



Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Scalable Computing CS7NS1

Project 3 – Software Defined Networking

Ernests Kuznecovs, Kaaviya Karunanidhi, Francis Long

Wangying Jiang

School of Computer Science and Statistics

Trinity College,

College Green, Dublin 2.

MSc Computer Science

QQI Qualification Level 9

November 2021

Table of Contents

Introduction	3
Design	5
Execution	8
Results	10
Conclusions	11
Problems and Workarounds	12
Future Work	13
References	14

Introduction

This assignment seeks to design and implement a sufficient peer-to-peer networking protocol, with mandatory Software Defined Networking and Network Function Virtualisation principles and implementation at its core.

The proposed Software Defined Architecture is modelled around a Smart Home Application consisting of numerous Internet of Things Devices. The devices communicate through the network interface using two Raspberry Pi's. The Raspberry Pi's collect data from virtual sensors. This data is collected and presented using http network programming wrapped in a python shell. A centralised network is proposed, which will control and adjust the devices and form a scalable environment. The environment is designed to be powerful, easy to manage, and flexible. Software defined networking allows a network administrator to quickly and easily manage network services from centralized locations without the need to configure routers or switches manually. It offers flexibility, scalability and efficiency

The virtual network contains virtual devices which in turn offer a range of sensor data to the server. Each network and device has been configured to emulate scalable entities and network functionalities. Moreover the application contains a database to which sensor data is sent and stored. This information can be called by any of the nodes and retrieved by any of the devices on the network.

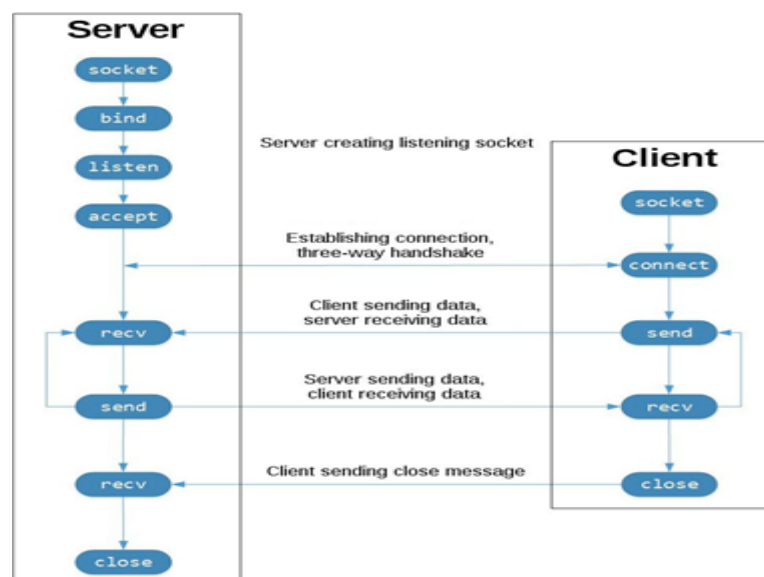


Fig 1: API TCP calls

The graphic displays the operation of a server client relationship featuring four key steps to a successful connection between server and client. These are socket, bind, listen and accept. The http API implemented for the project has a similar function and

connections are created and maintained using a designated port. (In the design stage this port was deemed to be 8001, however the port designation was changed to 33001 on the Raspberry Pi in order to sidestep a firewall rule.)

Design

The core network design involves the use of two college administered Raspberry Pi devices onto which is loaded the python package named “sensible.” Sensible is available as a github package at the following online address. <https://github.com/ErnestKz/sensible>

There are a range of sensor devices implemented. These include:

- Temperature Sensor
- Motion Sensor
- Face Detection Sensor
- Light Sensor
- Gas level
- Humidity
- Motion
- PM2.5 (particulate matter 2.5 nanometres in diameter)
- Water temp
- Carbon dioxide
- Door state
- Window State
- Timestamp
- User ID

Key Features: Sensor Data Lock and Thread

A key feature of the design is that sensors can be easily added or removed. This offers a degree of scalability to facilitate the growth of the virtual network. Moreover the number of nodes can be easily configured to provide maximum scalability and ease of configuration. Another key feature is that each sensor has its own lock and its own thread. This was decided in order to safeguard the data from corruption during the reading and writing stages. Using the lock and the separate sensor only one operation can be completed at any one time. A random sleep interval is added to each sensor in order to avoid writing data from different sensors at the same time. This key feature is responsible for the superior stability of the platform which performs to a high degree of reliability.

Key Features: Functional Programming

As a positive training exercise it was decided to code the project using the principles of Functional Programming as promoted by Richard Bird in his book “*Thinking Functionally with Haskell*.” Using these principles greater efficiencies can be created to improve the program workflow. Functional programming is explained in greater detail below.

Raspberry Pi Devices

Two college administered Raspberry Pi devices were loaned to the group for testing. These were device number 25 and device number 26. The Raspberry Pi devices can be accessed by an ssh command into the TCD School of Computer Science server at macneill.scss.tcd.ie. Then a pivot ssh command is issued to log into the Pi devices with the network address rasp-025.berry.scss.tcd.ie and rasp-026.berry.scss.tcd.ie. Active port numbers on the Pi devices start at 30001.

Platform

The python package was tested on Microsoft Windows platform and on a 64 bit Ubuntu Linux 20 platform installed with python3. The program performance was stable on each platform with no environmental issues resulting.

Control Plane

The Control Plane and the Data Plane are on two separate virtual layers. Control information is propagated to the Raspberry Pis on one layer and sensor data is propagated to the Pi's on another. Control signals determine which Pis are actively accepting data. The Control Plane is used to request all node data and transmit control signals to route data to individual Raspberry Pi devices. The Software Defined Network architecture represents fast and easy management to regulate network services from a centralised location. This avoids the necessity of manual configuration of a router or switch, a measure which increases efficiency in the design.

Network Function

Receivers can be blocked using a network function designed to protect data from corruption.

Control Lock

The lock prevents reading and writing on the same interface where the sensor is defined.

User Interface

A user interface was implemented to select devices and examine the sensor log from each one. This can be displayed in a range of detail including “all” or “device_summary”. Data from the local device or a network device can be selected and displayed. The user interface brings a deceptively easy management feature to control the complexity of the peer to peer network and retrieve data from it.

Database

A database (Structured Query Language Lite) was also created to store and process the sensor data and minimise the risk of data loss or corruption. This data analysis is based on a network of sensors feeding data to the nodes. In an Internet of Things scenario each small device can store information and export it if necessary. Each device can store its data on an SQLite database. This is suitable for small devices because it enables concurrent reads and writes rapidly. Moreover SQLite is suitable for Embedded and IoT devices.

Execution

Functional Programming

A key feature of the programming technique employed in the successful completion of this project was Functional Programming. Using this technique defined functions can be contained and called from within other functions. This creates a degree of efficiency which has a knock on benefit of reducing CPU cycles.

```
10 def create_virtual_sensor(sleep_time, sensor_name, run_sampler):
11     def run_virtual_sensor(lock, data, deviceAddress):
12         while True:
13             sampled_data = run_sampler()
14             timeStamp = time.time()
15
```

Fig 2: Functional Programming.

In the example reproduced above a function (run_virtual_sensor) is contained within a function (create_virtual_sensor). The purpose of the exercise is to generate data from a single sensor inside the auspices of the create_virtual_sensor function. This practice avoids the need for a loop to create episodes of data. Using functional programming the data is generated in tandem with a lock that adds a sleep timer which is designed to separate the reading and writing processes.

Key Features

- A http server (Pi_1) listens for incoming requests and handles them.
- A client (Pi_2) asks for data sent to other Pi's on the network.
- Each request has a timestamp which is used to check for old data.
- Only data created after the timestamp is sent to the server.
- The data from virtual sensors is randomised.
- The application includes a facility to add genuine sensors easily.
- The client loop continues indefinitely.
- A range of ports is checked for active connections and incoming data streams.
- Each device has its own sensors.
- Broadcasts connect the devices to one another.
- Each device has information about all the other devices.
- Each device recognises all the other devices.
- Each sensor has its own thread and lock. The thread sleeps while idle.
- Each sensor has a flexible sampling rate.
- Threading is implemented on each port that is being read.

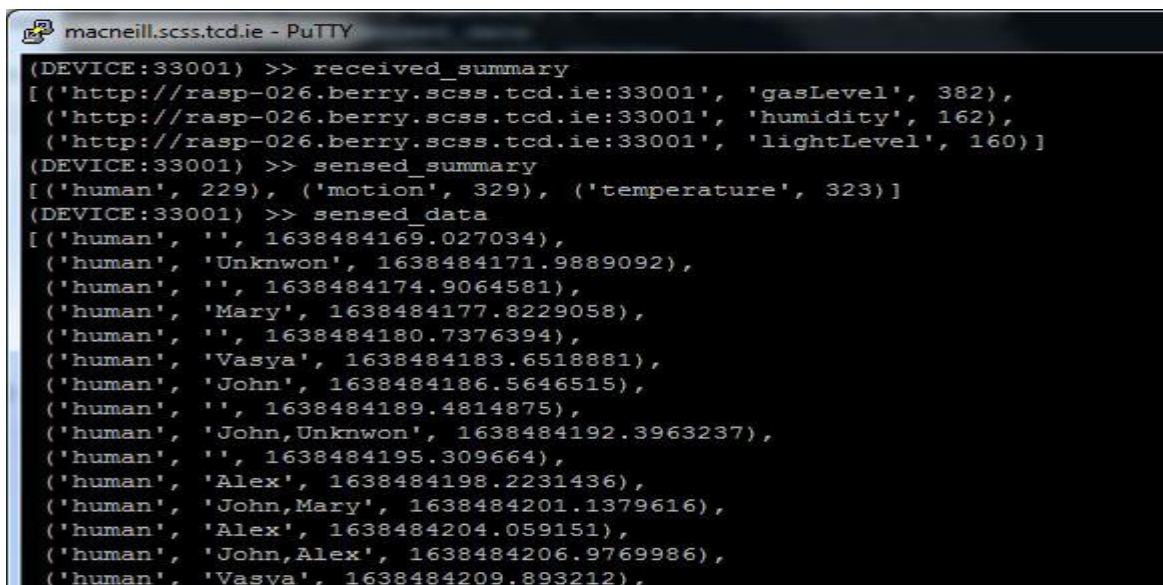
The Software Defined Network is activated from the control plane. The virtual sensors on the network are entirely defined through software. The Network Function is also Virtualised.

Error Handling

Exceptions and warnings are generated for 'incorrect data format', 'no devices listening' or timeouts.

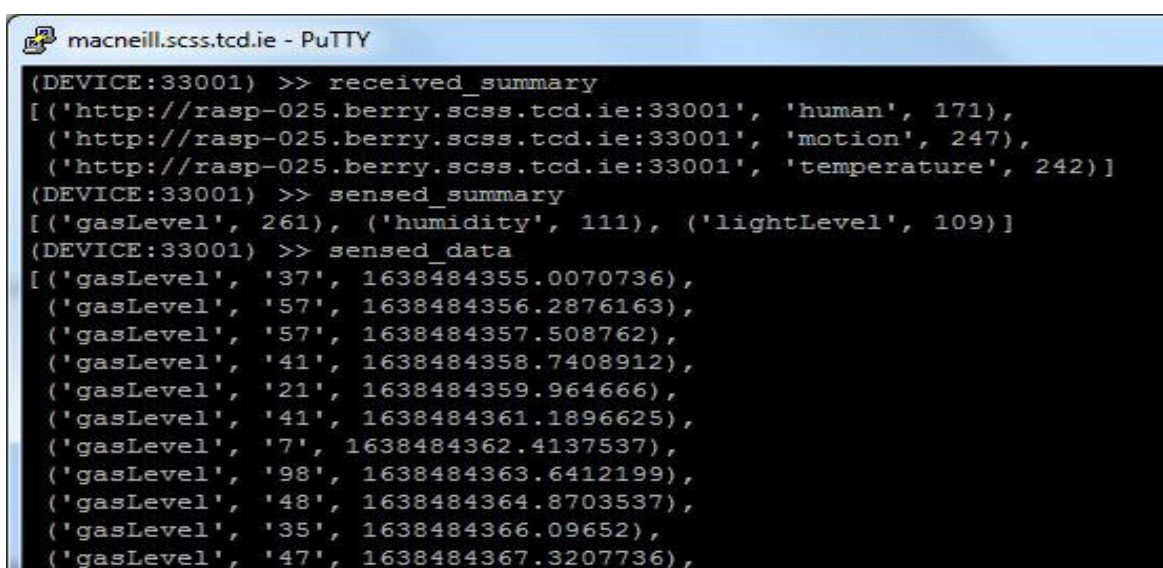
Usage:

- 1) Create data folder in sensible folder
- 2) python3 -m sensible pi25 or python3 __main__.py pi25

A screenshot of a terminal window titled 'macneill.scss.tcd.ie - PuTTY'. The terminal shows a series of commands and their outputs. The first command is 'received_summary', which returns a list of three tuples: ('http://rasp-026.berry.scss.tcd.ie:33001', 'gasLevel', 382), ('http://rasp-026.berry.scss.tcd.ie:33001', 'humidity', 162), and ('http://rasp-026.berry.scss.tcd.ie:33001', 'lightLevel', 160). The second command is 'sensed_summary', which returns a list of three tuples: ('human', 229), ('motion', 329), and ('temperature', 323). The third command is 'sensed_data', which returns a list of 18 tuples, each containing a name, a status, and a numerical value. The names include 'human', 'Unknwon', 'Mary', 'Vasya', 'John', and 'John,Unknwon'.

```
(DEVICE:33001) >> received_summary
[('http://rasp-026.berry.scss.tcd.ie:33001', 'gasLevel', 382),
 ('http://rasp-026.berry.scss.tcd.ie:33001', 'humidity', 162),
 ('http://rasp-026.berry.scss.tcd.ie:33001', 'lightLevel', 160)]
(DEVICE:33001) >> sensed_summary
[('human', 229), ('motion', 329), ('temperature', 323)]
(DEVICE:33001) >> sensed_data
[('human', '', 1638484169.027034),
 ('human', 'Unknwon', 1638484171.9889092),
 ('human', '', 1638484174.9064581),
 ('human', 'Mary', 1638484177.8229058),
 ('human', '', 1638484180.7376394),
 ('human', 'Vasya', 1638484183.6518881),
 ('human', 'John', 1638484186.5646515),
 ('human', '', 1638484189.4814875),
 ('human', 'John,Unknwon', 1638484192.3963237),
 ('human', '', 1638484195.3096664),
 ('human', 'Alex', 1638484198.2231436),
 ('human', 'John,Mary', 1638484201.1379616),
 ('human', 'Alex', 1638484204.059151),
 ('human', 'John,Alex', 1638484206.9769986),
 ('human', 'Vasya', 1638484209.893212),
```

Fig 3 Sample Data in Pi25 received from Pi26.

A screenshot of a terminal window titled 'macneill.scss.tcd.ie - PuTTY'. The terminal shows a series of commands and their outputs. The first command is 'received_summary', which returns a list of three tuples: ('http://rasp-025.berry.scss.tcd.ie:33001', 'human', 171), ('http://rasp-025.berry.scss.tcd.ie:33001', 'motion', 247), and ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', 242). The second command is 'sensed_summary', which returns a list of three tuples: ('gasLevel', 261), ('humidity', 111), and ('lightLevel', 109). The third command is 'sensed_data', which returns a list of 14 tuples, each containing a name, a status, and a numerical value. The names include 'gasLevel' and 'gasLevel' followed by a number.

```
(DEVICE:33001) >> received_summary
[('http://rasp-025.berry.scss.tcd.ie:33001', 'human', 171),
 ('http://rasp-025.berry.scss.tcd.ie:33001', 'motion', 247),
 ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', 242)]
(DEVICE:33001) >> sensed_summary
[('gasLevel', 261), ('humidity', 111), ('lightLevel', 109)]
(DEVICE:33001) >> sensed_data
[('gasLevel', '37', 1638484355.0070736),
 ('gasLevel', '57', 1638484356.2876163),
 ('gasLevel', '57', 1638484357.508762),
 ('gasLevel', '41', 1638484358.7408912),
 ('gasLevel', '21', 1638484359.964666),
 ('gasLevel', '41', 1638484361.1896625),
 ('gasLevel', '7', 1638484362.4137537),
 ('gasLevel', '98', 1638484363.6412199),
 ('gasLevel', '48', 1638484364.8703537),
 ('gasLevel', '35', 1638484366.09652),
 ('gasLevel', '47', 1638484367.3207736),
```

Fig 4: Sample data in Pi 26 received from Pi25.

Results

- Virtual devices were easily added and interconnected demonstrating the scalability of the solution.
- Threads were successfully implemented on the sensors in combination with a sensor data lock to safeguard data from read write conflicts and race conditions that could corrupt the data. A sleep timer randomised to each sensor avoids collisions.
- A simple interface for configuring virtual and actual sensors was successfully configured. (Interface commands include 'device', 'all', 'clear', 'exit', 'received_summary', 'sensed_summary', 'sensed_data' and 'exit'. Output is sent to the console to enable monitoring.
- The Server Log and Client Log - prints to the console
- A Control Plane was defined and implemented to define the network structure. The Control Plane samples at a different rate to the data plane, separating the two SDN Layers.
- Each small device can store sensor information and transmit it to a database running SQL Lite..
- Each device stores information regarding the active port range. The control plane can update the port range.
- Automation – a useful script was developed to download and run python code directly to the Raspberry Pi devices. This bypasses privilege issues on the devices.
- A fully operational Application programming Interface was developed to integrate device hardware into software. Through this interface the integration of real or virtual devices was made easier.
- A port range scanner was deployed to listen for connected devices and receive data.

Conclusions

Group 13 successfully designed, implemented and demonstrated a working model of a Software Defined Networking instance that satisfied the objectives of project 3. A separate control and data plane was implemented. The peer to peer design demonstrates scalability, flexibility and Network Function Virtualisation. The use case of an Internet of Things environment was used to successfully develop network nodes and network sensors to operate together.

Hardware and software sensors can be easily connected or removed for enhanced flexibility.

The application enables continuous monitoring of the data in a log file sent to a database. A generous amount of data is produced.

```
/rasp-025.berry.scss.tcd.ie:33001', 'temperature', '19', 1638485829.4631162), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '20', 1638485831.5313416), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '16', 1638485833.599103), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '21', 1638485835.6673253), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '9', 1638485837.7325168), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '23', 1638485839.8001604), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '23', 1638485841.8699284), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '15', 1638485843.935451), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '22', 1638485846.0019133), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '18', 1638485848.0709693), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '23', 1638485850.1389642), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '18', 1638485852.2052145), ('http://rasp-025.berry.scss.tcd.ie:33001', 'temperature', '7', 1638485854.2747078)]
```

=====

```
clientLog: ['Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200',  
    , 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to:  
: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://r  
asp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.be  
rry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.t  
cd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:3300  
1, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 20  
0', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent  
to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://  
/rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.berry.scss.tcd.ie:33001, code 200', 'Sent to: http://rasp-025.
```

Fig 5: Sample Output from “all” command on Pi26

The use cases for this application include a warehouse environment or installation containing multiple Internet of Things devices. The Sensor Data generated has applications in predictive behaviour to calculate footfall or frequency traffic for a commercial enterprise.

Problems and Workarounds

An additional sensor device that would offer a timestamp was not implemented because of a firewall rule on the Raspberry Pi devices which prevented a network Time Protocol (NTP) connection to timeout.org from which the timestamps were to be obtained.

A portion of the project time was spent researching OpenFlow and Mininet solutions which were later deemed unnecessary.

Time constraints precluded the extra features detailed in the section on “Further Work.”

Covid restrictions hampered the smooth workflow of the group as communications were restricted to online platforms where the normal academic discourse was handicapped.

Future Work

It is anticipated that future versions may include a facial recognition element which could match a user's preferences and data trends to a particular person. In turn this information could be used in a predictive manner to tailor the home device settings to each user. This would avoid the necessity for the user's manual intervention and create flexibility through Machine Learning based on examination of the volumes of data generated. For example, the temperature setting of a building's water supply could be prepared in advance of a certain user's arrival. Other settings can be actuated based on data from other sensors. In this manner the information allows predictions to be made based on previous user behaviour and preferences.

The application is suited to an enterprise reporting system such as Splunk which could be used to create graphically rich representations of the data in a more human readable form.

References

Bird, R. (2014). *Thinking Functionally with Haskell*. Cambridge: Cambridge University Press. doi:10.1017/CBO9781316092415

Socket Programming: <https://realpython.com/python-sockets/>

https://commons.wikimedia.org/wiki/File:InternetSocketBasicDiagram_zhtw.png