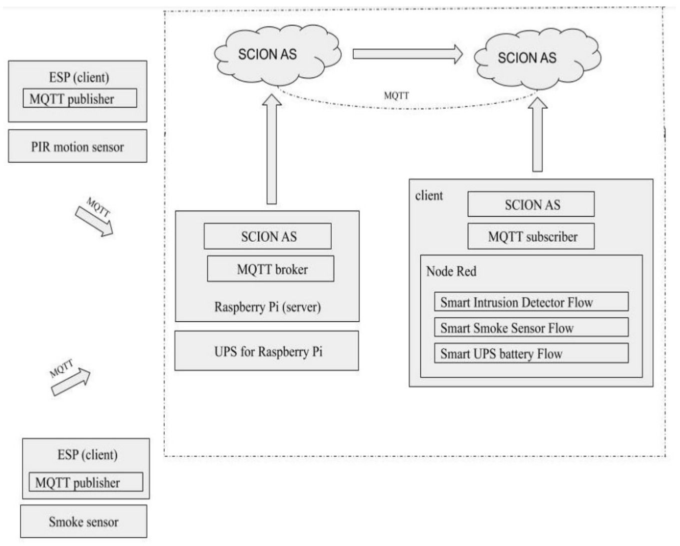
Implementation of SCION on IoT Applications

Abhijith Remesh
Email: abhijith.remesh@st.ovgu.de
Manish Ramachandran
Email: manish.rama@st.ovgu.de

Abstract— **There are several Internet of things (IoT) applications running on legacy networks which are not flexible in terms of path selections, network outages and security concerns. Thus, the current situation demands switching over to a new network platform which is highly scalable, secure, and isolated in terms of its architecture. Thus, a basic IoT application is being attempted which will use Scalable Control and Isolation on Next generation networks (SCION) architecture and hence, surpasses the existing concerns on the current network protocols. The chosen Internet of things (IoT) application is based on Automatic Burglar Detection,where an Intruder is identified without indulging camera setups and rather employing optimal motion sensor, mac ids and virtual switches. The implementation involves a simple client server architecture where there are 2 clients connected to a single server (Raspberry pi) running a SCION AS instance and MQTT broker. The ESP module with PIR sensor act as one of the clients publishing values to the server (Raspberry pi) via MQTT protocol. The SCION AS enabled local machine being the other client and the user interface to the user connects to the server (Raspberry pi) via MQTT over SCION sig configuration. The key process includes (I) The Raspberry pi is configured as sever with MQTT broker and a SCION AS instance.(ii) the client ESP module integrated with PIR sensors publishes motion detection over MQTT protocol to the SCION AS enabled Raspberry pi server. (iii) the other client side, SCION AS enabled machine subscribes and receives the data from the server (Raspberry pi) via MQTT protocol running over the SCION network.**

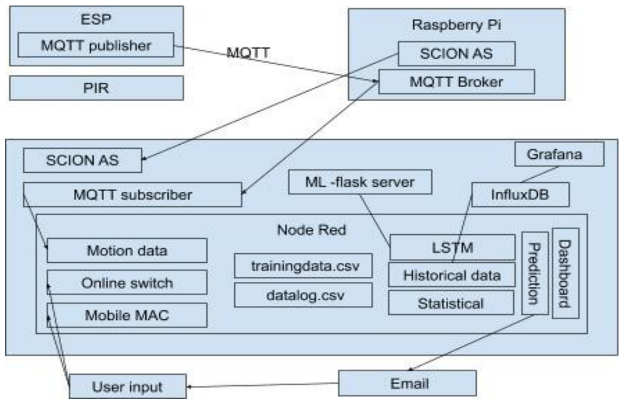
I. INTRODUCTION

We propose to develop an intruder detection application for the Smart Home use case. The main objective of our project is to detect the presence of an intruder at home without the application of cameras as installing cameras within the home is a security concern for the personnel residing at home. Inorder to detect the presence of a person at home,an idea was proposed to monitor three parameters (i) motion , (ii) a switch functionality made available to the user and (iii) another parameter which conveys if the user's mobile is connected to the home network. The passive infrared motion sensor has been employed to detect the motion readings at home which will be installed at an appropriate space at home which would produce HIGH output when a motion is detected and produces LOW output otherwise. An online toggle switch functionality has been developed so that the user can manually operate it whenever he arrives or leaves home. A MAC ID look-up in the ARP table has been used which would look up the MAC address of the user's mobile in the Address Resolution Protocol (ARP) table so that it would produce a HIGH output whenever the mobile is connected to the home network and produces LOW output otherwise. These three parameters namely motion data which will be either a 0 or 1, switch data which will be either a 0 or 1 and mobile MAC data which also will be either a 0 or 1 along with the respective timestamps are considered as the source data for the application which processes this data based on various approaches to determine the times when the person will be likely to be present/absent at home and further determines the presence of an intruder. Various analytical and machine learning approaches have been used in our application to predict the intruder presence namely 'historical data analysis', 'Long short term memory model' , 'Classic Statistical methods'. The prediction results produced by these approaches are taken into account and are compared and evaluated against each other considering the merits and demerits of each of the approach. The best performing approaches are eventually specified in the conclusion.



II. COMMUNICATION FLOW

THIS chapter discusses the data and communication pathways in our application.



- The PIR motion sensor is coupled with ESP module so that motion data from the PIR sensor is sent over MQTT protocol (WiFi) through an ESP module as a MQTT message “Motion/Voltage” to the MQTT broker installed at the raspberry pi end with topic “**/feeds/motion**”. The ESP module is configured to act as a MQTT publisher which publishes the message “Motion/Voltage” to the MQTT broker installed at the raspberry pi end whenever a motion is being detected by the PIR sensor.

- A SCION AS is configured to run on the pi which provides the SCION Internet Gateway (SIG) through specific defined ip addresses”______”. The MQTT broker installed at the raspberry pi end will be always listening on the port 1883 of the pi's network interface. The ESP module publishes the MQTT messages to the pi on a particular topic '**/feeds/motion**'. Upon receipt of this MQTT message at the defined ip address and port 1883 of the pi, the MQTT broker at the pi end will forward this message to the respective subscriber (local remote machine running another defined SCION IP address “ “) who is subscribed on this topic via MQTT over SCION network.

- This MQTT message from the PIR motion sensor acts as the trigger point and captures the instantaneous mobile MAC data and online switch data along with the timestamp.

- This data is logged in two different csv files namely trainingdata.csv and da talog.csv The datalog.csv acts as the source file for historical data analysis and is uploaded weekly to the InfluxDB for the same in a specific format whereas the trainingdata.csv acts as the source file for training the LSTM model and the model is trained on a monthly basis. Hence, the datalog.csv is refreshed on a weekly basis and the trainingdata.csv is refreshed on a monthly basis. The training data.csv is also used for the statistical approaches and providing the weekly summary detection report.

- The motion data, mobile mac data and the online switch data is visualized using Grafana and the predictions results are visualized using the Node-Red dashboard. Whenever the prediction results suggests an abnormal intrusion behavior, an email notification is sent out to the user.

III. HARDWARE DEVICES IN CONCERN

A. Device Specifications

The PIR motion sensor has been configured with different wired setups over the time and some of these setups were sidelined for their inefficient functionality and portability.

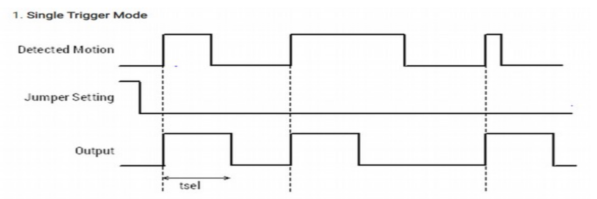
The PIR sensor opted for the application is HC-SR501 whose specifications are as below

- View area :110 degree like cone
- Operating voltage: 5-20 V
- Power consumption: 65mA
- PIN 1: Ground
- PIN 2 Output: 3.5V when the motion is detected.
- PIN 3: Vcc - 5V

The PIR motion sensor is operated single trigger mode.

The time delay begins immediately when the motion is detected. No Continued detection.

- Jumper setting on PIR sensor must be low
- Output becomes high when motion detected.
- Output becomes low after a certain time delay even if the object is in motion.
- After the output being low for a certain time (3s), it again becomes high



The Time delay range for the sensor ranges from 5 seconds to 5 minutes.

- Clockwise towards the extreme right position gives a delay of 5 minutes.
- Counter-clockwise towards the extreme left position gives a delay of 5 seconds.

If the motion is detected, and if the time delay is set as 5 seconds, then the PIR output becomes HIGH for 5 seconds and then it becomes LOW for 3 seconds.

The Sensitivity ranges from 3m to 7m.

- Clockwise towards the extreme right position gives a range of 3 meter.
- Counter-clockwise towards the extreme left position gives a range of 7 meter.

Hence, the PIR is physically configured to operate at a delay of 5 seconds and at a sensitivity of 7m.

The application uses a Raspberry Pi [2] as the micro-controller unit which acts as the server with a SCION AS instance and a MQTT Broker is running.

B. *Hardware Interfacing methodologies*

There were a couple of circuits designed over time to interface the PIR sensor and raspberry Pi in an optimal way. Some of them were dropped eventually considering the efficiency and portability factors. The methodologies that were developed and enhanced over time are described below.

Method 1: The conventional wired setup wherein the motion detection values and person's presence are obtained from the serially interfaced, PIR sensor and hardware switch with Raspberry Pi.

The PIR motion sensor and a hardware switch have been interfaced with Raspberry pi serially. The basic functionality of the hardware switch is to indicate the presence of person so that the user is expected to turn on the switch (Switch value 1) at his presence and turn off the switch (Switch value 0) at his absence. The script at the server side " Client_PIR.go" fetches the motion values ,the corresponding switch values and amalgamate timestamps along with it which forms the data. This data is then delivered to the client-side script at the remote system in subsequent periods over the SCION network.

Because of these concerns, a new idea has been proposed to replace the existing wired setup with the ESP8266 module so that the PIR motion sensor can be connected to this ESP module wirelessly, thus establishing a wireless communication over Wi-Fi via MQTT protocol.

Advantages	Disadvantages
Easy set up	No portability
Easy defect probing	High latency from Client local machine side because of serial communication.
Low latency from server side	Limitation of PIR sensor placement due to lack of portability.
High duration of ON time.	
Easier to deploy	

Method 2: ESP8266 (Wemos d1 mini) and PIR sensor

This setup employs the state of the art IoT protocol, Message Queuing Telemetry Transport (MQTT). It is a wireless protocol that sends message to any other network in a device by configuring the devices as broker and client.

The ESP8266 is a Wi-Fi microcontroller which enables it to connect to a Wi-Fi network. However, Wemos D1 mini is employed for the use case which is a mini board with 4mb flash based on ESP-8266EX. Since a TTL CH340G converter module adapter is required to burn programs in ESP-01 module, the Wemos D1 mini has been used which can be connected directly to any machine to burn the programs. As the Wemos d1 mini is powered by an external battery, there exist a privileges for Battery monitoring by supplying upto 3.3 V across its analog pin which will be discussed in later part.

MQTT Brokers, Publishers and Subscribers

MQTT comprises of publishers and subscribers, connected to the same broker. The broker can be acts as the server and the publishers/subscribers acts as the clients.

MQTT Topics and Messages

All clients attempting to connect via MQTT protocol needs to know the address of the broker .Any client connected to a broker can publish a message on a specific topic and any client connected to the same broker and same topic can subscribe and receive the respective message.
Publisher(client) publishes a message on a topic. Topic is a intermittent channel via which any subscriber enrolled to the same broker and topic can receive the message.

Hosting an MQTT broker

A MQTT broker can be hosted in a computer, Pi or any virtual machine. Similarly any program (python or go or Arduino etc) can enable MQTT(publish/subscribe) with the latest

MQTT libraries like Paho MQTT for Python and PubSub for Arduino.

Mosquitto Installation and Service Activation
Sudo apt install mosquito mosquito-clients
Sudo systemctl enable mosquitto

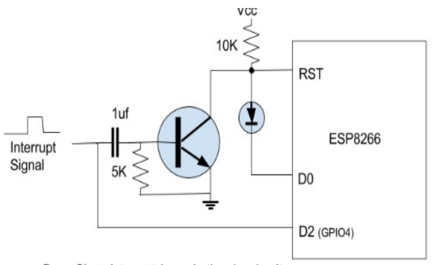
Arduino IDE has been used to burn the code into Wemos di mini. The PubSubClient library has been used in the code for the ESP to publish motion value to the pi where the MQTT broker is installed.
Topic : /feeds/motion
Value : Motion/(Person_Value) for eg Motion/5.2

As described earlier, the Hardware switch has been used to indicate the presence of person at home. If the person is present at home, it would indicate 1 (person switches on) and otherwise 0.

Advantages	Disadvantages
Wireless Set up	Heavy battery drain
flexible to mount at remote location	
Away from the raspberry pi.	

Because of the concern of high battery drain, a new idea has been proposed to design a PCB which would put the ESP in deep sleep when not in use.

Method 3: ESP8266 (Wemos) with Sleep PCB Circuit and Physical Switch



Deep Sleep interrupt (one shot) wake circuit

The above circuit has been designed to put the ESP in deep sleep mode when not in use and its corresponding PCB circuit has been developed. With this PCB circuit we were able to cut down the power usage of ESP hence saving the battery life. The module wakes with an interrupt which is nothing but the detected motion which would further publish a MQTT message and goes back to sleep.

Battery Usage Calculation

During the normal mode, the PIR motion sensor and the ESP consumes 3.3V and 80mA of current.

In the deep sleep mode, the estimated / calculated current consumption of ESP is around 8mA (substantial drop)

We assume 80 motion in an hour which varies from person to person.

Motion in an hour = 80

Current consumption by ESP during a motion = 80mA

Time taken by ESP to wake up, publish the message and sleep = 2.5sec (on an avg)

In a hour = 80 * 2.5 = 200seconds
Remaining 3400 sec in hour is in deep sleep mode.

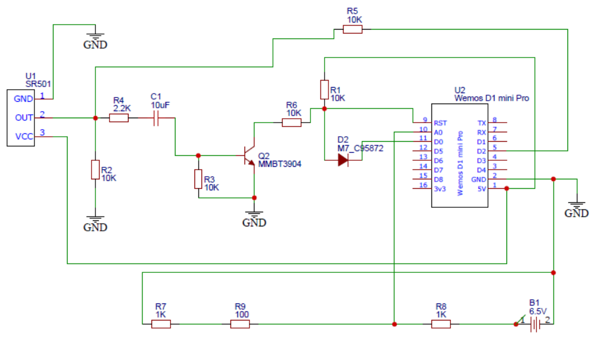
200 sec * 80mA + 3400sec * 8mA = 12mAh in an hour

Source = Battery capacity of our setup = 2900 mAh (4 AA batteries in series 6.5 V)

Output efficiency of battery = 0.80 * 2900 = 2320mAh

2310 mAh /12mAh = 193 hours = 8days

The battery should approx. last 8 days if there is 80 motion in an average in an hour. Hence, it is required to find much optimal solutions.



Hence, the battery voltage needs to be stepped down from 6.5v to 3.3 v to feed it across the analog pin of ESP. This has been successfully implemented with the help of voltage divider (1 and 1.1K) where voltage across 1.1 K (total 2..1k) is fed across the ESP analog pin (3.3v). This is scaled back in the Arduino code to detect the real voltage and sense the health of battery.
Because the MQTT message from ESP is “Motion/(battery voltage)”, the user in the client side get continuous notification about the battery health via Node-RED UI. Alerts is also set when the battery voltage steps down below 4.2V.
The last enhancement in this setup has been the implementation of Wi-Fi Manager library in ESP (Wemos d1 mini).

The WiFi network is never constant and it is required for the ESP to connect to different Wi-Fi networks/broker/topic dynamically as per our needs. Technically, it's a repetitive task as the program needs to be burned to ESP using Arduino every time, to connect to a new Wi-Fi network. This concept has been made super easy with the Wi-Fi manager library. Every time the code is burned into ESP (without Wi-Fi cred) it creates a mount point (server) which can be accessed for setting all the parameters (Wi-Fi/topic/broker). Once the ESP is reset, next time it connects to the parameters that were set in its configuration file earlier.

Whenever there is a need to edit any of these parameters the ESP is required to create a mount point so that the parameters could be reconfigured. A small two-way physical switch has been implemented to realize this functionality. When the switch is turned on (set high) it resets all the parameters in the ESP config file and hence creates a mount point. Once all our parameters are set, the switch needs to be turned off (set low) so as to save the parameters. This process can be repeated to wipe out all parameters and set again.

Also, as an enhancement, the hardware switch (indicating person presence) has been replaced with an online switch so that the user could have an access to it via the webpage rather than expecting the user to enable and disable the switch physically. This is achieved by the "Online switch node" configured in Node-RED. The switch can be turned on/off via Node-RED UI.

Apart from switch functionality, Address Resolution protocol (ARP) has been incorporated in the application which is realized using the ARP node in Node-RED. The mac id of the user's mobile is specified in the ARP node. Whenever, the user's mobile is connected to Wi-Fi at his presence, the ARP outputs a value of "1" indicating that the user is at home.

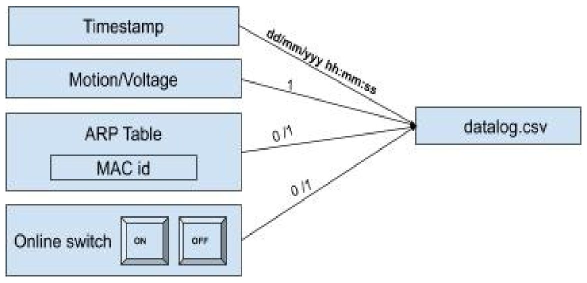
IV. DATA ANALYSIS APPROACHES

This chapter discusses about the establishment of client server communication over the SCION [1] network. The sequence of operation is as follows:

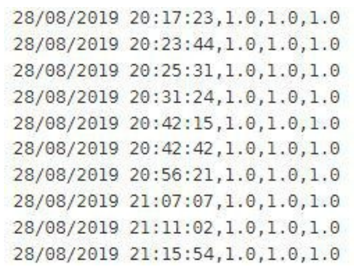
A. Historical data analysis

1) Logging Data

This approach involves the periodical querying of a InfluxDB database containing the historical data (timestamp data, week index data, time index data, motion data, online switch data, mobile mac data). This historical data in the InfluxDB database is updated on a weekly basis. In every five minutes, a query is passed to the database to return the number of detections for that five minute interval from the historical data which conveys the usual number of detections that normally happen on that day, for that five minute interval.This query result is then compared with the current number of detections observed in the current five minute interval. The detailed description of this approach is as follows:



When the MQTT message "Motion/Voltage" is received at the server end which as the MQTT subscriber, the corresponding online switch data and the mobile MAC data at that instant of time is extracted. The MQTT message is fetched using the "mqtt in" node which is configured by providing the ip address of the raspberry pi "________" and the topic of the MQTT message "/feeds/motion". The MQTT message "Motion/Voltage" is then further mapped as 1. The online switch value is extracted using the "switch" node which is a toggle button representing "0" at OFF state and "1" at ON state. The mobile mac data is derived from the "ARP" node where the MAC id of the user's mobile is specified so that this MAC id is checked in the ARP table of the home's network. Presence of the MAC id corresponds to "1" and absence of the same corresponds to "0". As a result, the motion data will be always "1", the online switch value will be either "0" or "1" and the mobile mac value will be either "0" or "1". All these data are then combined together along with the corresponding timestamp and is logged in a csv file,datalog.csv. An entry in the log file only happens upon a motion detection and on further receipt of the MQTT message"Motion/Voltage" at the server end The format of the datalog.csv is as shown below :



```
28/08/2019 20:17:23,1.0,1.0,1.0
28/08/2019 20:23:44,1.0,1.0,1.0
28/08/2019 20:25:31,1.0,1.0,1.0
28/08/2019 20:31:24,1.0,1.0,1.0
28/08/2019 20:42:15,1.0,1.0,1.0
28/08/2019 20:42:42,1.0,1.0,1.0
28/08/2019 20:56:21,1.0,1.0,1.0
28/08/2019 21:07:07,1.0,1.0,1.0
28/08/2019 21:11:02,1.0,1.0,1.0
28/08/2019 21:15:54,1.0,1.0,1.0
```

2) Uploading logged data to InfluxDB

This datalog.csv file acts as the basis file for the historical data analysis.This datalog.csv file is then uploaded to the InfluxDB database "historicaldb" on a weekly basis and hence, the datalog.csv file is also refreshed on a weekly basis to avoid duplicate data being uploaded to the "historicaldb". A retention policy of six months is applied on the database so that the historical data older than six months are automatically removed from the database as the time progresses and eventually, the database will only contain the recent historical data.

The datalog.csv file is uploaded to “historicaldb” in a particular format by passing the data over a series of python scripts namely “timestamp_formating.py”, “timestamp_influxconversion_format.py” and “timestamp_influx_lineprotocol_format.py”.

The input data to “timestamp_formating.py” is of the format:

Timestamp	motion	Online switch	Mobile Mac
dd/mm/yyyy hh:mm:ss	1	0 or 1	0 or 1

Which is then transformed into the format as below:

Timestamp	WDay	Time index	moti -on	Online switch	Mobile Mac
dd/mm/ yyyy hh:mm:ss	1-7	0-287	1	0 or 1	0 or 1

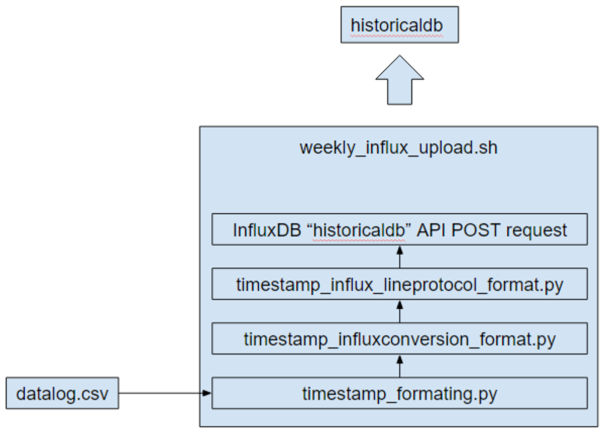
The “timestamp_formating.py” script aggregates the datalog.csv on a five-minute interval basis such that the timestamps are in five-minute intervals in the sequence dd/mm/yyyy hh: 00:00, dd/mm/yyyy hh:05:00, dd/mm/yyyy hh:10:00 and so on.

The script also derives the WDay and the time index from the date and time information of the timestamp respectively. The weekday, WDay is derived from the date information of the timestamp ie 1 for Sunday, 2 for Monday, 3 for Tuesday, 4 for Wednesday, 5 for Thursday, 6 for Friday and 7 for Saturday. Hence, the WDay value will be any value between 1 and 7. The time index value is also derived from the time information of the timestamp. The time index value is the incremental value for every five minutes starting from 0 to 287 such that the first five minutes of a day (00:00:00 - 00:05:00) corresponds to 0 and the last five minutes of a day (23:55:00 - 00:00:00) corresponds to 287.Thus, the time index will have values ranging from 1 to 287 as a day contains 287 five-minute intervals.
As a result, an entry in the output file conveys the following information: five minute interval timestamp, the WDay which tells the day when the motion was detected, time index which tells five minute interval on that day, motion which tells the number of times, motion has been detected for that five minute interval on that day ,the online switch and the mobile mac value for that five minute interval on that day.

The “timestamp_influxconversion_format.py” script converts the timestamp into the corresponding UNIX timestamp format.

Timestamp (UNIX)	WDay	Time index	moti -on	Online switch	Mobile Mac
dd/mm/ yyyy hh:mm:ss	1-7	0-287	*	0 or 1	0 or 1

The “timestamp_influx_lineprotocol_format.py” also transforms the existing format to a particular line protocol format such that the data suits to be fed into a measurement “sensor” within the “historicaldb”.The output file from this script is then fed to the InfluxDB “historicaldb” using the InfluxDB API POST request. The data gets updated in the “historicaldb” as follows:

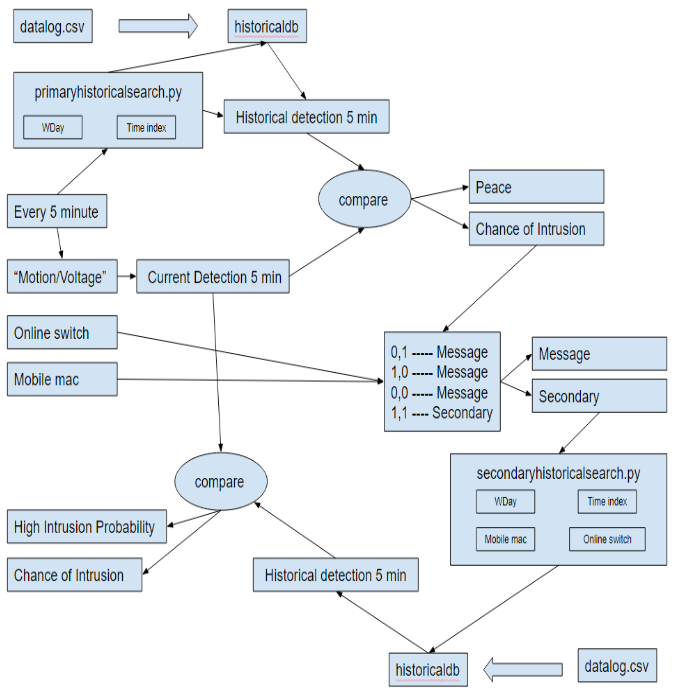

historicaldb
weekly_influx_upload.sh
InfluxDB “historicaldb” API POST request
timestamp_influx_lineprotocol_format.py
timestamp_influxconversion_format.py
timestamp_formating.py
datalog.csv

The weekly_influx_upload.sh is run weekly using the crontab services which is scheduled to run on every Sunday at 23:20:00 so that the historical data update happens automatically every week. and the datalog.csv is also refreshed weekly so that every week’s datalog.csv gets uploaded to the “historicaldb” to avoid duplicate upload.

```
20 23 * * 0 /bin/sh /home/pi/weekly_influx_upload.sh
```

3) *Historical Search logic*

In every five minutes of a day, a SELECT query is passed to the “historicaldb” to return the number of detections happened for that five minuteon that day. This is done using the “primary historicalsearch.py” script which is embedded within the “Python Shell” node in the Node-Red.



In every five minute of a day, the instantaneous timestamp is passed as arguments to the "primary historicalsearch.py" script.This script takes this timestamp and derives the respective current WDay and current time index from the timestamp information.So,the week index will be any value between 1 to 7 and the time index will be any value between 0 to 287.

The "historicaldb" already contains the historical data in the format (timestamp,WDay, time index,detections, online switch, mobile MAC). The script then passes a couple of select queries on this historical data :

- "Select the "sum"of detections from"historicaldb" where time index = current time index and WDay = current WDay "
    - This will fetch out the total sum of detections that happen on this particular week day and time index. This value is nothing but the sum of detections that happen in a five minute interval on specific week day.
- "Select the "count" of WDay from "historicaldb" where time index = current time index and WDay = current WDay"
    - This will fetch out the count of specific WDays existing in the "histroicaldb"

The division of the two results corresponds to the usual number of detections that happen on a five minute interval of a specific day based on the historical data. This corresponds to the historical five minute detections value.

In every five minute, the number of times , the MQTT message "Motion/Voltage" being received at the "mqtt in" node is counted. This value corresponds to the current five minute detections value.

This historical five minute detections value and current five minute detections value are compared with each other on a function node. If the current five minute detections value exceeds the historical five minute detections value by a nominal percentage,then the situation can be considered as abnormal situation and the situation is classified as a "Chance of Intrusion" and otherwise as "Peace".

Condition	Situation
Current five min detection > historical five min detection	Chance of Intrusion
Current five min detection < historical five min detection	Peace

If an abnormal situation or "Chance of Intrusion" case arises, the current online switch value and current mobile MAC value are taken into consideration and different cases are formulated based on their value as follows:

Situation	Online Switch	Mobile mac	Action
Chance of Intrusion	0	0	message
	1	0	message
	0	1	message
	1	1	Secondaryhistoricalsearch.py
Peace	nil	nil	nil

The message corresponds to "Detections are observed or check your dashboard to see if the switch value and mobile MAC value is correctly configured or not". This message is displayed in the NodeRed dashboard.

This corresponds to the historical primary search.

If both online switch and mobile MAC are in HIGH state, a SELECT query is passed to the script "secondaryhistoricalsearch.py" embedded in the pythonshell of Node-red along with the current timestamp value as arguments.The script takes this timestamp and derives the respective current WDay and current time index from the timestamp information.The script passes a couple of queries to the "historicaldb"

- "Select the "sum"of detections from"historicaldb" where time index = current time index and WDay =

current WDay " and mobile mac = current mobile mac and online switch = current online switch value.
    - This will fetch out the total sum of detections that happen on this particular week day and time index if the person is present at home. This value is nothing but the sum of detections that happen in a five minute interval on specific week day at user's presence. The result can be even empty which indicates that the passed timestamp does not exist at all at user's presence in the historical data.
- "Select the "count" of WDay from "historicaldb" where time index = current time index and WDay = current WDay".
    - This will fetch out the count of specific WDays existing in the "histroicaldb"

The division of these two results corresponds to the usual number of detections that happen on a five minute interval of a specific day when the person is present based on the historical data.This corresponds to the historical five min detections.The number of times , the MQTT message "Motion/Voltage" being received at the "mqtt in" node is counted in every five minute which corresponds to the current five minute detections value. Both these values are compared against each other as follows:

Condition	Situation
Current five min detection > historical five min detection	High Intrusion probability
Current five min detection < historical five min detection	Chance of Intrusion
Historical five min detection	High Intrusion probability

This corresponds to the historical secondary search.

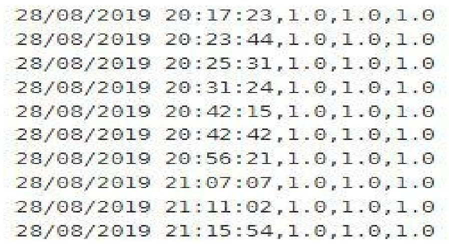
The result status of the primary and secondary search is displayed at the node red dashboard for user reference which displays and refreshes the status in every five minute interval. Despite of that, whenever an abnormal situation arises, the respective message is sent out to the user via an email notification.

B. *Statistical methods*

Two statistical methods are used to derive relevant information from the logged data ,trainingdata.csv.

1) Standard deviation method

Since the data is only logged in the trainingdata.csv file whenever a motion is detected, the timestamps logged in the dataset will be the timestamps of motion detection and the motion value logged in the dataset will be always "1' as follows :



```
28/08/2019 20:17:23,1.0,1.0,1.0
28/08/2019 20:23:44,1.0,1.0,1.0
28/08/2019 20:25:31,1.0,1.0,1.0
28/08/2019 20:31:24,1.0,1.0,1.0
28/08/2019 20:42:15,1.0,1.0,1.0
28/08/2019 20:42:42,1.0,1.0,1.0
28/08/2019 20:56:21,1.0,1.0,1.0
28/08/2019 21:07:07,1.0,1.0,1.0
28/08/2019 21:11:02,1.0,1.0,1.0
28/08/2019 21:15:54,1.0,1.0,1.0
```

The script "stdprediction.py" transforms the datalog.csv into another format as follows

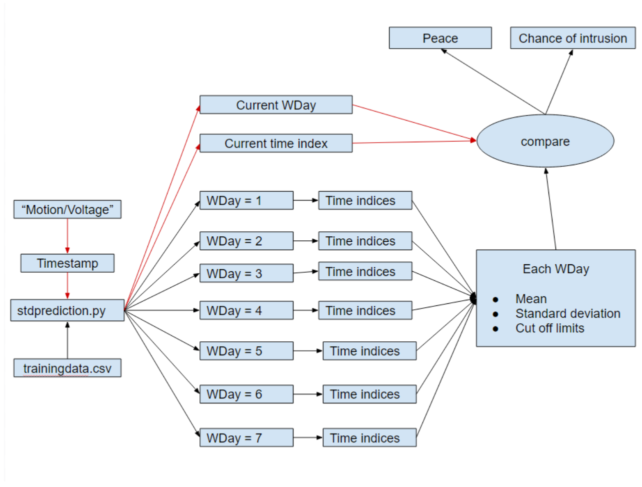
Timestamp	WDay	Time index	moti -on	Online switch	Mobile Mac
dd/mm/ yyyy hh:mm:ss	1-7	0-287	1	0 or 1	0 or 1

The script also filters the dataset by applying the selection condition such that online switch value = 1 , mobile MAC value = 1 and motion = 1.

Timestamp	WDay	Time index	moti -on	Online switch	Mobile Mac
dd/mm/ yyyy hh:mm:ss	1-7	0-287	1	1	1

Hence, the dataset only contain the corresponding WDay and time indices when the person is present and when the motion is detected which indicates the time periods of motion detections at users presence.

The script groups the motion detected time indices with respect to the WDay values such that the motion detected time indices of the WDay 2, Monday is stored in one list and the motion detected time indices of the WDay 3, Tuesday is stored in another list and so on. As a result, the motion detected time indices of each day is stored in each specific list or each group.



The mean and standard deviation of the time indices of each WDay is calculated dynamically with every new detection that happens logged in the csv file. The cut off limit is defined as the thrice times the standard deviation. Likewise, the upper and the lower cut off limit with respect to the standard deviation value is calculated for each WDay. These values gets changed dynamically when a new detection gets logged into the csv file. This logic is implemented on the script which runs on every detection that happens.On every detection, the instantaneous timestamp is passed as arguments to the script which triggers the script to run and the respective week index and time index is derived from the timestamp.

The current WDday specifies the list to be checked , that is the WDay to be checked and if the current time index exceeds the upper cut off limit or fall short of the lower cut off limit of that WDay, then the situation is regarded as an abnormal situation like chance of Intrusion otherwise is regarded as a peaceful situation.

2) *Interquartile region method*

The trainingdata.csv acts as the source file for this method.

The script "iqrprediction.py" transforms the datalog.csv into another format as follows :

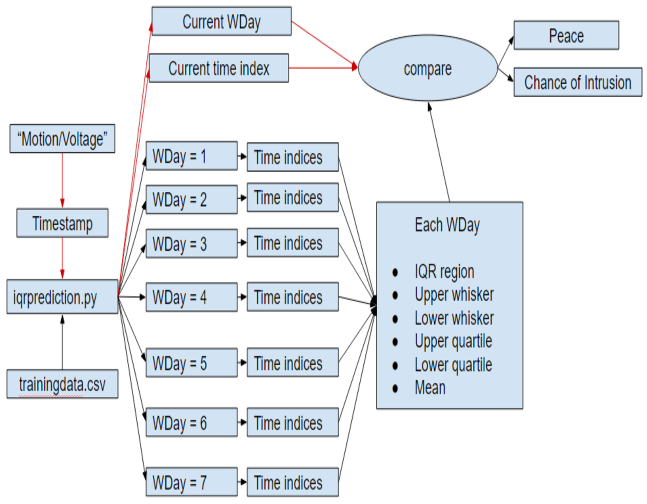
Timestamp	WDay	Time index	moti-on	Online switch	Mobile Mac
dd/mm/ yyyy hh:mm:ss	1-7	0-287	1	0 or 1	0 or 1

The script also filters the dataset by applying the selection condition such that online switch value = 1 , mobile MAC value = 1 and motion = 1.

Timestamp	WDay	Time index	moti-on	Online switch	Mobile Mac
dd/mm/ yyyy hh:mm:ss	1-7	0-287	1	1	1

Hence, the dataset only contain the corresponding WDay and time indices when the person is present and when the motion is detected which indicates the time periods of motion detections at users presence.

The script groups the motion detected time indices with respect to the WDay values such that the motion detected time indices of the WDay 2, Monday is stored in one list and the motion detected time indices of the WDay 3, Tuesday is stored in another list and so on. As a result, the motion detected time indices of each day is stored in each specific list or each group.



The interquartile region, upper whisker, lower whisker , upper quartile, lower quartile and mean of the time indices of each WDay is calculated dynamically and stored with every new detection that gets logged in the datalog.csv file.These values gets changed dynamically when a new detection gets logged into the csv file. This logic is implemented on the script which runs on every detection that happens.On every detection, the instantaneous timestamp is passed as arguments to the script which triggers the script to run and the respective WDay and time index are derived from the timestamp. If the passed current time index of a particular WDay is greater than the upper whisker/ upper quartile or lesser than the lower whisker/ lower quartile or not within the interquartile region of that WDay, then that passed timestamp is regarded as an abnormal situation, that is Chance of Intrusion otherwise is regarded as a peaceful situation.

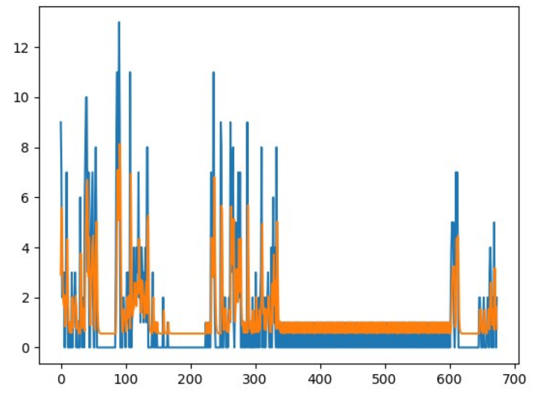
C. Machine Learning Approach

Since the application deals with motion and time series, it was decided to choose one of the best suited Time series forecasting model for the Machine Learning functionality. The idea is to track, predict and compare the new motions instances with past occurred instances . The motion data is un-structured, huge and pattern less data (having bigger duration gap length). So LSTM with its capability to remember past data even older ones, was the perfect choice. Apart from these, the lack of many training parameters, for instance in our case, time and motion values are only the parameters that are passed to the machine learning model. LSTM is preferred in such univariate time cases with plethora amount of data.
Whenever there is a motion detected, from the sensor, the data is recorded in the client side (Node-RED) as **Timestamp** : mm/dd/yy HH:mm:SS and **motion value** as "1". This is the data format which is fed into our machine learning model as the input for training.

The machine learning model was built with 67 percent of motion data (timestamp aggregated as 5min duration and the corresponding motion values for the duration is added.) as training set and remaining set of test data was used to validate the same. The other parameters used to train model was

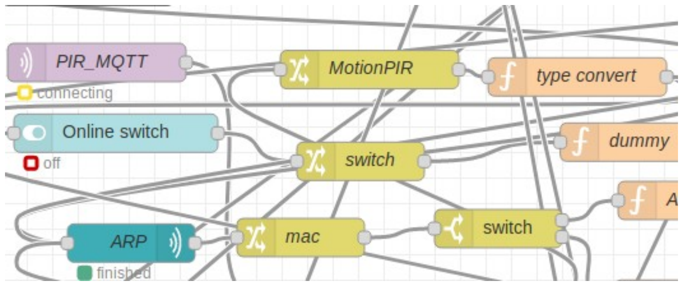
Batch Size : 1
Epochs : 1500
Neurons : 4
Loss : mean_squared_error
Optimizer: adam

The LSTM model obtained was quite satisfactory with the RMSE score of 1.6 as per test data validation given below where the orange plot refers to the predictions.

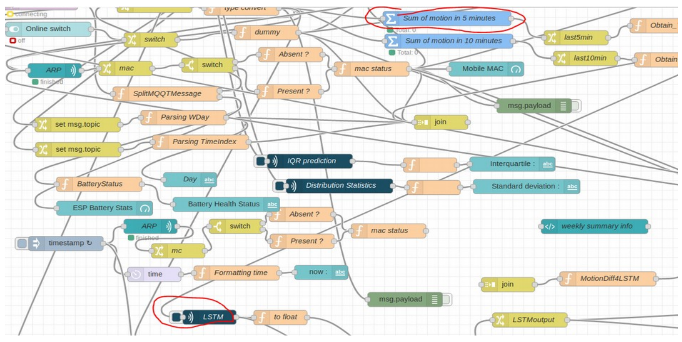


The model was saved offline using appropriate libraries and used for real time prediction by inputting real time motion values. This practical move was achieved by creating a Flask application (ML server) based on python where we would load the model once when the server is run initially. The server is good to accept API calls which in turn are motion values. These motion values obtained are passed to the model where it would predict the next motion value(for next 5 min). The output of the model which is the predicted value is fetched back to the client and compared with the real time motion value from the sensor in that specific instance. Based on the variation of the values it is decided, if it's a Intrusion or Motion caused by the user. The application server is enabled as a service " modelstart.service" which is autorun every time the machine boots. Also the service restarts monthly once the training is completed to invoke and load the newly obtained model.

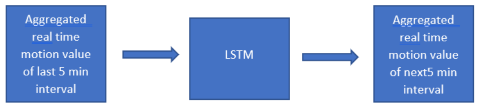
1) Working of LSTM prediction based on Node-red implementation

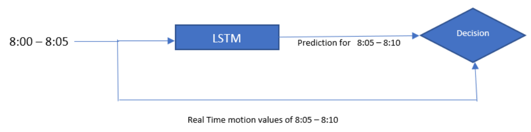


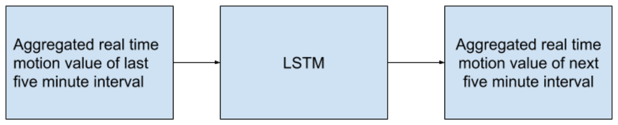
The PIR_MQTT node in Node-RED is connected to the broker installed in pi via Scion protocol. Whenever the data is sensed by the sensor, the ESP publishes value "Motion/(battery voltage)" onto the mosquito broker installed in pi. The PIR_MQTT node in Node-RED (Client VM machine) subscribed to the same broker and topic in pi (data transfer between remote systems via Scion sig protocol) receives this data instantly and feeds it to the member nodes. Based on this, the battery status and the motion value is processed. This motion data is saved in a CSV file " Trainingdata.csv" which is used monthly to train the model. The training duration is up to every user and also depend on the accuracy of the model obtained.

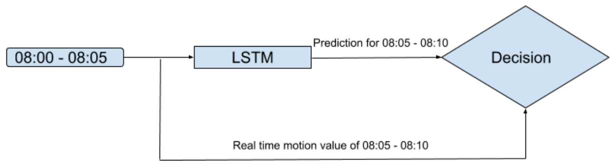


As discussed, in case of motion, the Node-RED detects the motion value as “1”. Now since the model was trained with the data aggregated for 5min interval, the model predictions will always be for future 5 mins. Hence during a motion, the real time motion values across last 5 min interval is fed into LSTM to predict motion value for next 5 min interval. As discussed this is done by running a python file in “Python shell node” in node-red. The script would take the motion value as its input argument and make an API call to our Flask application server where the model is loaded and running. This value is fed to model for the prediction. The predicted value is fed back as the output of the script. This predicted value from the python shell is put in a register and compared with the real time motions at next 5min interval end. Based on the deviation or the overlapping of data, the final decisions are made.


Aggregated real time motion value of last 5 min interval
LSTM
Aggregated real time motion value of next5 min interval


8:00 – 8:05
LSTM
Prediction for 8:05 – 8:10
Decision
Real Time motion values of 8:05 – 8:10


Aggregated real time motion value of last five minute interval
LSTM
Aggregated real time motion value of next five minute interval


08:00 - 08:05
LSTM
Prediction for 08:05 - 08:10
Decision
Real time motion value of 08:05 - 08:10

After comparison of real time and predicted values, a difference of these values are calculated. If the difference is greater than 5 motions (which is a customization parameters and could vary with different applications and users), the final decision from the Machine learning side as Intrusion. The output decision/response again depends on the value of ARP and online switch obtained. Again this decisive part depends on person to person who can customize the algorithm as per their choice

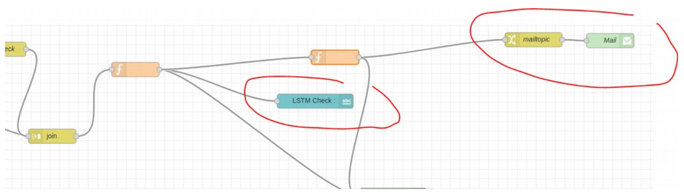
If the output from LSTM is in a considerable range (threshold 5 motions) it is concluded as “Peace” ignoring switch and mac values.

However if the output from LSTM is considerably deviated from the normal range, then the decision is based on the obtained switch and mac values

- Switch = 1 Mac = 1 message : No chance of Intrusion as Switch and Mac configured rightly
- Switch = 0 Mac = 1 message : Detections are observed. Either Switch/Mobile misconfigured, Else Inrusion
- Switch = 1 Mac = 0 message : Detections are observed. Either Switch/Mobile misconfigured, Else Inrusion

Online switch	Mobile MAC	Message
1	1	“No chance of Intrusion”
0	1	“Detections are observed. Either Switch/Mobile misconfigured, else Intrusion”
1	0	“Detections are observed. Either Switch/Mobile misconfigured, else Intrusion”

If the decision turns out to be any of the above 3 other than “Peace” (from the machine learning side) the user is notified by an alert mail sent to their respective mail id. This is done using the mail node in Node-RED.



V. REPORTS

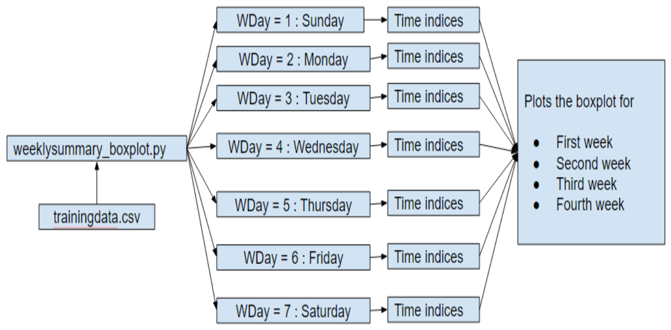
A. *Weekly Detection Summary Report*

The trainingdata.csv acts as the source file for this application.

The script “weeklysummary_boxplot.py” transforms the trainingdata.csv into another format as follows:

Timestamp	WDay	Time index	motion

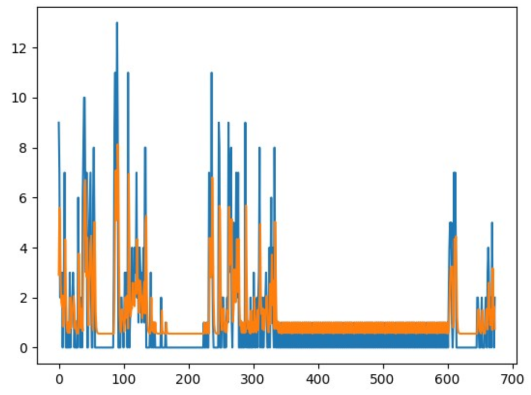
dd/mm/ yyyy hh:mm:ss	1-7	0-287	1



The box plot maps the time indices values with respect to the WDay value. In other words, the script creates the box plot for sunday by considering the time indices belonging to the WDay = 1 and creates the boxplot for Monday by considering the time indices belonging to the WDay = 2 and so on. This process is iterated for every week so that daily box plots of each of the week is obtained. A week's box plot summary comprises of boxplots of Sunday, Monday, Tuesday, Wednesday, Thursday, Friday and Saturday. The Summary consists of box plot of the last four weeks. The script is scheduled to run daily at 21:00:00 using the crontab services as shown below:

```
00 21 * * * /bin/sh /home/pi/weeklysummary_boxplot.py
```

B. *LSTM Predictor model Report*



After each cycle of lstm model training based on the recent updated dataset, a report on the lstm model created is generated specifying the accuracy of the model with respect to the RMSE score. The lower the RMSE value, the better is the model accuracy and performance.With this report, the user is able to access and evaluate the model performance. The blue line indicates the actual prediction values and the orange line indicates the expected prediction values. This report is available to the user via the Node Red dashboard.

VI. IMPLEMENTATION OF GUI

THIS chapter explains how the graphical user interface has been implemented at the client SCION [1] AS node. The flow-based, visual programming development tool, Node-Red [6] has been employed for this purpose. Several types of nodes has been used and configured accordingly for several use cases like to retrieve the MQTT message over the SCION network, to realize the online switch functionality, to realize the ARP based mobile mac functionality, to run the python scripts at real time, to customize and parse the data, to provide dashboard facility for the users and so on. The most important nodes and it's configuration and it's requirement has been discussed below

- "*MQTT In*", a network node which is configured with the raspberry pi's ip address/ scion instance (MQTT broker) so as to receive the MQTT message 'Motion/Voltage' whenever motions are being detected.
- "Switch", a dashboard node which provides a switch functionality at the node red dashboard so that the user can either turn ON or OFF depending on his presence at home.
- "ARP" node, a network node which provides the content of the ARP table with which we can detect the ip address and the MAC address of the devices connected in the network. The node is configured by specifying the MAC address of the user's mobile. This tells if the user's mobile is connected to the network by checking if the MAC address of the mobile is present in the ARP table.
- "Join", a sequence node which is used at several instances to combine the different messages together so that certain functionalities and comparison checks can be made among the different messages being received.
- Custom "Function", a function nodes which is used to parse, format the messages being received, to create custom context dependent functionalities and comparison checks with the messages.
- "Python shell", a input node which has been used to run the python scripts upon triggering of the node. The node gets triggered whenever it receives a input which can be also passed as arguments into the python script. The python shell has been used to run various scripts which gets triggered on motion detected timestamps (weeklysummary_boxplot.py, iqrprediction.py, stdprediction.py) or every five minute intervals(primaryhistoricalsearch.py), every five minute current detection value.
- "Email, a network node which is used to sent an email notification whenever any abnormalities in detections are observed at peculiar timings based on the implemented

approaches. The node is configured by specifying the email credentials of the user.

- "Csv", A parser node to parse the data in the csv format which has been used to log the data being received: t imestamp, PIR motion data, online switch data, mobile mac data in two different csv files namely datalog.csv and trainingdata.csv. The former acts as the source file for InfluxDB upload and the latter acts as the source file for various already discussed approaches.
- Dashboard nodes which comprises of text nodes, gauge nodes, chart nodes has been used to serve the purpose of visualization to the user so that the user is able to know the live status of the home remotely.

This acts as a Graphical User Interface for the users so that they can monitor the live status of the home remotely. On this GUI, the user can control the online switch value based on his/ her presence. He/She can turn the switch to either ON or OFF based on his presence/absence. Whenever any email alerts are sent to the user stating any abnormalities, the user is able to monitor the system from any remote location.

VII. RESULTS

THIS chapter describes the results observed after the completion of relevant development tasks during the entire course of the application development which involves the following:

Reception of MQTT messages which are being published from the ESP module on motion detections at the raspberry Pi MQTT broker end.

Generation of trainingdata.csv and datalog.csv which combines four parameters namely timestamp, PIR MQTT motion message, online switch value, mobile mac value.

Successful upload of datalog.csv to the InfluxDB database.

LSTM prediction status based on the current five minute detection value. The model takes the current five minute detection count as the input and outputs the predicted five minute detection count for the next five minute interval and classifies the situation as normal or abnormal by comparing them over a custom percentage.

Historical Search status in every five minute interval by comparing the current five minute detection count with the past five minute detection count of that respective day which is stored in the InfluxDB "histroicaldb".

Statistical methods prediction status in every detection which classifies the detection as normal or abnormal based on the timing of the detection and the usual pattern. It checks if the detected timestamp lie inside or outside the inter-quartile range and if the detected timestamp deviates from the usual median value.

Weekly summary view which creates a box plot of detected timestamps of each day for four weeks. With this, the user can get a summary of timings when he is usually present/absent at home and can view any abnormal detection (outliers) happened on any day which he never observed at first place and hence, he/she can be cautious for this regard.

VIII. CONCLUSION AND FURTHER WORK

This scientific work focuses on the implementation of a Smart Home IoT use case, Burglar detection system with the use of motion sensor rather than the camera via the SCION network using the SCION IP gateway (SIG). The application involves a client server architecture where the SCION AS instance configured, MQTT broker enabled raspberry pi acts as the SCION AS server which receives the motion data from the remotely placed ESP module coupled PIR motion sensor over WiFi and the SCION AS instance configured local machine acts as the SCION AS client. This application posses wide range of development possibilities by employing more number of motion sensors so that the readings from each motion sensor can be sequenced together to derive the velocity ,direction of movement and even can suggest the approximate number of personnel present in an area. With this, it is possible to identify the intruder in a more accurate way and hence, the accuracy of the burglar detection can be improved. There is no need to employ cameras within the home to detect the burglar presence which is actually a security concern for the user as it disturbs the privacy of the user because the user might have a feeling of insecurity when he becomes conscious of the fact that a camera is being monitoring him continously.

REFERENCES

[1]SCIONLab Coordination Service, SCION is the first clean-slate Internet architecture designed to provide route control, failure isolation, and explicit trust information for end-to-end communication. Available: https://www.scionlab.org

[2]Raspberry Pi - A small and affordable computer that you can use to learn programming. Available: https://www.raspberrypi.org/

[3]balenaEtcher, An open source project by balena. Available: https://www.balena.io/etcher/

[4] Micro load cell data sheet
Available:
https://www.robotshop.com/media/files/pdf/datasheet3133.pdf

[5] HX711, 24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

[6] Node-Red, Flow-based programming for Internet of things. Available: https://nodered.org/

[7]Go, The Go Programming Language. Available: https://golang.org/

[8] Python, Python is a programming language that lets you work quickly and integrate systems more effectively. Available: https://www.python.org/

[9]MFRC522 Standard performance MIFARE and NTAG front end. Available: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

[10] Raspbian, Raspbian is the Foundation's official supported operating system. Available: https://www.raspberrypi.org/downloads/raspbian/

[11] docker, Build, Ship, and Run Any App, Anywhere. Learn about the only enterprise-ready container platform to cost-effectively build and manage your application Available :https://www.docker.com/

[12] Wireshark, Wireshark is the world's foremost and widely-used network protocol analyzer. Available: https://www.wireshark.org/

[13] SCION tutorials, Welcome to SCION Tutorials. Available: https://netsec-ethz.github.io/scion-tutorials/

[14] Ubuntu Mate for Raspberry Pi 2 and 3, Available: https://ubuntu-mate.org/raspberry-pi/

[15] Github, GitHub brings together the world's largest community of developers to discover, share, and build better software. Available: https://github.com/

[16] HX711-Go library, MichaelS11
Available : https://github.com/MichaelS11/go-hx711

[17] SCION Browser AS Visualizations. Available: https://netsec-ethz.github.io/scion-tutorials/as_visualization/browser_asviz/

[18] SCION bwtester application. Available :https://netsec-ethz.github.io/scion-tutorials/sample_projects/bwtester/

[19] Network Security Group at ETH Zürich. Scion-apps. Available: https://github.com/netsec-ethz/scion-apps

APPENDIX

(a)RFID_TCPserver.Py

```
import RPi.GPIO as GPIO
import MFRC522
import signal
from multiprocessing import Process, Value, Array
import Adafruit_DHT
import socket
import time
import json


continue_reading = True
sensor = Adafruit_DHT.DHT11
pin = 17
MIFAREReader = MFRC522.MFRC522()
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
sock.bind(('0.0.0.0',10000))
sock.listen(1)

def readTempAndHumidity(arr):
while continue_reading:
arr[1], arr[0] = Adafruit_DHT.read_retry(sensor, pin)
# if arr[0] is not None and arr[1] is not None:
#     print('Temp={0:0.1f}*  Humidity={1:0.1f}
%'.format(arr[0],arr[1]))
# else:
#     print('Failed to get reading. Try again!')
# if continue_reading == False:
#     return
time.sleep(2)

# Capture SIGINT for cleanup when the script is aborted
def end_read(signal,frame):
global continue_reading
# print "Ctrl+C captured, ending read."
continue_reading = False
sock.close()
#TempReadProcess.join()
GPIO.cleanup()

# Hook the SIGINT
signal.signal(signal.SIGINT, end_read)

arr = Array('d',(0.0,0.0))

TempReadProcess = Process(target=readTempAndHumidity,
args=(arr,))
TempReadProcess.start()
```

```
uid_str = "[]"
while True:
c,a = sock.accept()

while True:
data = c.recv(128)
temperature = int(arr[0])
humidity = int(arr[1])

if continue_reading:

# Scan for cards
(status,TagType) =
MIFAREReader.MFRC522_Request(MIFAREReader.PICC_
REQIDL)

# If a card is found

(status,uid) = MIFAREReader.MFRC522_Anticoll()
if status == MIFAREReader.MI_OK:
uid_str = str(uid)

(status,TagType) =
MIFAREReader.MFRC522_Request(MIFAREReader.PICC_
REQIDL)


data = {"UID" : str(uid),
"Temperature" : temperature,
"Humidity" : humidity}
print(str(data))
data = json.dumps(data)
c.send(bytes(data))
if not data:
c.close()
break
```

(b)weight_server_full_rv2.go

```
package main

import (
//"bufio"
"encoding/json"
"flag"
"fmt"
"log"
"net"
"time"

//"os"
//"strings"

"sync"

"github.com/MichaelS11/go-hx711"
"github.com/scionproto/scion/go/lib/sciond"
"github.com/scionproto/scion/go/lib/snet"
)

type Rfidth struct {
Humidity    int
UID         string
Temperature int
}

type Message struct {
Name       string
Exp        string
Time       string
Unitweight float64
Capacity   int
CurrentNo  int
TempL      int
TempH      int
HmdL       int
HmdH       int
Temp       int
Hmd        int
Wght       int
UID        string
}

func check(e error) {
if e != nil {
log.Fatal(e)
}
}

var weightData map[string]string
var weightDataLock sync.Mutex

func init() {
weightData = make(map[string]string)
}

// Obtains input from weight observation application

func printUsage() {
fmt.Println("weightserver -s ServerSCIONAddress")
fmt.Println("The SCION address is specified as ISD-
AS,[IP Address]:Port")
fmt.Println("Example SCION address 17-ffaa:0:1102,
[192.33.93.173]:42002")
}

func main() {

var (
serverAddress  string
sciondPath     string
sciondFromIA   bool
dispatcherPath string
```

```
err    error
server *snet.Addr

udpConnection snet.Conn
)

var msg = []Message{Message{"Sallos",
"10/10/2020", "", 5.4, 21, 0, 0, 40, 10, 90, 20, 30, 0, "[249, 20,
56, 86, 131]"},
Message{"Toffee", "20/10/2020", "", 4, 50,
0, 0, 50, 10, 90, 20, 30, 0, "[41, 106, 118, 72, 125]"}}
var noPallette = Message{"NoPallete", "", "", 0, 1, 0,
0, 50, 0, 100, 0, 0, 0, "[]"}
var sendData Message
sendData = noPallette
var rfth = Rfidth{UID: "[]"}
// Fetch arguments from command line
flag.StringVar(&serverAddress, "s", "", "Server
SCION Address")
flag.StringVar(&sciondPath, "sciond", "", "Path to
sciond socket")
flag.BoolVar(&sciondFromIA, "sciondFromIA",
false, "SCIOND socket path from IA address:ISD-AS")
flag.StringVar(&dispatcherPath, "dispatcher",
"/run/shm/dispatcher/default.sock",
"Path to dispatcher socket")
flag.Parse()

// Create the SCION UDP socket
if len(serverAddress) > 0 {
        server, err =
snet.AddrFromString(serverAddress)
        check(err)
} else {
        printUsage()
        check(fmt.Errorf("Error, server address
needs to be specified with -s"))
}

if sciondFromIA {
        if sciondPath != "" {
                log.Fatal("Only one of -sciond or -
sciondFromIA can be specified")
        }
        sciondPath =
sciond.GetDefaultSCIONDPath(&server.IA)
} else if sciondPath == "" {
        sciondPath =
sciond.GetDefaultSCIONDPath(nil)
}
snet.Init(server.IA, sciondPath, dispatcherPath)
udpConnection, err = snet.ListenSCION("udp4",
server)
check(err)

defer udpConnection.Close()

err = hx711.HostInit()

hx711, err := hx711.NewHx711("GPIO6", "GPIO5")
check(err)
// SetGain default is 128
// Gain of 128 or 64 is input channel A, gain of 32 is
input channel B
// hx711.SetGain(128)
// make sure to use your values from calibration
above
hx711.AdjustZero = -72782
hx711.AdjustScale = -1960

tcpreq := "hello from client"

tcpAddr, err := net.ResolveTCPAddr("tcp",
"0.0.0.0:10000")
if err != nil {
        panic(err)
}
conn, err := net.DialTCP("tcp", nil, tcpAddr)
defer conn.Close()
if err != nil {
        panic(err)
}

receivePacketBuffer := make([]byte, 2500)
sendPacketBuffer := make([]byte, 3000)
tcpReply := make([]byte, 1024)

for {
        _, clientAddress, err :=
udpConnection.ReadFrom(receivePacketBuffer)
        check(err)

        // Packet received, send back response to
same client

        //time.Sleep(200 * time.Microsecond)
        conn.Write([]byte(tcpreq))

        n, err := conn.Read(tcpReply)
        check(err)
        err = json.Unmarshal(tcpReply[:n], &rfth)
        check(err)
        if rfth.UID != "[]" {
                for i := 0; i < len(msg); i++ {
                        if string(rfth.UID) ==
(msg[i].UID) {
                                sendData =
msg[i]
                                break
                        }
                }
        }
        data, err := hx711.ReadDataMedian(11)
        check(err)

        if string(sendData.UID) != "[]" {
                sendData.CurrentNo = int(data /
sendData.Unitweight)
sendData.Time =
string(time.Now().Format("Jan 2 15:04:05"))
} else {
sendData = noPallette
}
sendData.Temp = rfth.Temperature
sendData.Hmd = rfth.Humidity
sendData.Wght = int(data)
b, _ := json.Marshal(sendData)
fmt.Println(string(b))

copy(sendPacketBuffer, b)

for i := 0; i < 3; i++ {
_, err =
udpConnection.WriteTo(sendPacketBuffer[:len(b)],
clientAddress)
if err == nil {
break
} else {
log.Println(err)
}
}
check(err)
sendData = noPallette
}

}
```

(c) weight_client_retry.go

```
package main

import (
"flag"
"fmt"
"log"
"net/http"

"github.com/scionproto/scion/go/lib/sciond"
"github.com/scionproto/scion/go/lib/snet"
)

func check(e error) {
if e != nil {
log.Fatal(e)
}
}

func printUsage() {
fmt.Println("scion-sensor-server -s
ServerSCIONAddress -c ClientSCIONAddress")
fmt.Println("The SCION address is specified as ISD-
AS,[IP Address]:Port")
fmt.Println("Example SCION address 1-1,
[127.0.0.1]:42002")
}

var (
clientAddress  string
serverAddress  string
sciondPath     string
sciondFromIA   bool
dispatcherPath string
udpConnection  snet.Conn
err            error
local          *snet.Addr
remote         *snet.Addr
)

func main() {

// Fetch arguments from command line
flag.StringVar(&clientAddress, "c", "", "Client
SCION Address")
flag.StringVar(&serverAddress, "s", "", "Server
SCION Address")
flag.StringVar(&sciondPath, "sciond", "", "Path to
sciond socket")
flag.BoolVar(&sciondFromIA, "sciondFromIA",
false, "SCIOND socket path from IA address:ISD-AS")
flag.StringVar(&dispatcherPath, "dispatcher",
"/run/shm/dispatcher/default.sock",
"Path to dispatcher socket")
flag.Parse()

// Create the SCION UDP socket
if len(clientAddress) > 0 {
local, err =
snet.AddrFromString(clientAddress)
check(err)
} else {
printUsage()
check(fmt.Errorf("Error, client address
needs to be specified with -c"))
}
if len(serverAddress) > 0 {
remote, err =
snet.AddrFromString(serverAddress)
check(err)
} else {
printUsage()
check(fmt.Errorf("Error, server address
needs to be specified with -s"))
}

if sciondFromIA {
if sciondPath != "" {
log.Fatal("Only one of -sciond or -
sciondFromIA can be specified")
}
sciondPath =
sciond.GetDefaultSCIONDPath(&local.IA)
} else if sciondPath == "" {
sciondPath =
sciond.GetDefaultSCIONDPath(nil)
```

```
                                    }
              snet.Init(local.IA, sciondPath, dispatcherPath)
          udpConnection, err = snet.DialSCION("udp4", local,
                              remote)
                                 check(err)

                    http.HandleFunc("/", dataHandler)

                    http.ListenAndServe(":4000", nil)

                                 }

func dataHandler(w http.ResponseWriter, r *http.Request) {
              receivePacketBuffer := make([]byte, 2500)
                 sendPacketBuffer := make([]byte, 0)
                                   n := 0
                         for i := 0; i < 3; i++ {
                                     n, err =
           udpConnection.Write(sendPacketBuffer)
                                    check(err)

                                    n, _, err =
       udpConnection.ReadFrom(receivePacketBuffer)
                                  if err == nil {
                                         break
                                     } else {
                                      log.Println(err)
                                         }
                                    }

                                 check(err)

              fmt.Println(string(receivePacketBuffer[:n]))

                   w.Write(receivePacketBuffer[:n])
                                 }
```

(d)SCION Raspberry installation method