J COMPONENT PROJECT REPORT

Vigilance in Mobile Camps Using MQTT Protocol

Submitted for ECE4011 – WIRELESS SENSOR NETWORKS

BY –

K VAMSI KRISHNA	–	15BEC0049
P HAMSA DATTA	–	15BEC0173

Slot – E1

Submitted under the guidance of **PROF. SUNDAR S**

VIT® UNIVERSITY (Estd. u/s 3 of UGC Act 1956) VELLORE ▪ CHENNAI www.vit.ac.in

MARCH, 2018

ACKNOWLEDGEMENTS

We sincerely thank the management of VIT University, The Dean of SENSE School, and most importantly, Prof. Sundar S for his continued support in completing our project successful on time.

We extend our gratitude to VIT University, for giving us this wonderful opportunity, wherein we have learnt so much in the due course of completion of the project.

Sincerely

K VAMSI KRISHNA	– 15BEC0049
P HAMSA DATTA	– 15BEC0173

TABLE OF CONTENTS

ABSTRACT

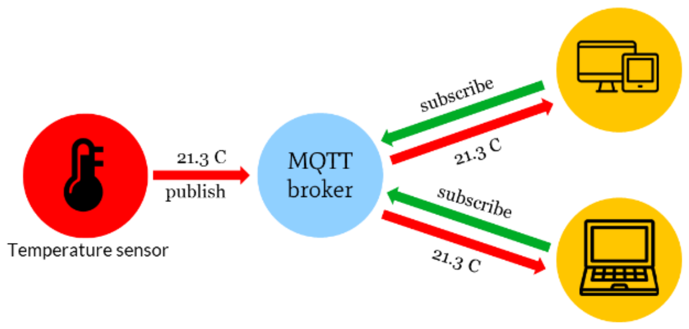
Wireless Sensor Networks (WSN) is emerging rapidly and we can use them as a powerful tool for connecting the physical and digital world. WSN tend to be a great deal in mission-critical applications. The scenario of our project is a WSN with one master node (inside the camp – Raspberry Pi) and three communicating nodes (placed around the camp – NodeMCU's), each interfaced with sensors. The striking aspect of the project is the use of MQTT protocol for communication with a star topology, which is generally an IoT protocol and is popular for its reliability in low power devices. Finally, incorporating a mobile node (associated to individual soldier to communicate with base while on run) would create an Ad Hoc network scenario. All the communicating nodes and the master node are connected through a router, with the master node serving as Broker cum Server. As in short, the project has WSN with IoT protocol in Ad Hoc scenario.

AIM AND OBJECTIVE

- Our chief objective is to create a Wireless Sensor Network with the least bandwidth usage and low power consumption providing best results and robustness.
- The concepts we are using here are MQTT Protocol and Wireless Networking.
- There will be 3 wireless nodes in the scenario, among which two are immobile and the remaining one is a mobile node. Each node is interfaced with a specific set of sensors.
- All these nodes are connected to a central server to control and monitor.
- Our expected output will be an effective Wireless Sensor Network with 3 Client nodes and One Server node, each communicating with the server node.

MQTT Protocol

MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.



Schematic data flow from sensor (machine) to devise (machine)

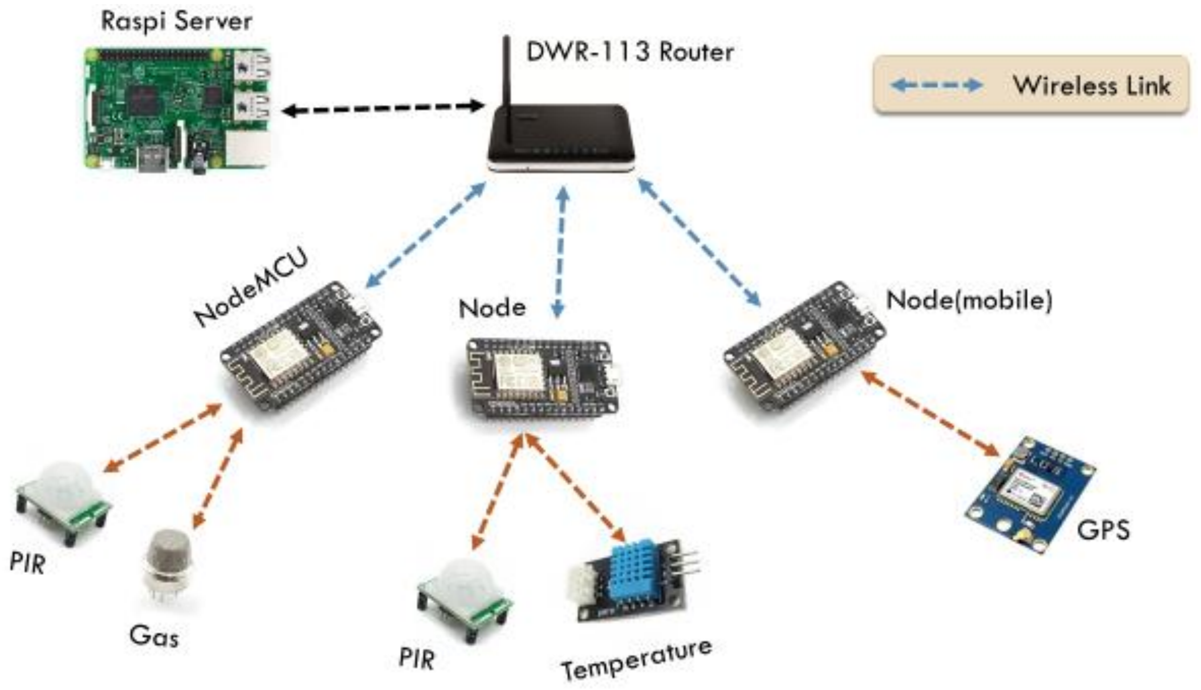
Client:

The client includes publisher or subscribers, both of them label an MQTT client that is only doing publishing or subscribing. In general, an MQTT client can be both a publisher & subscriber at the same time. An MQTT client is any device from a micro controller up to a fully-fledged server that has an MQTT library running and is connecting to an MQTT broker over any kind of network. This could be a really small and resource constrained device that is connected over a wireless network and has a library strapped to the minimum or a typical computer running a graphical MQTT client for testing purposes, basically any device that has a TCP/IP stack and speaks MQTT over it. The client implementation of the MQTT protocol is very straight-forward and really reduced to the essence. That’s one aspect, why MQTT is ideally suitable for small devices.

Broker:

The counterpart to an MQTT client is the MQTT broker, which is the heart of any publish/subscribe protocol. Depending on the concrete implementation, a broker can handle up to thousands of concurrently connected MQTT clients. The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients. It also holds the session of all persisted clients including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients. And at most of the times, a broker is also extensible, which allows to easily integrate custom authentication, authorization, and integration into backend systems. Especially the integration is an important aspect, because often the broker is the component, which is directly exposed on the internet and handles a lot of clients and then passes messages along to downstream analyzing and processing systems.

Block Diagram



Components Description

Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

DWR-113 Router

DWR-113 is a Wi-Fi 150N pocket router, it supports 3G dongle up to HSPA+. The router features include NAT, Routing, Firewall, VPN pass-through, WPS, Auto-3G-Dial-Up Backup Connection, DHCP Server and so on. It is managed easily by Web UI and is UPnP. SPI Firewall, Packet Filtering and Access Control is embedded in the product. It supports various connection approaches to ISP such as Static IP, Dynamic IP (DHCP client), PPTP, L2TP, PPPoE, and 3G. It is also compliant with IEEE 802.11n/b/g standard for WLAN.

NodeMCU

The NodeMCU (Node Microcontroller Unit) is an open source software and hardware development environment that is built around a very inexpensive System-on-a-Chip (SoC) called the ESP8266. The ESP8266, designed and manufactured by Espressif Systems, contains all crucial elements of the modern computer: CPU, RAM, networking (Wi-Fi), and even a modern operating system and SDK. It uses many open source projects, such as lua-cjson, and spiffs. It includes firmware which runs on the ESP8266 Wi-Fi SoC, and hardware which is based on the ESP-12 module. NodeMCU can be programmed using Arduino IDE.

PIR Sensor

A passive infrared sensor (PIR sensor) is an electronic sensor that measures infrared (IR) radiation is emitted from objects in its field of view. They are most often used in PIR-based motion detectors. When the sensor is idle, both slots detect the same amount of IR, the ambient amount radiated from the room or walls or outdoors. When a warm body like a human or animal passes by, it first intercepts one half of the PIR sensor, which causes a positive differential change between the two halves.

Gas Sensor

The Gas Sensor(MQ2) module is useful for gas leakage detection (home and industry). It is suitable for detecting H2, LPG, CH4, CO, Alcohol, Smoke or Propane. Due to its high sensitivity and fast response time, measurement can be taken as soon as possible. The sensitivity of the sensor can be adjusted by the potentiometer. This is an Analog output sensor. The output voltage from the Gas sensor increases when the concentration of gas increases.

Temperature Sensor

The DHT11 is a basic, ultra-low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal on the data pin (no analog input pins needed). Its fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old.

GPS Sensor

The GPS module for Arduino and Raspberry Pi is a small electronic circuit that allows connecting to your Arduino/Raspberry Pi board to get position and altitude, as well as speed, date and time on UTC (Universal Time Coordinated). It uses the standard NMEA protocol to transmit the position data via serial port.

CODES

Server

MQTT Receiver Code

```
def c_broker():
            import os
            import time
            os.system('sudo /etc/init.d/mosquitto start')

def main1():
       import paho.mqtt.client as mqtt
       import paho.mqtt.subscribe as subscribe
       import os
       import time
       msg = subscribe.simple("test1", hostname="192.168.0.50")
       tem=str(msg.payload)
       return tem

def main2():
              import paho.mqtt.client as mqtt
              import paho.mqtt.subscribe as subscribe
              import os
              import time
              msg = subscribe.simple("test2",
hostname="192.168.0.50")
              tem=str(msg.payload)
              return tem

def main3():
            import paho.mqtt.client as mqtt
            import paho.mqtt.subscribe as subscribe
            import os
            import time
            msg = subscribe.simple("test3", hostname="192.168.0.50")
            tem=str(msg.payload)
            return tem

def main4():
                  import paho.mqtt.client as mqtt
                  import paho.mqtt.subscribe as subscribe
                  import os
                  import time
                  msg = subscribe.simple("test4",
hostname="192.168.0.50")
                  tem=str(msg.payload)
                  return tem

def main5():
                  import paho.mqtt.client as mqtt
                  import paho.mqtt.subscribe as subscribe
                  import os
```

```
                 import time
                 msg = subscribe.simple("test5",
hostname="192.168.0.50")
                 tem=str(msg.payload)
                 return tem

def main6():
                 import paho.mqtt.client as mqtt
                 import paho.mqtt.publish as publish
                 import os
                 import time

                 publish.single("test6", "1",
hostname="192.168.0.50")

def main7():
                 import paho.mqtt.client as mqtt
                 import paho.mqtt.subscribe as subscribe
                 import os
                 import time
                 msg = subscribe.simple("test7",
hostname="192.168.0.50")
                 tem=str(msg.payload)
                 return tem

def main8():
                 import paho.mqtt.client as mqtt
                 import paho.mqtt.subscribe as subscribe
                 import os
                 import time
                 msg = subscribe.simple("test8",
hostname="192.168.0.50")
                 tem=str(msg.payload)
                 return tem

def main9():
                 import paho.mqtt.client as mqtt
                 import paho.mqtt.subscribe as subscribe
                 import os
                 import time
                 msg = subscribe.simple("test9",
hostname="192.168.0.50")
                 tem=str(msg.payload)
                 return tem

def main10():
                 import paho.mqtt.client as mqtt
                 import paho.mqtt.publish as publish
                 import os
                 import time
```

```
                    publish.single("test6", "f",hostname="192.168.0.50")
```

GUI Code

```
########

from PyQt4 import QtCore, QtGui
import simple_mqtt_test

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig,
_encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class Ui_MainWindow(object):
      def setupUi(self, MainWindow):
                  MainWindow.setObjectName(_fromUtf8("MainWindow"))
                  MainWindow.resize(657, 380)
                  MainWindow.setMinimumSize(QtCore.QSize(657, 380))
                  MainWindow.setMaximumSize(QtCore.QSize(657, 380))
                  MainWindow.setIconSize(QtCore.QSize(24, 24))
                  self.centralwidget = QtGui.QWidget(MainWindow)

      self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
                  self.gridLayout =
QtGui.QGridLayout(self.centralwidget)

      self.gridLayout.setObjectName(_fromUtf8("gridLayout"))
                  self.label = QtGui.QLabel(self.centralwidget)
                  self.label.setMinimumSize(QtCore.QSize(162, 21))
                  font = QtGui.QFont()
                  font.setPointSize(13)
                  self.label.setFont(font)
                  self.label.setTextFormat(QtCore.Qt.PlainText)
                  self.label.setObjectName(_fromUtf8("label"))
                  self.gridLayout.addWidget(self.label, 0, 0, 1, 1)
                  self.pushButton_ConBrok =
QtGui.QPushButton(self.centralwidget)

      self.pushButton_ConBrok.setMinimumSize(QtCore.QSize(125, 23))
      self.pushButton_ConBrok.setObjectName(_fromUtf8("pushButton_Con
Brok"))
                  self.gridLayout.addWidget(self.pushButton_ConBrok,
0, 1, 1, 1)
                  self.Status = QtGui.QLineEdit(self.centralwidget)
                  self.Status.setMinimumSize(QtCore.QSize(130, 20))
                  self.Status.setReadOnly(True)
                  self.Status.setObjectName(_fromUtf8("Status"))
                  self.gridLayout.addWidget(self.Status, 0, 2, 1, 2)
                  self.groupBox = QtGui.QGroupBox(self.centralwidget)
                  self.groupBox.setMinimumSize(QtCore.QSize(201, 291))
                  self.groupBox.setObjectName(_fromUtf8("groupBox"))
                  self.label_2 = QtGui.QLabel(self.groupBox)
                  self.label_2.setGeometry(QtCore.QRect(10, 30, 71,
16))
                  self.label_2.setObjectName(_fromUtf8("label_2"))
                  self.lineEdit_temp = QtGui.QLineEdit(self.groupBox)
                  self.lineEdit_temp.setGeometry(QtCore.QRect(80, 30,
41, 20))
                  self.lineEdit_temp.setAcceptDrops(False)
                  self.lineEdit_temp.setReadOnly(True)

      self.lineEdit_temp.setObjectName(_fromUtf8("lineEdit_temp"))
                  self.label_3 = QtGui.QLabel(self.groupBox)
                  self.label_3.setGeometry(QtCore.QRect(10, 80, 61,
16))
                  self.label_3.setObjectName(_fromUtf8("label_3"))
                  self.lineEdit_hum = QtGui.QLineEdit(self.groupBox)
                  self.lineEdit_hum.setGeometry(QtCore.QRect(80, 80,
41, 20))
                  self.lineEdit_hum.setReadOnly(True)

      self.lineEdit_hum.setObjectName(_fromUtf8("lineEdit_hum"))
                  self.label_4 = QtGui.QLabel(self.groupBox)
                  self.label_4.setGeometry(QtCore.QRect(10, 140, 46,
13))
                  self.label_4.setObjectName(_fromUtf8("label_4"))
                  self.lineEdit_motion =
QtGui.QLineEdit(self.groupBox)
                  self.lineEdit_motion.setGeometry(QtCore.QRect(80,
140, 90, 20))
                  self.lineEdit_motion.setReadOnly(True)

      self.lineEdit_motion.setObjectName(_fromUtf8("lineEdit_motion")
)
                  self.gridLayout.addWidget(self.groupBox, 1, 0, 1, 1)
                  self.groupBox_2 =
QtGui.QGroupBox(self.centralwidget)
                  self.groupBox_2.setMinimumSize(QtCore.QSize(201,
291))
```

```
      self.groupBox_2.setObjectName(_fromUtf8("groupBox_2"))
                  self.lineEdit_smoke =
QtGui.QLineEdit(self.groupBox_2)
                  self.lineEdit_smoke.setGeometry(QtCore.QRect(80, 30,
90, 20))
                  self.lineEdit_smoke.setAcceptDrops(False)
                  self.lineEdit_smoke.setText(_fromUtf8(""))
                  self.lineEdit_smoke.setReadOnly(True)

      self.lineEdit_smoke.setObjectName(_fromUtf8("lineEdit_smoke"))
                  self.label_5 = QtGui.QLabel(self.groupBox_2)
                  self.label_5.setGeometry(QtCore.QRect(10, 30, 71,
16))
                  self.label_5.setObjectName(_fromUtf8("label_5"))
                  self.lineEdit_motion_2 =
QtGui.QLineEdit(self.groupBox_2)
                  self.lineEdit_motion_2.setGeometry(QtCore.QRect(80,
80, 90, 20))
                  self.lineEdit_motion_2.setReadOnly(True)

      self.lineEdit_motion_2.setObjectName(_fromUtf8("lineEdit_motion
_2"))
                  self.label_6 = QtGui.QLabel(self.groupBox_2)
                  self.label_6.setGeometry(QtCore.QRect(10, 80, 46,
13))
                  self.label_6.setObjectName(_fromUtf8("label_6"))
                  self.gridLayout.addWidget(self.groupBox_2, 1, 1, 1,
2)
                  self.groupBox_3 =
QtGui.QGroupBox(self.centralwidget)
                  self.groupBox_3.setMinimumSize(QtCore.QSize(211,
291))

      self.groupBox_3.setObjectName(_fromUtf8("groupBox_3"))
                  self.label_7 = QtGui.QLabel(self.groupBox_3)
                  self.label_7.setGeometry(QtCore.QRect(10, 30, 81,
16))
                  self.label_7.setObjectName(_fromUtf8("label_7"))
                  self.label_8 = QtGui.QLabel(self.groupBox_3)
                  self.label_8.setGeometry(QtCore.QRect(10, 62, 81,
16))
                  self.label_8.setObjectName(_fromUtf8("label_8"))
                  self.label_9 = QtGui.QLabel(self.groupBox_3)
                  self.label_9.setGeometry(QtCore.QRect(10, 92, 81,
16))
                  self.label_9.setObjectName(_fromUtf8("label_9"))
                  self.label_10 = QtGui.QLabel(self.groupBox_3)
                  self.label_10.setGeometry(QtCore.QRect(10, 122, 81,
16))
                  self.label_10.setObjectName(_fromUtf8("label_10"))
                  self.label_11 = QtGui.QLabel(self.groupBox_3)
```

```
                 self.label_11.setGeometry(QtCore.QRect(120, 123, 81,
16))
                 self.label_11.setObjectName(_fromUtf8("label_11"))
                 self.lineEdit_GPSLoc =
QtGui.QLineEdit(self.groupBox_3)
                 self.lineEdit_GPSLoc.setGeometry(QtCore.QRect(70,
60, 80,20))
                 self.lineEdit_GPSLoc.setReadOnly(True)

      self.lineEdit_GPSLoc.setObjectName(_fromUtf8("lineEdit_GPSLoc")
)
                 self.lineEdit_GPSLon =
QtGui.QLineEdit(self.groupBox_3)
                 self.lineEdit_GPSLon.setGeometry(QtCore.QRect(70,
90, 80,20))
                 self.lineEdit_GPSLon.setReadOnly(True)

      self.lineEdit_GPSLon.setObjectName(_fromUtf8("lineEdit_GPSLon")
)
                 self.lineEdit_GPSDis =
QtGui.QLineEdit(self.groupBox_3)
                 self.lineEdit_GPSDis.setGeometry(QtCore.QRect(70,
120, 50,20))
                 self.lineEdit_GPSDis.setReadOnly(True)

      self.lineEdit_GPSDis.setObjectName(_fromUtf8("lineEdit_GPSDis")
)
                 self.pushButton_Loc =
QtGui.QPushButton(self.groupBox_3)
                 self.pushButton_Loc.setGeometry(QtCore.QRect(10,
160, 75, 23))

      self.pushButton_Loc.setObjectName(_fromUtf8("pushButton_Loc"))
                 self.pushButton_FB =
QtGui.QPushButton(self.groupBox_3)
                 self.pushButton_FB.setGeometry(QtCore.QRect(70, 250,
75, 23))

      self.pushButton_FB.setObjectName(_fromUtf8("pushButton_FB"))
                 self.gridLayout.addWidget(self.groupBox_3, 1, 3, 1,
1)
                 MainWindow.setCentralWidget(self.centralwidget)
                 self.statusbar = QtGui.QStatusBar(MainWindow)
                 self.statusbar.setObjectName(_fromUtf8("statusbar"))
                 MainWindow.setStatusBar(self.statusbar)
                 self.actionClose = QtGui.QAction(MainWindow)

      self.actionClose.setObjectName(_fromUtf8("actionClose"))

                 self.retranslateUi(MainWindow)
                 QtCore.QObject.connect(self.pushButton_ConBrok,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.now_this)
                 QtCore.QObject.connect(self.pushButton_Loc,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.now_loc)
                 QtCore.QObject.connect(self.pushButton_FB,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.now_fb)
                 #QtCore.QObject.connect(self.pushButton_ConBrok,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.now_this)
                 QtCore.QObject.connect(self.pushButton_Loc,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.pushButton_Loc.toggle)
                 QtCore.QObject.connect(self.pushButton_FB,
QtCore.SIGNAL(_fromUtf8("clicked()")), self.pushButton_FB.toggle)
                 QtCore.QMetaObject.connectSlotsByName(MainWindow)


      def now_this(self):
                 print('this is just test!!!')
                 simple_mqtt_test.c_broker()
                 self.Status.setText(_translate("MainWindow",
"connected", None))

      def now_loc(self):
                       simple_mqtt_test.main6()

      def now_fb(self):
                       simple_mqtt_test.main10()

      def retranslateUi(self, MainWindow):
                 MainWindow.setWindowTitle(_translate("MainWindow", "
Surviellance", None))
                 self.label.setText(_translate("MainWindow", "Connect
to the Broker", None))

      self.pushButton_ConBrok.setText(_translate("MainWindow",
"Connect", None))
                 #self.Status.setText(_translate("MainWindow",
"statuscon", None))

      self.Status.setPlaceholderText(_translate("MainWindow",
"Status", None))
                 self.groupBox.setTitle(_translate("MainWindow",
"Node 1", None))
                 self.label_2.setText(_translate("MainWindow",
"Temperature:", None))

      self.lineEdit_temp.setPlaceholderText(_translate("MainWindow",
"*C", None))
                 self.label_3.setText(_translate("MainWindow",
"Humidity:", None))

      self.lineEdit_hum.setPlaceholderText(_translate("MainWindow",
"%", None))
```

```
                 self.label_4.setText(_translate("MainWindow",
"Motion:", None))

      self.lineEdit_motion.setPlaceholderText(_translate("MainWindow"
, "Neutral", None))
                 self.groupBox_2.setTitle(_translate("MainWindow",
"Node 2", None))

      self.lineEdit_smoke.setPlaceholderText(_translate("MainWindow",
"Normal", None))
                 self.label_5.setText(_translate("MainWindow",
"Smoke", None))

      self.lineEdit_motion_2.setPlaceholderText(_translate("MainWindo
w", "Neutral", None))
                 self.label_6.setText(_translate("MainWindow",
"Motion:", None))
                 self.groupBox_3.setTitle(_translate("MainWindow",
"Node 3", None))
                 self.label_7.setText(_translate("MainWindow", "GPS
Location", None))
                 self.label_8.setText(_translate("MainWindow",
"Latitude:", None))
                 self.label_9.setText(_translate("MainWindow",
"Longitude:", None))
                 self.label_10.setText(_translate("MainWindow",
"Distance:", None))
                 self.label_11.setText(_translate("MainWindow", "Kms
from Base", None))

      self.lineEdit_GPSLoc.setPlaceholderText(_translate("MainWindow"
, "Latitude", None))

      self.lineEdit_GPSLon.setPlaceholderText(_translate("MainWindow"
, "Longitude", None))

      self.lineEdit_GPSDis.setPlaceholderText(_translate("MainWindow"
, "Dist.", None))
                 self.pushButton_Loc.setText(_translate("MainWindow",
"Get Location", None))
                 self.pushButton_FB.setText(_translate("MainWindow",
"Fall Back", None))
                 self.actionClose.setText(_translate("MainWindow",
"Close", None))


if __name__ == "__main__":
    import sys
    app = QtGui.QApplication(sys.argv)
    MainWindow = QtGui.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
```

```
        MainWindow.show()
        sys.exit(app.exec_())
```

Main Code

```
################

from PyQt4 import QtCore,QtGui
import gui
import sys
#import sysinfo
import simple_mqtt_test


class MainUiClass(QtGui.QMainWindow, gui.Ui_MainWindow):
      def __init__(self, parent=None):
            super(MainUiClass, self).__init__(parent)
            self.setupUi(self)
            self.threadclass=ThreadClass()
            self.threadclass.start()
            self.threadclass2=ThreadClass2()
            self.threadclass2.start()
            self.threadclass3=ThreadClass3()
            self.threadclass3.start()
            self.connect(self.threadclass,
QtCore.SIGNAL('temp1'),self.updateValue)
            self.connect(self.threadclass,
QtCore.SIGNAL('hum1'),self.updateValue2)
            self.connect(self.threadclass,
QtCore.SIGNAL('motion1'),self.updateValue3)
            self.connect(self.threadclass2,
QtCore.SIGNAL('gas2'),self.updateValue4)
            self.connect(self.threadclass2,
QtCore.SIGNAL('motion2'),self.updateValue5)
            self.connect(self.threadclass3,
QtCore.SIGNAL('lat1'),self.updateValue6)
            self.connect(self.threadclass3,
QtCore.SIGNAL('lon1'),self.updateValue7)
            self.connect(self.threadclass3,
QtCore.SIGNAL('dis1'),self.updateValue8)


      def updateValue(self,temp):
            self.lineEdit_temp.setText(str(temp))

      def updateValue2(self,hum):
            self.lineEdit_hum.setText(str(hum))

      def updateValue3(self,md1):
            self.lineEdit_motion.setText(str(md1))
```

```
    def updateValue4(self,gas):
        self.lineEdit_smoke.setText(str(gas))

    def updateValue5(self,md2):
        self.lineEdit_motion_2.setText(str(md2))

    def updateValue6(self,lat):
        self.lineEdit_GPSLoc.setText(str(lat))

    def updateValue7(self,lon):
        self.lineEdit_GPSLon.setText(str(lon))

    def updateValue8(self,dis):
        self.lineEdit_GPSDis.setText(str(dis))

class ThreadClass(QtCore.QThread):
    def __init__(self, parent=None):
        super(ThreadClass, self).__init__(parent)

    def run(self):
        while 1:
            temp=simple_mqtt_test.main1()
            hum=simple_mqtt_test.main2()
            md1=simple_mqtt_test.main3()
            self.emit(QtCore.SIGNAL('temp1'),temp)
            self.emit(QtCore.SIGNAL('hum1'),hum)
            self.emit(QtCore.SIGNAL('motion1'),md1)

class ThreadClass2(QtCore.QThread):
    def __init__(self, parent=None):
        super(ThreadClass2, self).__init__(parent)

    def run(self):
        while 1:
            gas = simple_mqtt_test.main4()
            md2=simple_mqtt_test.main5()
            self.emit(QtCore.SIGNAL('gas2'),gas)
            self.emit(QtCore.SIGNAL('motion2'),md2)



class ThreadClass3(QtCore.QThread):
    def __init__(self, parent=None):
        super(ThreadClass3, self).__init__(parent)

    def run(self):
        while 1:
            lat=simple_mqtt_test.main7()
            lon=simple_mqtt_test.main8()
            dis=simple_mqtt_test.main9()
            self.emit(QtCore.SIGNAL('lat1'),lat)
```

```
                  self.emit(QtCore.SIGNAL('lon1'),lon)
                  self.emit(QtCore.SIGNAL('dis1'),dis)



if __name__=='__main__':
      a=QtGui.QApplication(sys.argv)
      app=MainUiClass()
      app.show()
      a.exec_()
```

Clients

Node 1

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <String.h>
#include <dht.h>
dht DHT;

#define dhtp 5// dht11 pin - nodemcu 1
int pirp=4; // output of pir sensor- nodemcu 2

// Update these with values suitable for your network.
const char* ssid = "vamsi1";
const char* password = "12345678";
const char* mqtt_server = "192.168.0.50";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
char msg2[50];
//////////////////////////////////////////////////////////////////////

void setup_wifi()

{

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

////////////////////////////////////////////////////////////////////

void reconnect()
{
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "node1";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("test2", "hello world");
      // ... and resubscribe
      client.subscribe("inTopic");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

////////////////////////////////////////////////////////////////////

void setup()
{
  pinMode(pirp, INPUT);// setting pir output as arduino input

  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

////////////////////////////////////////////////////////////////////

void loop()
{
  if (!client.connected())
  {
    reconnect();
  }
  client.loop();

  long now = millis();
  if (now - lastMsg > 2000)
  {
    lastMsg = now;
    char msg3[]="Motion Detected";
    char msg4[]="Neutral";

    int chk = DHT.read11(dhtp);
    snprintf (msg, 75, "%.2f", DHT.temperature);//test1-temp
    Serial.println(msg);
    client.publish("test1", msg);

    snprintf (msg2, 75, "%.2f", DHT.humidity);
    Serial.println(msg2);
    client.publish("test2", msg2);//test2-humidity

    if(digitalRead(pirp) == HIGH) // reading the data from the pir
sensor
    {
     Serial.println("Motion detected");
     client.publish("test3", msg3);//test3-intrusion
    }
    else {
     Serial.println("No Motion detected");
     client.publish("test3", msg4);//test5-intrusion
    }
    delay(2000);
  }
}
```

Node 2

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <String.h>


int gasPin = A0;//gas sensor output to analog pin
int sensorThres = 900;//threshold value
int pirp=4; // output of pir sensor- nodemcu 2

// Update these with values suitable for your network.
const char* ssid = "vamsi1";
const char* password = "12345678";
const char* mqtt_server = "192.168.0.50";
```

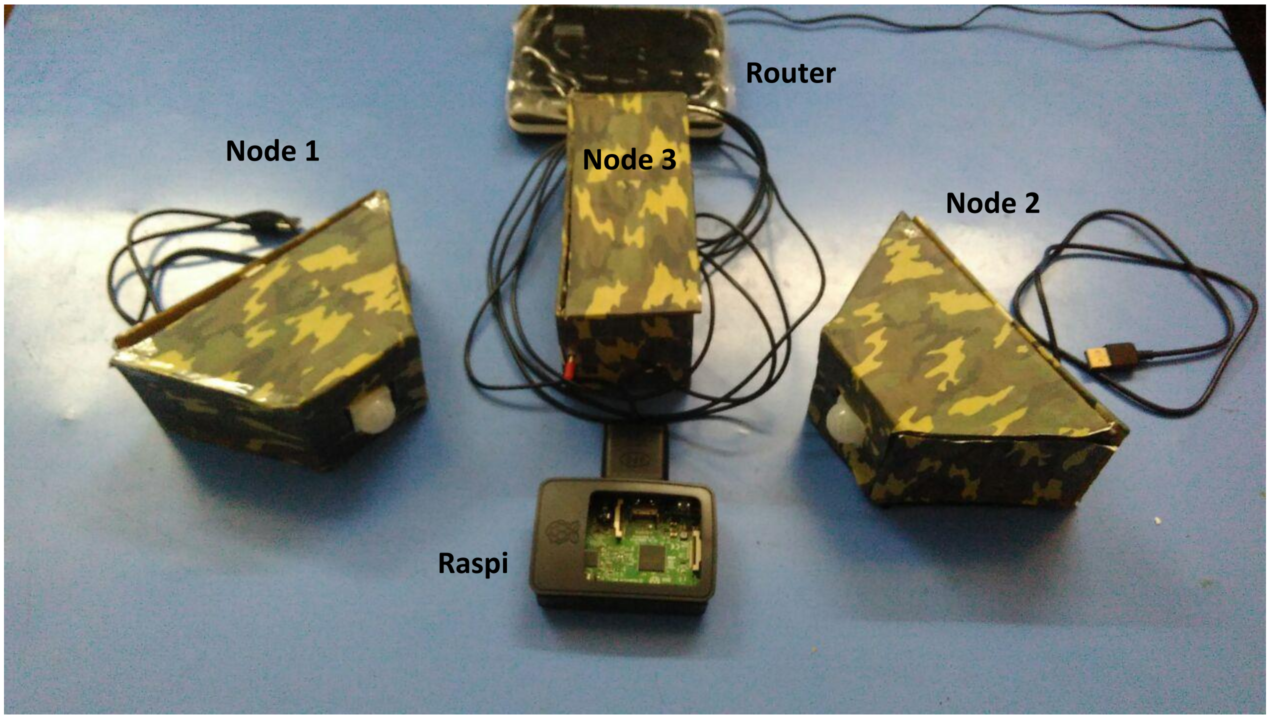
```
WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
char msg2[50];
/////////////////////////////////////////////////////////////////////

void setup_wifi()

{

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}


/////////////////////////////////////////////////////////////////////

void reconnect()
{
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "node1";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("test2", "hello world");
      // ... and resubscribe
      client.subscribe("inTopic");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

//////////////////////////////////////////////////////////////////////

void setup()
{
  pinMode(gasPin, INPUT);
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

//////////////////////////////////////////////////////////////////////

void loop()
{

  if (!client.connected())
  {
    reconnect();
  }
  client.loop();

  long now = millis();
  if (now - lastMsg > 2000)
  {
    lastMsg = now;
    char msg3[]="Motion Detected";
    char msg2[]="Neutral";
    char msg4[]="Smoke Detected";
    char msg5[]="Neutral";
    int gsvalue = analogRead(gasPin);

    if(gsvalue>sensorThres) // reading the data from the pir sensor
    {
     Serial.println("Smoke detected");
     client.publish("test4", msg4);//test4-smoke
    }
    else {
     Serial.println("Smoke Neutral");
     client.publish("test4", msg5);
    }

    if(digitalRead(pirp) == HIGH) // reading the data from the pir
sensor
    {
     Serial.println("Motion detected");
     client.publish("test5", msg3);//test5-intrusion
    }
    else {
     Serial.println("No Motion detected");
     client.publish("test5", msg2);//test5-intrusion
    }
    delay(1000);
  }
}
```

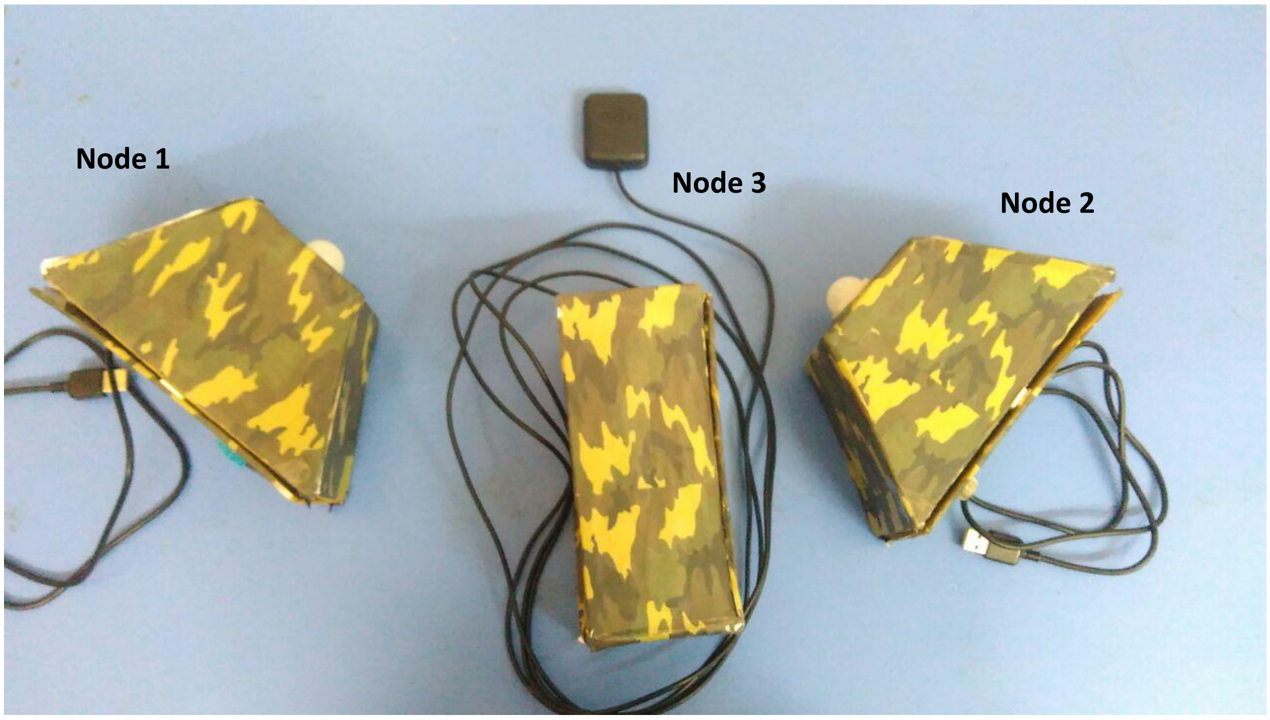
Node 3

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#define LED D0
char msg[9];
char msg1[9];
char msg2[9];
static const double LONDON_LAT = 13.61050457, LONDON_LON = 79.42950;

static const int RXPin = 12, TXPin = 13;
static const uint32_t GPSBaud = 9600;

// The TinyGPS++ object
TinyGPSPlus gps;

// The serial connection to the GPS device
SoftwareSerial ss(RXPin, TXPin);

const char* ssid = "vamsi1";
const char* password = "12345678";
const char* mqtt_server = "192.168.0.50";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
int value = 0;

void setup()
{
  Serial.begin(115200);
  ss.begin(GPSBaud);
  pinMode(LED,OUTPUT);
  digitalWrite(LED, HIGH);
  setup_wifi();
  client.setServer(mqtt_server, 1883);

}
```

```
void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("ESP8266Client")) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("outTopic", "hello world");
      // ... and resubscribe
      client.subscribe("inTopic");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Switch on the LED if an 1 was received as first character
  if ((char)payload[0] == '1') {
    snprintf (msg, 9, "%2.6f", gps.location.lat());
    Serial.println(msg);
    snprintf (msg1, 9, "%3.6f", gps.location.lng());
    Serial.println(msg1);
    unsigned long distanceKmToLondon =(unsigned
long)TinyGPSPlus::distanceBetween(gps.location.lat(),gps.location.lng
(),LONDON_LAT, LONDON_LON) / 1000;
    snprintf (msg2, 9, "%d", distanceKmToLondon);
    Serial.println(msg2);
    client.publish("test7",msg); //test7=lat
    client.publish("test8",msg1); //test8=lon
    client.publish("test9",msg2); //test9=dist
  }
  else if((char)payload[0] == 'f') {

       digitalWrite(LED,LOW);
       delay(2000);
       digitalWrite(LED,HIGH);
  }
}


void loop()
{

  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  long now = millis();
  if (now - lastMsg > 2000) {

  lastMsg = now;
  client.subscribe("test6");
  client.setCallback(callback);
  smartDelay(1000);
  }
}

// This custom version of delay() ensures that the gps object
// is being "fed".
static void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}
```

System Setup

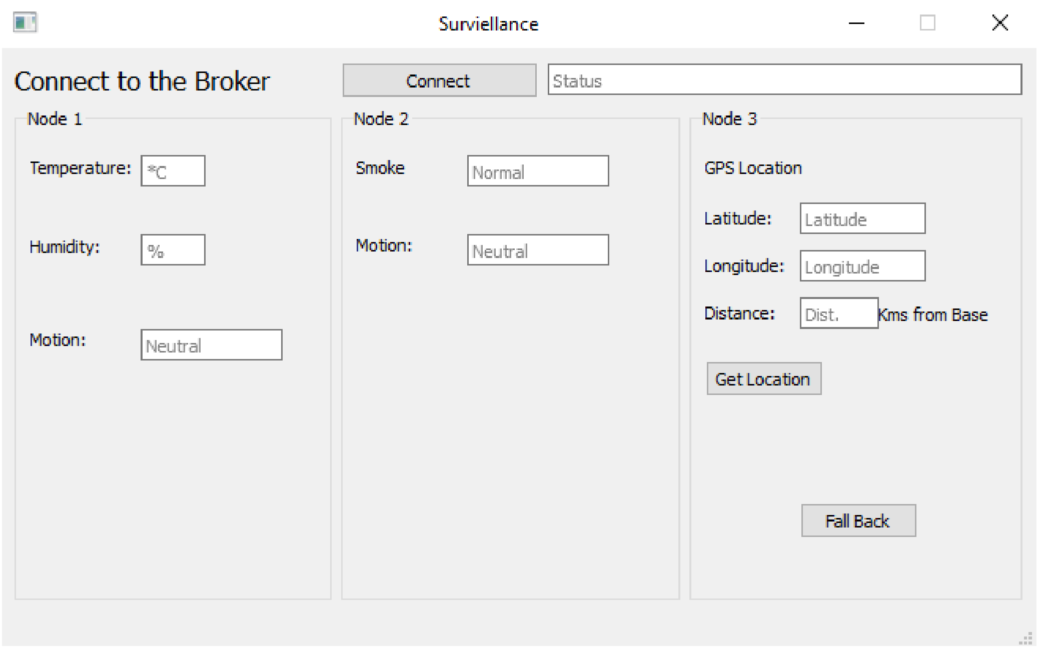




Results

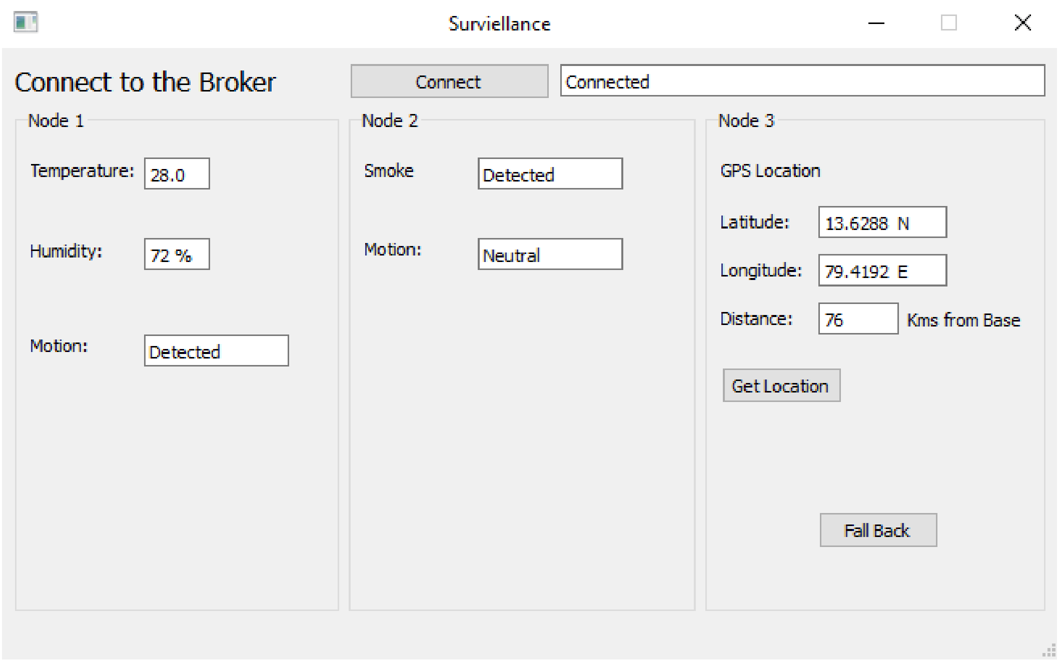
Graphical User Interface – Monitoring Nodes

Normal State



Status	Status
Temperature:	°C
Humidity:	%
Motion:	Neutral
Smoke	Normal
Motion:	Neutral
Latitude:	Latitude
Longitude:	Longitude
Distance:	Dist. Kms from Base

When the Nodes are Connected



Status	Connected
Temperature:	28.0
Humidity:	72 %
Motion:	Detected
Smoke	Detected
Motion:	Neutral
Latitude:	13.6288 N
Longitude:	79.4192 E
Distance:	76 Kms from Base

▶ When the network is switched on and running, we can see that all the nodes are communicating with the server. The data is visualized in a custom-made GUI window.

▶ Connect to the Broker button will allow us to connect the broker to the network.

▶ When the nodes are switched on, they automatically connect to the server via MQTT.

▶ Node 1 and Node 2 are immobile nodes; hence they only transfer the data which they sense.

▶ Node 3, however, is a mobile node which is associated with GPS module. We can see that in the Node 3 group there are two buttons for “Get Location” and “Fall Back”. The former one fetches the live location of the soldier, who is carrying the node. While the latter one commands the soldier to return back to the base by alarming him.

CONCLUSIONS

▶ Successfully created a Wireless Sensor Network with the least bandwidth usage and low power consumption providing best results and robustness.

▶ Achieved the concepts of MQTT Protocol and Wireless Networking.

▶ Created three wireless nodes in the scenario, among which two are immobile and the remaining one is a mobile node. Each node is interfaced with a specific set of sensors.

▶ All these nodes are connected to a central server to control and monitor.

▶ Achieved output with an effective Wireless Sensor Network with 3 Client nodes and One Server node, each communicating with the server node.

Future Scope:

The future scope of this project is to extend the MQTT service to military using satellite communication. The low bandwidth provides an effective utilization over the channel. By enhancing the security using encryption, data can be protected and could be leveraged to household networks. MQTT has several industrial applications that can be processed through secured brokers and helps to deploy multiple sensors.

REFERENCES

[1] Internet of Things: Survey and open issues of MQTT protocol, Muneer Bani Yassein; Mohammed Q. Shatnawi; Shadi Aljwarneh; Razan Al-Hatmi, 2017 International Conference on Engineering & MIS (ICEMIS)

[2] Architectural design of token based authentication of MQTT protocol in constrained IoT device, Adhitya Bhawiyuga; Mahendra Data; Andri Warda 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)

[3] https://www.eclipse.org/paho/clients/python/docs/

[4] http://mqtt.org/