NGN Project Proposal:

# Evaluation of ThingsBoard Implementation in the Context of an IoT Based Home Network

Submitted by:

Steffen Klein (11117177)

Rubaiya Kabir Pranti (11146364)

Laila Afreen (11142479)

## Table of Contents

## Abstract

This project aims to evaluate the performance of open source IoT platform, ThingsBoard on the basis of a simplified IoT based home device management network. To emulate a home device network, light bulbs and thermo hygrometer (to measure temperature and humidity), connected via Raspberry Pi and ESP32 will be operated and monitored using ThingsBoard dashboard. Tests will be carried out to gauge parameters such as latency of real time data, accuracy and responsiveness.

The project will also serve as a detailed study on communication protocols, MQTT and HTTP which will be employed for the purpose of the project.

## Objectives

**Specific Objectives:**

- Analyze the open source IoT platform, ThingsBoard to identify its advantages and limitations in relation to the characteristics desired for the execution of the project.
- Study of IoT communication protocols: MQTT, HTTP
- Study of functionality of devices, controllers, sensors and other components used in project
- Study of NoSQL databases for IoT: Cassandra
- ThingsBoard environment setup and produce inference based on operability
- Configure the platform: Users, rules & devices
- Practical implementation testing the communication of the IoT platform and associated protocols
- Validation of real-time data visualization tools and widgets provided by ThingsBoard

## Project Plan:

| PHASE - 1: Setting Environment | | | |
|---|---|---|---|
| **SL.** | **Task Name** | **Deadline** | **Person** |
| 1 | Software setup: ThingsBoard Installation | 25.04.21 | All |
| 2 | Hardware setup: Raspberry Pi & ESP 32 OTA | 06.05.21 | Steffen, Laila |

| PHASE - 2: System Setup | | | |
|---|---|---|---|
| **SL.** | **Task Name** | **Deadline** | **Person** |
| 1 | Connect client/controllers to ThingsBoard with connection troubleshooting | 16.05.21 | Steffen, Laila |
| 2 | Dashboard Integration Setup | 16.05.21 | Rubaiya |

| PHASE - 3: Implementation | | | |
|---|---|---|---|
| **SL.** | **Task Name** | **Deadline** | **Person** |
| 1 | Rule setup | 19.05.21 | All |
| 2 | Connectivity testing and data collection | 31.05.21 | Rubaiya |
| 3 | Functional testing and data collection | 31.05.21 | Rubaiya |
| 4 | Protocol testing and data collection | 09.06.21 | Rubaiya |

| PHASE - 4: Data Consolidation & Report Writing | | | |
|---|---|---|---|
| **SL.** | **Task Name** | **Deadline** | **Person** |
| 1 | Data Consoldation and evaluation of results | 17.06.21 | All |
| 2 | Report writing | 24.06.21 | Rubaiya,Laila |

# Introduction

## Impact of IoT

Over the past few years, IoT has become one of the most important technologies of the 21st century. Now that we can connect everyday objects—kitchen appliances, cars, thermostats, baby monitors—to the internet via embedded devices, seamless communication is possible between people, processes, and things.

All of this is has been made possible due to advancements such as:

- **Low-cost, low-power sensor technology**. Low cost and reliable sensors are making IoT technology available to everyone from for applications in a DIY project to industrial applications
- **Custom Connectivity Protocols**. A host of network protocols designed to support machine to machine and to user has made it easy to connect sensors to the cloud and to other "things" for efficient data transfer.
- **Cloud computing platforms**. The increase in the availability of cloud platforms enables both businesses and consumers to access the infrastructure they need to scale up without actually having to build it from scratch or having to manage it all.
- **Machine learning and analytics**. With advances in machine learning and analytics, along with access to varied and vast amounts of data stored in the cloud, businesses can gather insights faster and more easily.
- **Conversational artificial intelligence (AI).** Advances in neural networks have brought natural-language processing (NLP) to IoT devices (such as digital personal assistants Alexa, Cortana, and Siri) and made them appealing, affordable, and viable for home use.

Ref: https://www.oracle.com/internet-of-things/what-is-iot/

The scale of growth has been so immense that by the end of 2018, there was an estimated 22 billion internet of things (IoT) connected devices in use around the world. According to Statista, this number will be 38.6 billion by 2025, creating a massive web of interconnected devices.
(Ref.:https://findstack.com/internet-of-things-statistics/#:~:text=Internet%20of%20Things%20(IoT)%20emerged,and%2075.44%20billion%20by%202025.)

This proliferation of IoT applications has led to the availability of a number of IoT based platforms that allows individuals and organizations alike, to setup their own automation projects and modules. One such platform is the ThingsBoard. This is an open-source IoT platform that enables rapid development, management, and scaling of IoT projects. The platform supports both IoT cloud or on-premises solution that will enable server-side infrastructure for IoT applications. Thus a basic level home device management system will be implemented and operated to evaluate its capabilities in terms of performance, function and security.

## System Architecture

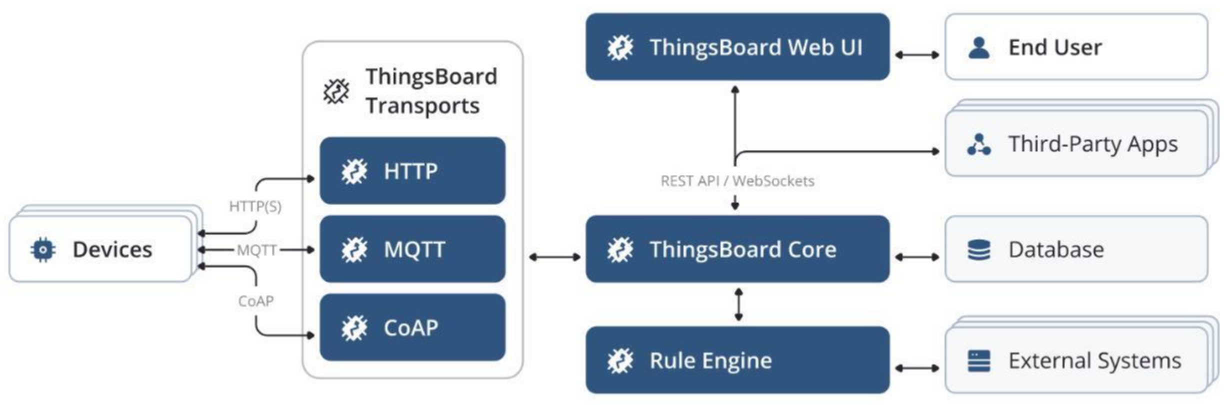


Figure 1: ThingsBoard Monolithic Architecture

1. **Devices**

   The home device management system comprises of devices which are IP enabled and can be controlled via ThingsBoard using the MQTT protocol on a dashboard. For the purpose of the project, two types of devices will be connected;

   - **Lighting Control Device** – signaling data is exchanged between device and ThingsBoard using RPC (Remote Procedure Calls) to remotely control lighting of bulbs attached. The state of the lights will be displayed to end user from the IoT platform.
   - **Temperature and Humidity Measurement Device** - telemetry data from this device will be collected and processed to display parameters such as heat index, humidity in percentage and temperature in degrees using external weather API.

   For both devices, users will receive notifications indicating success, failure, telemetry and errors in execution of remote functionality

2. **Transport Component**

   ThingsBoard Transport layer provides API for transport protocols, MQTT, CoAP and HTTP. Each of these APIs comes with their own server component. Here, each transport component pushes data to the rule engine.

3. **ThingsBoard Core**

   The core is responsible for handling:

   - REST API calls
   - WebSocket subscriptions on entity telemetry and attribute changes
   - Storing up to date information on active device sessions and monitoring device connectivity state (active/inactive).

   ThingsBoard Core uses Actor System under the hood to implement actors for main entities: tenants and devices.

4. **Rule Engine**

Incoming messages from devices connected to ThingsBoard is processed by user defined logic and flow set in the rule engine.

ThingsBoard Rule Engine operates in two modes: shared and isolated. In shared mode, rule engine process messages that belong to multiple tenants. In isolated mode Rule Engine may be configured to process messages for specific tenant only.

5. **External Systems**

   It is possible to push messages to external systems from ThingsBoard platform and to receive data in desired format via Rule Engine. In terms of the scope of the project, a REST API calls will be made to a weather app to derive temperature and humidity data and also to a sunrise-sunset app to derive time data for switching lights as per the time of the day.

6. **Database**

   All signaling and telemetry data will be stored in ThingsBoard's inbuilt NoSQL database,

   Cassandra. Due to its low latency and highly stable characteristics, this database is ideal for systems requiring high throughput or high availability.

7. **Third-Party Apps**

   Notifications to end users will be communicated via third party applications such as messaging apps (Eg: Telegram) and mailing service such as Gmail.

## Devices

To emulate a home device management system, two separate devices, one for controlling light and the other, for reading temperature and humidity is added to the ThingsBoard platform. Details regarding building the system are as follows:

### Lighting Control Device

This device is constructed to control lighting of our system via the GPIO (General Purpose Input/Output) on ThingsBoard as per the user’s preference. To enable this, the Raspberry Pi and relay board are separately programmed and then enabled the MQTT protocol on it. This will allow for communication between the Raspberry Pi device and ThingsBoard to demonstrate functionality.

#### *Components*

1. **Raspberry Pi 4 Model B** – A single board computer that utilizes a SD card inserted into a slot on the board for hard drive and is powered by USB. A Raspberry Pi has the ability to interact with external devices and environments using simple and intuitive languages such as Scratch and Python.

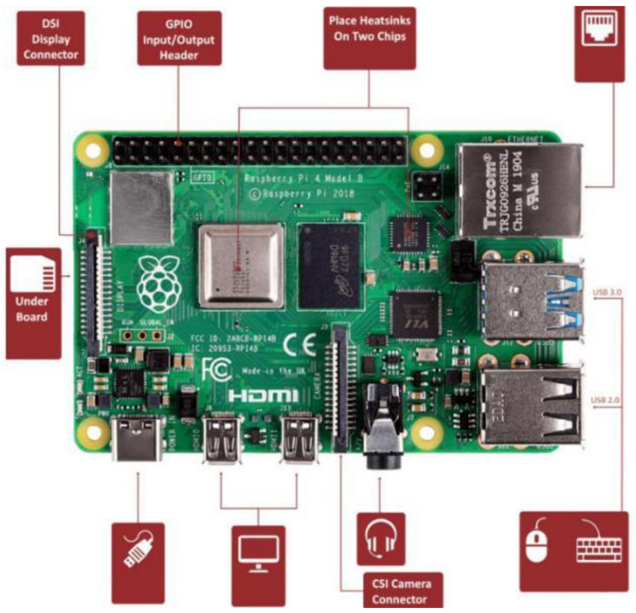


Figure 2: Raspberry Pi 4 Schematic

An important feature of the Raspberry Pi that is used to control and monitor external devices is the GPIO pins. Using these pins, a raspberry Pi can for instance, switch on or off lights, run motors and many other functions. A brief overview of pins relevant to the device management system is discussed below:

**Voltages**

Two 5V pins and two 3V3 pins are present on the board, as well as a number of ground pins (0V), which are unconfigurable. The remaining pins are all general purpose 3V3 pins, meaning outputs are set to 3V3 and inputs are 3V3-tolerant.

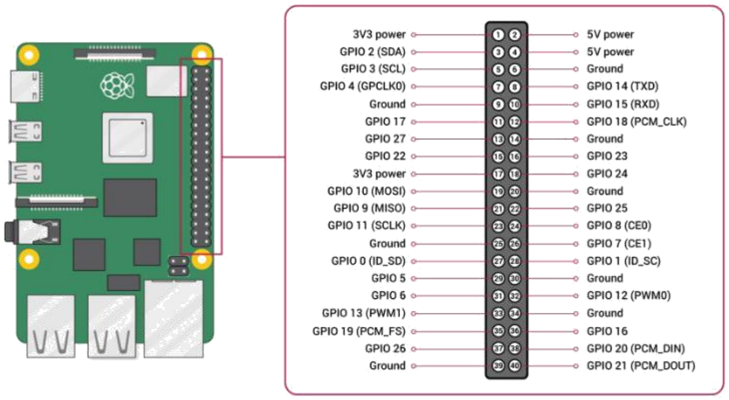


Figure 3: Raspberry Pi 4 Pin Layout

**Outputs**

A GPIO pin designated as an output pin can be set to high (3V3) or low (0V).

**Inputs**

A GPIO pin designated as an input pin can be read as high (3V3) or low (0V). This is made easier with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software.

2. **Raspberry Pi Relay Board** - This Relay Board gives Raspberry Pi the ability to control high voltage/high current devices, converting everyday home appliances to smart appliances.

The details of the pins are as follows:

1. Raspberry Pi GPIO interface: for connecting Raspberry Pi
2. Relay screw terminal: for connecting target devices
3. Relays
4. Photocoupler:PC817
5. Relay indicator
6. Power indicator
7. Relay control jumper

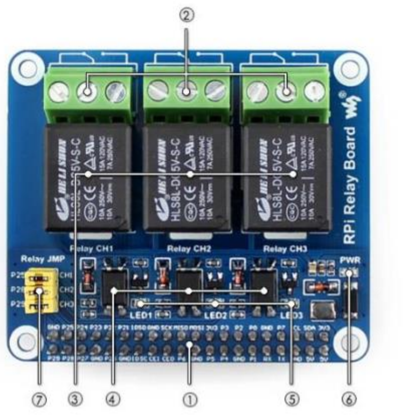


Figure 4: Raspberry Pi Relay Board

The Relay Board's features include:

- Standard Raspberry Pi 40PIN GPIO extension header, supports Raspberry Pi series boards
- High quality relays, loads up to 5A 250V AC or 5A 30V DC
- Photo coupling isolation, prevent interference from high voltage circuit
- Onboard LEDs for indicating relays status

- Relay control jumper, allows to control the relays by custom pins other than the default pins
- Comes with development resources, including examples in wiringPi, WebioPi, shell, python, and bcm2835

The above mentioned components are then connected as per the illustration below and the board is connected to power supply via a USB port.

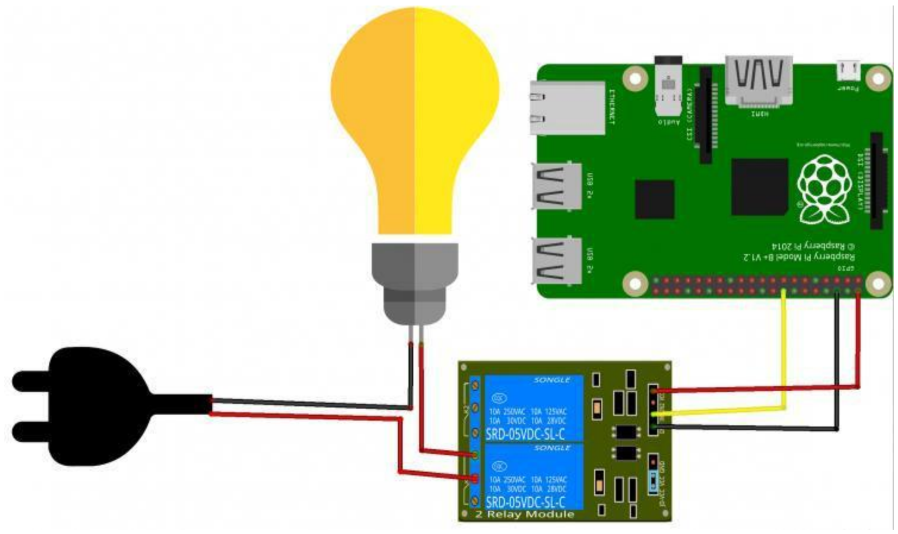


Figure 5: Lighting Device Using Raspberry PI Board and Relay

## Temperature & Humidity Device

### *Components*

The IoT Sensor consists of 2 basic modules:

- ESP8266 (https://joy-it.net/en/products/SBC-NodeMCU)
- DTH22 (https://joy-it.net/en/products/SEN-DHT22)

In Addition to these modules the IoT Sensor also has an indicator LED. It is switched on every time a measurement is undertaken and data is transmitted to the dashboard.

Power is supplied over a USB-Cable to ESP8266 which in turn supplies power to the DHT22 with its internal regulator. Input Voltage to the ESP8266 is roughly 5V, Output of the internal regulator is 3,3V.

The connection of the device is made as following:

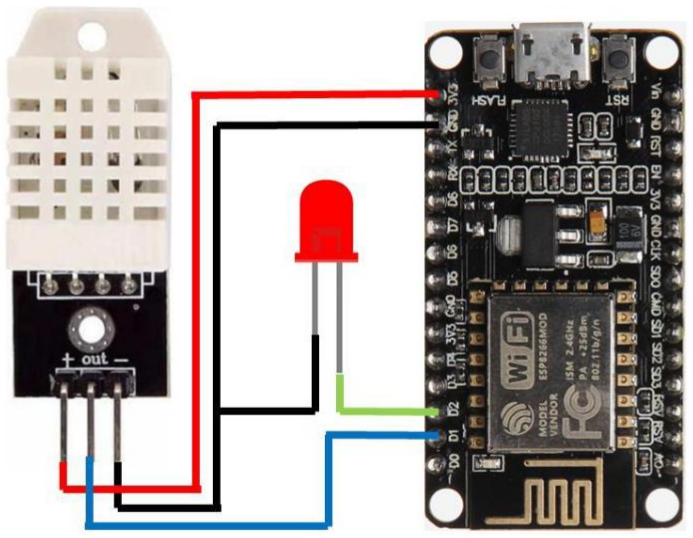


Figure 6: Humidity 7 Temperature Measurement Device using ESP8266 & DTH21

## *Firmware*

The Firmware is based on example code from ThingsBoard for ESP8266 and Adafruit's “DHTtester”.
The library ESP8266Wifi is included for Wifi capabilities.

The Firmware only fulfills some rudimentary tasks. After including said libraries defines are set for the WiFi which is to use, access to the ThingsBoard-Server as well as Pins used to connect the Sensor and LED.

In the setup() function a serial communication for outputs, the WiFi client, the DHT Sensor and the output pin for the LED is initialized.

Code in the loop function is repeated indefinitely. As measurement and data transmission is only to be done every 2 seconds the first instruction is delay (2000). While the delay function is an easy way to implement a delay, it is not advisable to use as it is a blocking function meaning no code can be executed while waiting. But as the ESP8266 has no other tasks and therefore does not need to execute any other instructions while waiting.

To indicate that a measurement is undertaken the LED is switched on in the beginning. Then humidity and temperature data is read from the sensor, and validated before it is written into the variables “temperature” and “humidity” which are used as input for the transmission to ThingsBoard. Also a serial output with the current values is printed.

If the sensor is not connected to ThingsBoard it will reconnect and send the sensor data as two float values to ThingsBoard. Afterwards the LED is switched off and a measurement cycle is completed.

## Tool Stack for Home Management System

The device management system is made operational with the help of the following tool stack:

**Programming language:** Python and C – for programming the Raspberry Pi and relay board

**Databases:** Cassandra – an open source, NoSQL database that is scalable and is ideal for storing time series data from devices. Some of its features include:

- Low latency - withstands data loss across cloud and on premise data storage
- High fault tolerance – database replication and replacement of failed nodes with no downtime
- Highly stable

  Ref: https://cassandra.apache.org/

**Server providers:** Virtual private server, VServer rented from Zap-hosting.com. This server comes with own IP address and protection against DDoS attack.

## ThingsBoard

(WIP)

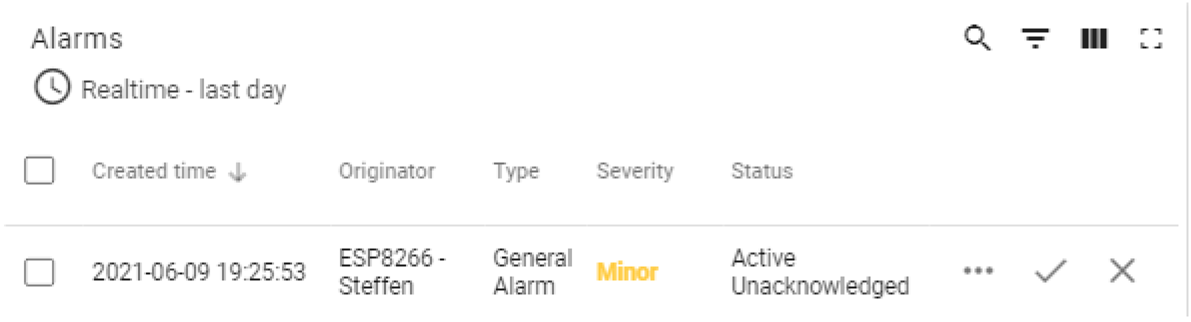
## Test Areas:

### 1. Connectivity testing:

This testing is done to check network connectivity performance between two endpoints in the network. The round trip time(RTT) is found less than 300ms which is considered as a considerable RTT while connectivity testing. Here, RTT is denoted as time it takes for a packet to travel from a user(customer) to server (ThingsBoard server) and back to user(customer). In addition, RTT encompasses queueing delay, propagation delay and processing delay. Therefore, after proper network connection, the presence of telemetry data is detected. When there is no sudden connection, data gets restored in database and latest telemetry is shown in dashboard. However, it is ensured that all the devices integrated with internet of things testing are able to register to the network.

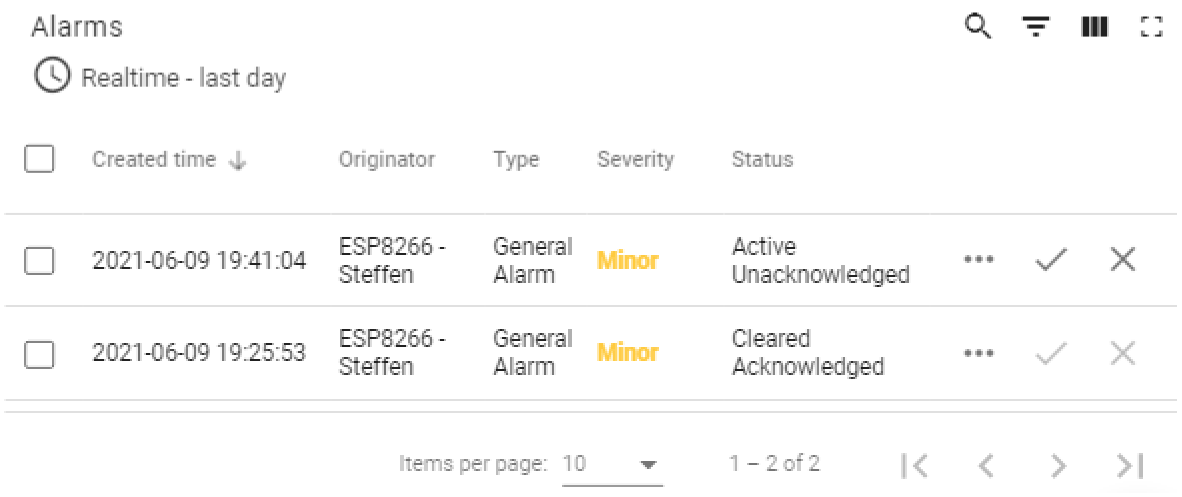
### 2. Functionality testing:

For functional testing, some input values have been changed to observe respective expected outputs to verify if the system is working perfectly or not. In rule engine, the values in scripting are modified to get corresponding outputs which gives errors also in case of inappropriate inputs. Here, it is acknowledged that when scripting function is changed with critical value from 40 to 10. A minor severity is showing as output alarm as presumed for providing low input value. Consequently, when heat index value is set to 30 as critical value, another alarm is generated. Therefore, the rule engine functions are working properly with any changing values by delivering distinct outputs.


Alarms

Realtime - last day

| | Created time | Originator | Type | Severity | Status |
|---|---|---|---|---|---|
| | 2021-06-09 19:25:53 | ESP8266 - Steffen | General Alarm | Minor | Active Unacknowledged |


Alarms

Realtime - last day

| | Created time | Originator | Type | Severity | Status |
|---|---|---|---|---|---|
| | 2021-06-09 19:41:04 | ESP8266 - Steffen | General Alarm | Minor | Active Unacknowledged |
| | 2021-06-09 19:25:53 | ESP8266 - Steffen | General Alarm | Minor | Cleared Acknowledged |

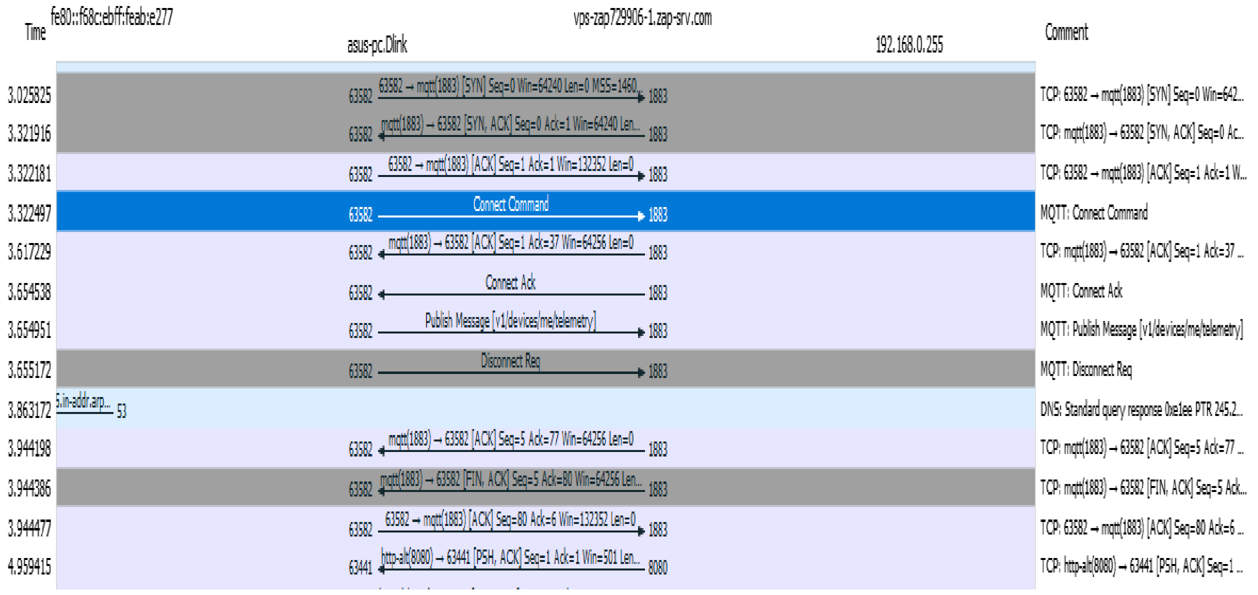
Items per page: 10 1 – 2 of 2

*Figure 7: Dashboard changes with Function changing test*

## 3. Protocol testing:

For devices to be connected with ThingsBoard, transport protocols are used such as MQTT, HTTP which flow back and forth between devices and ThingsBoard Web UI (Rule Engine). This Rule Engine is connected with other core services such as with devices and their credentials, Rule Chains and Rule Nodes, tenants and customers, widgets and dashboard and alarms and events. So if transport protocols MQTT, HTTP do not transfer data, there will be no output data or visualization from ThingsBoard platform. Here, MQTT protocol is transferring data by publish-subscribe method by applying following command as:

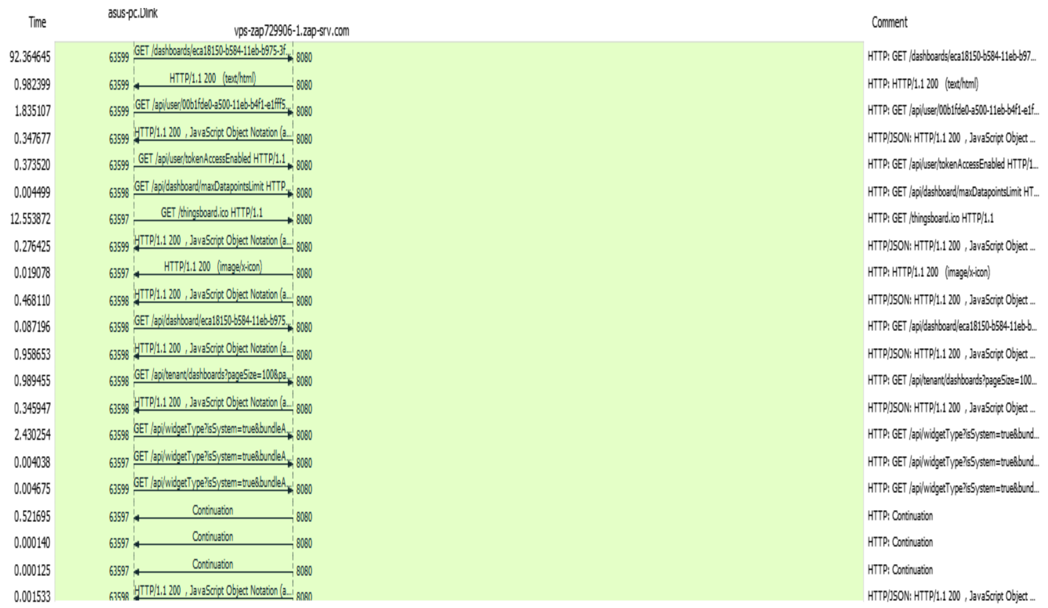
```
mosquitto_pub -h 45.146.254.245 -u l5mC2Vds66TcElwyx27v -t v1/devices/me/telemetry -m "{temparature":45}"
mosquitto_pub -h 45.146.254.245 -u l5mC2Vds66TcElwyx27v -t v1/devices/me/telemetry -m "{humidity":30}"
```



```
Time     fe80::f68c:ebff:feab:e277   asus-pc.Dlink   vps-zap729906-1.zap-srv.com   192.168.0.255   Comment
3.025825  63582  63582 → mqtt(1883) [SYN] Seq=0 Win=64240 Len=0 MSS=1460...  → 1883   TCP: 63582 → mqtt(1883) [SYN] Seq=0 Win=642...
3.321916  63582 ← mqtt(1883) → 63582 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len...  1883   TCP: mqtt(1883) → 63582 [SYN, ACK] Seq=0 Ac...
3.322181  63582  63582 → mqtt(1883) [ACK] Seq=1 Ack=1 Win=132352 Len=0  → 1883   TCP: 63582 → mqtt(1883) [ACK] Seq=1 Ack=1 W...
3.322497  63582  Connect Command  → 1883   MQTT: Connect Command
3.617229  63582 ← mqtt(1883) → 63582 [ACK] Seq=1 Ack=37 Win=64256 Len=0  1883   TCP: mqtt(1883) → 63582 [ACK] Seq=1 Ack=37 ...
3.654538  63582 ← Connect Ack  1883   MQTT: Connect Ack
3.654951  63582  Publish Message [v1/devices/me/telemetry]  → 1883   MQTT: Publish Message [v1/devices/me/telemetry]
3.655172  63582  Disconnect Req  → 1883   MQTT: Disconnect Req
3.863172  5.in-addr.arp... 53   DNS: Standard query response 0xe1ee PTR 245.2...
3.944198  63582 ← mqtt(1883) → 63582 [ACK] Seq=5 Ack=77 Win=64256 Len=0  1883   TCP: mqtt(1883) → 63582 [ACK] Seq=5 Ack=77 ...
3.944386  63582 ← mqtt(1883) → 63582 [FIN, ACK] Seq=5 Ack=80 Win=64256 Len...  1883   TCP: mqtt(1883) → 63582 [FIN, ACK] Seq=5 Ack...
3.944477  63582  63582 → mqtt(1883) [ACK] Seq=80 Ack=6 Win=132352 Len=0  → 1883   TCP: 63582 → mqtt(1883) [ACK] Seq=80 Ack=6 ...
4.959415  63441 ← http-alt(8080) → 63441 [PSH, ACK] Seq=1 Ack=1 Win=501 Len...  8080   TCP: http-alt(8080) → 63441 [PSH, ACK] Seq=1 ...
```

*Figure 8: IoT-Client to  MQTT Broker*

*Fig:8* shows the flow graph where after the TCP connection setup, the MQTT connection command and MQTT messages are transmitted. When commands like above are applied, data gets published in MQTT Broker which is in built in 45.146.254.245 (ThingsBoard server). Then customers get accessed to this new published data from their respective dashboards. Whereas, there is HTTP protocol which is the basic protocol for transportation of data in network between client and server. The flow graph of this protocol in *Fig:9* is shown below:



```
Time        asus-pc.Dlink   vps-zap729906-1.zap-srv.com                                  Comment
92.364645  63599  GET /dashboards/eca18150-b584-11eb-b975-3f...  → 8080   HTTP: GET /dashboards/eca18150-b584-11eb-b97...
0.982399   63599 ← HTTP/1.1 200  (text/html)  8080                      HTTP: HTTP/1.1 200  (text/html)
1.835107   63599  GET /api/user/00b1fde0-a500-11eb-b4f1-e1fff5...  → 8080   HTTP: GET /api/user/00b1fde0-a500-11eb-b4f1-e1f...
0.347677   63599 ← HTTP/1.1 200 , JavaScript Object Notation (a...  8080   HTTP/JSON: HTTP/1.1 200 , JavaScript Object ...
0.373520   63599  GET /api/user/tokenAccessEnabled HTTP/1.1  → 8080   HTTP: GET /api/user/tokenAccessEnabled HTTP/1...
0.004499   63598  GET /api/dashboard/maxDatapointsLimit HTTP...  → 8080   HTTP: GET /api/dashboard/maxDatapointsLimit HT...
12.553872  63597  GET /thingsboard.ico HTTP/1.1  → 8080          HTTP: GET /thingsboard.ico HTTP/1.1
0.276425   63599 ← HTTP/1.1 200 , JavaScript Object Notation (a...  8080   HTTP/JSON: HTTP/1.1 200 , JavaScript Object ...
0.019078   63597 ← HTTP/1.1 200  (image/x-icon)  8080                   HTTP: HTTP/1.1 200  (image/x-icon)
0.468110   63598 ← HTTP/1.1 200 , JavaScript Object Notation (a...  8080   HTTP/JSON: HTTP/1.1 200 , JavaScript Object ...
0.087196   63598  GET /api/dashboard/eca18150-b584-11eb-b975...  → 8080   HTTP: GET /api/dashboard/eca18150-b584-11eb-b...
0.958653   63598 ← HTTP/1.1 200 , JavaScript Object Notation (a...  8080   HTTP/JSON: HTTP/1.1 200 , JavaScript Object ...
0.989455   63598  GET /api/tenant/dashboards?pageSize=100&pa...  → 8080   HTTP: GET /api/tenant/dashboards?pageSize=100...
0.345947   63598 ← HTTP/1.1 200 , JavaScript Object Notation (a...  8080   HTTP/JSON: HTTP/1.1 200 , JavaScript Object ...
2.430254   63598  GET /api/widgetType?isSystem=true&bundleA...  → 8080   HTTP: GET /api/widgetType?isSystem=true&bund...
0.004038   63597  GET /api/widgetType?isSystem=true&bundleA...  → 8080   HTTP: GET /api/widgetType?isSystem=true&bund...
0.004675   63599  GET /api/widgetType?isSystem=true&bundleA...  → 8080   HTTP: GET /api/widgetType?isSystem=true&bund...
0.521695   63597 ← Continuation  8080                               HTTP: Continuation
0.000140   63597 ← Continuation  8080                               HTTP: Continuation
0.000125   63597 ← Continuation  8080                               HTTP: Continuation
0.001533   63598 ← HTTP/1.1 200 , JavaScript Object Notation (a...  8080   HTTP/JSON: HTTP/1.1 200 , JavaScript Object ...
```

*Figure 9: IoT-Client to MQTT Broker*

Furthermore, there is another protocol named WebSocket which is responsible for visualization of real-time analytics or time-series table which has collected data from devices using different protocols HTTP, MQTT and has stored those time-series data in Cassandra database.

### 4. Regression testing:

This test assured that any changes to the rule engine (new function implementation, code changes, debug mode) has not distorted the prevailed functions or has not created any kind of instabilities in ThingsBoard platform.

### 5.Usability testing:

Usability testing is done to gauge user satisfaction with the product interface by the customers. After creation of final product, customers are assigned with defined credentials with which they can access ThingsBoard platform according to their need to visualize and observe so that they can take actions accordingly. Also after having users' feedback, required improvements can be implemented through scripting and coding.

## Testing Challenges faced:

1. Lack of Integration Accessibility (different features integration are possible only for professional edition)
2. Lack of compatible software resources to test ThingsBoard platform
3. Complexity of ThingsBoard Framework
4. Compatibility Issues
5. Security Issues (security implementation cannot be done progressively for unavailability of professional edition)
6. Lack of QA environment

## Appendix

### Installation commands and source code for Raspberry pi based Lighting device:

- Raspberry Pi Imager: https://www.raspberrypi.org/software/
- Raspberry Pi Relay Setup:
    - Open ‘ssh’ for Raspberry Pi in terminal o
      To access GPIO switches, use commands:

```
su
apt install -y git-core
cd /tmp/
git clone git://git.drogon.net/wiringPi
cd wiringPi/
git pull origin
./build
```

    - Check installation with:

```
which gpio
gpio -v
gpio readall
```

    - Sample code to test the relays:

```
PIN=25
gpio mode $PIN output ; gpio write $PIN 0 ; sleep 1 ; gpio write $PIN 1

# for PIN in `seq 1 30`;
do for PIN in 25 28 29; do
  echo driving pin
  $PIN gpio mode $PIN
  output gpio write
  $PIN 0 sleep 1
  gpio write $PIN 1
  sleep 1
done
```

- To connect applications to MQTT broker (inbuilt in ThingsBoard), the following commands and code are run:
    - MQTT library installation:

```
sudo pip install paho-mqtt
```

- Application source code to be saved in a file in the directory:

```
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO
import json

THINGSBOARD_HOST = 'YOUR_THINGSBOARD_IP_OR_HOSTNAME'
ACCESS_TOKEN = 'RASPBERRY_PI_DEMO_TOKEN'

# We assume that all GPIOs are LOW
gpio_state = {7: False, 11: False, 12: False, 13: False, 15:
False, 16: False, 18: False, 22: False, 29: False,
              31: False, 32: False, 33: False, 35: False, 36:
False, 37: False, 38: False, 40: False}


# The callback for when the client receives a CONNACK
response from the server.
def on_connect(client, userdata, rc, *extra_params):
    print('Connected with result code ' + str(rc))
    # Subscribing to receive RPC requests
    client.subscribe('v1/devices/me/rpc/request/+')
    # Sending current GPIO status
    client.publish('v1/devices/me/attributes',
get_gpio_status(), 1)


# The callback for when a PUBLISH message is received from
the server.
def on_message(client, userdata, msg):
    print 'Topic: ' + msg.topic + '\nMessage: '
+ str(msg.payload)
    # Decode JSON request
    data = json.loads(msg.payload)
    # Check request method
    if data['method'] == 'getGpioStatus':
        # Reply with GPIO status
        client.publish(msg.topic.replace('request',
'response'), get_gpio_status(), 1)
    elif data['method'] == 'setGpioStatus':
        # Update GPIO status and reply
        set_gpio_status(data['params']['pin'],
data['params']['enabled'])
        client.publish(msg.topic.replace('request',
'response'), get_gpio_status(), 1)
        client.publish('v1/devices/me/attributes',
get_gpio_status(), 1)
```

```
def get_gpio_status():
    # Encode GPIOs state to json
    return json.dumps(gpio_state)


def set_gpio_status(pin, status):
    # Output GPIOs state
    GPIO.output(pin, GPIO.HIGH if status else GPIO.LOW)
    # Update GPIOs state
    gpio_state[pin] = status


# Using board GPIO layout
GPIO.setmode(GPIO.BOARD)
for pin in gpio_state:
    # Set output mode for all GPIO
    pins GPIO.setup(pin, GPIO.OUT)

client = mqtt.Client()
# Register connect callback
client.on_connect = on_connect
# Registed publish message callback
client.on_message = on_message
# Set access token
client.username_pw_set(ACCESS_TOKEN)
# Connect to ThingsBoard using default MQTT port and
60 seconds keepalive interval
client.connect(THINGSBOARD_HOST, 1883, 60)

try:
    client.loop_forever()
except KeyboardInterrupt:
    GPIO.cleanup()
```

- To launch the application, run the command: `python gpio.py`

## Installation commands and source code for Temperature and Humidity measuring sensor:

```
#include "ThingsBoard.h"
#include <ESP8266WiFi.h>
#include "DHT.h"

#define WIFI_AP             "SSID"
#define WIFI_PASSWORD       "PASSWORD"

#define TOKEN               "l5mC2Vds66TcEIwyx27v"
#define THINGSBOARD_SERVER  "45.146.254.245"
```

```
#define SERIAL_DEBUG_BAUD   115200
#define DHTPIN               5
#define DHTTYPE         DHT22
#define LEDPIN               4

WiFiClient espClient;
ThingsBoard tb(espClient);
int status = WL_IDLE_STATUS;

DHT dht(DHTPIN, DHTTYPE);
float humidity = 0.0;
float temperature = 0.0;

void setup() {
  Serial.begin(SERIAL_DEBUG_BAUD);
  WiFi.begin(WIFI_AP, WIFI_PASSWORD);
  InitWiFi();
  dht.begin();
  pinMode(LEDPIN, OUTPUT);
}

void loop() {
  delay(2000);
  digitalWrite(LEDPIN, HIGH);

  //get data from sensor
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  //check if data is valid
  if(!(isnan(h)||isnan(t))){
    temperature = t;
    humidity = h;

    //Serial Printout
    Serial.print(F("Humidity: "));
    Serial.print(h);
    Serial.print(F("%  Temperature: "));
    Serial.print(t);
    Serial.println(F("°C "));
  }

  if (WiFi.status() != WL_CONNECTED)
     { reconnect();
  }

  if (!tb.connected()) {
      // Connect to the ThingsBoard
      Serial.print("Connecting to: ");
      Serial.print(THINGSBOARD_SERVER);
      Serial.print(" with token ");
      Serial.println(TOKEN);

      if (!tb.connect(THINGSBOARD_SERVER, TOKEN)) {
        Serial.println("Failed to connect");
      return;
      }
  }

  Serial.println("Sending data...");

  // Uploads new telemetry to ThingsBoard using MQTT.
  tb.sendTelemetryFloat("temperature", temperature);
  tb.sendTelemetryFloat("humidity", humidity);

  tb.loop();
  digitalWrite(LEDPIN, LOW);
}

void InitWiFi()
{
  Serial.println("Connecting to AP ...");
  // attempt to connect to WiFi network
  WiFi.begin(WIFI_AP, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
  }
  Serial.println("Connected to AP");
}


void reconnect() {
  // Loop until we're reconnected
  status = WiFi.status();
  if ( status != WL_CONNECTED) {
      WiFi.begin(WIFI_AP, WIFI_PASSWORD);
      while (WiFi.status() != WL_CONNECTED) {
      delay(500);
```

```
        Serial.print(".");
        }
        Serial.println("Connected to AP");
    }
  }
```