore, there are five components which have been developed to do the thesis task.

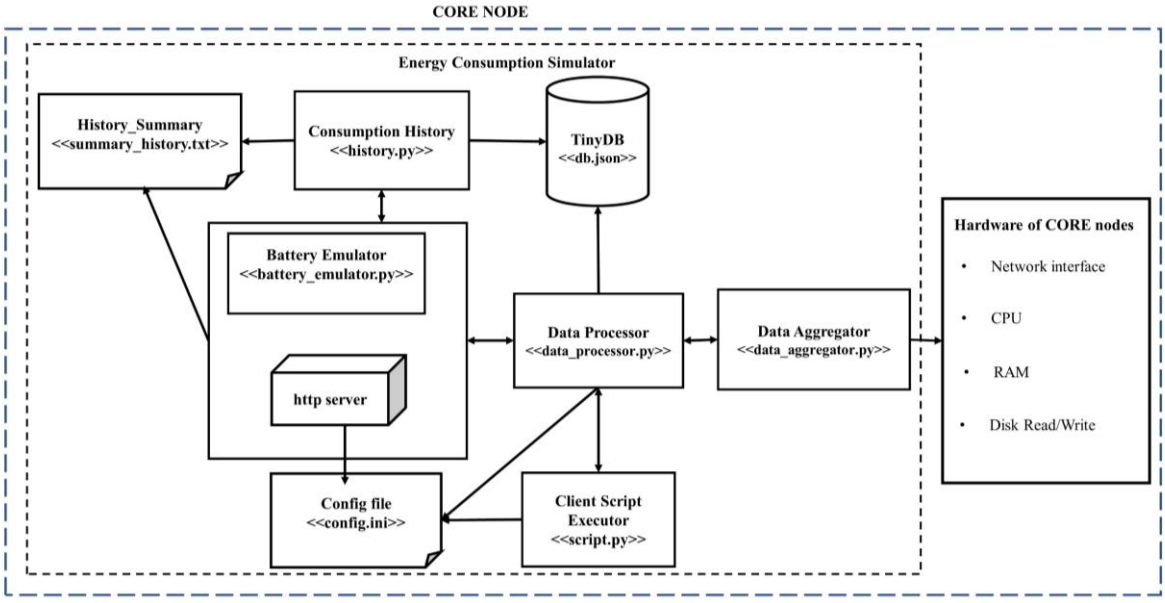


Figure 4. 26: Design of developed Energy Consumption Simulator

Above Figure 4. 26 shows all the components with corresponding python scripts of developed application. In following these components are discussed in short and in later portion of chapter 4 includes all the detail of these developed components.

Data Aggregator plays the role of aggregator which reads the parameters RX/TX, CPU/RAM, Disk Read/Write from CORE node. It can read these mentioned parameters of an individual node. As an example, if it runs in node n1 it retrieves the parameters of this node n1 only.

Client Script Executor component is developed with logic to run user defined shell script at different battery level given by the user in Config file.

And this aggregated data is processed by *Data Processor* based on configuration defined in *Config file.* After processing it will calculate the consumption. During processing the parameter, the average values of individual parameters and consumption will be stored in *TinyDB* database if user has activated to store history. And after processing these task *Data Processor* calls *Client Script Executor* to check if any script should be run. Therefore, *Data Processor* is connected to two python scripts: *data_aggregator.py, script.py* and these two should be imported by *Data Processor.* This process continues every one second until user interrupts.

And this *Data Processor is* imported by *Battery Emulator* and there will be a http server in *Battery Emulator* which will be running and gets the energy consumption from *Data Processor* with one second interval. So, user can retrieve current battery, history or change the configuration anytime while the server is running.

Consumption History component is responsible to get the average of the parameter history based on configuration and store it in a file and user can get this average whenever they need.

Besides these components, it has Configuration file to configure parameters(*config.ini*), a tiny database to store history(`db.json`) and a file to store average of history called *summary_history.txt.*

These components will be covered in detail in this chapter. To explain the implementation of individual components some figures, activity diagram and corresponding screenshots of the task will be used. Implementation source code will be found in attached CD.

Installing Python modules

To run the application at first python modules have to be installed. Functionality of all the modules have been covered and these modules can be installed individually as stated in section 2.6 or can be installed with `requirement.txt` file. The modules that has been used to develop the application should be installed from Linux terminal before running it. `requirements.txt` file includes all the dependencies of the modules.

The command is to install all these modules from requirements.txt is given below:

```
$ pip install -r requirements.txt
```

`requiremnents.txt` file can be found in attached CD.

Example network scenario

Figure 4. 27 depicts an example scenario of a network. This scenario will be used to discuss above mentioned python scripts.

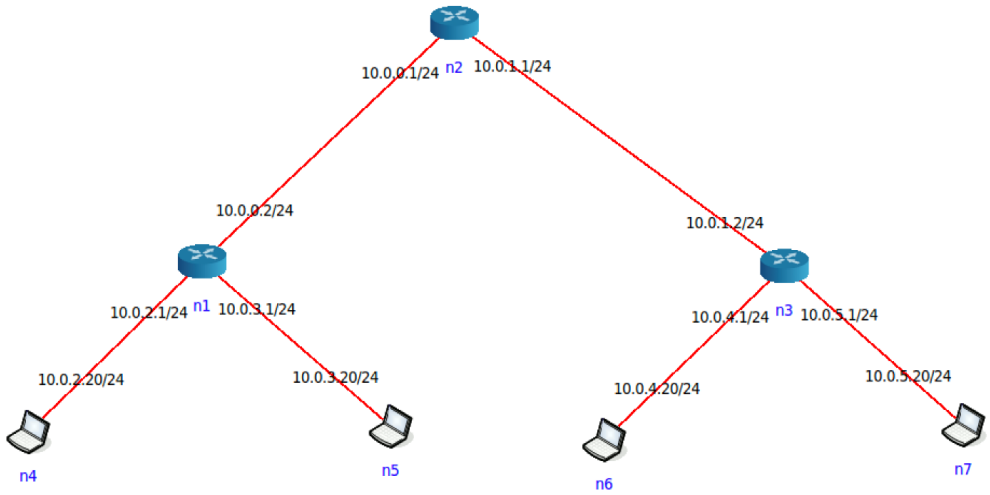


Figure 4. 27:Network scenario for testing python scripts

Testing from CORE terminal

To discuss about the application every script will be running from Desktop in CORE terminal. To do this, after running the session in CORE and shell terminal should be opened from any node where it is needed to run the python script. In Figure 4. 28 shows shell or terminal is opened from node n2 of network given in Figure 4. 27 and then move to Desktop directory.

As this VM is named on servercore2, so command will be:

```
$ cd /home/servercore2/Desktop
```

```
root@n2:/tmp/pycore.38681/n2.conf# cd /home/servercore2/Desktop
root@n2:/home/servercore2/Desktop#
```

Figure 4. 28: Running Python script from Desktop

4.4.1 Implementation of Data Aggregator

Data Aggregator is responsible to read consumption parameters from CORE node. In Figure 4. 29 it is shown that it reads RX/TX, CPU/RAM, Disk Read/Write parameters from node.

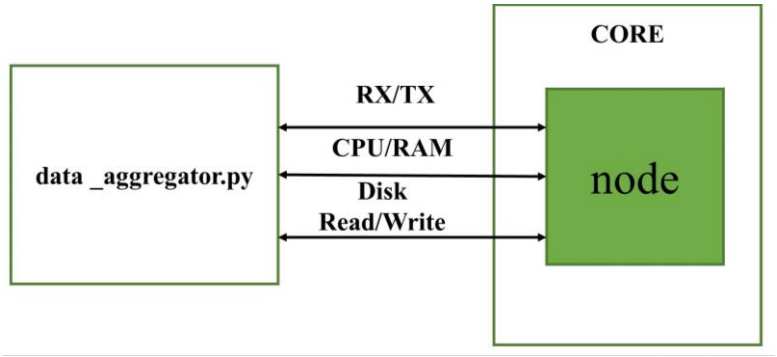


Figure 4. 29: Aggregating parameters from CORE node

To read the parameters from CORE nodes `data_aggregator.py` script has been developed and `aggregator ()` method is used to retrieve data from CORE nodes.

To develop this script some Linux command have been used as reference and these have been already mentioned in section 4.3. These commands have been first evaluated and then taken in consideration which is suitable to get the desired output.

To visualize the activities of Data Aggregator an activity diagram is given in Figure 4. 30

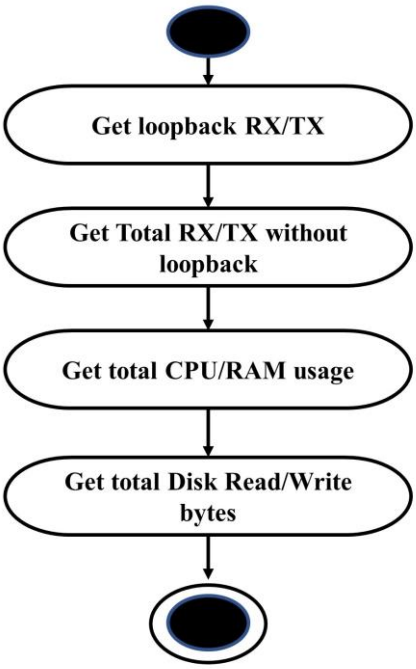


Figure 4. 30: Activity flow of data_aggregator.py

Get loopback RX/TX bytes

In given network scenario in Figure 4. 27 node n2 has three interfaces eth0, eth1 and loopback. To aggregate RX/TX bytes of interfaces from CORE node, python *psutil* module has been used. A screenshot can be shown as following Figure 4. 31. To run `data_aggregator.py`, terminal of node n2 should be opened and run from Desktop directory. In Figure 4. 31 it proves that same output as `ifconfig` can be retrieved by `data_aggregator.py`.

```
$ ifconfig lo && python data_aggregator.py
```

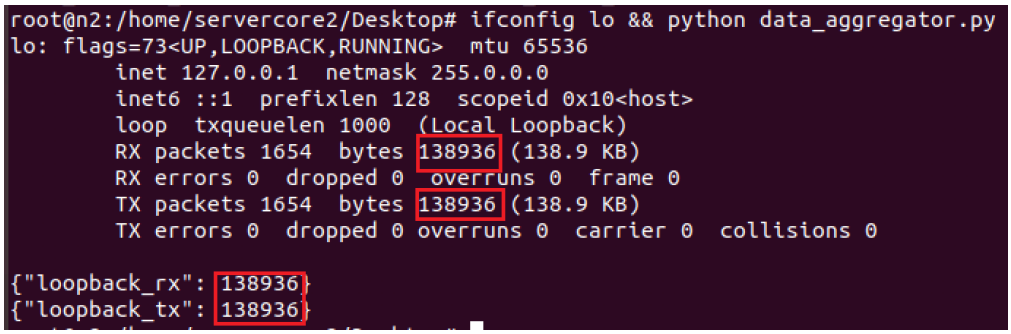


Figure 4. 31 Loopback byte RX/TX

Get total RX/TX without loopback

It is already mentioned that, total RX bytes of all ethernets will be calculated and then loopback RX bytes will be subtracted from this total RX. To get total RX/TX Python *psutil* module has been used. If loopback is greater than total RX then absolute value (positive) will be used. Similarly, total TX will be calculated, and loopback TX bytes will be subtracted.

Figure 4. 32 can prove the above explanation. In node n2(Figure 4. 27) total interfaces are two: eth0, eth1(without loopback). Then total RX=55592+54904=110496 and TX =48324+48330=96654 and result from `data_aggregator.py` is also same.

```
$ ifconfig && python data_aggregator.py
```

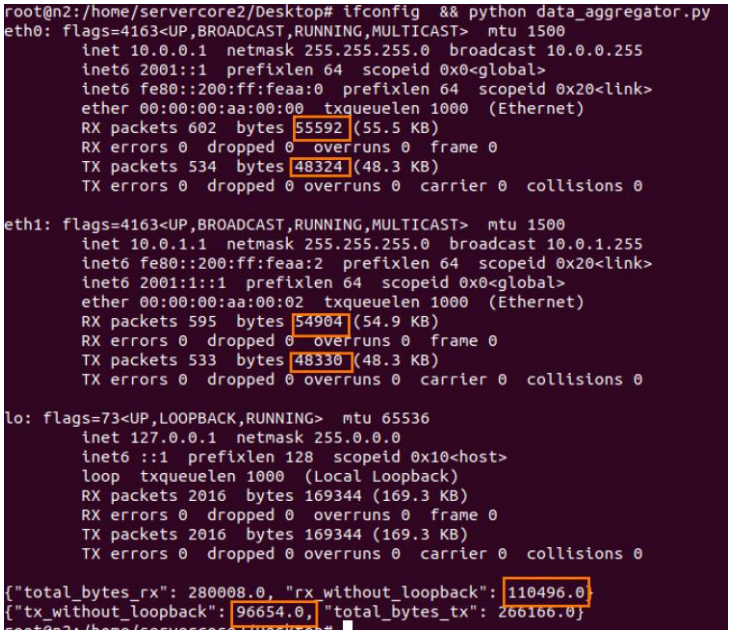


Figure 4. 32:Total RX/TX without loopback

Get total CPU RAM usage:
Python *os* module has been used to extract specific CPU column from command `ps aux`. And then the rows of this column have been added with python *pandas* module as shown in Figure 4. 33. It can be seen that at first CPU and RAM column have been extracted from `ps aux` command.

```
$ ps aux && python data_aggregator.py
```

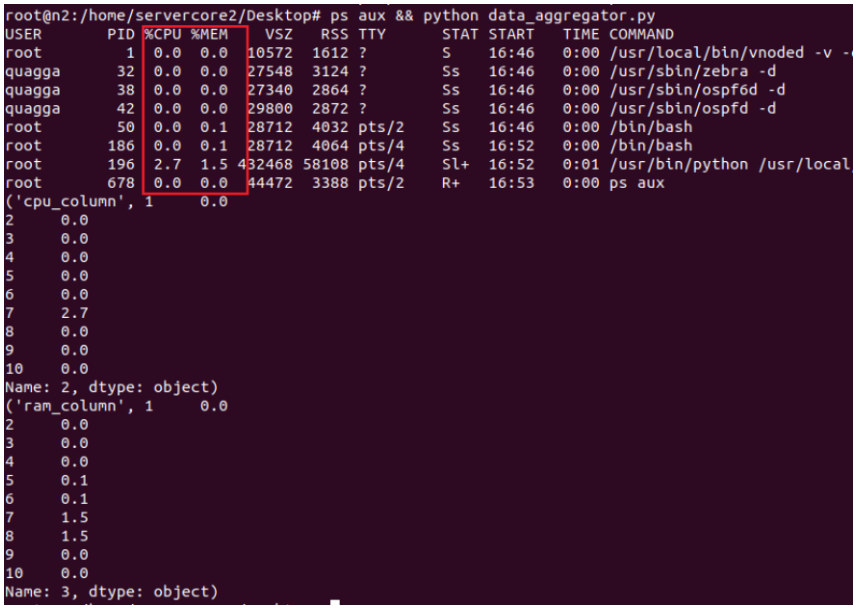


Figure 4. 33: Extracting CPU RAM usage from ps aux

Then total usage is calculated shown as Figure 4. 34.

```
$ python data_aggregator.py
```

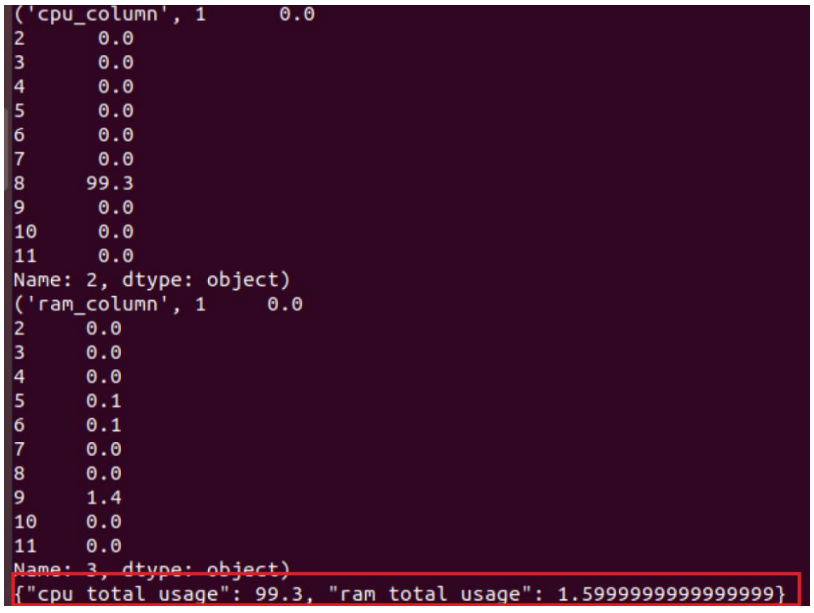


Figure 4. 34: Calculating total CPU/RAM usage

Get total Disk read/write bytes

To extract Disk read write bytes, shell input `pidstat -dl` is used by os module and then the same process has been followed as CPU RAM usage extraction to get the total of all read write bytes.

Below screenshots given in Figure 4. 35. states the total result of disk read write bytes and `pidstat -dl` command output of node n3.

It can be proved from Figure 4. 35 .

From `pidstat -dl` disk write per second in KB = 0.02

From `data_aggregator.py` disk write per second in KB = 0.02

From `pidstat -dl` disk read per second in KB = 0.01+1.64+1.18 =2.83

From `data_aggregator.py` disk read per second in KB = 2.83

```
$ data_aggreagtor.py && pidstat -dl
```

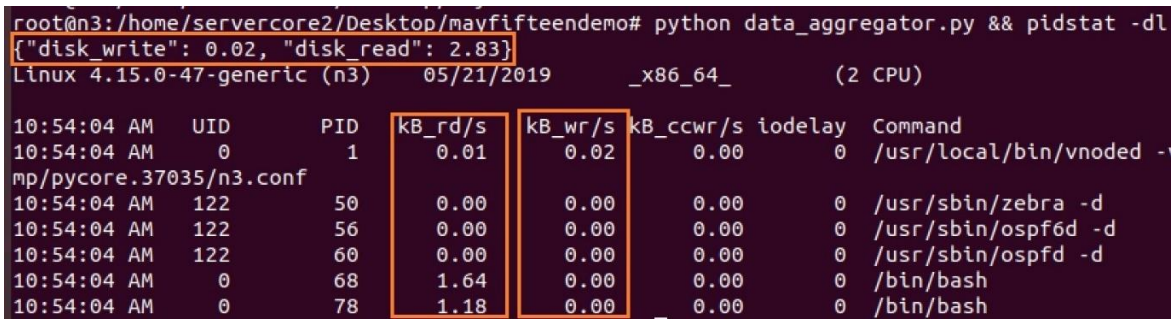


Figure 4. 35: Extract total DISK read write bytes

All parameters from data_aggregator.py

Total six parameters should be retrieved by Data Aggregator. Below an example screenshot is given of all parameters retrieved by `data_aggregator.py`.

So, `data_aggregator.py` have been used to aggregate total rx without loopback, total tx without loopback, total CPU usage, total RAM usage, total disk write bytes, total disk read bytes.

```
$ python data_aggregator.py
```

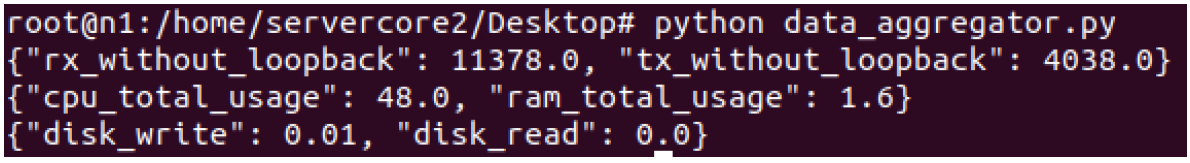


Figure 4. 36:Extract all parameters with data_aggregator.py

4.4.2 Implementation of Client Script Executor

The objective behind this implementation to run user defined shell script at different battery level. With this implementation three scripts can be defined at user defined battery level. This has to be defined in `config.ini` file. Therefore, this script is connected to only `config.ini` file as depicted in Figure 4. 37.

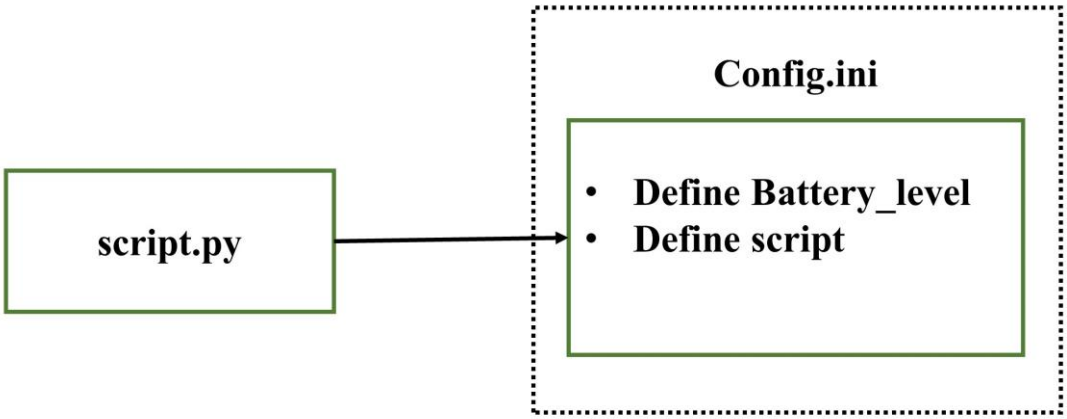


Figure 4. 37: Getting configuration to run script

A function called `battery_level(consumption)` is developed to do the task. It should be mentioned that this script is imported by `data_processor.py`. That means, actions of `data_processor`.py is dependent on `script.py` and this script will be running in the background when `data_processor.py` runs. That's why before discussing about *Data Processor* it will be necessary to discuss about `script.py`.

Defining battery level and shell scripts in `config.ini`

`Config.ini` file should include the battery levels and scripts name. And it should start from minimum battery level to maximum. As an example, if user want to run shell script at battery levels are 10,20,30 then it should start as given below `config.ini` starting from `battery_1 =10`. And the scripts that should be run at defined levels should be written in `script_1, script_2` and `script_3`. In this task maximum three scripts can be run. And this implementation can be extended for more scripts later if user needs.

`Config.ini`

```
[battery_lev]
battery_1 = 10
script_1 = script1.sh
battery_2 = 20
script_2 = script2.sh
battery_3 = 30
script_3 = script3.sh
```

To have an idea about the process of this function an activity diagram is given below in Figure 4. 38

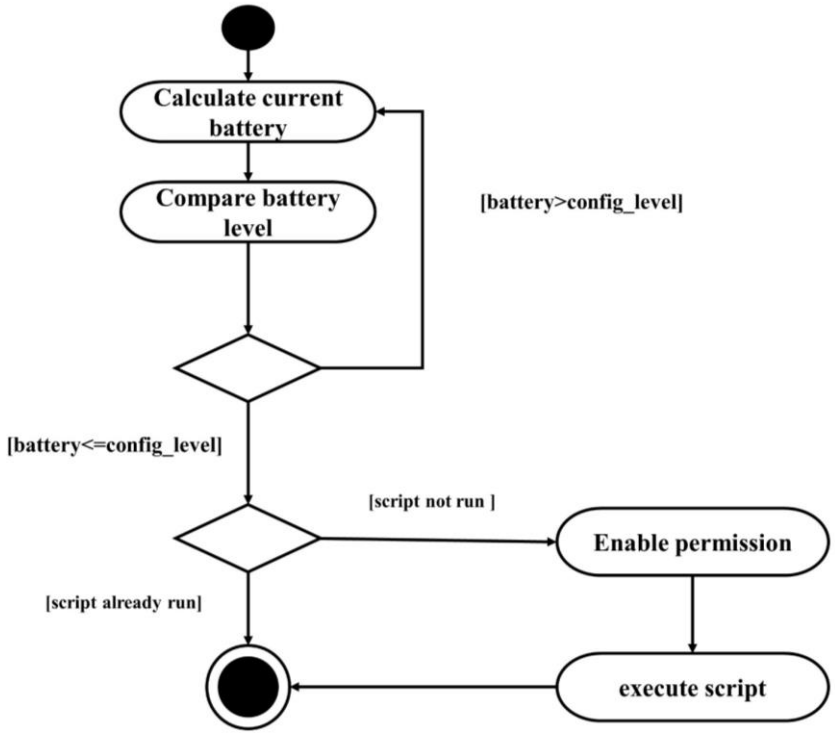


Figure 4. 38: Activity diagram of script.py

Calculate current battery and compare battery level
At first, it calculates the current battery. If current battery is less than defined battery level in `config.ini` then it checks if script has been run before, if it is not run then it takes the shell `.sh` script defined in `config.ini` and enable the permission. After enabling, it runs the shell script. Any shell `.sh` script can be run with this procedure.

For three battery level it has to be compared with three conditions. It starts checking from the maximum level, as an example it will start from 30% as defined in above `config.ini`, then 20 % and 10%. As in practical it is not possible to get exactly 10,20,30% battery level, that's why below logic of this implementation is established:

- For 30% battery level: if current battery is equal to 30% and greater than 20% then it will `run script_3`.
- For 20% battery level: if current battery is equal to 20% and greater than 10% then it will `run script_2`.
- For 10% battery level: if current battery is equal to 10% and less than 10% then it will `run script_1`.

If current battery matches with this any condition, then it enables permission of defined `.sh` script in `config.ini`.

To discuss this an example can be considered:

Let's assume, we have got current battery 29.2%. Then, `script_3` should be run as defined in `config.ini`. After running once, the `.sh` script will not run again even after it matches the battery level. But after consuming energy if battery level degrades less than 20%, as an example 19.9% then `script_2` will be run once. Then, if battery level goes less than 10% then `script_1` will run once. That's how the logic works.

Some screenshots can depict the scenario more. These screenshots are taken from node n2 of given scenario in Figure 4. 27.

In Figure 4. 39 it is shown that defined `script3.sh` is running at battery level 29.9% and running once only, it is running afterwards even it matches the battery level condition.

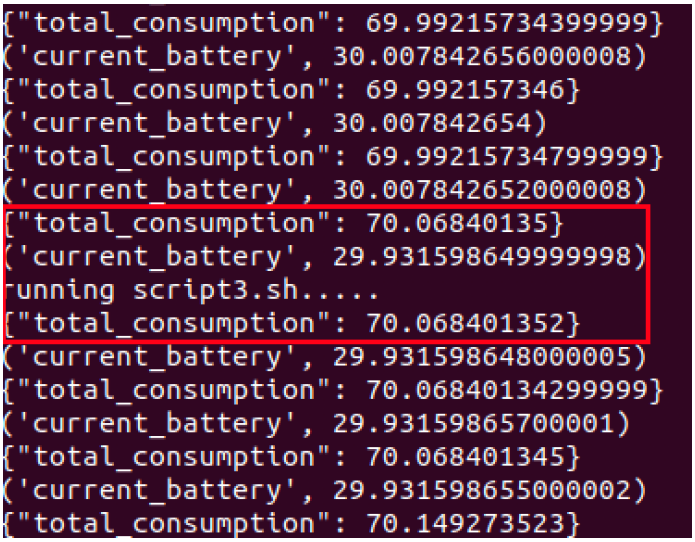


Figure 4. 39:Running script3.sh at battery level 30%

And in Figure 4. 40 depicts the scenario where `script2.sh` runs at battery level 20% or greater than 10% and it also runs once.

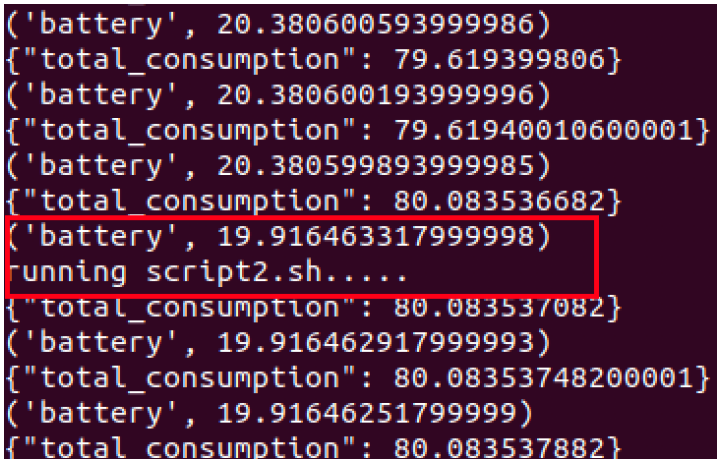


Figure 4. 40: Running script at defined battery level 20%

Figure 4. 41 depicts running script for battery level 10% as defined in `config.ini` It can be seen that `script1.sh` running after draining battery level to 9.3467% and it runs once only, after that it skips the procedure.

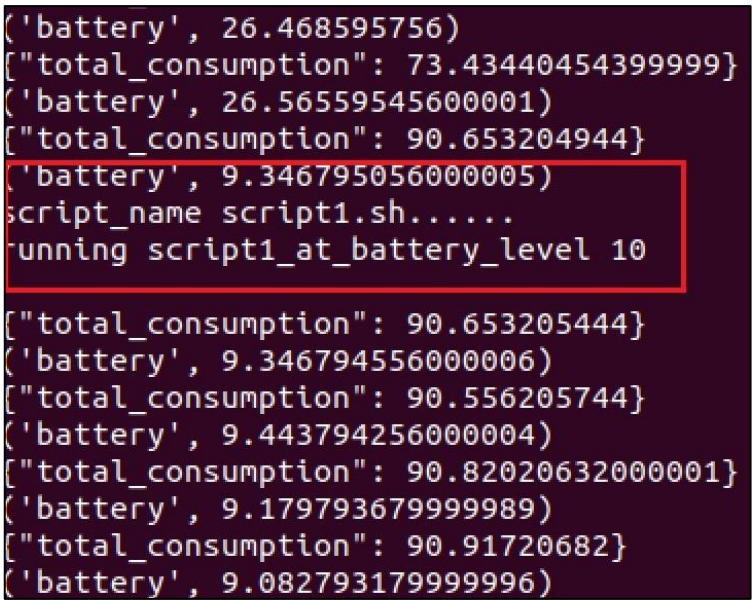


Figure 4. 41:Running script at defined battery level 10%

4.4.3 Implementation of Data Processor

Data Processor is responsible to process data after getting aggregated parameters from *Data Aggregator*. To process data, it uses the configuration from `config.ini` file. It is also connected to another component *called Client Script Executor* to compare battery level after getting the consumption. That means, it imports two python scripts called `data_aggregator.py` and `script.py`. And it also stores individual history of the parameters along with total consumption in a *TinyDB* database.

So, it has four components which it is connected as depicted in Figure 4. 42.

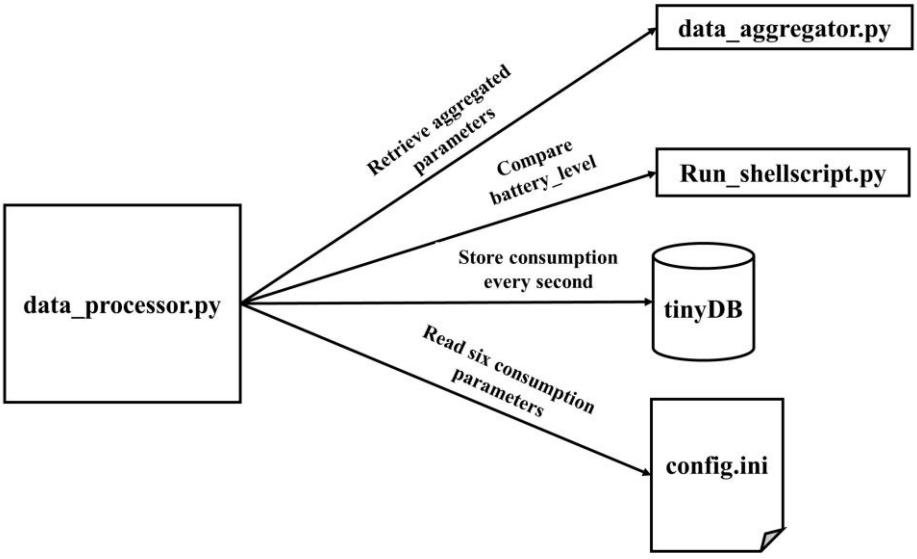


Figure 4. 42: Four main components of data_processor.py

A function called `process_parameter()` is developed to do all the tasks `in data_processor.py`. At first it reads six configuration parameters and then it processes the aggregated data getting from `data_aggregator.py`. If the history is activated to store parameter usage and total consumption data, then it will store the history otherwise it will ignore this process. After that it calls `battery_level(consumption)` function from `script.py` which therefore runs its own logic as given in 4.4.2. It calculates current battery percentage subtracting(100-consumption) % from present consumption and then compares it with given battery level in `config.ini` and run the `.sh` scripts if condition meets. This process continues with the interval 1 sec until user stops it. This developed application is also able to calculate consumption in mA, this has been discussed in later portion of this section and also a comparison is given in section 5.3.

These steps can be visualized with below activity diagram in Figure 4. 43.

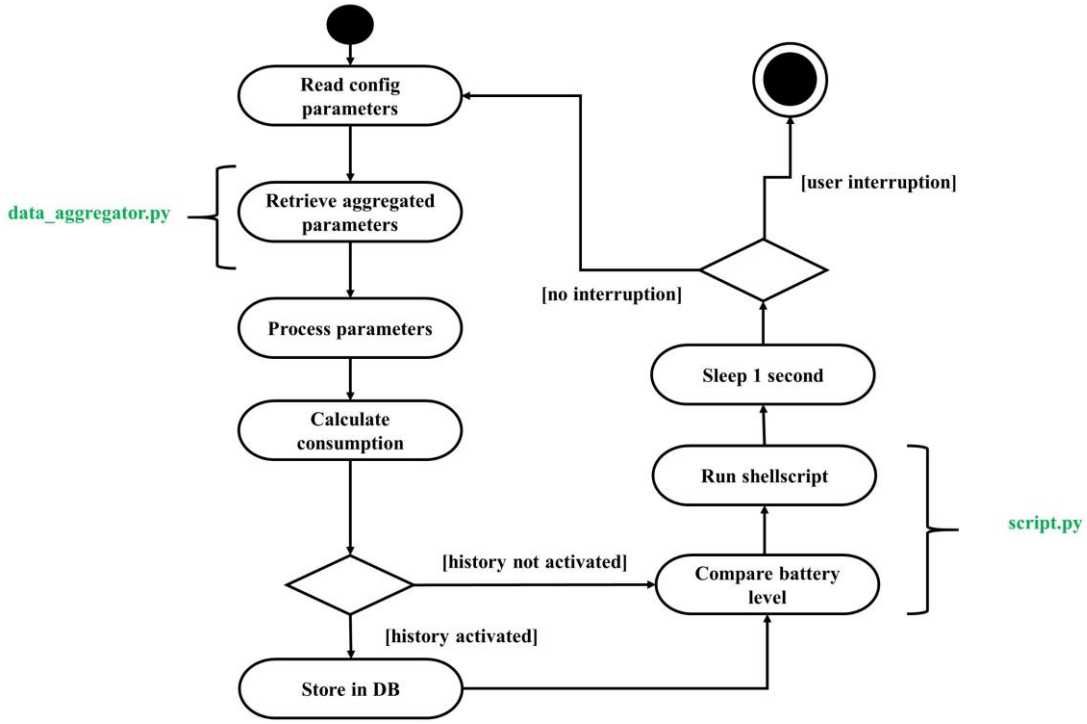


Figure 4. 43:Flow of processing parameter

Read configuration

`data_processor.py` reads six parameter specification and history activation section from `config.ini`. To read this file ConfigParser module of python has been used. Below an example of `config.ini` file is given. There are four sections: [rx_tx], [cpu_ram], [disk_rd_wrt],[battery_lev],[activate_history]. In every section parameter value are defined in percentage (Discussion with mA configuration can be found in last part of this section). As an example,

In section [rx_tx],rx =0.00001.That means 1000 bytes consume 0.00001% battery or energy.

In [cpu_ram] section cpu=0.000001, that means 1% CPU consumes 0.000001 % battery or energy.

In section [disk_rd_wrt] disk_rd is given as 0.00001.That means 1000 bytes consume 0.00001% battery or energy.

In section [activate_history] activate=1 means history is activated and all other number will be considered as history not activated.

Config.ini

```
#1000 bytes consume X% battery
[rx_tx]
rx = 0.00001
tx = 0.00001
#1% cpu/ram consumes X% battery
[cpu_ram]
cpu = 0.000001
ram = 0.000001
#1000 bytes read write consume X% battery
[disk_rd_wrt]
disk_rd = 0.00001
disk_wrt = 0.00001
#activate history
[activate_history]
activate = 1
```

Calculate consumption

After getting aggregated data `process_parameter()` processes the data based on `config.ini`. To understand this an example can be given:

From `config.ini`: rx =0.00001
From `aggregator()` : total_rx=2000
And, it is already mentioned that the assumption is based on per 1000 bytes. That means, 1000 bytes consume 0.00001% battery.
Then consumption by rx= (0.00001×2000) ÷1000=0.00002%.

Similarly, all other parameters have been calculated and the total consumption is taken as the addition of all six parameters. It should be also mentioned that, `pidstat -dl` shows disk read write usage in Kilobyte. So, it has been converted to bytes by multiplying with 1000.

When CPU usage is less than 4% it is assumed as idle mode. Therefore, if CPU usage is less than 4% it shows idle mode. In Figure 4. 44 it is shown the individual consumption in percentage in idle mode for all parameters.

```
root@n2:/home/servercore2/Desktop# python data_processor.py
Idle
{"consumption_byte_tx": 0.00090028, "consumption_byte_rx": 0.0010831000000000003}
{"consumption_by_ram": 1.6000000000000003e-05, "consumption_by_cpu": 0.0}
{"consumption_by_disk_write": 1e-10, "consumption_by_disk read": 1e-10}
```

Figure 4. 44:Consumption by parameters when Idle

Below Figure 4. 45 is a screenshot of all parameter consumption after processing aggregated data, and also total consumption taken in active mode (CPU usage more than 4%). It is taken from node n2 in given network scenario in Figure 4. 27.

```
$ python data_processor.py
```

```
root@n2:/home/servercore2/Desktop# python data_processor.py
active
{"consumption_byte_tx": 0.6641491, "consumption_byte_rx": 0.6643323}
{"consumption_by_ram": 1.8e-05, "consumption_by_cpu": 0.0006000000000000001}
{"consumption_by_disk_write": 9.000000000000001e-10, "consumption_by_disk read": 2e-10}
{"total_consumption": 1.3291005}
```

Figure 4. 45:Getting total consumption after processing

Store usage and total consumption in TinyDB

If history is activated in `config.ini` then usage of all parameters and total consumption will be stored in `db.json` with 1 second interval. In Figure 4. 46 it is shown that the parameters usage and total consumption have been stored in every second. It can be done by using built in function `db.insert()` of *TinyDB*. Data shown in this Figure 4. 46 taken with scenario mentioned in Figure 4. 27.

To clarify more about the stored parameters it is discussed a bit below:

- total_rx: total RX bytes of all ethernets except loopback of that particular time.
- total_tx: total TX bytes of all ethernets except loopback of that particular time.
- ram_usage: ram usage in % of that particular time.
- cpu_usage: cpu usage in % of that particular time.
- disk_read_usage: total read bytes of that particular time.
- disk_write_usage: total write bytes of that particular time.
- consumption:total consumption of that particular time after processing the parameter based on `config.ini`.
- time: showing present time of retrieving parameters.

Marked portion of following figure shows the particular time block of extracting mentioned parameters. As `data_processor.py` process data every second the time laps will be 1 second.

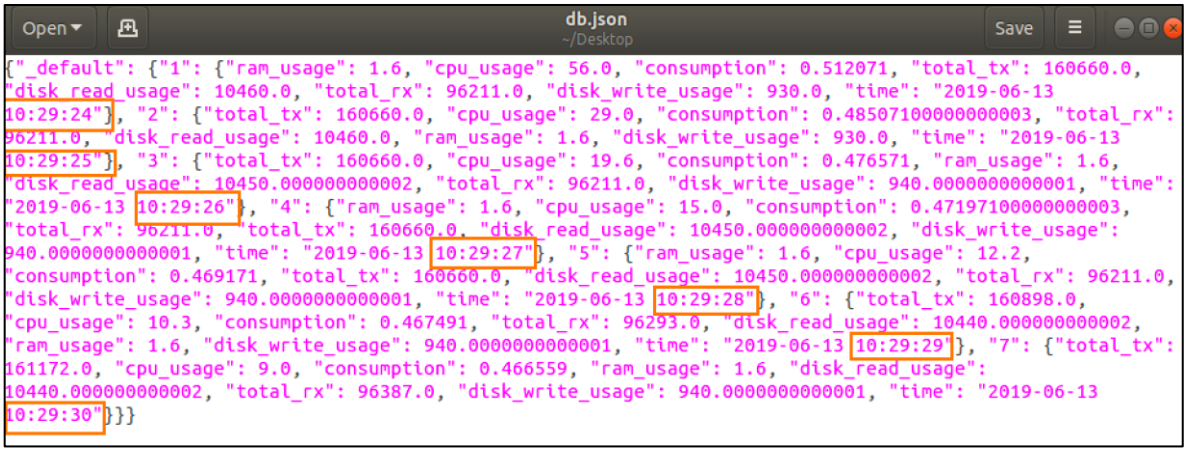


Figure 4. 46:Storing parameter usage and total consumption in TinyDB

Compare battery level

It is already mentioned in section 4.4.2 how battery level is compared and shell script is run by `script.py`. `data_processor.py` imports this python script and then `script.py` logic runs in the background.

Calculation with milliampere(mA)

This developed application is also able to calculate consumption in mA. To do this the specification should be added in `config.ini` file with above mentioned parameters. [battery_specification] includes the voltage and

current capacity of the system or device. Here, value is given voltage=5.7 volt and capacity 1500 mAh. Other parameters are also specified in mA. As an example, rx_ma=0.001 means 1000 bytes consume 0.001 mA, cpu_ma=0.01 means 1% CPU consumes 0.01 mA. The development code has been added in attached CD.

Config.ini

```
[battery_specification]
voltage=5.7
capacity=1500
#1000 bytes consume X% battery
[rx_tx]
rx = 0.00001
tx = 0.00001
rx_ma=0.001
tx_ma=0.001
#1% cpu/ram consumes X% battery
[cpu_ram]
cpu = 0.000001
ram = 0.000001
cpu_ma=0.01
ram_ma=0.01
#1000 bytes read write consume X% battery
[disk_rd_wrt]
disk_rd = 0.00001
disk_wrt = 0.00001
disk_rd_ma = 0.001
disk_wrt_ma = 0.001
#activate history
[activate_history]
activate = 1
```

After getting all the consumption in mA total consumption has been calculated. Below Figure 4. 47 shows an example of output in mA of all parameter consumption and then the total consumption calculated from these parameters. It is taken from node n2 of network scenario given in Figure 4. 27.

```
$python data_processor.py
```

```
root@n2:/home/servercore2/Desktop# python data_processor.py
('battery_voltage: ', 5.7)
('total capacity in mA', 1500.0)
('consumption by rx in mA', 0.0181641600000000002)
('consumption by tx in mA', 0.01030232)
('consumption by cpu in mA', 61.0)
('consumption by ram in mA', 0.72000000000000001)
('consumption by disk read in mA', 0.0)
('consumption by disk write in mA', 0.0)
('total consumption in mA', 61.74846648)
```

Figure 4. 47:Calculation in mA

4.4.4 Implementation of Consumption History

Consumption History component is responsible to get the average of history stored in `db.json` and after calculating the average it stores a summary of history in a `summary_history.txt` file. That means it is connected to three components as depicted in Figure 4. 48.

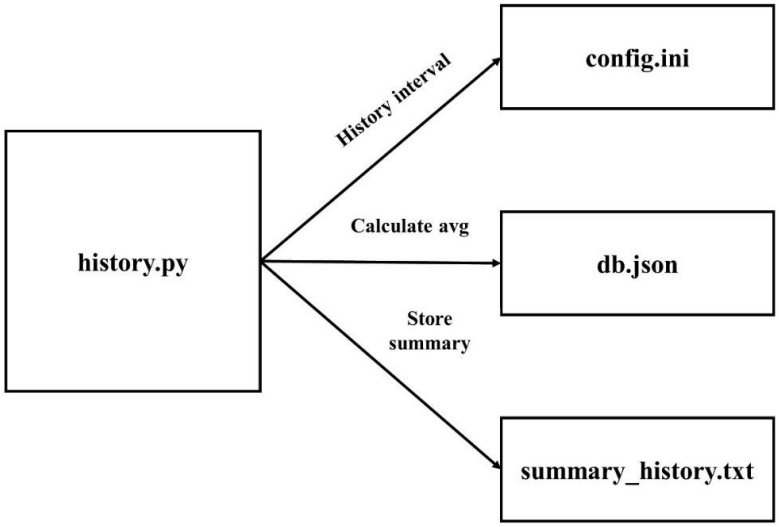


Figure 4. 48:Components connected to history.py

It is already stated in section 4.4.3 that `data_processor.py` stores parameters in a database if history is activated in `config.ini`.In `history.py` a method `db_read()` is developed where values are read and summary of a calculation based on configuration is stored in a file to allow the http server developed in `battery_emulator.py` to read it with http request. Managing history includes below steps:

- Read history from database based on configuration interval.
- Calculate average and get summary of history
- Store in a file after calculation

Below activity diagram in Figure 4. 49 can depict the steps of this implementation:

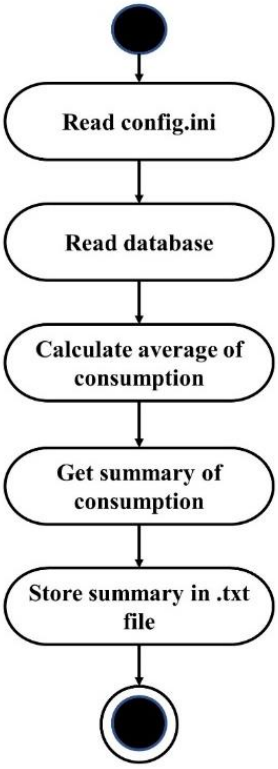


Figure 4. 49: Actions taken by history.py

Configuration for history

In `config.ini` file history configuration should be added, and time has to be defined to extract the history from database. For requesting history with given time when server will be running it is needed to define history configuration and other given parameters.

`config.ini`

```
[activate_history]

history=10
```

Read database and calculate average

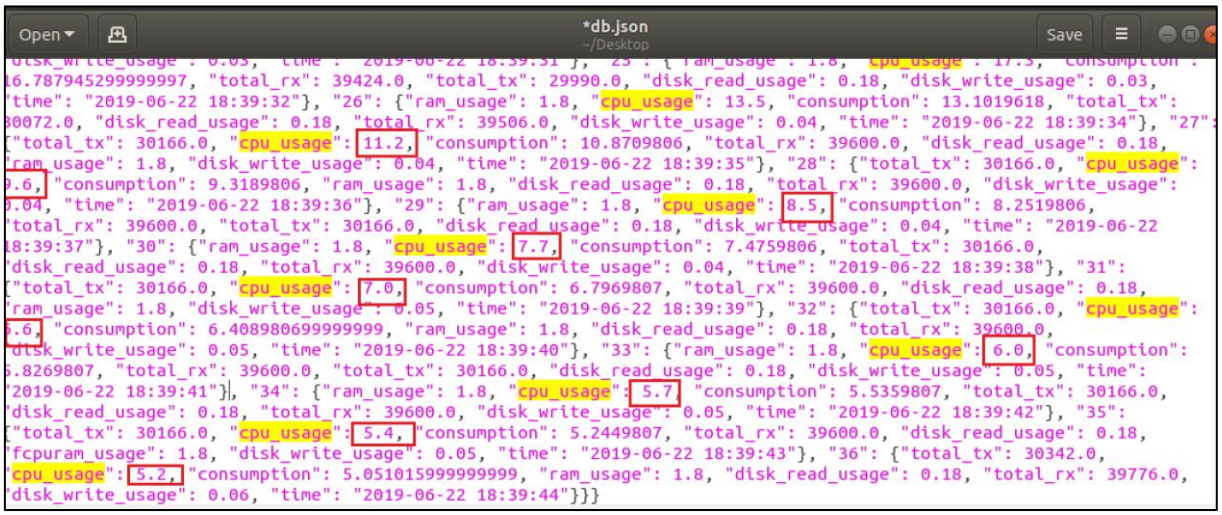
Stored parameters in `db.json` are read by `db_read()`.In Figure 4. 46 it is shown that history data is stored every second. As an example, if in `config.ini` section `[activate_history]` contains `history` section which contains user definable interval, for example if user sends http request to get history of last 10 seconds that means it will calculate the average of last 610seconds of all parameters stored in *TinyDB*..There are in total seven parameters which will be stored in *TinyDB*.It stores data in json format. It will be change as user sends request with the http request, it is described in later section 4.6.8.

Comparing calculated data of history with database

A small example is given below with screenshots to understand how it is working. If in `config.ini` history configuration is given, then the result is calculated as below. Assuming, `history`=10 given, then last 10 values will be taken, and average will be calculated.

For understanding the procedure only CPU parameter is taken as an example as shown in Figure 4. 50: .

db.json



16.787945299999997, "total_rx": 39424.0, "total_tx": 29990.0, "disk_read_usage": 0.18, "disk_write_usage": 0.03,
"time": "2019-06-22 18:39:32"}, "26": {"ram_usage": 1.8, "cpu_usage": 13.5, "consumption": 13.1019618, "total_tx":
30072.0, "disk_read_usage": 0.18, "total_rx": 39506.0, "disk_write_usage": 0.04, "time": "2019-06-22 18:39:34"}, "27":
{"total_tx": 30166.0, "cpu_usage": 11.2, "consumption": 10.8709806, "total_rx": 39600.0, "disk_read_usage": 0.18,
"ram_usage": 1.8, "disk_write_usage": 0.04, "time": "2019-06-22 18:39:35"}, "28": {"total_tx": 30166.0, "cpu_usage":
9.6, "consumption": 9.3189806, "ram_usage": 1.8, "disk_read_usage": 0.18, "total_rx": 39600.0, "disk_write_usage":
0.04, "time": "2019-06-22 18:39:36"}, "29": {"ram_usage": 1.8, "cpu_usage": 8.5, "consumption": 8.2519806,
"total_rx": 39600.0, "total_tx": 30166.0, "disk_read_usage": 0.18, "disk_write_usage": 0.04, "time": "2019-06-22
18:39:37"}, "30": {"ram_usage": 1.8, "cpu_usage": 7.7, "consumption": 7.4759806, "total_tx": 30166.0,
"disk_read_usage": 0.18, "total_rx": 39600.0, "disk_write_usage": 0.04, "time": "2019-06-22 18:39:38"}, "31":
{"total_tx": 30166.0, "cpu_usage": 7.0, "consumption": 6.7969807, "total_rx": 39600.0, "disk_read_usage": 0.18,
"ram_usage": 1.8, "disk_write_usage": 0.05, "time": "2019-06-22 18:39:39"}, "32": {"total_tx": 30166.0, "cpu_usage":
6.6, "consumption": 6.408980699999999, "ram_usage": 1.8, "disk_read_usage": 0.18, "total_rx": 39600.0,
"disk_write_usage": 0.05, "time": "2019-06-22 18:39:40"}, "33": {"ram_usage": 1.8, "cpu_usage": 6.0, "consumption":
6.8269807, "total_rx": 39600.0, "total_tx": 30166.0, "disk_read_usage": 0.18, "disk_write_usage": 0.05, "time":
"2019-06-22 18:39:41"}, "34": {"ram_usage": 1.8, "cpu_usage": 5.7, "consumption": 5.5359807, "total_tx": 30166.0,
"disk_read_usage": 0.18, "total_rx": 39600.0, "disk_write_usage": 0.05, "time": "2019-06-22 18:39:42"}, "35":
{"total_tx": 30166.0, "cpu_usage": 5.4, "consumption": 5.2449807, "total_rx": 39600.0, "disk_read_usage": 0.18,
"fcpuram_usage": 1.8, "disk_write_usage": 0.05, "time": "2019-06-22 18:39:43"}, "36": {"total_tx": 30342.0,
"cpu_usage": 5.2, "consumption": 5.051015999999999, "ram_usage": 1.8, "disk_read_usage": 0.18, "total_rx": 39776.0,
"disk_write_usage": 0.06, "time": "2019-06-22 18:39:44"}}}

Figure 4. 50: reading CPU usage by db_read() of history.py

From above database in Figure 4. 50 values are taken total 10 as given in `config.ini`. Then average is calculated taking last 10 usage of CPU as given in following Figure 4. 51.Below screenshot is taken from the terminal of node n2 in given network scenario in Figure 4. 27.

```
$ python history.py
```

```
root@n2:/home/servercore2/Desktop# python history.py
('cpu_values', array([11.2,  9.6,  8.5,  7.7,  7. ,  6.6,  6. ,  5.7,  5.4,  5.2]))
```

Figure 4. 51: Calculate CPU history as per config

Taking these 10 usage history of CPU in last 10 seconds is 7.29 as shown in Figure 4. 52. All other parameters are calculated like above given procedure of CPU history. Here, average of all parameters means average of last 10 seconds. As an example, avg_rx=39617.6 means node n2 received average of 39617.6 RX bytes every second, avg_tx=30183.6 means node n2 transmitted average of 30186.6 TX bytes every second, avg_cpu means at this node every second CPU usage in average is 7.29%. Similarly, other parameters are calculated. And, at last the time lap is shown of extracting these parameters.

```
root@n2:/home/servercore2/Desktop# python history.py
('avg_rx', 39617.6)
('avg_tx', 30183.6)
('avg_cpu', 7.290000000000001)
('avg_ram', 1.8000000000000003)
('avg_disk_read', 0.17999999999999997)
('avg_disk_write', 0.047)
('avg_consumption', 0.047)
('start', u'2019-06-22 18:39:35', 'end', array([u'2019-06-22 18:39:44']
```

Figure 4. 52: Average of history of all parameters

After calculation it is stored in a *.txt* file as shown in Figure 4. 53. It allows the user to retrieve summary of history with http request. It is shown in later section 4.6.8.

Open summary_history.txt ~/Desktop

```
('avg_rx', 39617.6)
('avg_tx', 30183.6)
('avg_cpu', 7.290000000000001)
('avg_ram', 1.8000000000000003)
('avg_disk_read', 0.17999999999999997)
('avg_disk_write', 0.047)
('avg_consumption', 0.047)
('start', u'2019-06-22 18:39:35', 'end', array([u'2019-06-22 18:39:44']
```

Figure 4. 53: Storing summary after calculating average of parameters

4.4.5 Implementation of Battery Emulator

This component consists of a http server. With this server user can get the necessary output with http request. This script imports `data_processor.py and history.py`. It also uses `summary_history.txt` to read summary of history saved by `history.py` and `config.ini` file to change the configuration by http request.

`battery_emulator.py` is connected or dependent on four components as shown in Figure 4. 54.

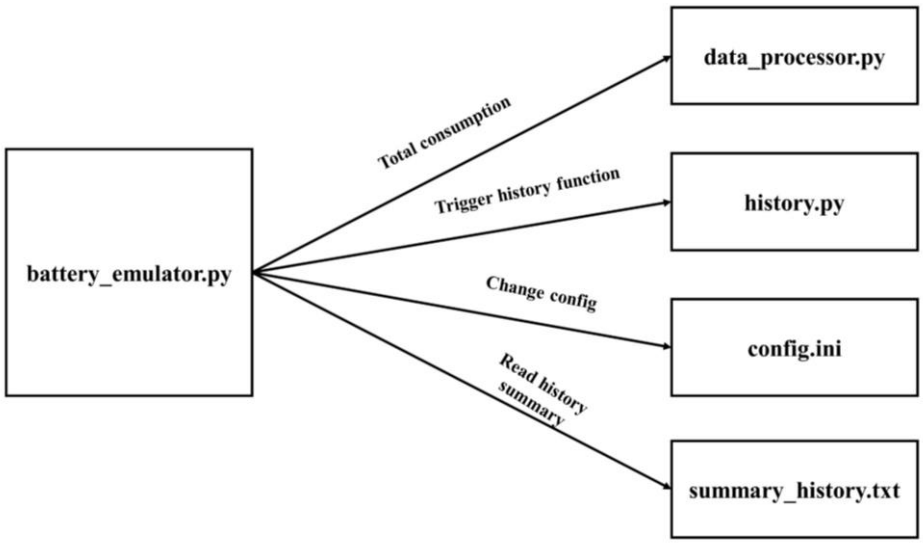


Figure 4. 54: Dependency on other components of batter_emulator.py

This http server is developed with Flask module of python. This server can facilitate user with below response:

- Current battery with highest CPU and RAM consuming PID.
- Changing configuration parameters while server is running.
- Get summary of consumption history.

To develop this http server three functions have been used with Flask module and using 5000 http port. It consists of three function. Such as:

`emulator()` : can retrieve current battery, total consumption and highest consuming CPU/RAM usage PID.

`Change_all_parameters()` : can change parameters causing consumption of `config.ini` while server is running with http request.

`get_history()` : takes request from user to know history interval and then response with summary of history.

Get current battery

After sending the request from user current battery can be retrieved with the developed function in this http server called `emulator()`. After sending the request server can verify the request. If it is valid then current battery is calculated and highest CPU/RAM usage by PID can be retrieved. If it is not invalid request it sends a bad response to the http client.

As this http server imports `data_processor.py`, that means consumption can be retrieved every second by `process_parameter()` of `data_processor.py`. And current battery is calculated subtracting this consumption from 100. And along with current battery highest CPU/RAM usage PID can be retrieved. GET request should be sent from user to get these mentioned responses.

Below activity diagram in Figure 4. 55 can depict the scenario of flow of handling GET request by the http server.

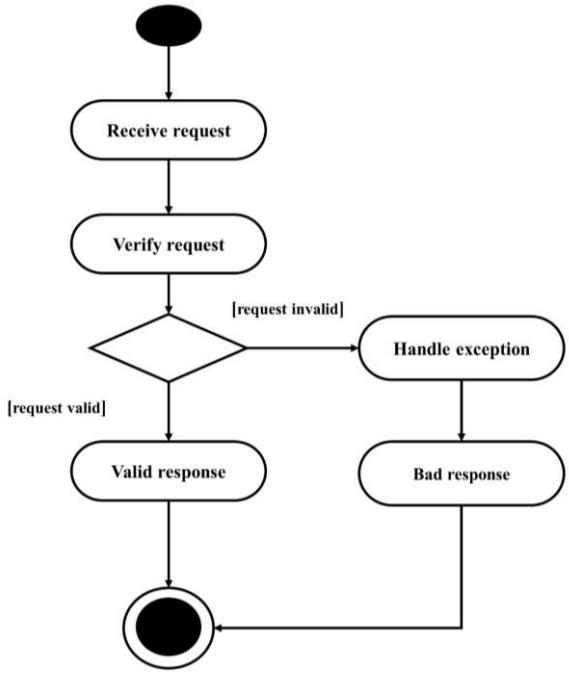


Figure 4. 55: Activity diagram of GET request

Therefore, as an example request is sent from localhost node n2 of network scenario given in Figure 4. 27 with `curl localhost:5000/emulate` user gets the response with current battery, total consumption and highest consuming CPU RAM usage PID.

It can be depicted in Figure 4. 56. It can show PID, USER, % usage consecutively as shown in Figure 4. 56.

```
root@n2:/tmp/pycore.36775/n2.conf# curl localhost:5000/emulate
{'total consumption': 1.375603, 'current_battery': 98.624397}
{'max_cpu_usage_pid': [['PID', 'USER', '%CPU'], ['12426', 'root', '72.5'],
 ['12427', 'root', '60.8'], ['10013', 'root', '14.2']]}
{'max_ram_usage_pid': [['PID', 'USER', '%MEM'], ['12427', 'root', '6.8'],
['12426', 'root', '4.6'], ['10013', 'root', '1.7']]}
```

Figure 4. 56: Response of GET request to the user

Change configuration parameters

It is possible to change the configuration parameters while server is running through POST request. After getting the request from user server can verify the request and change the parameters in `config.ini` file.

To change the parameters from localhost below http request can be sent:

```
curl -d'{"rx":0.0001,"tx":0.0002,"cpu":0.00001,"ram":0.00001,"read":0.005,
"write":0.03}' -H "Content-Type: application/json" -X POST http://lo-
calhost:5000/change_all.
```

After getting the POST request it reads `config.ini` file with `config.read()` method of ConfigParser module and then specified parameters have been set to new values with `config.set()` method of `ConfigParser` module given by user.

Below activity diagram in Figure 4. 57 shows the steps of POST request:

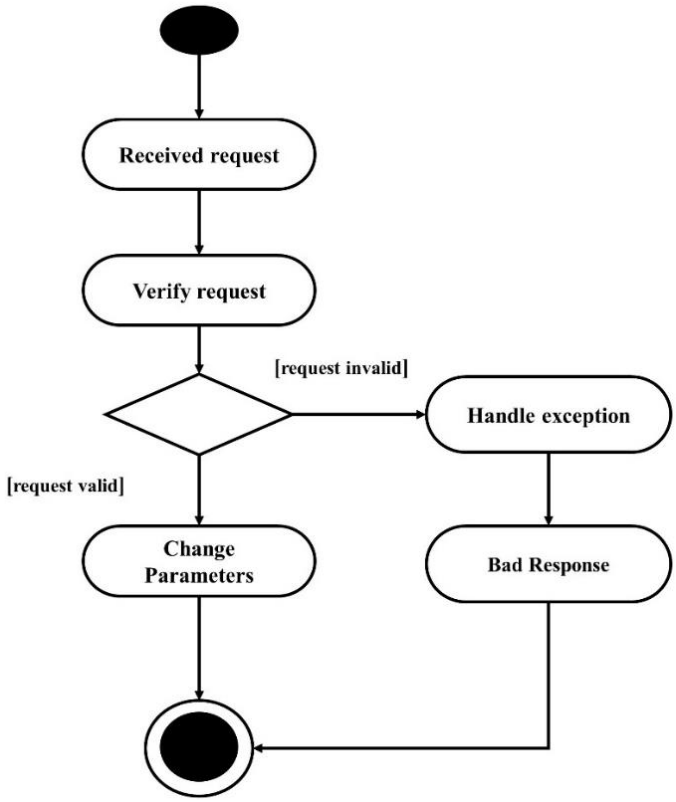


Figure 4. 57: Activity diagram of changing parameters with POST request

Get Consumption history

User can retrieve history by sending an interval of getting history average. As an example, if the request is `http://172.16.0.2:5000/history/60`, here 60 means to get the history of last 60 seconds. Then it changes the history portion of the section [activate history] in `cofig.ini`. And then summary of history can be retrieved from `summary_history.txt` file which has been calculated by `history.py`. it stores the value every second in `summary_history.txt` file, so that it can be read by the user whenever needed. Every time user changes interval of retrieving history `history.py` processes the history average with new interval.

Below activity diagram shown in Figure 4. 58 can give an idea of this part:

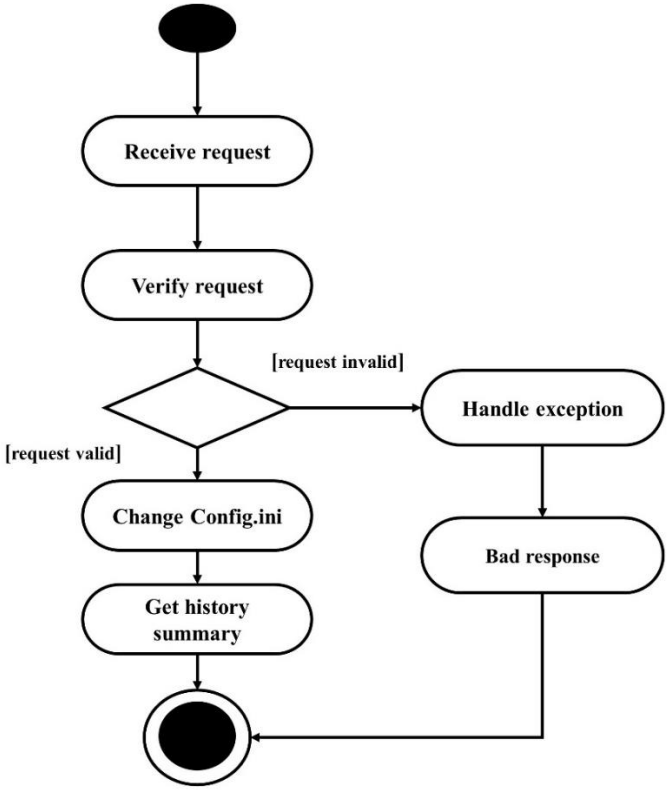


Figure 4. 58: Activity diagram of getting history with GET request

As an example, below screenshot in Figure 4. 59 from node n2 (Figure 4. 27). After sending the request with 60 seconds interval it is showing average of parameters calculated from last 60 seconds. start time shows when calculation is started, and end time shows when it ends.

Duration between this two is 60 seconds as shown in Figure 4. 59.

```
root@n2:/tmp/pycore.36775/n2.conf# curl localhost:5000/history/60
('avg_rx', 350360.76666666666)
('avg_tx', 841201.5666666667)
('avg_cpu', 4.233333333333333)
('avg_ram', 2.103333333333331)
('avg_disk_read', 2158.6666666666665)
('avg_disk_write', 4202.833333333333)
('avg_consumption', 1.639769)
('start', u'2019-06-13 16:27:22', 'end', array([u'2019-06-13 16:28:23'], dtype=object))
```

Figure 4. 59: Get summary of history with interval 60 seconds

4.5 Summary of Implementation

In previous sections detail explanation of the application has been given

There are five scripts which have been developed in this development task. Apart from these, a configuration file named `config.ini`,a tiny database `db.json` and another file to store summary of history called `summary_history.txt` have been used. After running `battery_emulator.py(http server)` all functionalities can be provided to user upon http request. After running this http server `data_processor.py` and `history.py` will be running and continuously process the parameters and gets consumption and stores history. In section 4.4 the functions used in individual scripts have been already mentioned. It can be stated again:

1. `aggregator()`: `data_aggregator.py`
2. `battery_level(consumption)`:`run_shellscript.py`
3. `process_parameter()`:`data_processor.py`
4. `db_reead()`:`history.py`

And `battery_emulator.py(http_server)` have three functions.

1. `current_battery()`: to get current battery.
2. `change_all_parameter()`: to change configuration parameters.
3. `get_history(id)`: to get the summary of history.
4. `app.run()`: flask built in function to run the server.

After running `battery_emualtor.py` the tasks shown in Figure 4. 60 always takes place and keep doing till there is any interruption by the user. After running it, `data_processor.py` calls `aggregator()` and this returns aggregated parameters. These parameters are processed by `data_processor.py` and then battery level is compared calling the function `battery_level()`.If battery level condition meets, script is run.`history.py` calculates average when user sends http request to retrieve summary of history. This task take place every second if there is no interruption.

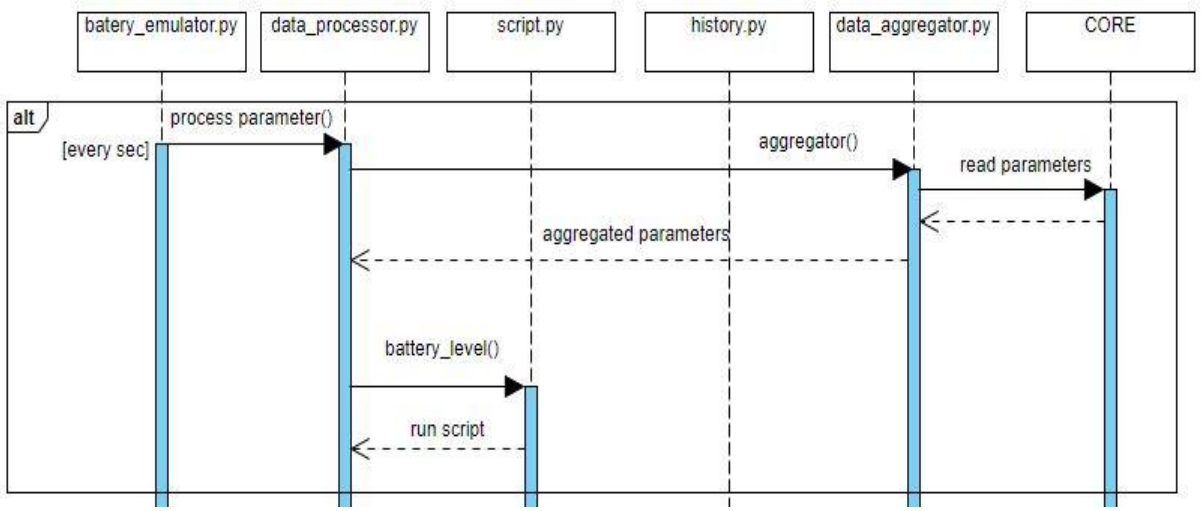


Figure 4. 60: Sequence of task after running battery_emulator.py

4.6 Deployment of Application in CORE as a Service

Developed application can be deployed in CORE as a service. In this section detail implementation of this http server and how it is deployed in CORE as a service, also how to initiate the `config.ini` and scripts from service tab will be shown. This section will replicate steps to run server from service tab. To deploy this application in a CORE `battery_emulator.py` is imported by `energy_simu.py`. Here, `energy_simu.py` is nothing but only to run http server without having much code as shown in Figure 4. 61 .

```
#!/usr/bin/python
from battery_emulator import*

if __name__ == "__main__":
    app.run()
```

Figure 4. 61: Running battery_emulator.py with energy_simu

4.6.1 Enable Python Scripts to Run from any Directory

Configuration file `config.ini` needs to be allowed to edit while server will be running. For this purpose, developed script should be allowed to run from any directory with `config.ini` file. It can be done placing all python scripts in `/usr/local/sbin`.

It facilitates to run script `enery_simu` from anywhere where the `config.ini` file is, without having all the scripts in same place. As an example, if `config.ini` file is in Desktop folder then `energy_simu` can be run from Desktop. Following Figure 4. 62 can be taken as an example, it is shown that in Desktop there is only `config.ini` and `energy_simu` can be run from Desktop without having other dependency.

```
servercore2@server:~/Desktop$ ls
config.ini  core
servercore2@server:~/Desktop$ energy_simu.py
 * Serving Flask app "battery_emulator" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 4. 62 Running server from Desktop with config.ini

4.6.2 Enabling Custom Services in CORE

To run any service in CORE it should be enabled first. To enable the service that is written in python in this thesis task following steps should be followed:

Activate custom service
To activate custom service on nodes `core.conf` file has been modified. `custom_services_dir = /home/servercore2/.core/myservices` needs to be uncommented in the `core.conf` file which is located in `/etc/core/core.conf` directory.

Following Figure 4. 63 depicts the procedure:

```
$ sudo nano /etc/core/core.conf
```

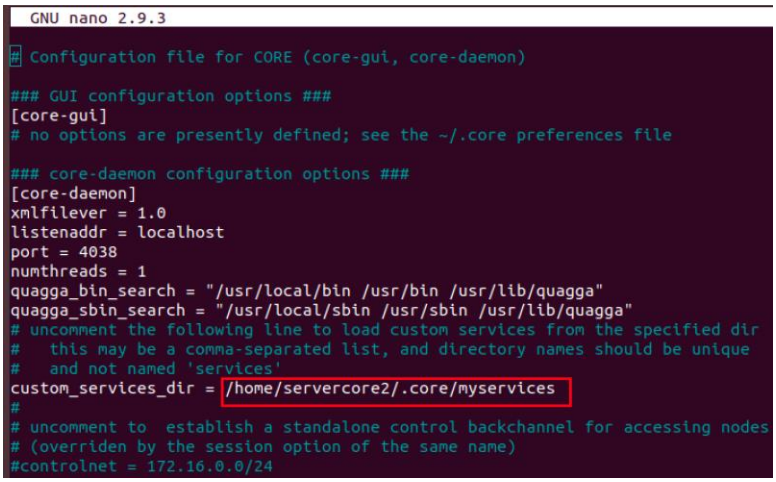


Figure 4. 63: Enabling custom service on nodes

Placing Python script to the destination folder:
The python script `samply.py` located in `/home/server/.core/myservices` has to be edited to specify the service name and other configurations. To run energy consumption service python scripts have been placed in the directory: `/usr/local/lib/python2.7/dist-packages/core/coreservices.` These scripts will be found in the attached CD.

The following Figure 4. 64 shows the *Energy_Consumption* service has been deployed in the CORE in node n2(Figure 4. 65)

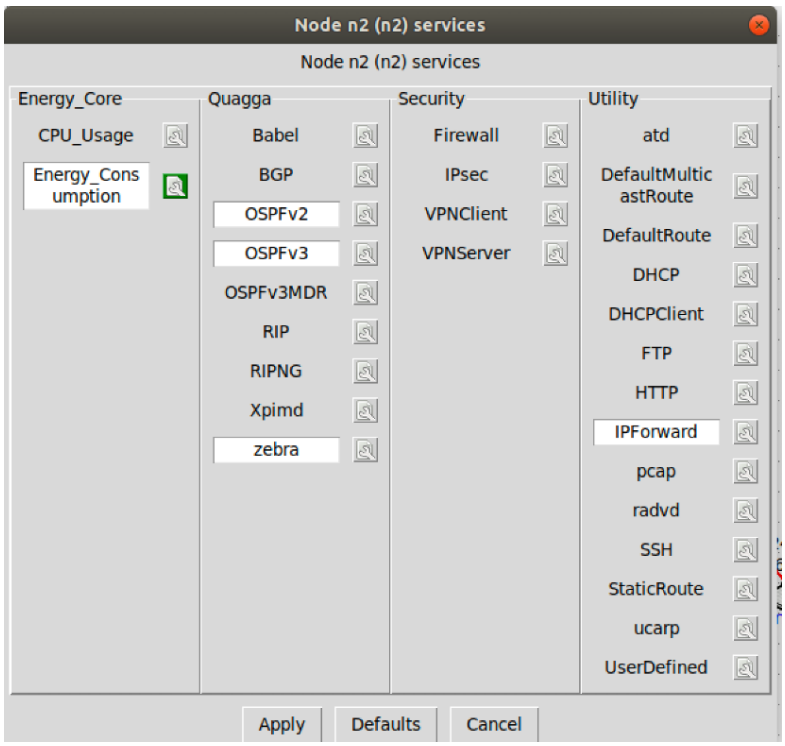


Figure 4. 64: Energy Consumption service on CORE nodes

4.6.3 Running Energy Consumption Service in CORE

After selecting the service from CORE the server will be running in the background. And then user can get the response of total consumption, current battery, history and change configuration parameters.

Following Figure 4. 65 is considered as an example of network scenario to deploy the application. To discuss all the aspects in this section this network has been taken as reference.

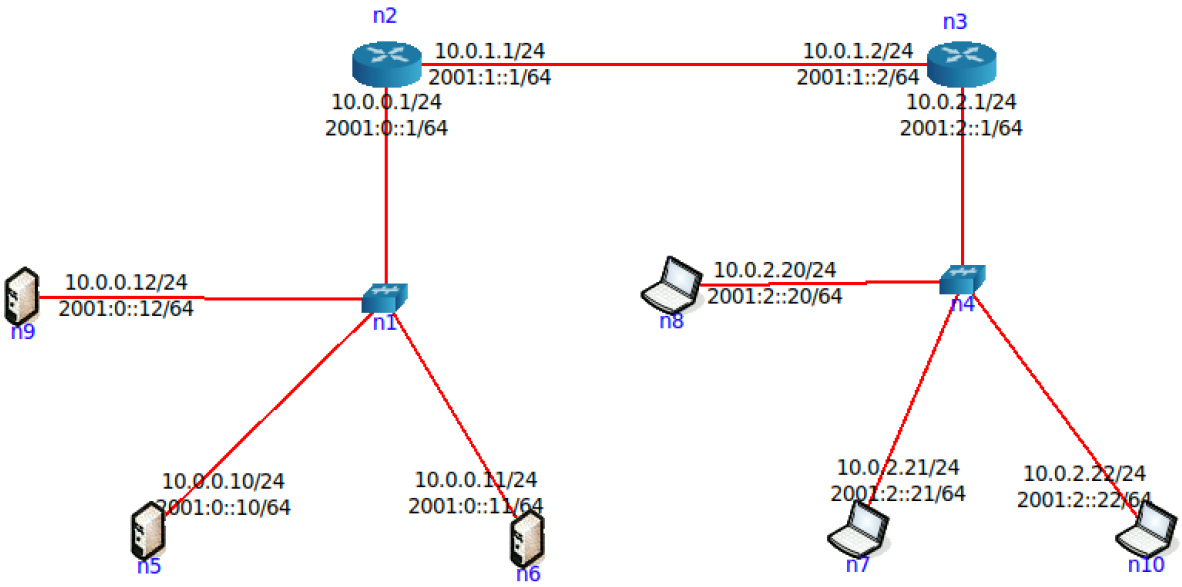


Figure 4. 65:Network scenario for server deployment

To deploy the application and to get the response with http requests following steps should be followed.

Open Service tab

To run custom service Services tab should be opened as in Figure 4. 66.

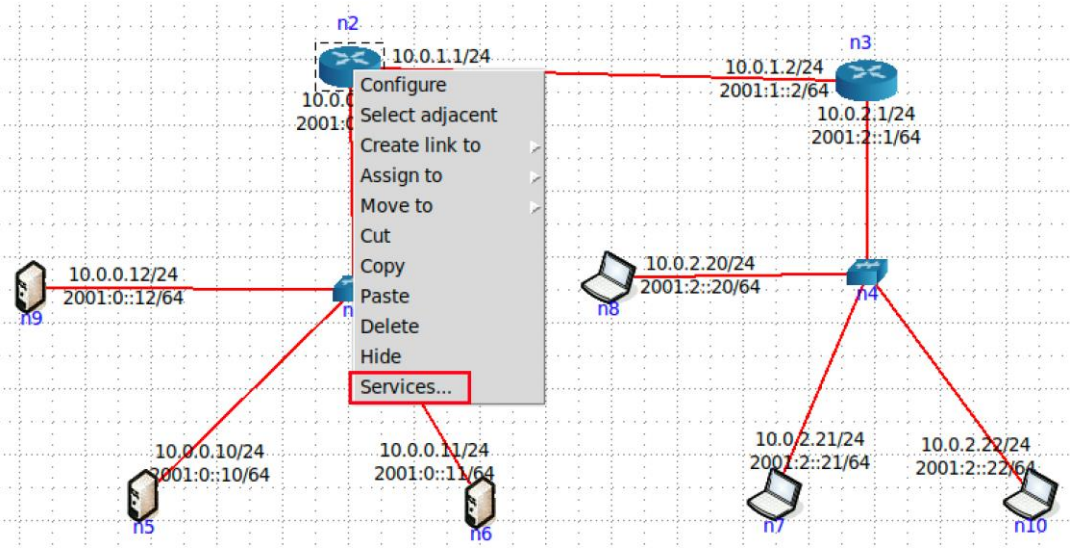


Figure 4. 66: Selecting service

After red marked option in Figure 4. 67 should be selected to edit configuration of *Energy_Consumption* service.

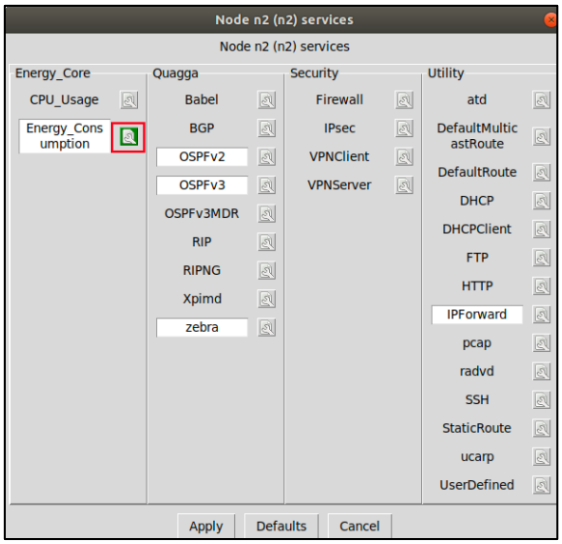


Figure 4. 67: Opening option to configure Energy_Configuration service

Selecting CORE service with `config.ini` and scripts

As all the parameters are initialized in `config.ini` and `*.sh` scripts need to be initialized from the Service tab. `energy_simu` will run with this `config.ini` and scripts, it will trigger the developed python scripts as it is programmed, after service is selected to run. These files should be initiated from Files tab of service option as depicted in Figure 4. 68. Here `config.ini` is shown with configuration only with battery percentage.

- Initializing `config.ini` from Service tab of node n2

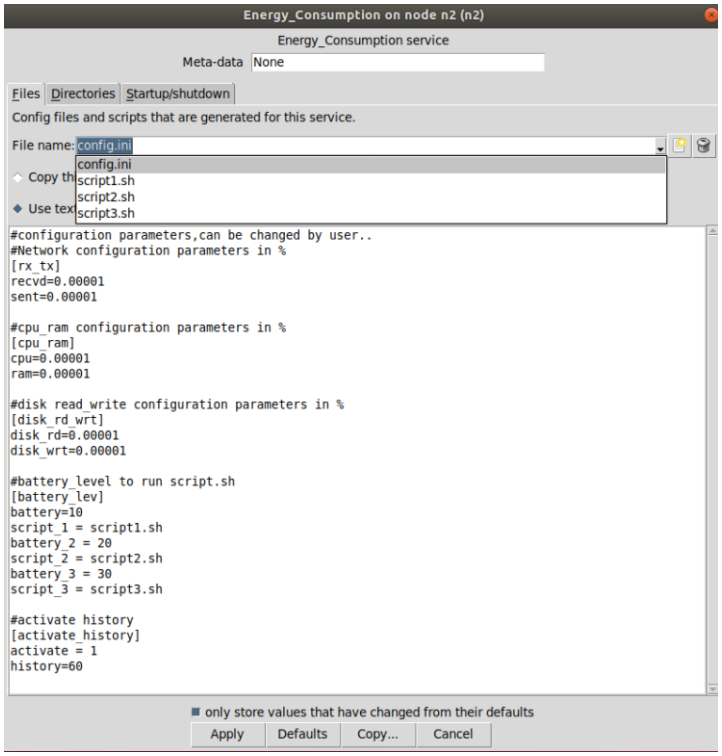


Figure 4. 68: config.ini file

Like `config.ini` scripts file can be initiated also. As an example, initialized three scripts in above `config.ini` have to be also initialized from same tab. It is done as following Figure 4. 69. Shell script program that has to be run at different battery level should be written here. This following example script will run at battery level 10 logic as mentioned in section 4.4.2. All other scripts can be written as a same way.

- Initializing script from Files option

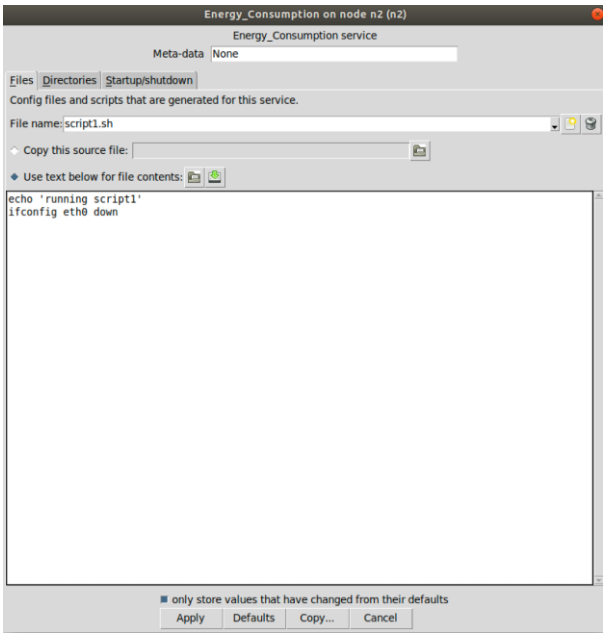


Figure 4. 69:Initialize .sh scripts

User can also add these files from the directory they want. For this they have select the directory from Use text below from file contents option. Following Figure 4. 70 can be considered as an example. After clicking the red marked box user can specify their directory.

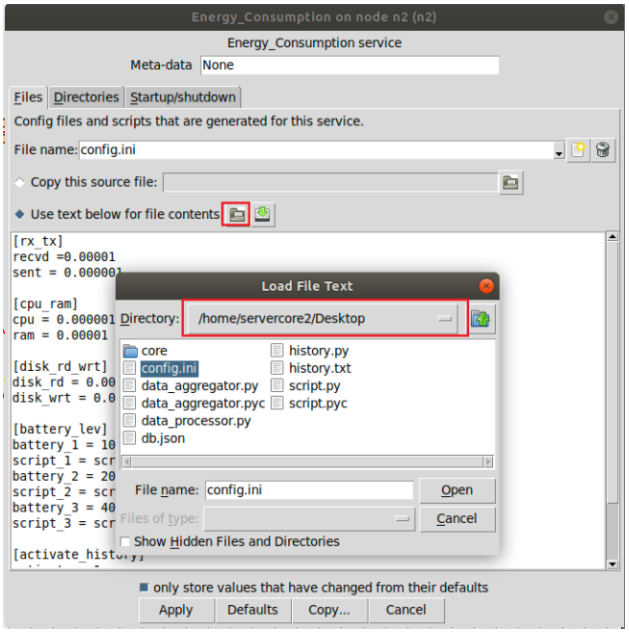


Figure 4. 70:Selecting file from Desktop

Configure Startup command

To run the server from service command should be added in Startup/shutdown option. Our command is energy_simu as this file runs the server. It should be added in this option with green marked tab as Figure 4. 71:

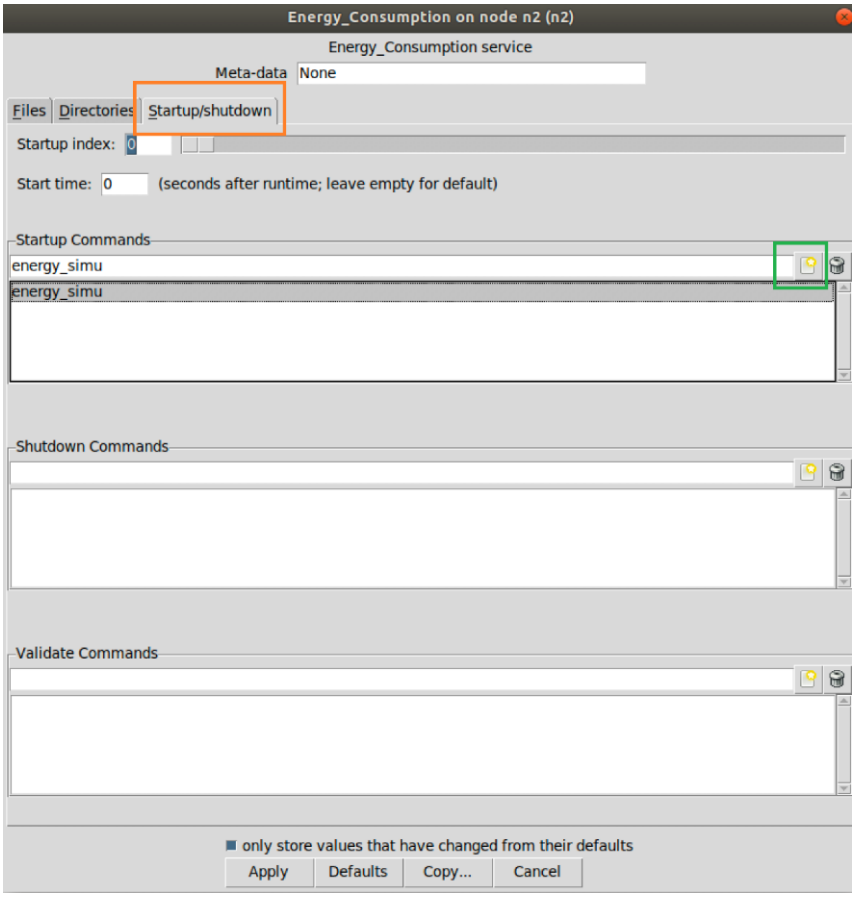


Figure 4. 71: Server Startup command

After running the service, the application will be running in the background. It can be checked by `ps aux` command.

```
$ps aux
```

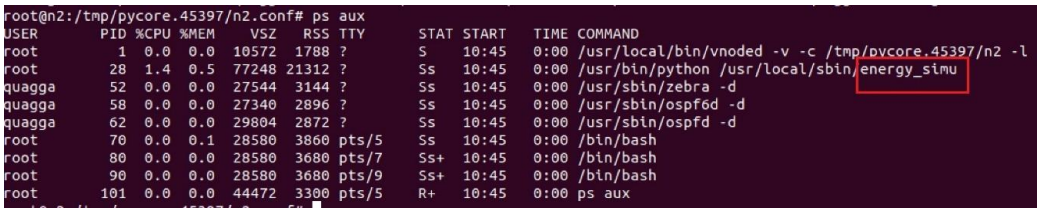


Figure 4. 72: Running energy_simu service after starting the session

4.6.4 Enabling Widgets

Widget can be used to hover on the node and get the specific result. And control network allows to send http request from outside of CORE. Steps to configure Widgets and Control Network in CORE are given in following sections 4.6.1 and 4.6.2.

- From CORE GUI Widgets option must be chosen and then Edit option should be selected like Figure 4. 73

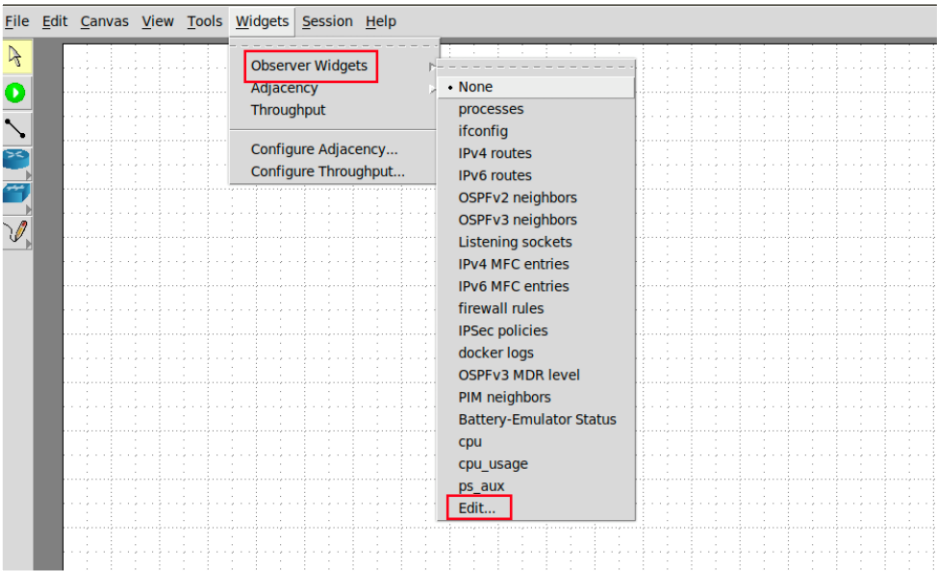


Figure 4. 73: Configure Widgets

- After selecting Edit option following window like Figure 4. 74 will appear then after clicking on new Widget name and command should be put like following Figure 4. 74.

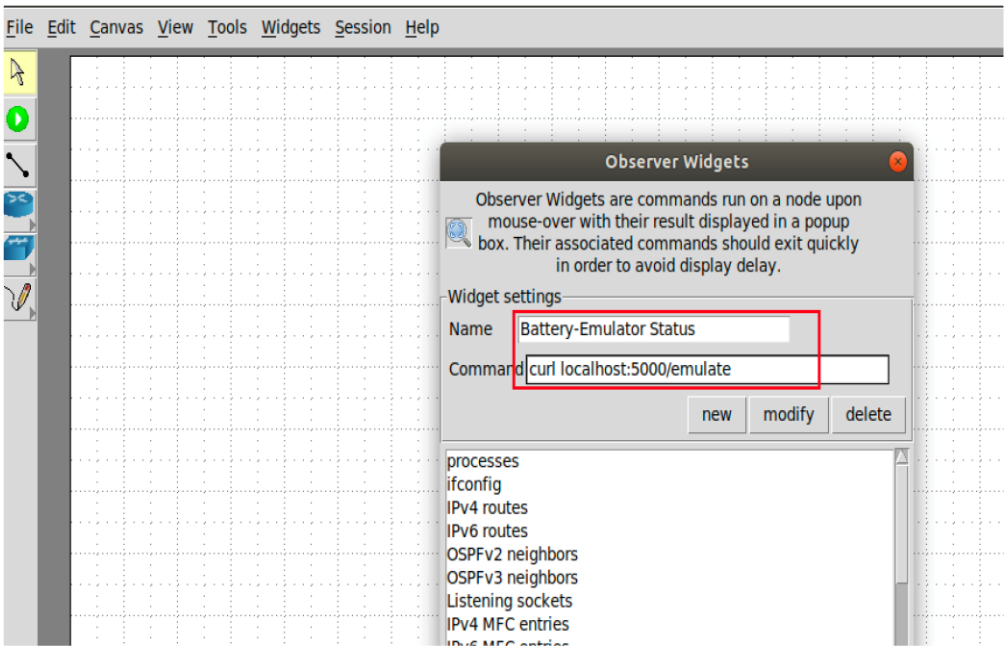


Figure 4. 74: Defining Widgets

4.6.5 Enabling Control Network

Control network should be configured to allow user to send request from outside CORE. After enabling it user can send request to the specific node using IP of that particular node. It can be done from Session option of GUI . Then following steps should be followed like Figure 4. 75 and Figure 4. 76

- Selecting Options tab

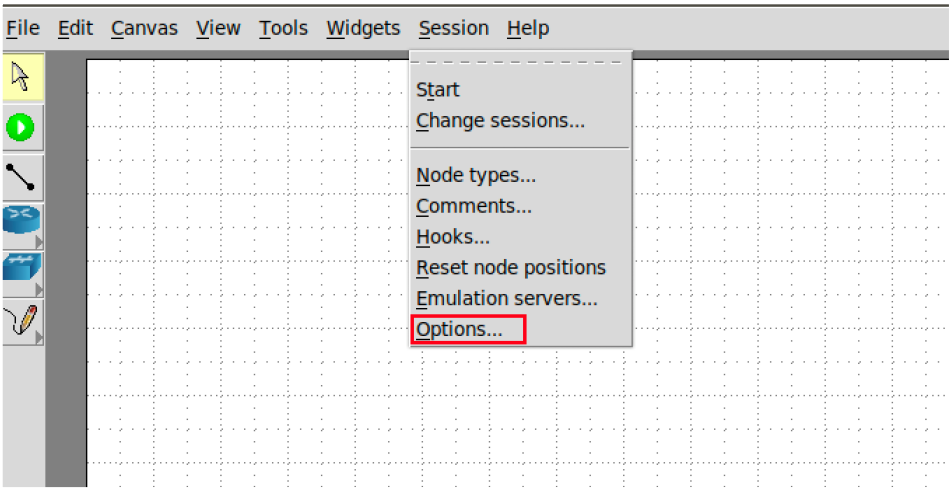


Figure 4. 75: Configure control network

- After selecting options following window will appear like Figure 4. 76 and need to define the control network IP

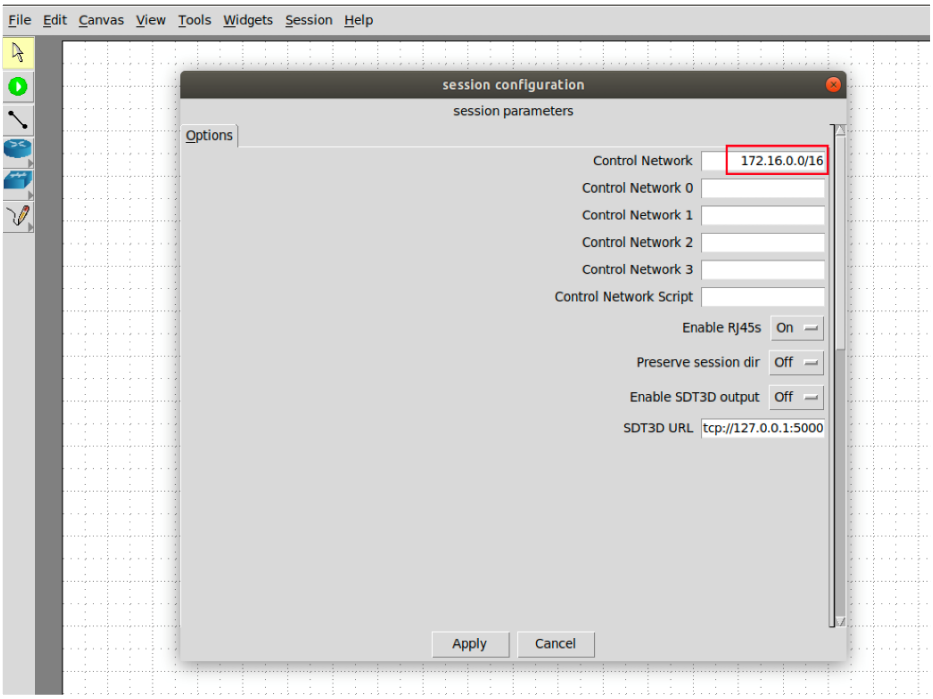


Figure 4. 76: Defining control network

4.6.6 Get Current Battery and Consumption

User can get response with http request. This service allows the user to get data:

1. From localhost
2. From outside of CORE node
3. From browser
4. From widget

To get the data from localhost request should send from same node where Energy Consumption service is running. As an example, here in following Figure 4. 77 request is sent from node n2(Figure 4. 65) as service is running in node n2. Request from other nodes will not work without allowing control network shown in 4.6.5. It shows current consumption, current battery, PID (consuming maximum CPU/RAM usage) as shown in Figure 4. 77.

```
$ curl http:localhost:5000/emulate
```

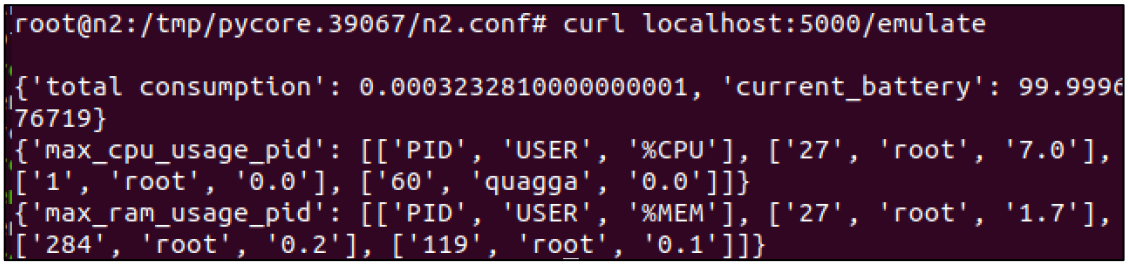


Figure 4. 77:Sending request from localhost

To get the data outside of CORE node control network IP should be used as mentioned in section 1.6.2. Here, for node n2 control network IP is 172.16.0.2. Below Figure 4. 78 And Figure 4. 79 replicate the scenario of sending request from Linux terminal

```
$ ps aux
```

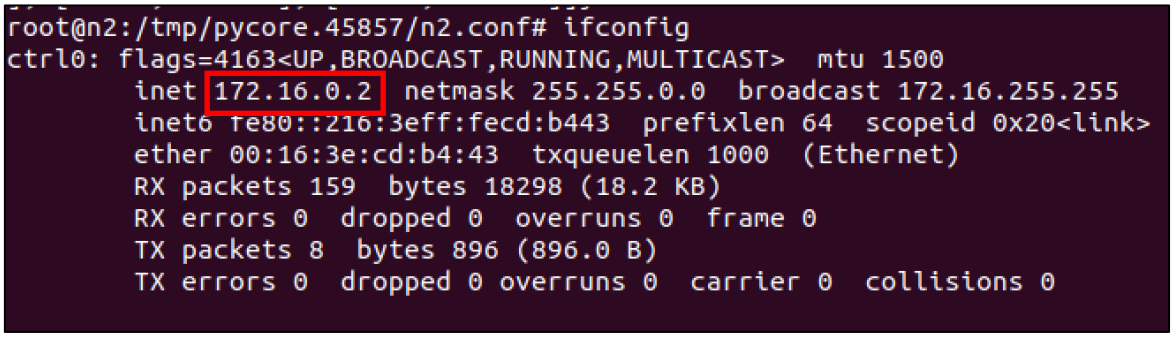


Figure 4. 78:Control network IP of node n2

```
$ curl http://172.16.0.2:5000/emulate
```

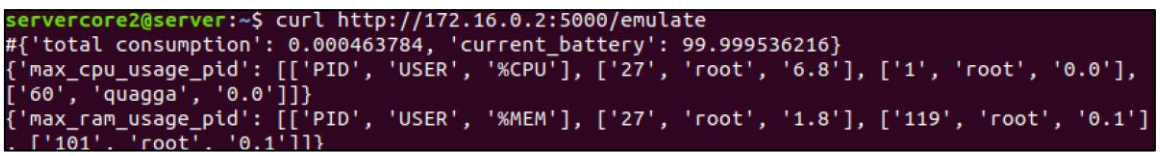


Figure 4. 79:Sending request from outside CORE

Request can be sent from browser in same way as Figure 4. 79. Below Figure 4. 80 depicts the result:

```
$ http://172.16.0.2:5000/emulate
```

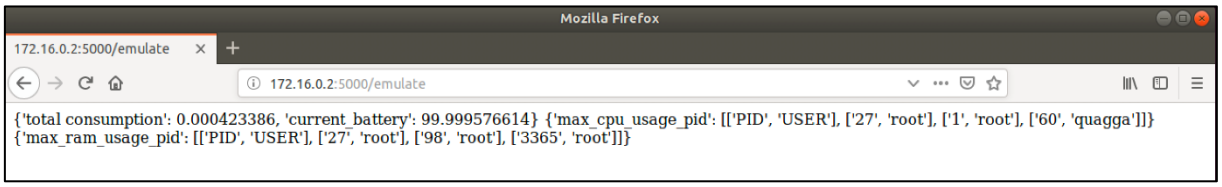


Figure 4. 80:Getting response from Browser

Current battery and consumption can be seen from widget after configuring customised widget as shown in section 4.6.4. And hovering on the will show the result like following Figure 4. 81

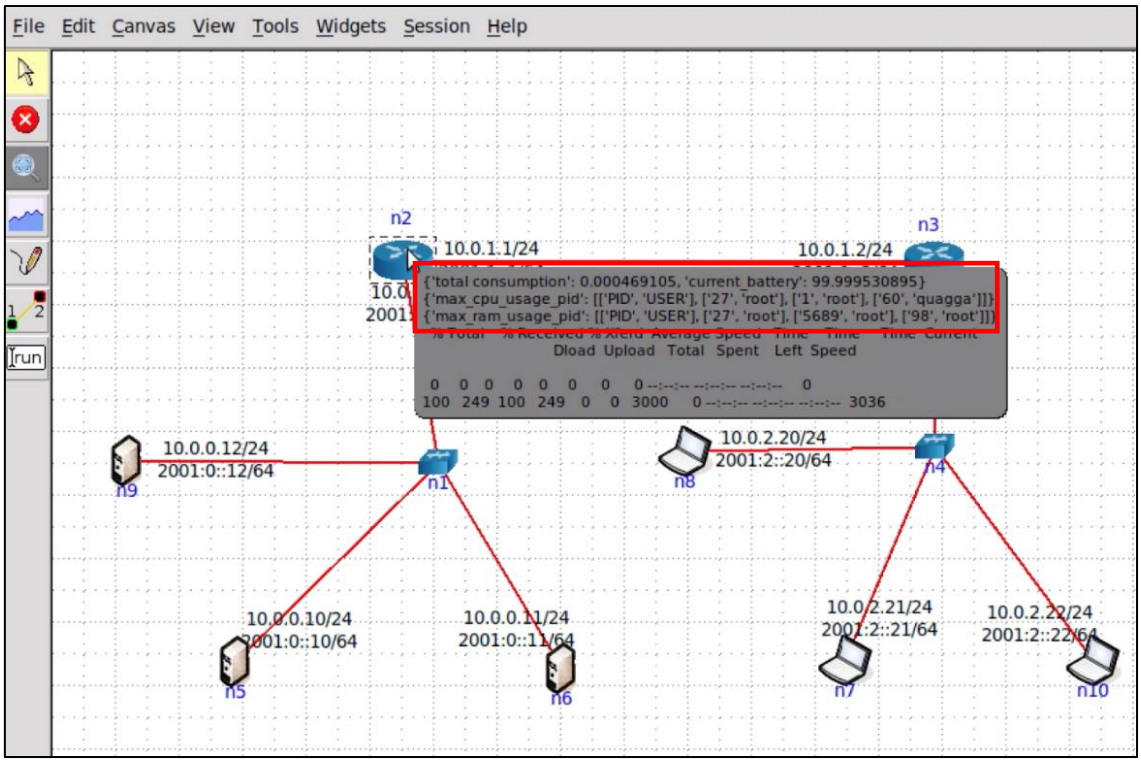


Figure 4. 81:Response from Widget

4.6.7 Change all Parameters with POST request

After running the service in CORE it is not possible to change the configuration, that's why a POST method can be used to change the configuration.

- Parameters can be changed by POST request from localhost as following

```
$ curl -d '{"rx":0.0001, "tx":0.0001,"cpu":0.0001,
"ram":0.0001,"read":0.04, "write":0.05}' -H "Content-Type: applica-
tion/json" -X POST http://localhost:5000/change_all
```

Figure 4. 82 it is shown taken when parameters have not been changed and in Figure 4. 83 it can be seen, after POST request parameters have been changed the `config.ini`.That means calculation will be continued with this new configuration.

```
root@n2:/tmp/pycore.35763/n2.conf# cat config.ini
[rx_tx]
recvd = 0.0001
sent = 0.0001

[cpu_ram]
cpu = 0.0001
ram = 0.0001

[disk_rd_wrt]
disk_rd = 0.0001
disk_wrt = 0.001
```

Figure 4. 82:Before changing parameters

```
root@n2:/tmp/pycore.35763/n2.conf# cat config.ini
[rx_tx]
recvd = 0.0001
sent = 0.0001

[cpu_ram]
cpu = 0.0001
ram = 0.0001

[disk_rd_wrt]
disk_rd = 0.04
disk_wrt = 0.05
```

Figure 4. 83: After changing parameters by POST request

After changing the configuration new calculated battery can be retrieved as shown in Figure 4. 84. It is shown that after changing the configuration parameters current battery decreases after recalculation.

```
oot@n2:/tmp/pycore.35763/n2.conf# curl localhost:5000/emulate
'total consumption': 2.0831777999999996, 'current_battery': 97.9168222}
H "Content-Type: application/json" -X POST http://localhost:5000/change_all
oot@n2:/tmp/pycore.35763/n2.conf# curl localhost:5000/emulate
'total consumption': 5.223926200000001, 'current_battery': 94.7760738}
oot@n2:/tmp/pycore.35763/n2.conf#
```

Figure 4. 84: Current battery changes after changing configuration

It can be changed outside of CORE also using control network IP. As an example, control IP of the node n2 in Figure 4. 65 is 172.0.0.2 then the request will be:

```
$ curl -d '{"recv":0.0001, "sent":0.0002,"cpu":0.00001,
"ram":0.00001,"read":0.005, "write":0.03}' -H "Content-Type: applica-
tion/json" -X POST http://172.0.0.2:5000/change_all
```

4.6.8 Get Summary of History

To get the summary of history GET request can be sent from any of the option mentioned above: from CORE and outside CORE, from browser As an example, to read last 60 seconds history from localhost: http request would be:`curl http:localhost:5000/history/60.`

Below screenshot in Figure 4. 85 shows the screenshot of getting history from localhost node n2 (Figure 4. 65)

```
$ curl localhost:5000/history/60
```

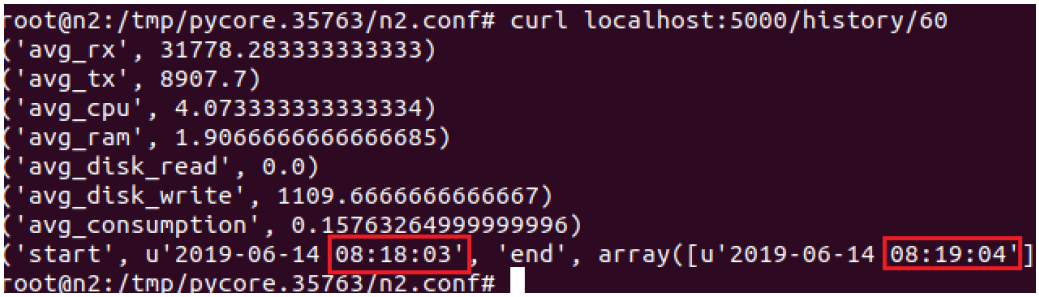


Figure 4. 85: Summary of history for last 60 seconds

5 Example Scenario Using the Application

The goal of this chapter is to illustrate some usages of the application. From chapter four the development of the thesis task has already been known. Two usage scenarios of this application will be described here. They are:

- Impact of application on network route
- Simulation with changing parameter usage
- Determination of real values

5.1 Impact of Application on Network Route

The application is able to run user definable shell script at different battery level. In section 4.4.2 it is already discussed how different scripts run at different battery levels. The goal of this section is to show the change in actual network topology if nodes or interfaces are shutting down by the application when running out of energy.

5.1.1 Test Scenario

To carry out this test following Figure 5.1 has been considered.

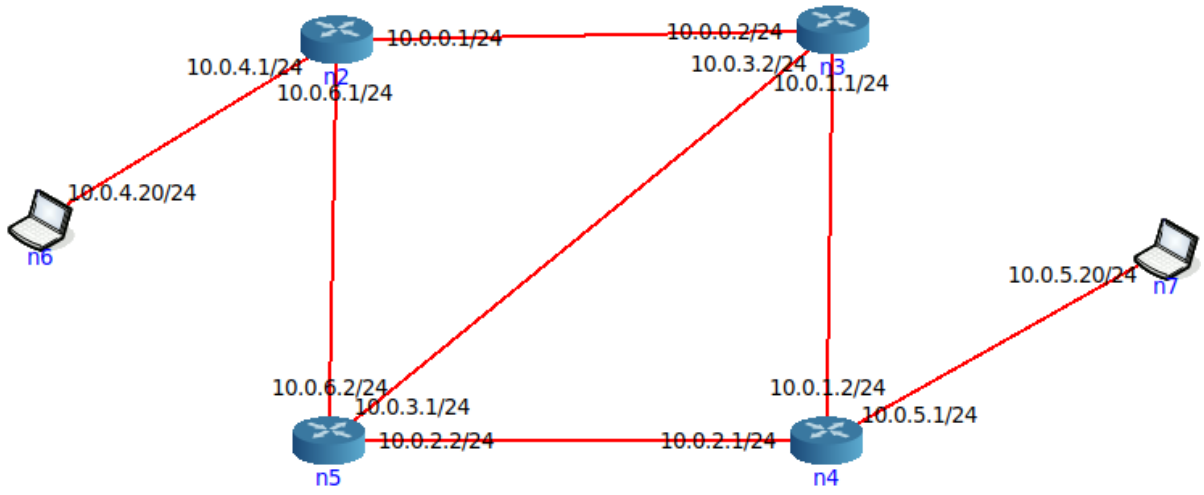


Figure 5. 1:Test network scenario

OSPF (Open Shortest Path First) routing protocol has been used in network scenario. The plan is to run the application *Energy Consumption Simulator* as a service from node n3 and n5 with defined scripts at different battery level and showing how it impacts the network route.

Energy Consumption service can be activated, and shell scripts can be activated as discussed in section 4.6. And configuration parameters and scripts can be enabled from the service option.

The *Energy_Consumption* service has been activated in all nodes. And configured with same configuration parameters. As an example, in below Figure 5. 2 in node n5 `config.ini` is configured with given values. All other nodes are configured with same values after selecting the service.

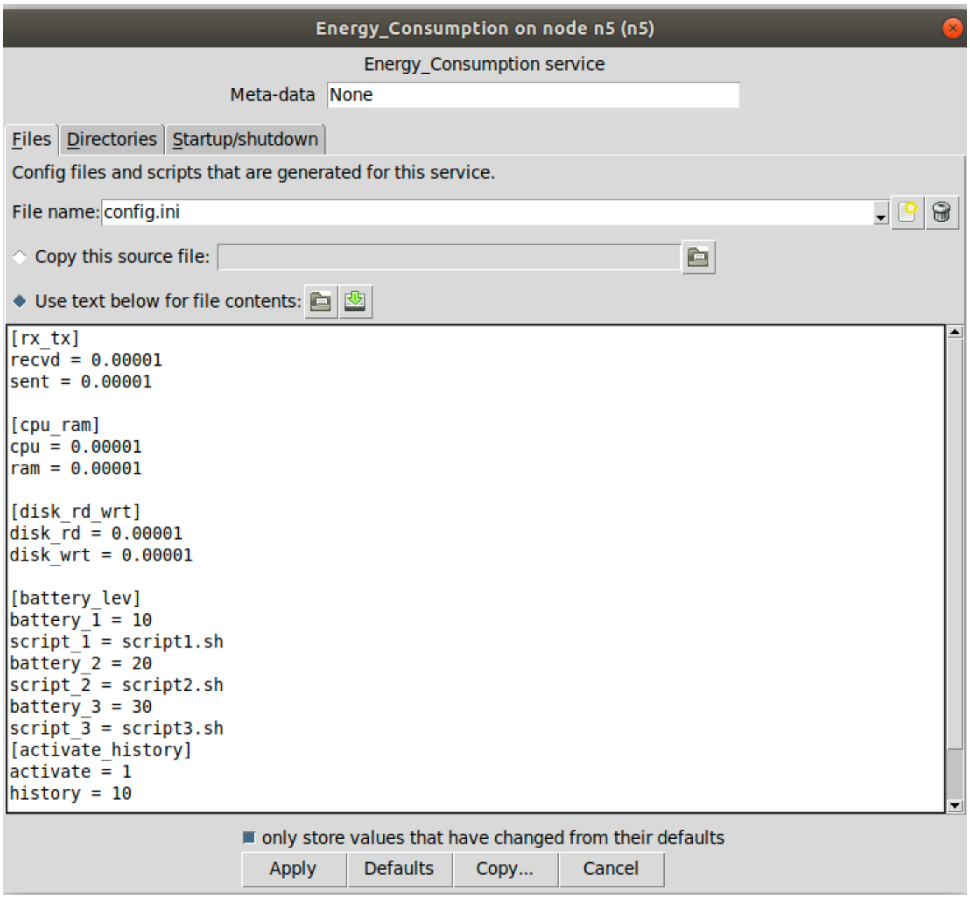


Figure 5. 2: Config.ini for node n5

Scripts have been configured in two nodes only: node n3 and node n5. In other nodes no shell scripts have been configured.

Before discussing about the task some network information should be mentioned. It can be shown that these nodes (n3, n5) have three interfaces each. Below table 5.1 shows all the ethernet IP addresses of node n3 and n5

Table 5.1: Ethernet IP addresses of node n3 and n5

nodes	eth0	eth1	eth2
n3	10.0.0.2	10.0.1.1	10.0.3.2
n5	10.0.2.2	10.0.3.1	10.0.6.2

Scripts that have been used in this testing scenario are given below:

Node n3:
Define script: `script3.sh.`

`script3.sh`

```
ifconfig eth1 down
```

In this node only script 3 is activated for battery level 30%, that means if battery level goes to 30% or less than that (till 20%) this script will run once. At this level eth1(10.0.1.1) should be down. It can be configured in CORE shown as below Figure 5. 3.

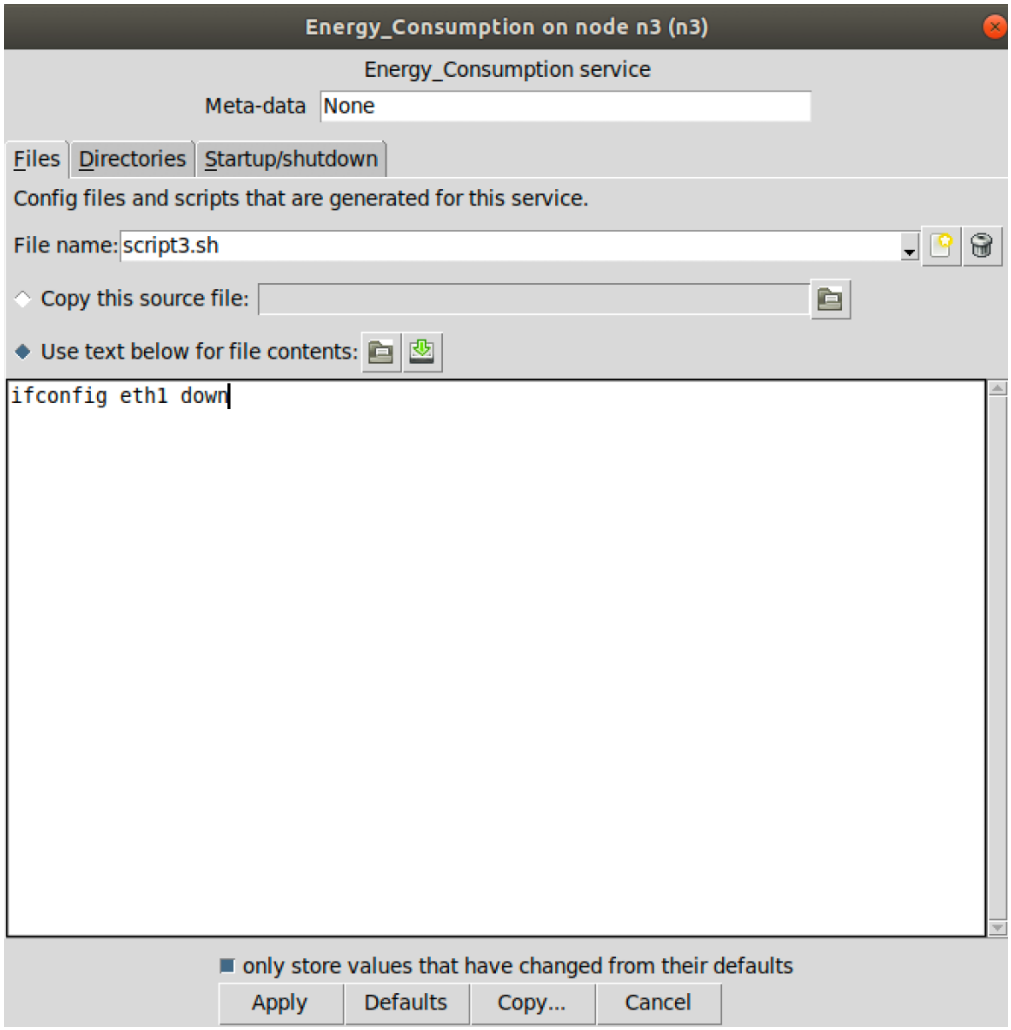


Figure 5. 3:Configured script3.sh at node n3

Node n5:
Define scripts

`script3.sh`

```
ifconfig eth2 down
```

`script1.sh`

```
ifconfig eth0 down

ifconfig eth1 down
```

In this node at first script3 is activated for battery level 30%, that means if battery level goes to 30% or less than that (till 20%) this script will run once. At this level eth2(10.0.6.2) interface should be down. It can be activated as depicted in Figure 5. 4.

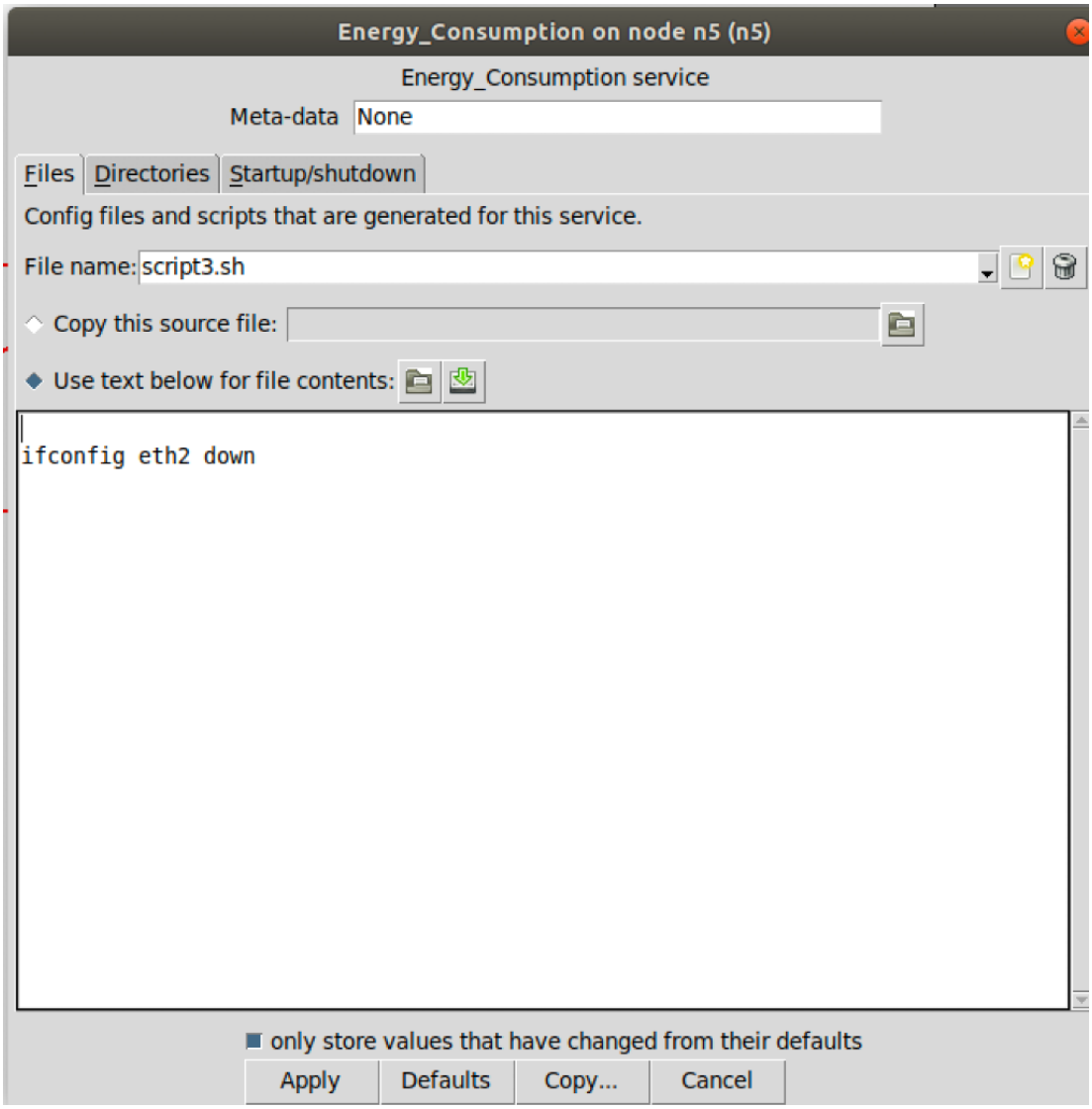


Figure 5. 4:Configured script3.sh at node n5

And another script `script1.sh` is activated at battery level 10%, that means if battery goes to 10% or less than that eth0(10.0.2.2) and eth1(10.0.3.1) goes down. That means all interfaces will be down at this battery level for node n5. It is configured as below Figure 5. 5.

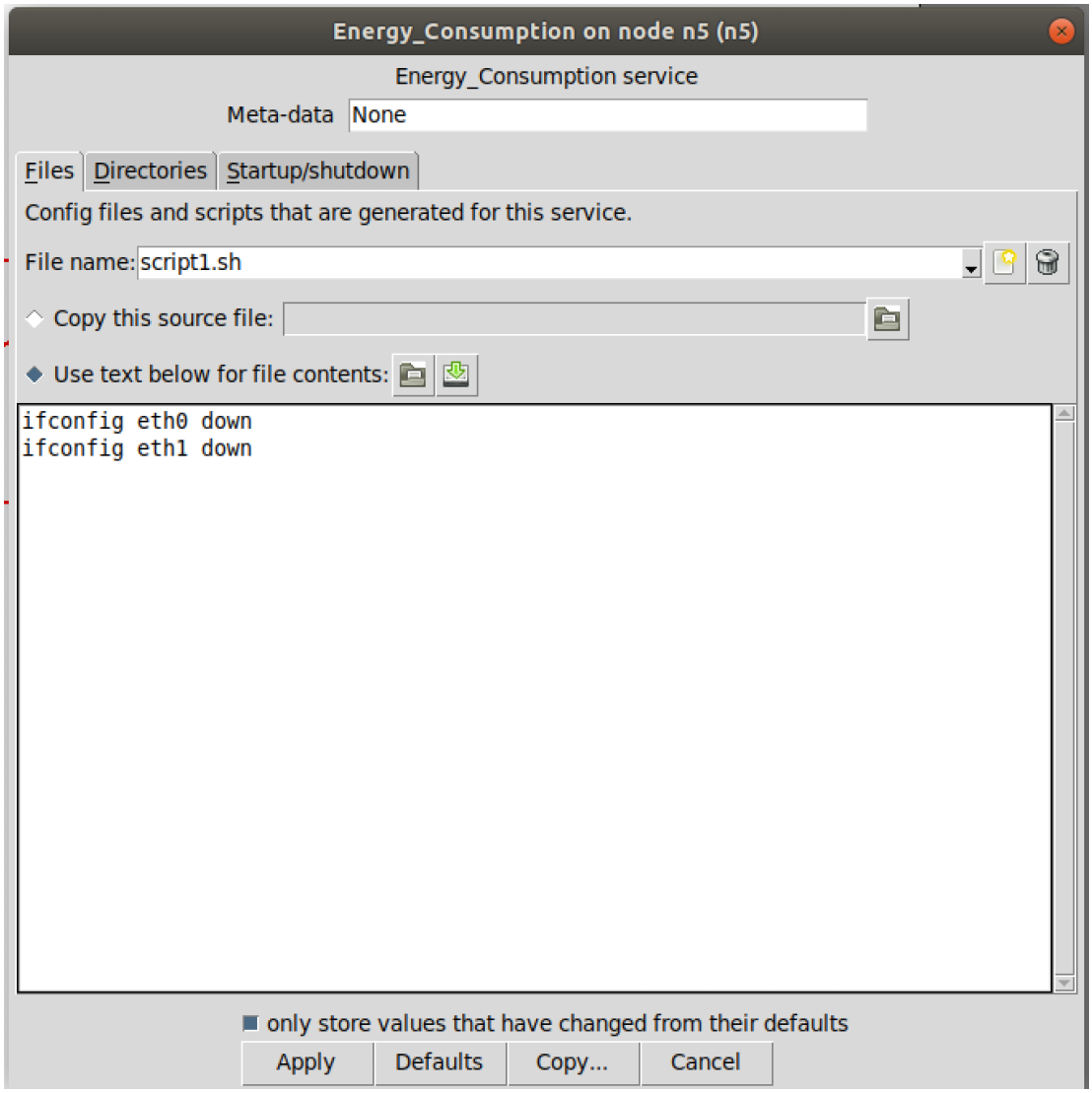


Figure 5. 5: configured script1.sh at node n5

For testing purpose one Linux command `ping -f` has been used to send and receive continuous data to the destination.

Ping -f: It is known as flood of ping or ICMP flood. For every ICMP ECHO_REQUEST sent a period '.' is printed while forever ECHO_REPLY received a backspace is printed. This provides a rapid display of how many packets are being dropped. It continues to flood ping if no interruption is given (Canonical LTD,2019g).
This command has been used to flood ping from node n6 to n7 shown in Figure 5.1 and that means there will be a change in the throughput in the route.

To show the throughput in route Throughput option should be selected from Widgets option as showed in Figure 5. 6

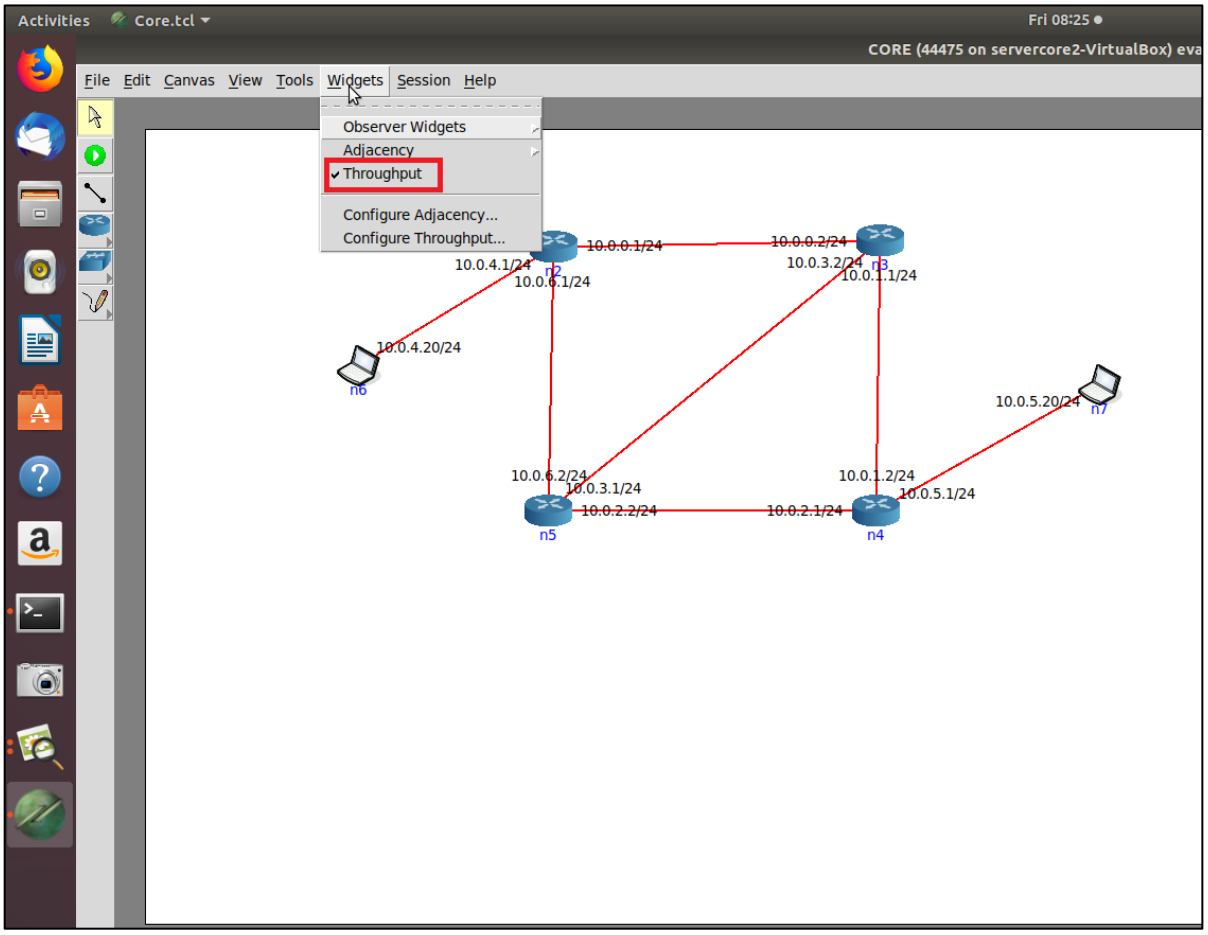


Figure 5. 6: Selecting Throughput from Widget

5.1.2 Result After Running the Service

In this section some screenshots of the result after running the Service will be shown. These screenshots have been taken from CORE GUI. After running the command `ping -f` from node n6 to n7 terminal what changes have taken place is shown below. This is already told `ping -f` command will be used to create a ping flood and different situations will be visualised with given screenshots.

```
$ ping -f 10.0.5.20
```

Route 1:
After ping flood the first route it has taken is as Figure 5. 7. It is taking route through n2 towards n7 using eth0 and eth2 of n5.

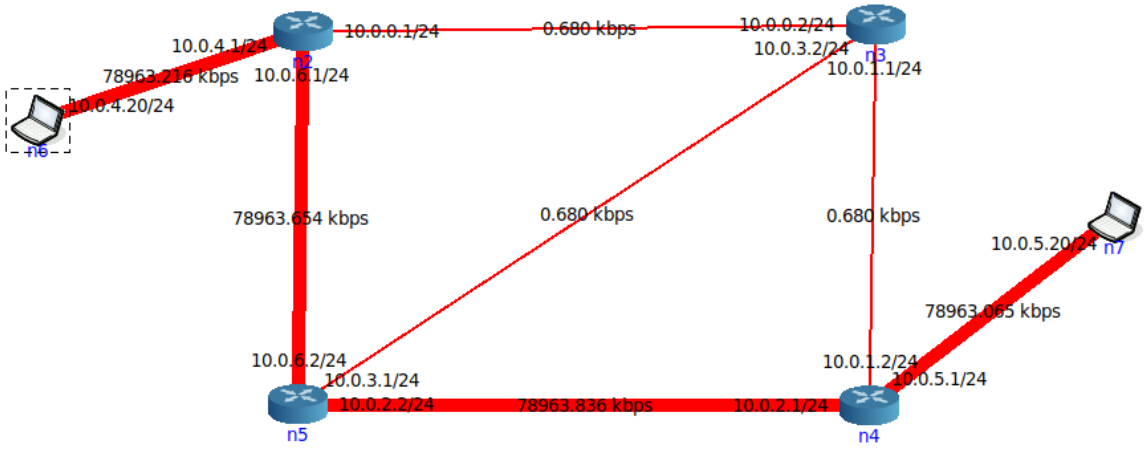


Figure 5. 7: Network route 1 after ping -f 10.0.5.20

Transforming phase from route 1
As it is mentioned, at node n5 `script3.sh` is activated to make eth2(10.0.6.2) down. That means at 30% level it should meet the condition and given ethernet should be down. In Figure 5. 8 it can be seen at battery level less than 30% it is in the transforming phase of changing route as eth2 got down with the application. Here, it is still highlighting the old route as it is distributing new route information to new node.

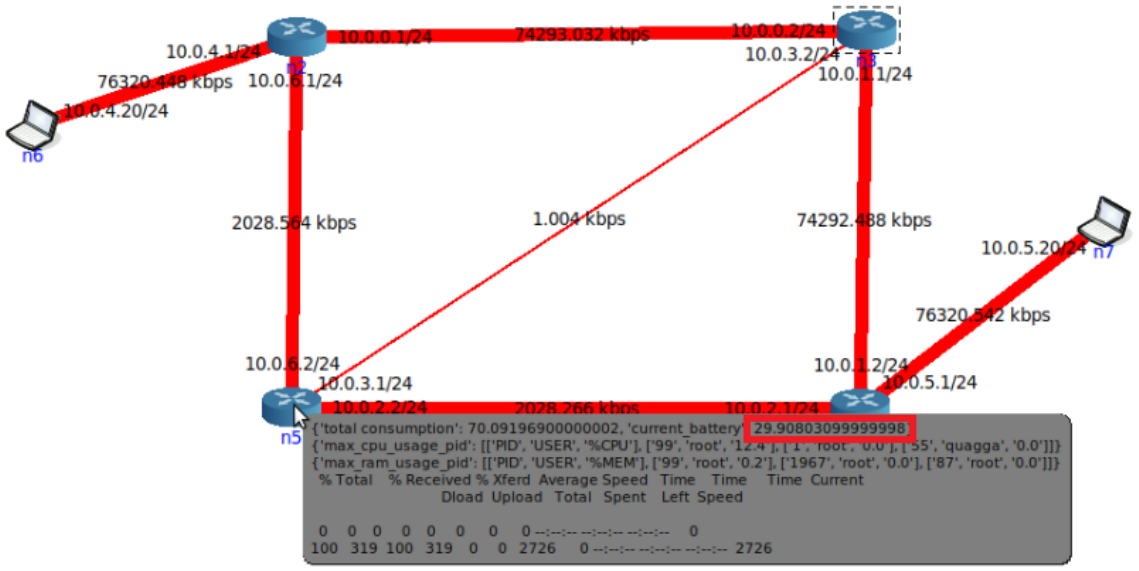


Figure 5. 8: Transform phase of route

Route 2

In Figure 5. 9 it can be shown that now it is taking new route using n2, n3,n4 towards n7.

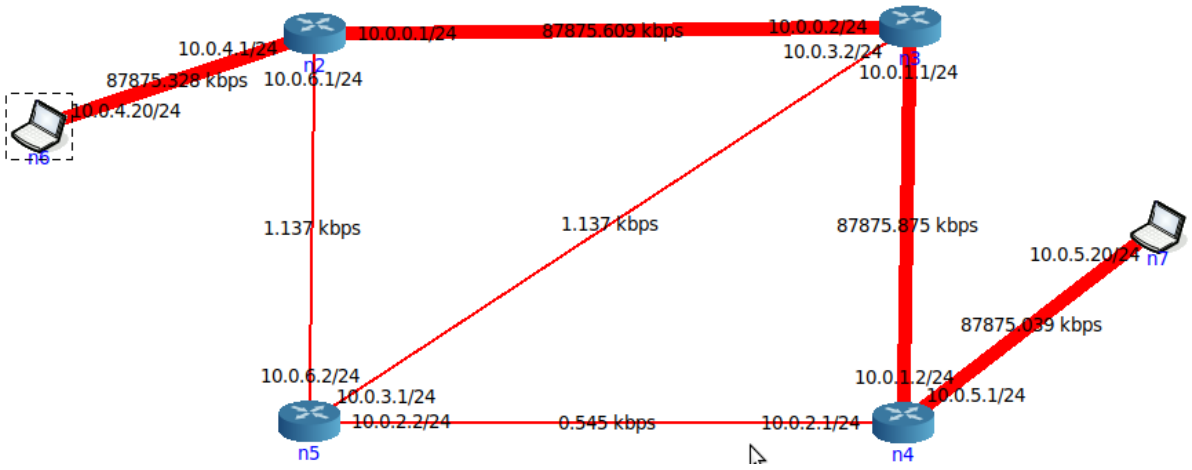


Figure 5. 9: Route 2 after making eth0 (n5) down

Transforming phase from route 2

It is already mentioned that in node n3 `script3.sh` is activated to run script at battery level 30% condition. So, logically if battery goes below 30% (not less than 20%) it should run the script once. And this is depicted in Figure 5. 10. As eth1(10.0.1.1) is down of node n3 it is changing the route from this phase.

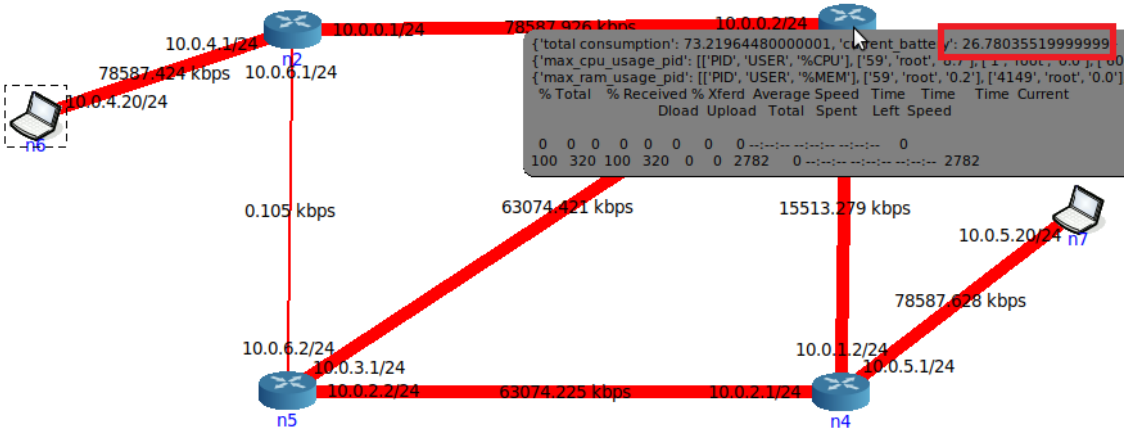


Figure 5. 10:Transform phase from route 2 to route 3

Route 3

After making the chosen ethernet down it takes another route as shown in Figure 5. 11. It is now taking the route through n2, n3, n5, n4 to reach node n7.

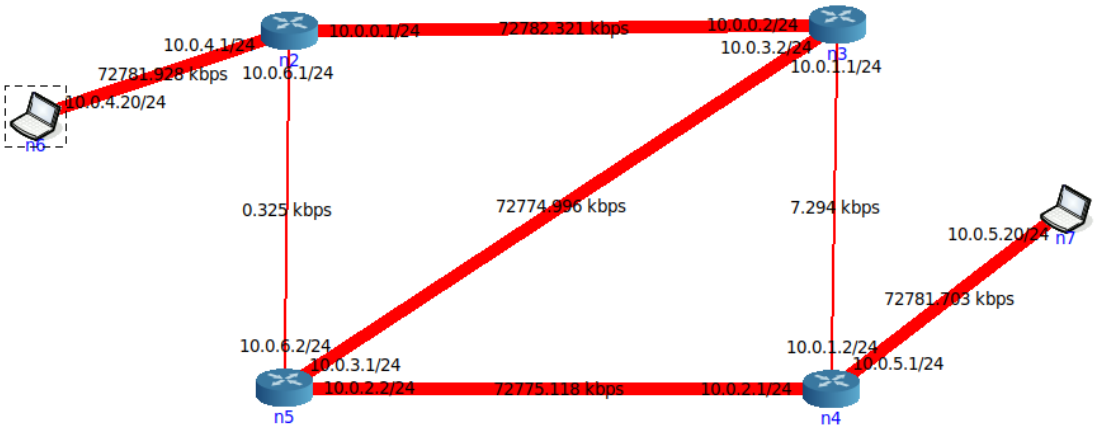


Figure 5. 11: Choosing route 3 after making the eth1(n3) down

Closing the node n5

It is already stated that at 10% battery level condition all nodes (eth0, eth1) should be down. It is shown in Figure 5. 12. All interfaces are down of node n5 at battery level 10% condition. That means it does not have any alternate route to send traffic to node n7. So, the amount of traffic has become less after this situation.

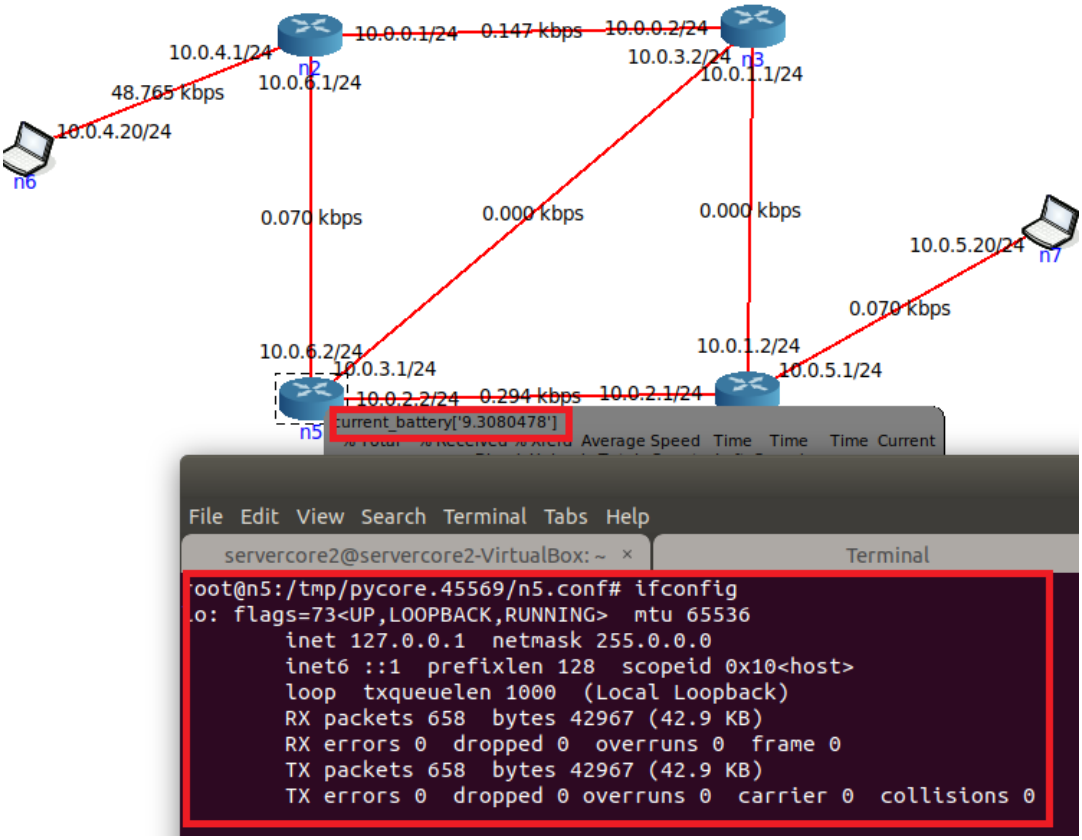


Figure 5. 12:: All interfaces(n5) down

Comparison of values of nodes (n3, n5, n7)

In previous section it is already shown how the scripts have affected the network routes. During this test, battery %, values of parameter of nodes n3, n5 and n7 have been taken from `db.json`. The values have been stored by `data_processor.py`. and a graph is drawn with the values using Microsoft Excel. In X-axis it has been denoted in second and in Y-axis it has been denoted in battery %.

In graph (Figure 5. 13) it can be shown that battery percentage of node n5 drains drastically from beginning, this is because the 1st route was through the interfaces(eth0,eth2) of node n5 as shown in Figure 5. 7. At this situation node n3 battery level remains stable as the traffic does not take the route with node n3. Node n7 battery level also goes down with time as it is the destination node and send and receive traffic is always available there which impacts the battery level.

After going to battery level 30% (at approx. 56 second in Figure 5. 13) of node n5 the interface(eth2) got down and then it takes the route 2 through node n3 interfaces(eth0,eth1) as shown in Figure 5. 9. That's why when the battery level of node n5 is at 30% traffic shifts to node n3 and the battery level of node n3 goes down drastically as shown in the graph in Figure 5.9.

At this situation node n5 battery % gets stable a bit, as there the route does not involve any of the interfaces of node n5 as shown in Figure 5. 9.

At 30% battery level node n3 interface(eth1) got down and it changes the route as shown in Figure 5. 11 using the interfaces (eth0, eth1) of node n5 again. Node n3 and n5 both battery level goes down as route 3(Figure 5. 11) involves interfaces of both n3 and n5.

At approximately battery level 10% (at approx. 99 second in Figure 5. 13) all interfaces of node n5 gets down and no traffic is blocked, and no route is there for the network to choose with OSPF. After shutting the nodes of node n5 it gets stable at this point.

Therefore, traffic gets blocked and battery level of all three nodes have got stable at this point as there is no changes in the parameters (RX, TX, CPU, RAM, Disk Read/Write) which affects battery percentage.

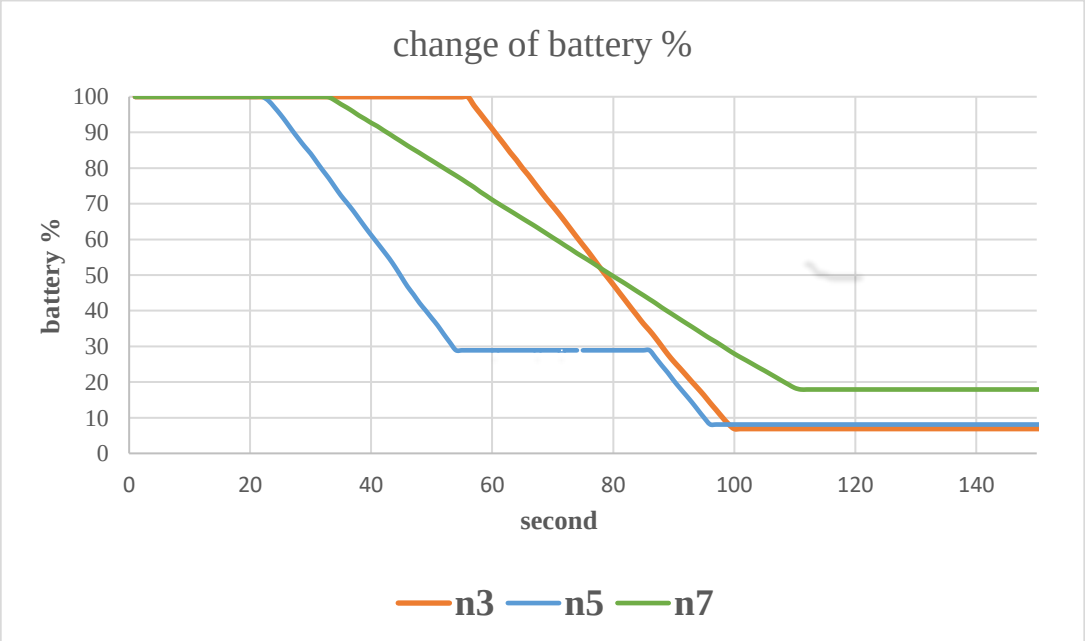


Figure 5. 13: Changing of battery % of nodes (n3, n5, n7)

5.2 Simulation with Changing Parameter Usage

In this section simulation will be tested with different scenarios. Some network nodes will be simulated and tested with an application. It aims at showing the change in consumption due to change in usage of energy configuration parameters. Therefore, current battery changes for the change in energy consumption. For discussion, some screen-shots will be shown and battery percentage values of corresponding node have been taken from `db.json` saved by `data_processor.py`. Added graphs have been drawn in Microsoft Excel.

Some Linux terminal commands will be used to load the network usage. These can be treated as service or application running on corresponding nodes. The commands are:

ping -f: Flooding the network with RX TX bytes and test simulation with changing network bytes.
stress: It is a simple workload generator for a system. It imposes a configurable amount of CPU, RAM, Disk I/O stress on system (NixCraft,2019).

5.2.1 Simulation with Changing Network Bytes

In Figure 5. 14 it is shown that node n1 is added in the network scenario and other network information are same as shown in previous section 5.1.1. The *Energy Consumption* service is activated, and it has been configured with below configuration parameter values:

Config.ini

```
#1000 bytes consume X% battery
[rx_tx]
rx = 0.00001
tx = 0.00001
#1% cpu/ram consumes X% battery
[cpu_ram]
cpu = 0.000001
ram = 0.000001
#1000 bytes read write consume X% battery
[disk_rd_wrt]
disk_rd = 0.00001
disk_wrt = 0.00001
#activate history
[activate_history]
activate = 1
history=30
```

Figure 5. 14 node n1 is sending traffic (through `ping -f`) to node n7, so the shortest path it takes is node n4 to n7.

At this point battery level of node n3 and n5 shown in Figure 5. 14 and Figure 5. 15 and does not change as it does not impact on these nodes.

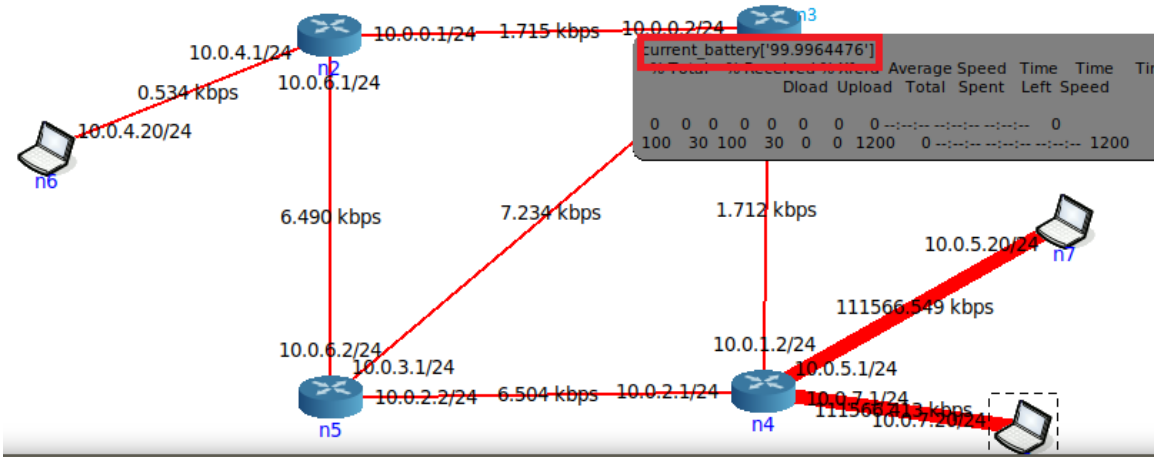


Figure 5. 14: Unchanged battery % for node n3

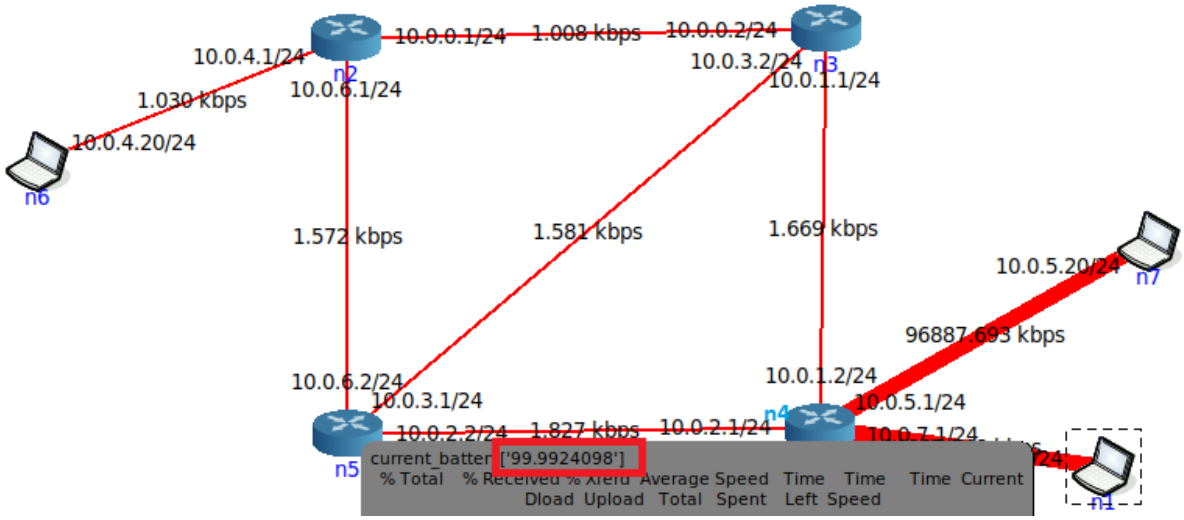


Figure 5. 15:Unchanged battery % for node n5

In Figure 5. 16 it can be shown that at node n7 the battery has been drained drastically to 16.95% as the traffic hit this node continuously.

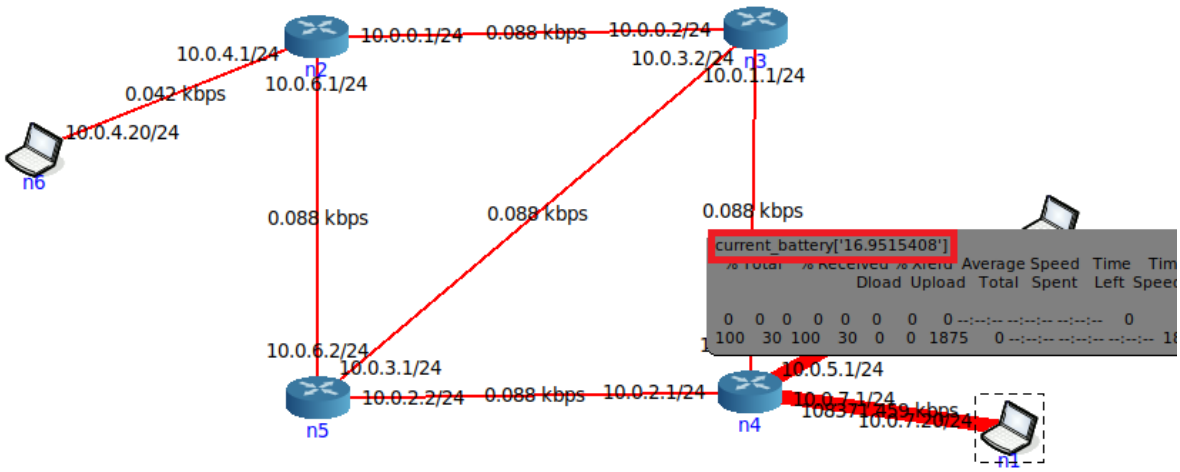


Figure 5. 16: Battery level draining at node n7

This scenario can be visualised clearly with the below graph given in Figure 5.13. In X-axis it is denoted in second and in Y-axis it is denoted as battery %.

This is visible that for this scenario battery level at node n3 and n5 is completely stable while at node n7 battery level drains drastically to 0.

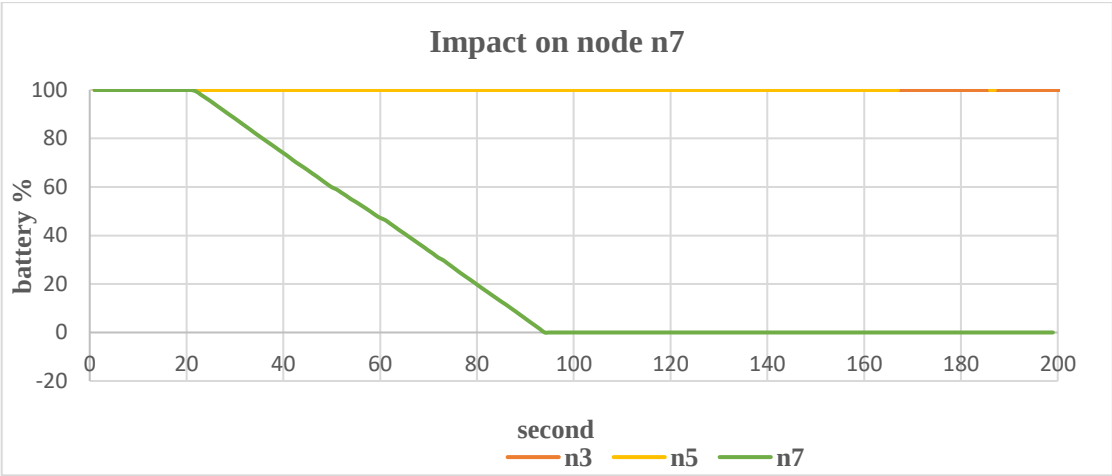


Figure 5. 17: Simulation at specific node

5.2.2 Simulation with Changing CPU RAM and Disk Usage

To test this scenario, CPU RAM and Disk have been loaded with usage. `stress` command of Linux has been used. Below network scenario is taken as in consideration for the testing.

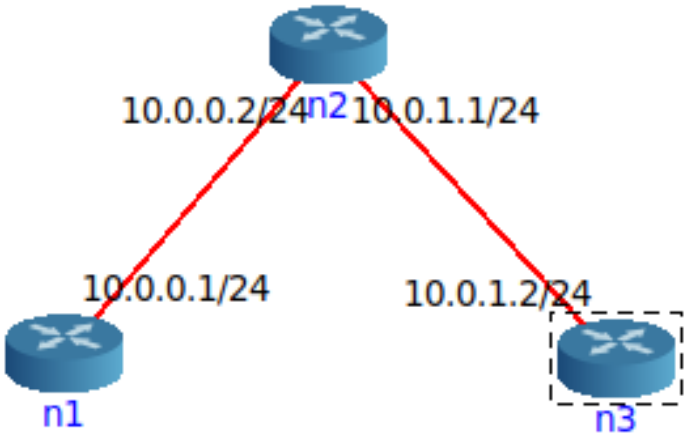


Figure 5. 18:Network scenario for CPU, RAM, Disk load testing

Service has been activated from all the nodes shown in Figure 5. 18. And configuration parameter has been also added from the service option.

Simulation with Loaded CPU and RAM
To cause load on node n2 with CPU and RAM, below command has been used.

`$stress -c 4 -m 4`(loading 4 process with CPU and RAM)

And below configuration has been given in `config.ini` from service option of CORE node.

`Config.ini`

```
#1000 bytes consume X% battery
[rx_tx]
rx = 0.00001
tx = 0.00001
#1% cpu/ram consumes X% battery
[cpu_ram]
cpu = 0.1
ram = 0.1
#1000 bytes read write consume X% battery
[disk_rd_wrt]
disk_rd = 0.00001
disk_wrt = 0.00001
```

```
#activate history

[activate_history]

activate = 1

history=30
```

The changes can be shown in Figure 5. 19.After loading the CPU RAM usage after certain period of time battery percentage drained to approx. 65.25%.

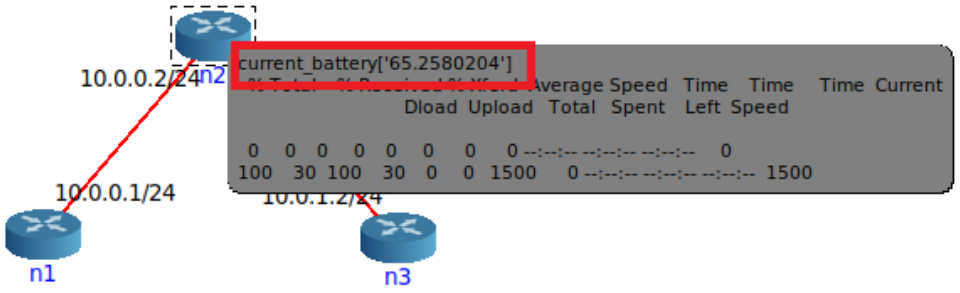


Figure 5. 19: Drained battery due to load in CPU and RAM usage

A graph can be shown Figure 5. 20 with values taken from `db.json`. After loading CPU and RAM the battery percentage drops drastically and with time battery draining become stable. The reason behind this is, CPU RAM load does not change with the time using `stress` command. As there is no other impact on consumption, therefore at some point it gets stable.

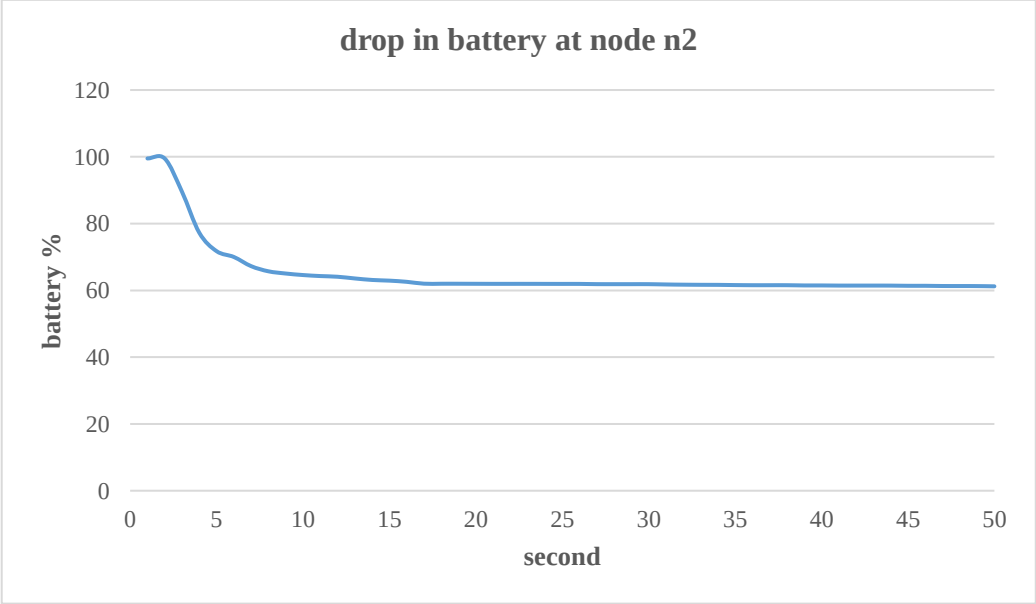


Figure 5. 20: Drop in battery percentage after loading CPU RAM

Simulation with loaded disk

This simulation has been tested in node n3 of Figure 5. 18. To cause load in hard disk below command can be used.

`$stress -d 2` (loading two process with Disk Read Write bytes)

And below configuration has been given in `config.ini` from service option of CORE node.

`Config.ini`

```
#1000 bytes consume X% battery
[rx_tx]
rx = 0.00001
tx = 0.00001
#1% cpu/ram consumes X% battery
[cpu_ram]
cpu = 0.00001
ram = 0.00001
#1000 bytes read write consume X% battery
[disk_rd_wrt]
disk_rd = 0.01
disk_wrt = 0.01
#activate history
[activate_history]
activate = 1
history=30
```

After loading the Disk battery percentage drops as shown in Figure 5. 21

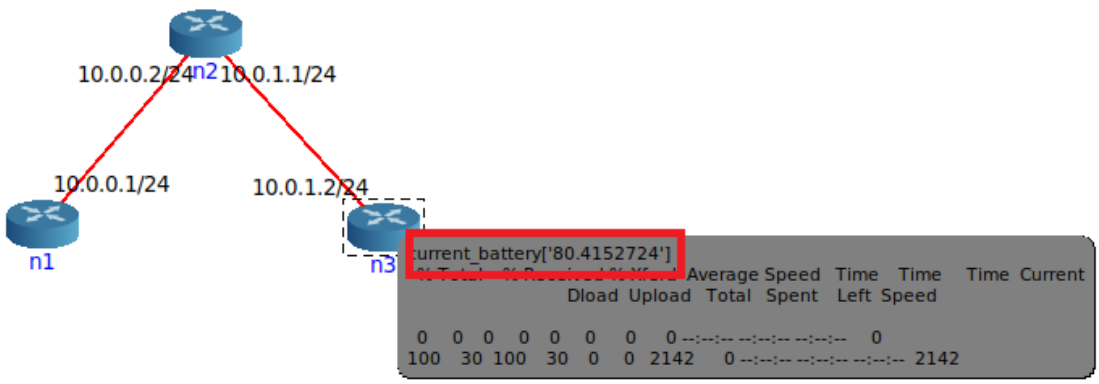


Figure 5. 21: Drop in battery percentage after loading disk

A graph can be shown in after stressing the load on node n3. It can be seen that after loading the disk battery percentage gradually drops to 0 %. As stressing disk increases the load with time, so the battery drops gradually.

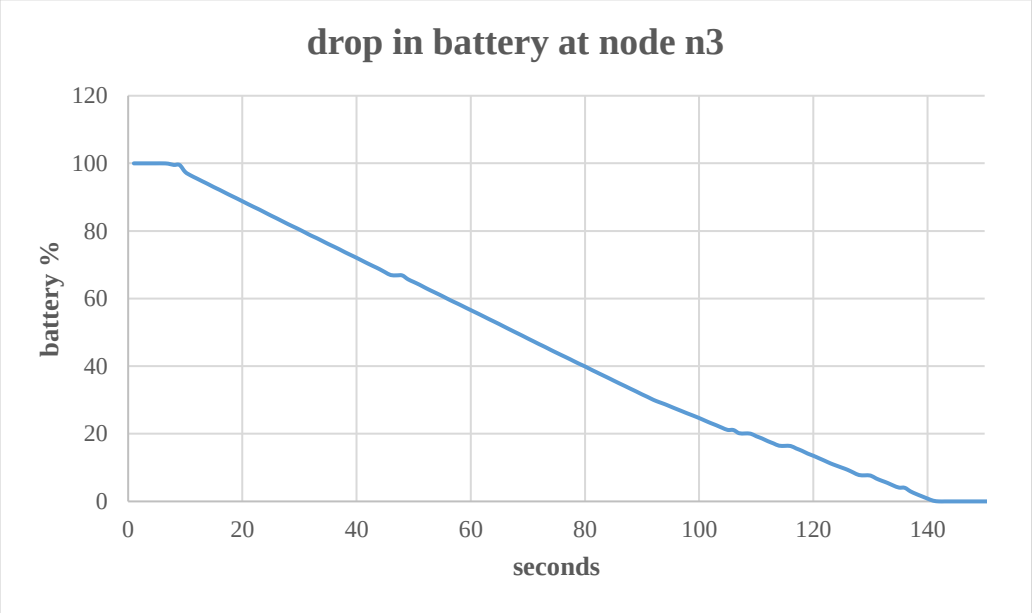


Figure 5. 22: Drop in battery percentage after loading Disk

5.3 Determination of Real Values

In this development task the values that have been used to measure battery percentage was an assumption. In this section some real values will be taken to calculate battery consumption. In real life scenarios consumption is used with milliampere(mA) unit or Watt. This developed application will be used to calculate the battery percentage by these parameters.

In this case, Raspberry Pi has been taken in consideration to take some values for calculation. The values are approximate and have been taken from some references.

Formula that has been used for RX/TX to convert watt to milliampere is:

$$\mathbf{mA=(W\times 1000)\div V}$$

As example, Rx =0.00014W, voltage capacity=5.7 V (assumption voltage)

$$\mathbf{mA= (0.0000017\times 1000)\div 5.7=0.00029\ mA}$$

TX calculation has been done following the same procedure. RX/TX values have been taken from a paper called 'Energy Models for NFV and Service Provisioning on Fog Nodes' (Kaup et al,2018).

Other parameter value in W and mA are considered according to Raspberry Pi 2B benchmark (Raspberry Pi,2019).

Here values are taken as:

- RX/TX consumption per 1000 bytes
- CPU/RAM consumption per 1% usage
- Disk read/write consumption per 1000 bytes.

The considered values are given in below Table 6.2.

Table 5.2: Approximate consumption of parameters in W and mA

Parameters	Hardware Specifications	Values in W	Values in mA
RX	Raspberry Pi 2B	0.0000017 W	0.00029
TX	Raspberry Pi 2B	0.0000025	0.00028
CPU	Raspberry Pi 2B	0.021	1
RAM	Raspberry Pi (DDR1 RAM,2GB)	0.0023	0.45
Disk Read	Raspberry Pi (USB 64GB SSD)	0.00000008593	0.0000171875
Disk Write	Raspberry Pi (USB 64GB SSD)	0.00000008593	0.0000171875

In section 4.4.3 it is already mentioned how the development procedure has been done to calculate the mA values. And for calculation Battery Power Specification is also specified in `config.ini`.

To test this given scenario in below Figure 5. 23 has been used.

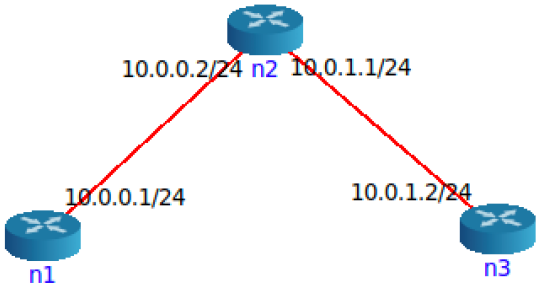


Figure 5. 23: Test network scenario for mA

To do this calculation Battery Specification has to be used, that means every battery has a capacity and it is denoted with Voltage and mAh or Wh basis. mAh or Wh is the highest capacity that load can consume. So, the higher the load, the lower the capacity.

Consumption in percentage has been calculated for node n1(Figure:5.18) from given mA configuration in `config.ini`. Battery capacity is assumed here: Voltage=5.7-volt, total capacity=1200mAh. Here, RX/TX consumption is calculated per 1000 bytes, for example 1000 bytes RX consume 0.00553792 mA and in % specification it is 0.00046149333 %. CPU/RAM consumption is calculated per 1% usage and Disk Read/Write consumption is calculated per 1000 bytes. In a nutshell, it is developed following the procedure discussed in section 4.4.3.Consumption by parameter is calculated with the formula is given below:

Consumption by parameter= (current consumption by parameter÷ total capacity)×100

```
root@n1:/home/servercore2/Desktop# python data_processor.py
('battery_voltage: ', 5.7)
('total capacity', 1200.0)
('consumption in mA-rx', 0.005537920000000001, 'consumption % rx:', 0.00046149333333333337)
('consumption in mA-tx', 0.0026706399999999997, 'consumption %-tx:', 0.00022255333333333333)
('consumption in mA-cpu', 47.0, 'consumption %- cpu:', 3.916666666666667)
('consumption in mA-ram', 0.765, 'consumption %-ram:', 0.06375)
('consumption in ma-disk rd', 6.7944851e-09, 'consumption %-read:', 5.662070916666667e-10)
('consumption_in_ma-disk wrt:', 1.2031250000000002e-09, 'consumption % -wrt:', 1.0026041666666669e-10)
('battery remaining percentage: ', 96.01889862019915)
```

Figure 5. 24: Consumption in percentage from mA

To have a more clear visualization a table can be given with the mA and % values that have been calculated. In Table 5.3 the values that have been used in `config.ini` to calculate consumption in mA, consumption in mA after calculation and consumption in % after calculation is given.

Table:5.3: Calculated consumption in mA and % from mA values in config.ini

Parameters	Config.ini(mA)	Consumption in mA	Consumption in %
RX	0.00034	0.0053792	0.000461493
TX	0.00028	0.002670639	0.0002225533
CPU	1	47	3.9167
RAM	0.45	0.765	0.06375
Disk Read	0.0000171875	0.0000000068	0.00000000057
Disk Write	0.0000171875	0.0000000012	0.0000000001

Therefore, *Energy Consumption Simulator* application is capable to calculate the consumption in mA and in % depending on the user definable configuration values given in `config.ini`.

6 Summary and Perspective

This chapter describes a summary of the implementation and some proposal of the future perspectives to extend the existing application and suggestions for improvements.

6.1 Summary

In this thesis to design and develop *Energy Consumption Simulator for the CORE Network* four hardware components: Network, CPU, RAM, Hard Disk have been taken as consideration which acted as energy consumption factor. More precisely, RX/TX bytes, CPU/RAM usage, Disk Read/Write bytes, these six parameters have been taken in consideration. Before developing the application some of the Linux commands have been evaluated. After the evaluation some commands have been chosen to complete the development task.

For developing the application python 2.7 has been used. For http client server approach *Flask* framework structure has been chosen. To realise the design of the application five components have been developed: *Data Aggregator, Data Processor, Client Script Executor, Consumption History, Battery Emulator (http server),*

For handling configuration file of parameters ConfigParser module of python has been used in this thesis task. *Tiny DB* has been used to store history which facilitates user with summary of history with given point of time interval.

And after whole development, it was integrated in CORE as a service. If this service is selected it can simulate energy of every single node individually depending on configuration.

User can get the current battery and total consumption through http request from corresponding node and outside of CORE through initialising control network. It can also be retrieved hovering on the Widget of CORE GUI. It is also possible to change the parameters while server is running, user can also retrieve summary of history through http request. This application also facilitates the user to get consumption with mA configuration.

Finally, in Chapter 5, usage of the application has been visualised with some network scenarios and impact of the application have been shown on network routes. Some simulation scenarios have been introduced in this chapter to visualise the changes in battery percentage due to change in parameter usage.

Though at the beginning of this thesis, python 3 version has been used to develop the application but later on shifted to python 2.7 as CORE itself is based on python 2.7 version.

6.2 Future Perspective

During this thesis task configuration parameters have been chosen from assumptions and later on analysed with some real values from Raspberry Pi taken as consideration. It will be interesting if real values of energy consumption parameters can be measured for CORE emulator and then consumption will be much more accurate.

At that moment user can run three scripts at three different battery levels, in future it will be beneficial to make it dynamic that user can initialise any script at any battery level.

This application does not include the facility to measure energy consumption in Wireless Meshed Network. To retrieve RX /TX values of a real WiFi interface is required to use *iw* Linux tool (Michel,2029). It will be interesting to introduce this aspect to the development.

In this development, the parameters in configuration are defined such that it increases the CPU consumption in linear way. As an example, if 1% CPU consumes 1% battery then 100% CPU consumes 100 mW. That means it will be linear.

But in real life it is not linear as shown in Figure 6.2. But during our development this have not been considered during the calculation. Below Figures 6.1,6.2 can be depict the scenario:

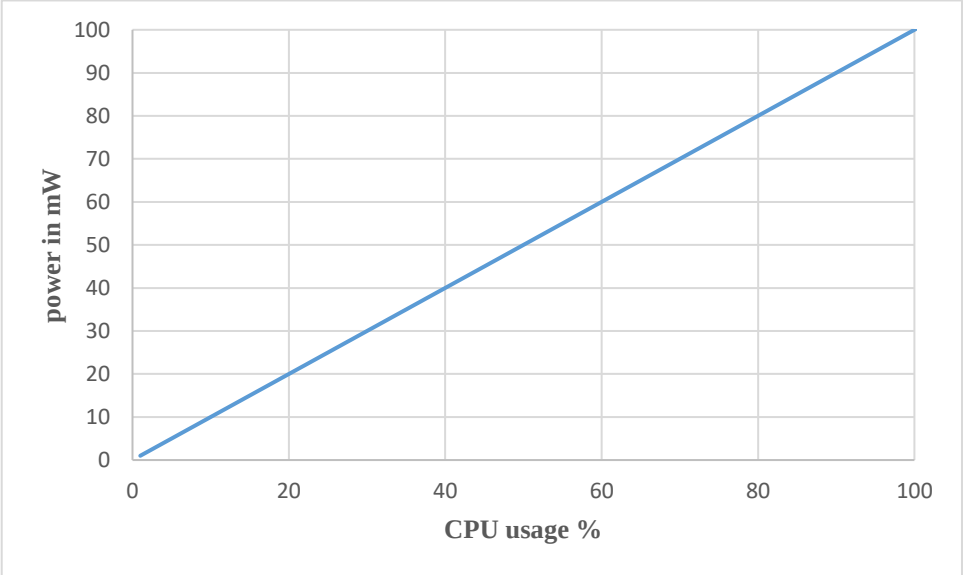


Figure 6.1: Consumption by developed application (assuming 1% CPU consuming 1 mW power)

In real life increase in consumption is not linear as shown in Figure 6.2. With accurate measurement and further research, it will be interesting to know the reason behind this difference.

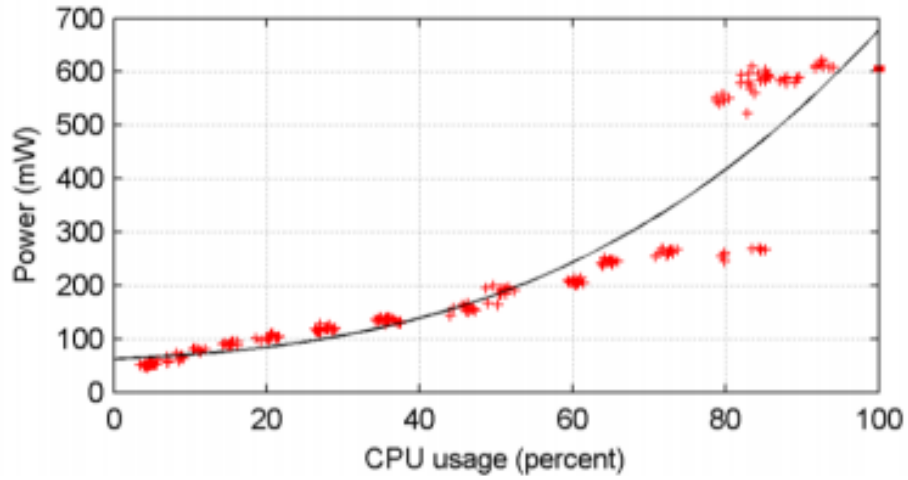


Figure 6.2: Power consumption in Real life scenario (Kennedy et al,2015)

7 Abbreviations

C

CLI	Command Line Interface
CORE	Common Open Research Emulator
CPU	Central Processing Unit

G

GUI	Graphical User Interface

H

HTTP	HyperText Transfer Protocol

I

IP	Internet Protocol
ICMP	Internet Control Message Protocol

M

MAC	Media Access Control
mA	Milliampere

O

OS	Operating System
OSPF	Open Shortest Path First

P

PC	Personal Computer

R

RAM	Random Access Memory
RX	Receive

T

TCP	Transmission Control Protocol
TX	Transmit

U

UI	User Interface

URI	Uniform Resource Identifier
V	
VM	Virtual Machine

8 References

1. Bleeping Computer LLC(2019):*When Computer is Idle,* https://www.bleepingcomputer.com/forums/t/633314/when-computer-is-idle-what-should-be-a-normal-cpu-usage/[accessed 12 February 2019].
2. Canonical LTD (2019):*Download Ubuntu Desktop*, https://ubuntu.com/download/desktop[accessed 14 January 2019]
3. Canonical LTD(2019a):*Ubuntu Manuals,* http://manpages.ubuntu.com/manpages/trusty/man8/ifconfig.8.html[accessed 25 January 2019].
4. Canonical LTD(2019b):*Ubuntu Manuals,* http://manpages.ubuntu.com/manpages/trusty/man8/iplink.8.html[accessed 25 January 2019].
5. Canonical LTD(2019c):*Ubuntu Manuals,* http://manpages.ubuntu.com/manpages/xenial/man5/proc.5.html[accessed 25 January 2019].
6. Canonical LTD(2019d):*Ubuntu Manuals*,http://manpages.ubuntu.com/manpages/xenial/man1/ps.1.html[accessed 27 January 2019].
7. Canonical LTD(2019e):*Ubuntu Manuals*, http://manpages.ubuntu.com/manpages/trusty/man1/sar.sysstat.1.html[accessed 25 January 2019].
8. Canonical LTD(2019f):*Ubuntu Manuals*, http://manpages.ubuntu.com/manpages/xenial/man8/vmstat.8.html[accessed 25 January 2019].
9. Canonical LTD(2019g):*Ubuntu Manuals*, http://manpages.ubuntu.com/manpages/cosmic/man8/ping.8.html[accessed 15 April 2019].
10. Carbajal, C.(2013):*Threading Programming Using Python,* http://homepage.cem.itesm.mx/carbajal/EmbeddedSystems/SLIDES/Python/Threading%20Programming%20using%20Python.pdf[accessed 5 March 2019].
11. Carroll, A. and Heiser, G.(2010):Analysis of power consumption in smartphone*, Proceedings of the 2010 USENIX conference,* [online] Available at https://dl.acm.org/citation.cfm?id=1855861[accessed 5 February 2019].
12. Ciliendo, E. and Kunimasa, T.(2007*):Linux Performance and Tuning Guidelines* https://lenovopress.com/redp4285.pdf[accessed 2 February 2019].
13. Code Yarns(2016):*Free Command in Linux*, https://codeyarns.com/2016/06/15/free-command-in-linux/[accessed 25 January 2019].
14. Computer Hope(2019):*Linux iostat command,* https://www.computerhope.com/unix/iostat.htm[accessed 28 January 2019].
15. CORE (2015): *CORE Documentation Release 4.8,* https://downloads.pf.itd.nrl.navy.mil/docs/core/core_manual.pdf *[accessed 2 February 2019].*
16. Kaup, F.et al(2018):Energy models for NFV and service provisioning on fog nodes,*2018 IEEE/IFIP Network Operations and Management Symposium,* pp 4-5, [online] Available at https://ieeexplore.ieee.org/document/8406158accessed (15 May 2019].
17. Kennedy, M. et al(2015): *Energy Consumption Analysis and Adaptive Energy Saving Solutions for Mobile Device Applications,* https://www.researchgate.net/publication/265819066_Energy_Consumption_Analysis_and_Adaptive_Energy_Saving_Solutions_for_Mobile_Device_Applications([accessed 24th March 2019].)
18. Linkletter,B.(2017):*Open-Source Routing and Network Simulation*, https://www.brianlinkletter.com/install-the-core-network-emulator-from-source-code/ [accessed 14 January 2019].
19. Maria, A.(1997): Introduction to Modeling and Simulation , *Proceedings - Winter Simulation Conference*, pp 12-13, [online] Available at https://dl.acm.org/citation.cfm?id=268440[accessed 20 January 2019]
20. Menner,W.(1995): *Introduction to Modeling and Simulation,*https://www.jhuapl.edu/techdigest/views/pdfs/V16_N1_1995/V16_N1_1995_Menner.pdf[accessed 1st March 2].
21. Michel,P.(2019):*Linux Wireless*, https://wireless.wiki.kernel.org/en/users/documentation/iw[accessed 25 June 2019]
22. Oracle (2016): *VirtualBox*, https://www.virtualbox.org/[accessed 14 January 2019].

23. Procaccianti,G. et al (2011):*Profiling Power Consumption Desktop Computer System,* https://www.researchgate.net/publication/235732703_Profiling_Power_Consumption_on_Desktop_Computer_Systems[accessed 2 February 2019].
24. Python Software Foundation(2019a):*Psutil 5.6.3*, https://pypi.org/project/psutil/[accessed 1 March 2019].
25. Python Software Foundation(2019b):*Os:Miscelleaneous System Interfaces*, https://docs.python.org/3.4/library/os.html[accessed 6 March 2019].
26. Python Software Foundation(2019c): *Flask 1.0.3*, https://pypi.org/search/?q=flask [accessed 1 March 2019].
27. Quora(2015): *How battery percentage is calculated in laptops or mobile,* https://www.quora.com/How-battery-percentage-is-calculated-in-laptops-or-mobile[accessed 7 May 2019].
28. Rodola, G.(2019): *Psutil 5.6.2* https://pypi.org/project/psutil/ [accessed 3rd February 2019].
29. ScienceDirect(2019):*Battery Capacity*, https://www.sciencedirect.com/topics/engineering/battery-capacity[accessed 15 February 2019].
30. Siemens, M.(2016): *TinyDB*, https://tinydb.readthedocs.io/en/latest/usage.html [accessed 6 March 2019].
31. TechConsumer(2018*): How to Estimate the Battery Life of Any Laptop Using This Formula,* https://www.techconsumerguide.com/how-to-estimate-laptop-battery-life/[accessed 15 February 2019].
32. Tecmint(2019):*Dstat-A Resourceful Tool to Monitor Linux Server Performance in Real-Time,* https://www.tecmint.com/dstat-monitor-linux-server-performance-process-memory-network/ [accessed 2 February 2019].
33. The Linux Foundation (2019): *Linux man page,* https://linux.die.net/man/1/pidstat [accessed 27 January 2019]
34. TutorialsPoint(2019):*Python Pandas- Introduction,* https://www.tutorialspoint.com/python_pandas/python_pandas_introduction.htm[accessed 1 March 2019].
35. White, P. and Ricki, I. (2009): Introduction to Simulation, *Proceedings - Winter Simulation Conference*, pp 12-13, [online] Available at https://www.researchgate.net/publication/221529490_Introduction_to_Simulation [accessed 20 January 2019]

9 Appendix

A CD in attachment which contain following

1. This thesis in PDF format
2. This thesis in doc format
3. Source code