Fire Management System using Raspberry Pi and Android App

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

Computer Science and Engineering

under the Supervision of

Ms. Reema Aswani

By

Sancheeta Kaushal (111273)

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled “Fire Management system using Raspberry Pi and Android App”, submitted by Sancheeta Kaushal in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science and Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date: **Supervisor’s Name : Ms. Reema Aswani**

Acknowledgement

I would like to express my gratitude and appreciation to all those who gave me the perfect environment for completion of this report. A special thanks to my final year project supervisor, Ms. Reema Aswani, whose stimulating suggestions and encouragement, helped me to get to the thrust of my topic and understanding the importance of the project. I would also like to acknowledge with much appreciation the crucial role of the staff of Computer Laboratory, who provided me with the lab facilities as and when required.

Additionally, I appreciate the guidance given by the panels especially during the previous project presentation which made me realize the various dimensions I was probably missing out and hence, they gave away a room for improvement in the project. Again a special thanks to my friends who gave me valuable suggestions regarding the project.

Date : **Sancheeta Kaushal**

111273

Table of Content

Topic	Page No.
Certificate	i
Acknowledgment	ii
Table of Content	iii
List of Figures	iv
List of Tables	v
List of Abbreviations	vi
Abstract	vii
1. Introduction	1-5.
1.1 Significance	1.
1.2 Attributes	2.
1.3 History	3.
1.4 Motivation	4-5.
2. Literature Survey	6-8.
2.1 Existing Systems	6.
2.2 Scope of Improvement	7.
2.3 Feasibility Study	8.
3. Project Design	9-17.
3.1 Overall Design	9.
3.2 Components	10-15.
3.3 Flow Diagram	16.
3.4 System Requirements	17.
3.5 Gantt Chart	17.
4. Implementation	18-19.
4.1 Phases	18.
4.2 Proposed Methodology	19.
5. Conclusion	20.
6. Future Work	21.
7. Appendix-I	22-33.
8. References	34.

List of Figures

S.No.	Title	Page No.
1.1	Pie chart depicting alarm effectiveness	12.
1.2	Pie chart depicting Alarm Status	13.
3.1	Overall Design	19.
3.2	Diagrammatic view of Raspberry Pi	20.
3.3	Bottom view of Raspberry Pi	21.
3.4	Top view of Raspberry Pi	22.
3.5	Top view of on-board pins	23.
3.6	Bottom view of sensor DS18B20	24.
3.7	Flow Diagram	27.
3.8	GCM Flow Diagram	28.
3.9	GCM Architecture Diagram	29.

List of Tables

S.No.	Title	Page No.
1.1	Fire statistics for 10 years in Delhi	14.

List of Abbreviations

- IoT – Internet of Things
- PaaS – Platform as a Service
- IBM – International Business Machines
- MAC Address – Media Access Control Address
- MQQT – Message Queue Telemetry Transport
- SSH – Secured Shell
- Telnet – Telecommunication Network
- TCP – Transmission Control Protocol
- IP – Internet Protocol
- GCM – Google Cloud Messaging

Abstract

Events of fire pose threat to human life and property. Hence, there is a need to have fire management systems that incorporate real-time survillence, monitoring and on time detection in case of fire such that the fire can be controlled before it spreads and causes huge losses. In this project we use IoT to achieve this.

The Internet of Things (IoT) is a rapidly emerging field and refers to the interconnection of uniquely identifiable embedded computing devices within the existing Internet infrastructure. This field has seen a sharp rise because of the increasing number of computing devices which every individual has these days. Therefore, there is a huge potential for early detection and early alert generation in case of fire.

This project thus aims for detection and alarming of the critical events of fire in a building using the IoT technology including Raspberry Pi and an Android App.The main objective is to build a system where in case of fire or undue rise in temperature in a room, the notification is sent to the concerned authority.Additionally, considering the case where an individual is struck in the fire and needs to notify it to nears and dears.In addition to this, the struck person can find the safest possible route to exit excluding the paths affected by fire.

If implemented properly, this system can also help in detection of the cause of fire based on the location of raspberry pi which reports the fire.

This system is unique in the sense that from the time immemorial, the fire mangement systems were typically the legacy systems and such systems were properitory in their use and hence only a few had the capability to use the then technology for fire detection. With the time, technology changed but the properiotory nature of the fire industry remained same which only increased the cost of fire management solutions. This project shows how to build better cost effective systems.

1. INTRODUCTION

In this project, we will get the data from the sensor and the CPU using Raspberry Pi and send that data to IBM Bluemix Cloud. This data will be checked continuously for the anomalous values. Then a cloud service notification will be sent to concerned authorities in case the sensor detects a temperature rise.

Secondly, in case a person is struck at such a place then this system sends notification to nears and dears whose numbers were saved when app was installed.

Thirdly, the app helps to find the safest exit by excluding the path on which the fire took place.

This fire management system falls under the category of Building Management System (BMS) or a (more recent terminology) Building Automation System (BAS) which is a computer-based control system installed in buildings that controls and monitors the building's mechanical and electrical equipment. BMS consists of software and hardware configured to generate proper signals and respond to them in near real time.

A fire alarm system is a set of electric/electronic devices/equipment working together to detect and alert people through visual and audio appliances when smoke/fire is present. These alarms may be activated from smoke detectors, heat detectors, water flow sensors, which are automatic or from a manual fire alarm pull station.

1.1 Significance:

- Most cost effective compact system.
- Easy to track locations vulnerable to fire.
- Easy to track locations out of easy human reach i.e. 24*7 presence of human is not possible.
- Easy to track labs which involve working with highly inflammable substances.
- Integration with cloud and app is an added advantage and gives a room for more tinkering with data from the sensor.

- Additional information about the environment can be known.

1.2 Attributes

- Real time monitoring and surveillance – Pi measures the data in real time and it can be observed while we are sending data to the cloud there is hardly any observable latency.This critical real-time system, in which timeliness (i.e., the ability of a system to meet time constraints such as deadlines) is significant.
- Reliable – In its literal meaning, reliability means being consistently good in quality or performance and to be able to be trusted. The system is highly reliable since any loss in wifi signal can be detected via cloud and notification will be sent immediately.
- Extensible nature – The system proposed involves greater flexibility and can be extended and operated upon by various types of algorithms at a later stage.
- Efficient in terms data processing – The cloud storage of streaming data provides a chance to process data on the cloud and that very efficiently.
- Sensitivity of the sensor - The digital sensor has a high degree of tolerance and very wide range over which the temperature can be measured.
- Good response time of sensor – The response tme is the time taken for a circuit or measuring device, when subjected to a change in input signal, to change its state by a specified fraction of its total response to that change. The sensor response time is quiet good.
- Scalable system – Right now we are having just one module to sense the temperature. We can scale the system by adding more sensors one at each vulnerable point in the room and make them communicate wirelessly.
- No false alarms – Since we are also measuring the temperature of the C.P.U. of the Raspberry Pi, we can detect any circumstances where the Pi is not functioning properly and is generating false alarms.

1.3 History

Events of fire occurred since the beginning of human civilizations. But with the increase in population and settlements we saw an increase in the losses due to fire.

- First fire alarms were illustrated by roving watchmen using hand bell-ringers or church sextons ringing church bells or factory steam whistles which alerted the fire brigades.
- Telegraph was later invented by Sam Morse in 1840's and led to faster and accurate fire reporting systems.
- In March, 1851 Municipal fire alarm system was installed in Boston which communicated alarm to all the fire centers across the city.
- On April 30, 1852 first alarm was transmitted.
- In March 1855, Gamewell bought rights to construct Public Fire Alarm Systems in USA.
- By early 1870's, Watkins has developed remotely monitored fire alarm systems using heat detectors.
- The first electric fire detector was developed in Brooklyn, NY in 1863 by Alexander Ross.
- In 1873, first private fire alarm company was established.
- Proposed Fire Research Program In 1959, Committee on Fire Research and the Fire Research Conference of the Division Of Engineering of the National Research Council.
- In 1968, Fire Research and Safety Act was formed in U.S.A.
- In the mid-1970's the National Bureau of Standards (Now the National Institute of

  Standards and Technology – NIST) contracted with Illinois Institute of Technology Research Institute and Underwriter's Laboratories to obtain data regarding the performance of smoke detectors and their effectiveness in residential environments.

- In 1968, Minnesota Fire Department resulted in conclusions nearly identical to the results of the Dunes Tests.
- In August of 1997, FDI requested for a proposal to research on Duct smoke detector research.

1.4 Motivation:

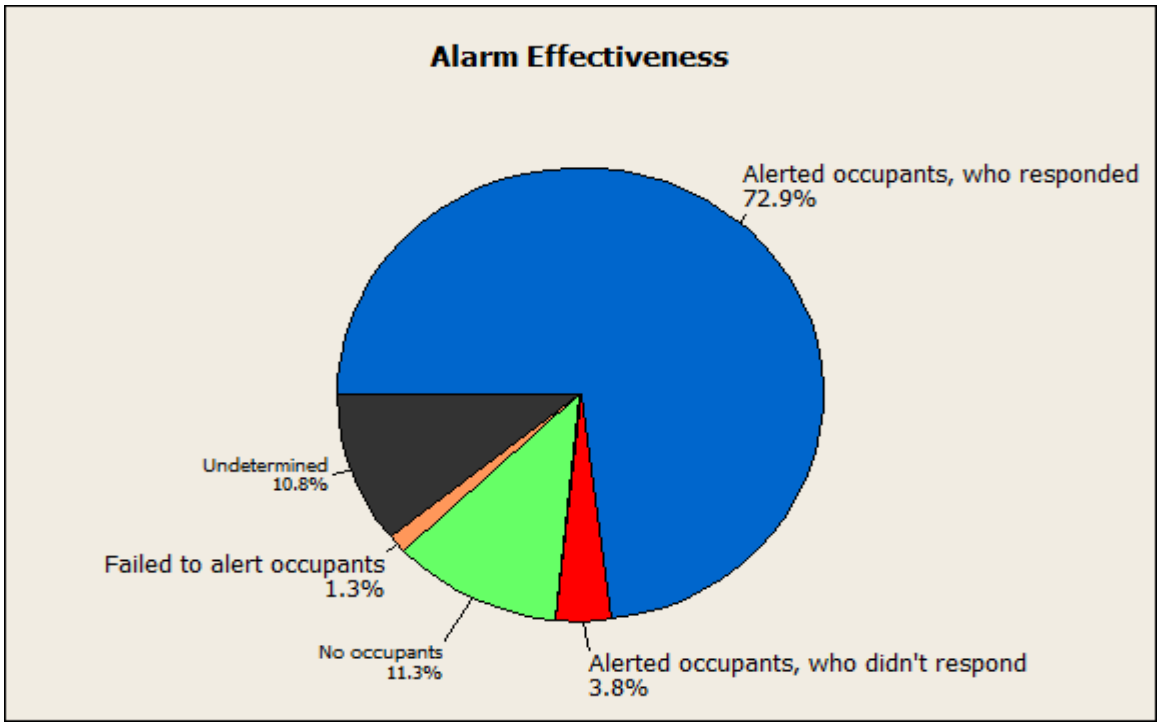

Fig. 1.1 Pie chart depicting alarm effectiveness

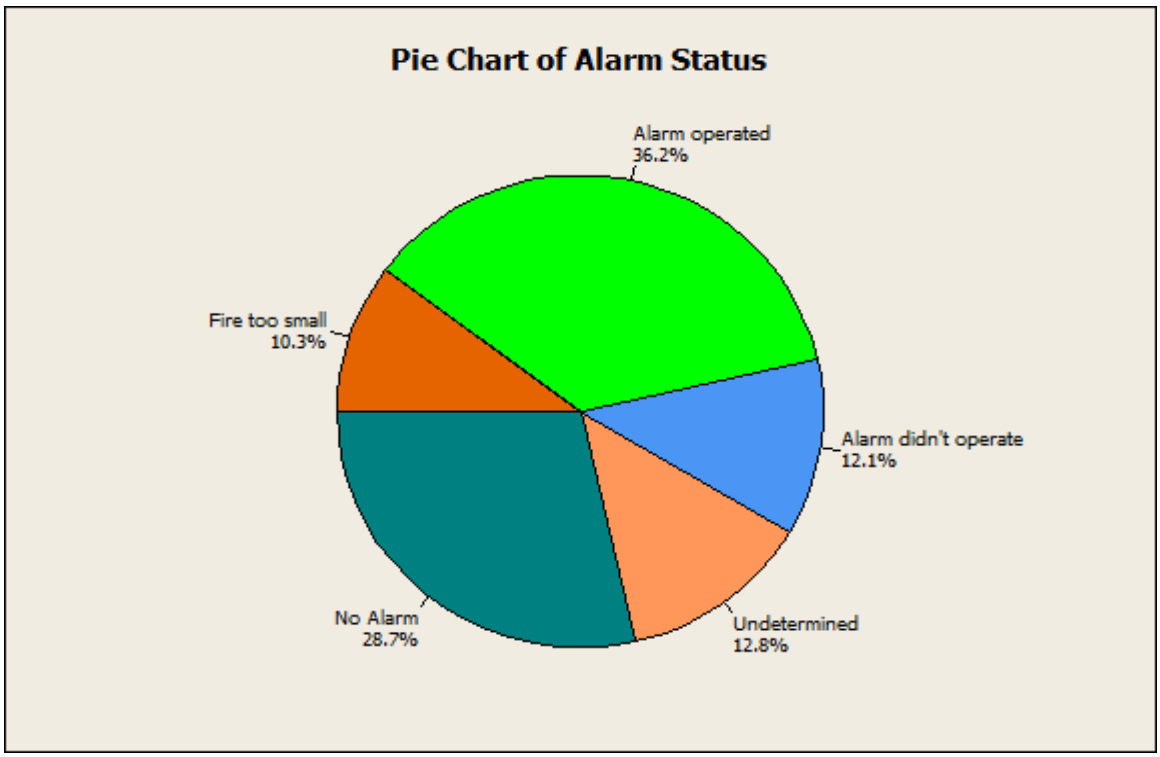

Fig. 1.2 Pie chart depicting Alarm Status whether determined or not in time

As of 2012 annual data, fires caused an estimated average of 25 injuries, $21 million property loss and 5 deaths in U.S.A.It has been a well known fact that a working smoke detector in the house goes a long way toward protecting the lives of individuals.But astonishingly, not all homes have these detectors.

According to Delhi Service Fire report:

S.No.	Year	No. of Calls	Approx. Loss in Lakhs	Property Saved in Lakhs	Injured	Deaths	Medium	Serious	Major
1	2003-04	14595	5874	8750	1334	235	17	5	-
2	2004-05	14208	4681	6629	1687	272	27	5	-
3	2005-06	16340	4720	6457	2191	470	16	1	-
4	2006-07	14291	5587	14903	1743	303	16	3	-
5	2007-08	15718	5922	29369	2057	351	09	2	-

6	2008-09	16452	5902	29471	2225	380	06	2	-
7	2009-10	21314	-	-	2598	423	10	02	-
8	2010-11	22187	-	-	243	447	10	03	
9	2011-12	18143	-	-	2132	357	13	1	-
10	2012-13	22581	-	-	1979	285	9	2	-

Table 1.1 Fire statistics for 10 years in Delhi

The figure in terms of injuries, deaths and loss of property is really alarming.The situation becomes more difficult due to the negligence of people. Indian households never showed any interests towards fire alarm systems the reason being high cost and difficult maintainence. This drove the motivation for the current project scenario.The reason for high ocsts invlolved is the hardware and the proprieotry nature of the industry.

These statistics also prove that the situation is not improving with time rather it's almost constant in terms of death and injury tolls. Hence, I feel that this area demands immediate attention.

2. LITERATURE SURVEY

2.1 Existing Systems:

a.) Fire Alarms:

- Manual Fire Alarm: Such systems need human input to indicate an event of fire. The issue is that they are not automated.
- Automatic Fire Alarms: These are automated systems using wired sensors and generate warning in case the sensor values go beyond a threshold limit. Such systems cost very high due to involvement of highly technical systems. The microprocessors being used are not that easy to handle both for the customer and the manufacturer and the programmer.
- Wireless sensor networks: These sensors have wireless capabilities and provide low cost solutions for such applications. They consist of small size, low-power and low cost devices integrated with limited computation, sensing and communication capabilities.
- Simulation Systems: These systems help in simulating real life situations where loss can be major. These systems prove to be of great in situations where the probability of fire event is very high. For example, Beijing Olymics may be considered for such a situation. This system used multi-dimensional integration model. Two types of studies were done one where we analysed fire resistant behaviours of whole structure.Second one being the study on fire evaluation and emergency rescue of campus based Geography Information System (GIS).
- Computer vision Systems: Such systems can also be used as a type of multi-function sensor. Computer vision applications have included building security, improving response rate, sensing and control and monitoring a fire. Additional fire detection capability can therefore be added with minimal cost through changes in software and correlating results between the computer vision system and other. Using combination of video cameras, computers, and artificial intelligence techniques, it processes multiple spectral images in real

time to reliably detect a small fire at large distances in a very short time. It can also identify the location of a fire, track its growth and monitor fire suppression. Further combined with radiation sensors (UV and IR) to enhance its detection capabilities or a CCD camera to automatically evaluate the scene through identification of bright regions associated with the fire radiation and increase system reliability.

b.) Notification Systems:

- GCM: Google Cloud Messaging (GCM) for Android is a service that allows you to send data from your server to your users' Android-powered device, and also to receive messages from devices on the same connection. The GCM service handles all aspects of queuing of messages and delivery to the target Android application running on the target device, and it is completely free.
- Rabbit MQ: RabbitMQ is open source message broker software (sometimes called message-oriented middleware) that implements the Advanced Message Queuing Protocol (AMQP). The RabbitMQ server is written in the Erlang programming language and is built on the Open Telecom Platform framework for clustering and failover. Client libraries to interface with the broker are available for all major programming languages.
- Others: This includes Tokodu Notification, Parse Library and Deacon Project.

c.) Approaches to find Location:

- Map of building
- Fused Location Provider API: It is a simple, power efficient API which is known for it’s versatile nature and immediate availability.
- Geolocation API: The Google Maps Geolocation API returns a location and accuracy radius based on information about cell towers and WiFi nodes that the mobile client can detect.

2.2 Scope of Improvement:

- Fire systems are really cost extensive. Since, the chances of a fire mishap taking place are very rare so the cost is important.
- But, this rare event may also cost lives and huge losses if we compromise on system's quality.
- So, we need to strike a balance between cost and quality.
- Lastly, the microcontrollers have less memory involved.
- Thus, sensor data cannot be stored and analyzed in on chip memory.
- Since there cannot be on chip storage we definitely need to have cloud services.
- Most of the sensors are analog sensors and hence required complex circuitry.
- A lot of systems require multiple sensors for detection of fire which is complex in the sense that the computation gets doubled and the cost and complexity of getting appropriate results is relatively higher.
- Also, in some wireless sensors the operating frequency of sensor interferes with the frequency of the processor leading to time delays in the results.
- Wireless sensors only possess low computation capabilities while in case of fire we need computation intensive sensors. In addition to this, we need repeaters which increase the cost.

2.3 Feasibility Study:

The feasibility study concerns with the consideration made to verify whether the system is fit to be developed in all terms. Once an idea to develop software is put forward the question that arises first will pertain to the feasibility aspects.
There are different aspects in the feasibility study:

- Operational Feasibility**:**

There is no difficulty in using the system, since the system will be made available as an

Android App and since apps are a common feature these days. Therefore, it is assumed that he will not face any problem in running the system. As users are responsible for initiating the development of a new system this is rooted out.

- Technical Feasibility**:**

Technical feasibility deals with the study of function, performance, and constraints like resources availability, technology, development risk that may affect the ability to achieve an acceptable system and as we know handling Raspberry Pi is quiet easy a task than use complicated microcontrollers. Even IBM Bluemix has a rich variety of resources already available so as to make the task of a technical person easier.

- Economic Feasibility**:**

One of the factors, which affect the development of a new system, is the cost it would incur. The proposed system involves Raspberry Pi which is really cost effective in front of other microcontroller. Additionally, the sensor is cheap and IBM Bluemix costs zero to us. Hence, very little cost has to beincurred to develop the system. Thereby decreasing the cost of the system to be sold which is one of the reasons that makes this system a better one.

3. PROJECT DESIGN

The project design involved various designing the system and software after realizing the components that were required. The overall system architecture was built and analysed.These system designs will serve as input to next phase of the model.

3.1 Overall Diagram:

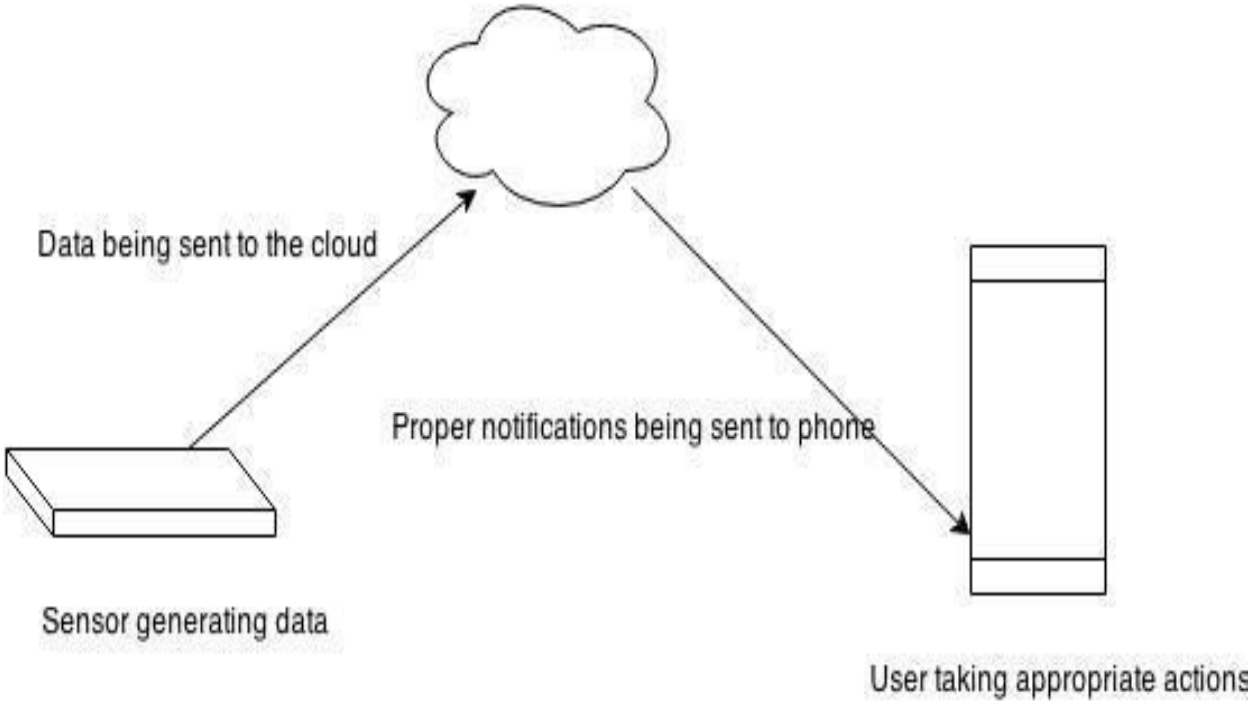

Fig 3.1 Overall system diagram

- In our system design, we generate data using temperature sensor which is interfaced to Raspberry Pi.
- The data is being sent to the cloud so that notification can be triggered in case of anomalous values.
- The notifications are being sent to the owner so that appropriate actions can be taken.
- Also, an emergency situation can be reported using the same android app.

- Lastly, in case of an emergency situation, the best possible route to exit will be determined.

3.2 Components:

1. Raspberry Pi:

Fig. 3.2 Diagrammatic view of Raspberry Pi

- Raspberry Pi is a credit card-sized single-board computer which can be used for wide variety of applications from General Purpose Computing to Project Platform.
- Also known as Device for Makers it allows bare metal computer hacking, it has a Broadcom System on a chip.
- The model being used is Model B constituting ARM based processor , 512 MB RAM, GPU, 700mA power input, 16 GB Class 10 Memory Card, 2 USB Ports and 1 Ethernet Port and 26 pin GPIO.
- The Operating System configuration being installed is Raspabian which is a stock OS and is a flavor of Linux.

- The GPIO is expandable and can be interfaced to another microcontroller like Arduino for analog sensors since there is no analog to digital interface on the pi.

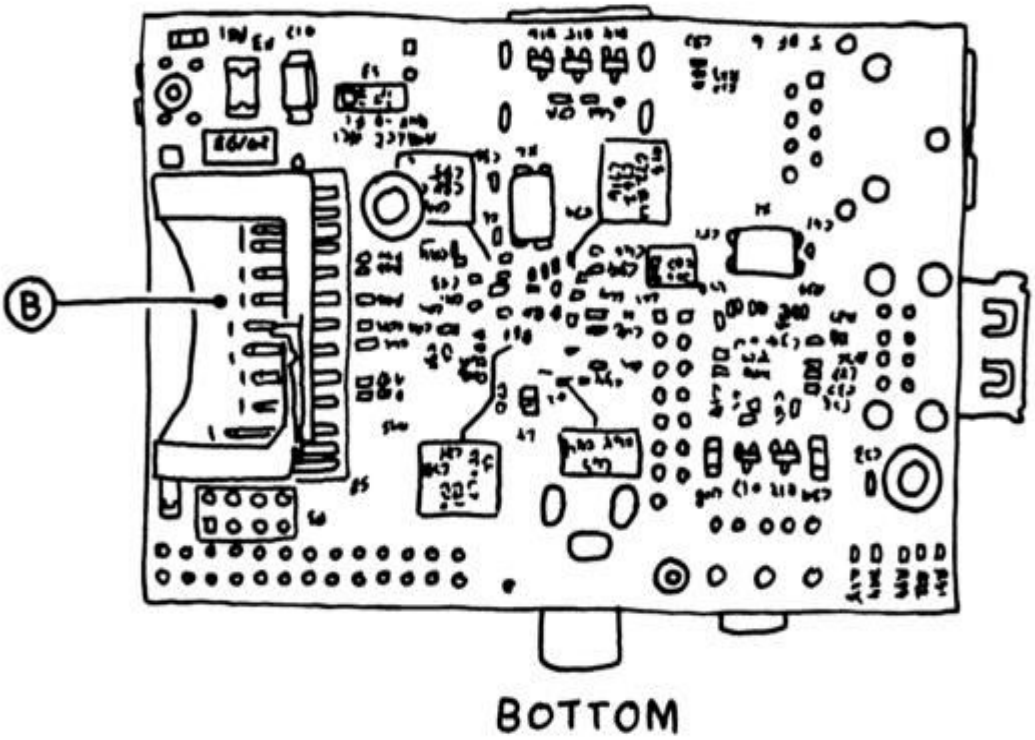

Fig 3.3 Bottom view of Raspberry Pi

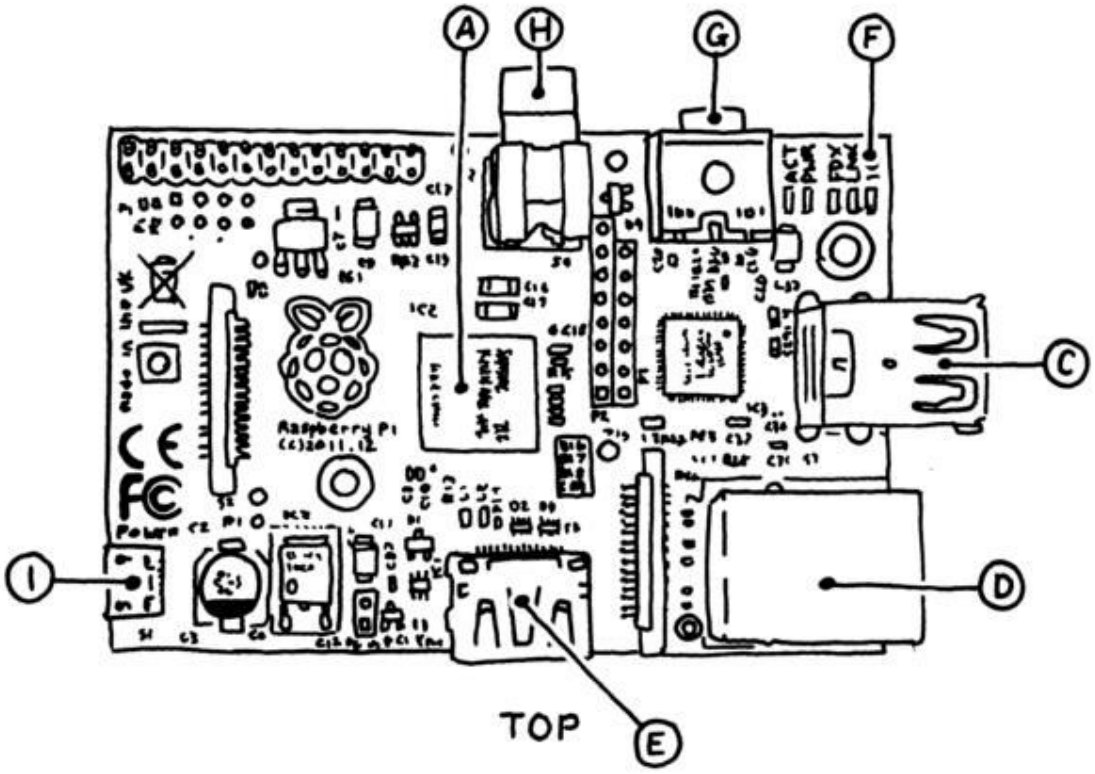

Fig 3.4 Top view of Raspberry Pi

A. **Processor:** At the heart of the Raspberry Pi is the same processor found in the iPhone 3G and the Kindle 2. This chip is a 32 bit, 700 MHz System on a Chip, which is built on the ARM11 architecture. The Model B has 512MB of RAM.

B. **Secure Digital Card Slot**: Everything on Pi is stored on SD Card so there is nothing like hard disk for storage.

C. **The USB Port:** On model B there are two USB 2.0 ports.

D. **Ethernet Port:** The model B has a standard RJ45 Ethernet port. The port on the Model B is actually an onboard USB to Ethernet adapter. WiFi connectivity via a USB dongle is another option.

E. **HDMI Connector:** The HDMI port provides digital video and audio output.14 different video resolutions are supported.

F. **Status LED's:**

- ACT - Green - Lights when the SD card is accessed
- PWR - Red - Hooked up to 3.3V power
- FDX - Green - On if network adapter is full duplex

- LNK - Green - Network activity light
- 100 - Yellow- On if the network connection is 100Mbps

G. Analog Audio Output: This is a standard 3.5mm mini analog audio jack, intended to drive high impedance loads (like amplified speakers). Headphone or unpowered speakers won't sound very good.

H. Composite Video Out: This is a standard RCA-type jack that provides composite NTSC or PAL video signals. This video format is extremely low-resolution compared to HDMI. Prefer to use a HDMI television or monitor, rather than a composite television.

I. Power Input: There is no power switch on the Pi. This microUSB connector is used to supply power (this isn't an additional USB port; it's only for power). MicroUSB was selected because the connector is cheap USB power supplies are easy to find.

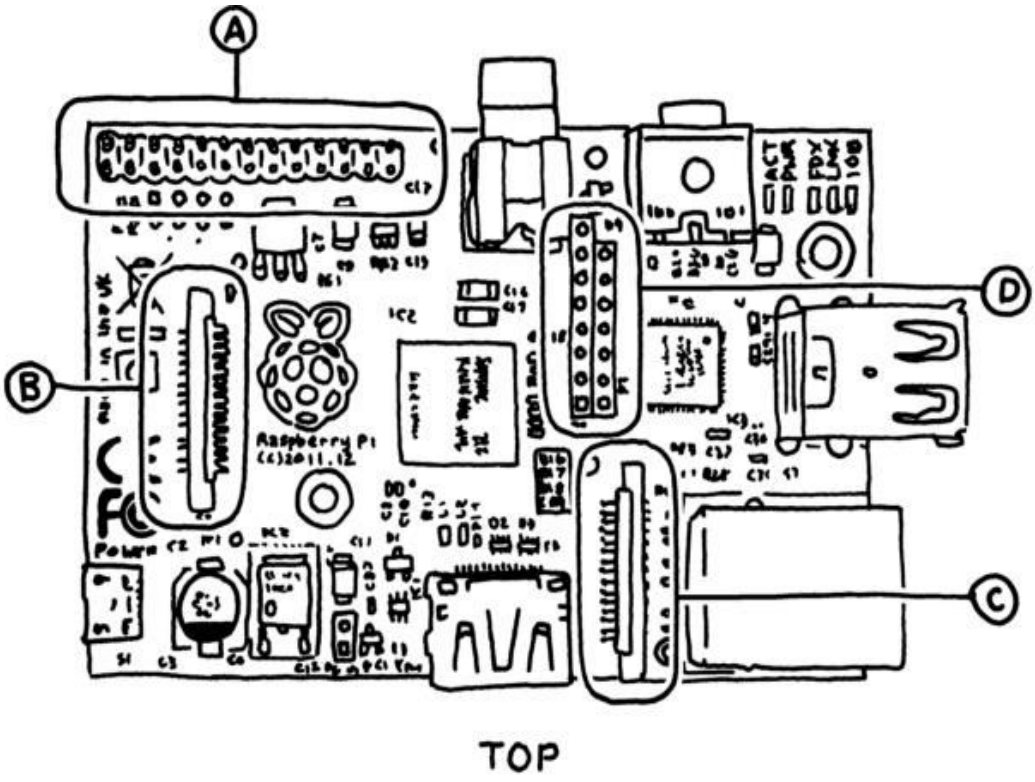

Fig 3.5 Top view of on-board pins

A. General Purpose Input and Output: These pins to read buttons and switches and control actuators like LEDs, relays, or motors.

B. **Display Serial Interface:** This connector accepts a 15 pin flat ribbon cable that can be used to communicate with a LCD or OLED display screen.

C. **Camera Serial Interface:** This port allows a camera module to be connected directly to the board.

D. **P2 and P3 headers:** These two rows of headers are the JTAG testing headers for the Broadcom chip (P2) and the LAN9512 networking chip (P3).

2. DS18B20 Temperature Sensor

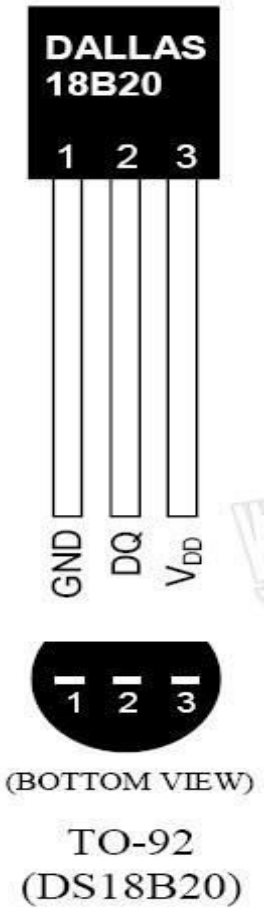

Fig 3.6 Bottom view of sensor DS18B20

- DS18B20 Temperature Sensor is a digital thermometer which provides 9 to 12 bit Celsius temperature measurement.

- With only 1 data line, it measures temperatures from -55°C to +125°C (-67°F to +257°F). It has a tolerance of ±0.5°C and its accuracy is maximum over the range -10°C to +85°C**.**
- This sensor can derive power directly from data line using the concept of parasite power and thereby eliminates the need of an external power supply.

3. PuTTY

PuTTY is an SSH and telnet client. As a free and open source terminal emulator, serial console and network fiile transfer application.It supports several network protocols including SCP, SSH, Telnet, rlogin and raw socket connection. In this project it will be used for SSH login.

4. Wifi module and LAN cable:

The concept of network bridging is used in the project. Network bridging is the action taken by network equipment to create an aggregate network from either two or more communication, or two or more network segments. Bridging is distinct from routing which allows the networks to communicate independently as separate networks. Also, if one or more segments of the network are wireless, it is known as bridging. A network bridge is a network device that connects multiple network segments. In the OSI model bridging acts in the first two layers, below the network layer. The wifi signal is bridged to Raspberry Pi using LAN Cable.

5. IBM Bluemix:

Bluemix is an open standards, cloud based platform for building, managing and running apps of all types, such as web, mobile, big data and smart devices. Capabilities include Java, mobile back-end development and application monitoring, as well as features from ecosystem partners and open-source all provided as a service in the cloud.

Being an enterprise Platform as a Service (PaaS), the services provided are helpful for integration of data from Raspberry pi with the cloud. It supports several programming languages and services[1]as well as integrated DevOps to build, run, deploy and manage applications on the cloud. Bluemix is based on Cloud Foundry open

technology and runs on SoftLayer infrastructure. Bluemix supports Java, Node.js, Ruby and can be extended to support other languages such as PHP, Python or Scala. In February 2014,Bluemix was one of the largest Cloud Foundry deployments in the world.

6. MQTT :

Message Queue Telemetry Transport is publish - subscribe based light weight messaging protocol for use on top of the TCP/IP protocol. Having designed for connections with remote locations where a small code footprint is required and network bandwidth is limited. The Publish-Subscribe messaging pattern requires a message broker. The broker is responsible for distributing messages to interested clients based on topic of a message.

7. Notification Based Strategies:

a.) Push Strategy: Push notifications let our application notify a user of new messages or events even when the user is not actively using your application. On Android devices, when a device receives a push notification, your application's icon and a message appear in the status bar. When the user taps the notification, they are sent to your application. Notifications can be broadcast to all users, such as for a marketing campaign, or sent to just a subset of users, to give personalized information.

b.) Pull Strategy: It is referred to as "polling" where the phone will periodically ask a server for new information or content.

3.3 Flow Diagram:

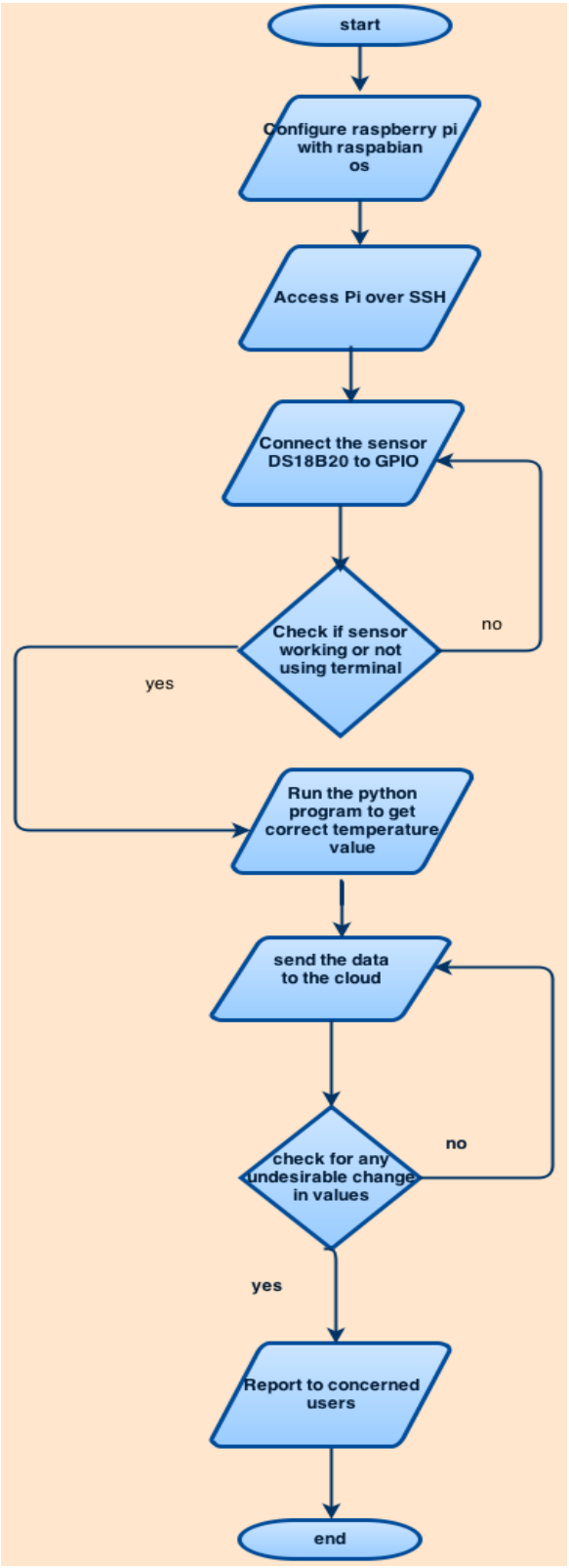

Fig 3.7 Data Flow Diagram

3.4 System Requirements:

1.Hardware Requirements**:**

- Raspberry Pi

  CPU : 700 MHz

  RAM : 512 MB

  Display : 1024*768 Monitor

  SD Card Slot : 16 GB

- Sensor
- Android Phone(for installation of App)

2.Software Requirements:

- OS : Raspbian
- Software : Eclipse
- Cloud Service : IBM Bluemix

3.5 GCM Flow Diagram:

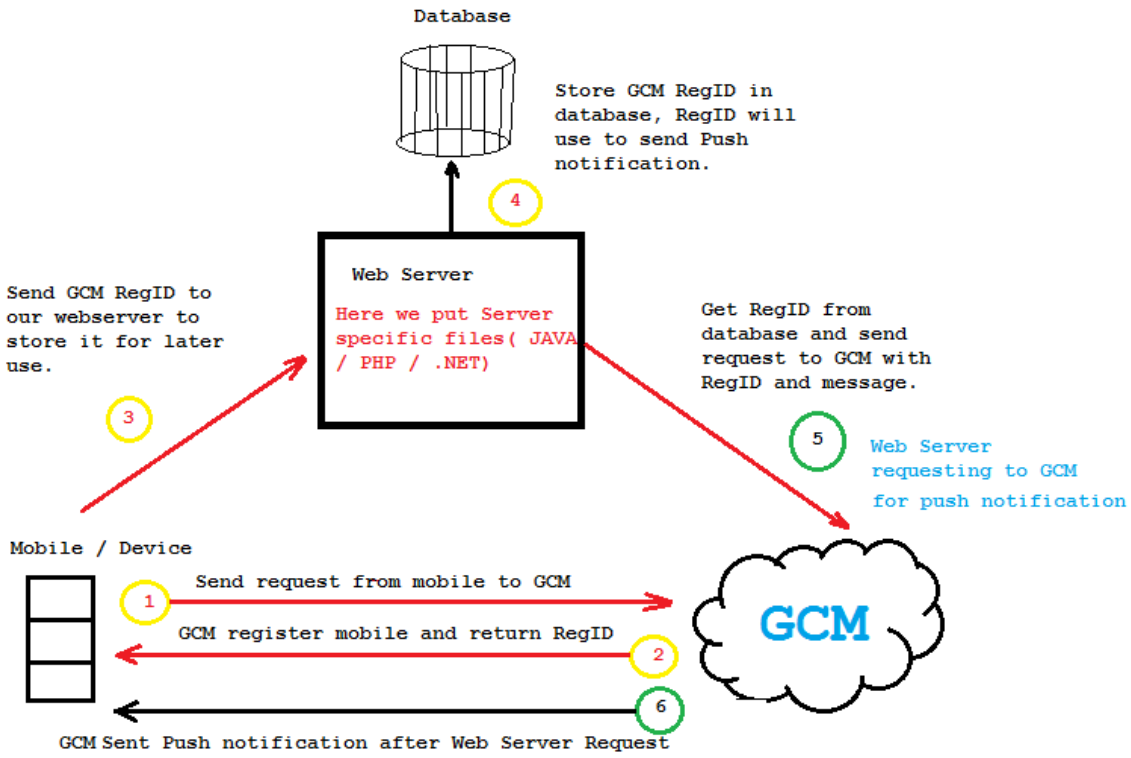

Fig 3.7 GCM Flow Diagram

3.6 GCM Architecture:

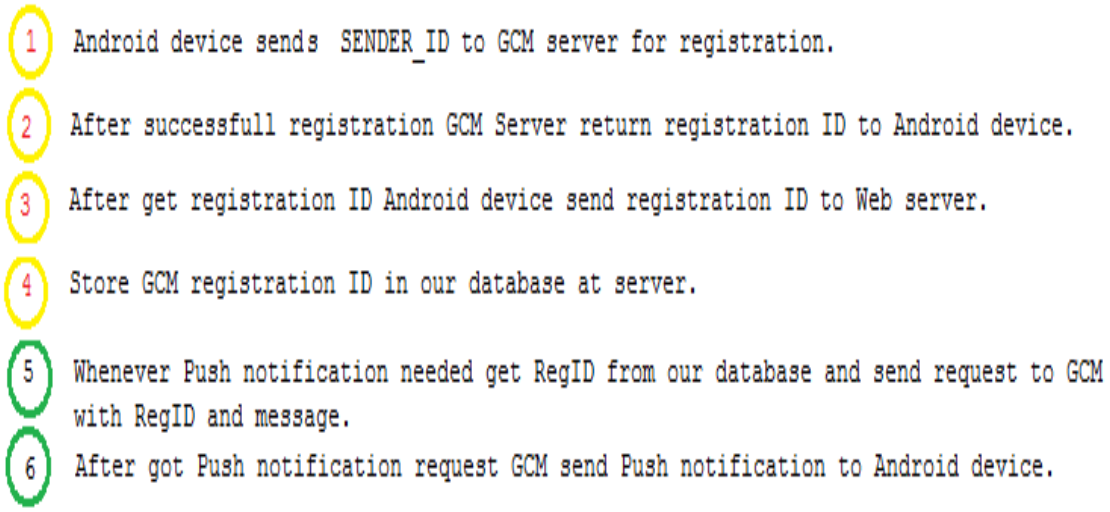

Fig 3.8 GCM Architecture Diagram

4. IMPLEMENTATION

4.1 Phases:

The project will be done in two phases:

- First phase –

  1. Getting data from sensor using Raspberry Pi

  2. Sending the data to the cloud using IBM Bluemix and building a service to ensure communication between the Raspberry Pi and the Cloud.

- Second phase –

  1. A cloud service notification in case of rise in temperature to the concerned authorities. Concerned authorities being administration and lab assistants in case of an institution and in case of houses, the owner is concerned authority.

  2. In case, you are struck at such a place then this system allows you to send notifications to near and dear ones.

3.We will use push notification strategy since it saves power consumption, time and other resources.

4. GCM will be required both for downstream and upstream notifications.

5. Downstream Notifications from BlueMix Cloud via GCM

6.Upstream Notifications to nears and dears via GCM

4.2 Proposed Methodology and Algorithm:

The following things were done in sequence to achieve an executable version of the project:

- Raspabian OS was installed on the Raspberry Pi.
- The DS18B20 sensor was mounted on breadboard. It was then connected to GPIO on Raspberry Pi. The left pin of sensor is connected to ground which is 3rd pin on right. The middle data pin of sensor is connected to 4th GPIO pin on left. The right pin of the sensor is connected to 1st pin on the left. All these references are made with respect to the flat face of the sensor.
- Raspberry Pi is connected to power supply. Fist two status LED's start blinking.
- The wifi was connected to laptop and the Ethernet cable was connected to Raspberry Pi from Laptop and these two connections were bridged. Rest three LED's also start blinking.
- Using PuTTY, we access the Raspberry Pi terminal and using modprobe commands we activate the sensor to get the sensor output on the terminal.
- We then make changes to our code and debuild the current file to form a .deb file which is a binary only file.
- Using the dpkg command we then run the service.
- Once the service is running check the status and get the device id.
- Go to IBM Bluemix Cloud platform, put the device id and check the streaming data son NodeRed Editor.
- We can also visualize the data streaming graph.
- Finally, this data will be further put to use when we build Android App.

5. CONCLUSION

From the project done till now, we can conclude that this system can prove to be a promising system. If implemented on large scale, streaming data can provide us interesting statistics. But as of now, the data sent to cloud can be used to detect alarming situations by using triggers if the temperature goes beyond a certain limit. Additionally, this system can detect false alarms, which is very crucial these days. Furthermore, the data from this sensor can improvise the results in simulation systems by giving us a general idea of the surrounding temperature thereby telling the optimal temperature range and the threshold temperature in a much better way. Lastly, I conclude that first phase of the project is complete in its form such that it deployed next semester as the input data for the Android App.

6. FUTURE WORK

Several approaches can be followed to improvise the current system

- If integrated with camera module we can very precisely reason about the cause of fire in the room. Additionally, the camera module can help us in knowing the exact location of fire.
- We can also use augmented reality and neural network algorithms to enhance the efficiency of our system.
- The sensor data has a potential to be used in fire simulation techniques. It can be used to determine the threshold temperature value depending on various circumstances where each circumstance may lead to a situation responsible for generation of false alarms.

7. APPENDIX – I

- iotmain.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <signal.h>
#include "iot.h"
#include "MQTTAsync.h"
#include <syslog.h>

char configFile[50] = "/etc/iotsample-raspberrypi/device.cfg";
float PI = 3.1415926;
float MIN_VALUE = -1.0;
float MAX_VALUE = 1.0;

char clientId[MAXBUF];
char publishTopic[MAXBUF] = "iot-2/evt/status/fmt/json";
char subscribeTopic[MAXBUF] = "iot-2/cmd/reboot/fmt/json";

//flag to check if running in registered mode or quickstart mode
// registered mode = 1
// quickstart mode = 0
int isRegistered = 0;

MQTTAsync client;

//config file structure
struct config {
```

```
        char org[MAXBUF];
        char type[MAXBUF];
        char id[MAXBUF];
        char authmethod[MAXBUF];
        char authtoken[MAXBUF];
};

int get_config(char* filename, struct config * configstr);
void getClientId(struct config * configstr, char* mac_address);
float sineVal(float minValue, float maxValue, float duration, float count);
void sig_handler(int signo);
int reconnect_delay(int i);

//cpustat.c
float getCPUTemp();
float GetCPULoad();
//mac.c
char *getmac(char *iface);
//jsonator.c
char * generateJSON(JsonMessage passedrpi);

//mqttPublisher.c
int init_mqtt_connection(MQTTAsync* client, char *address, int isRegistered,
                char* client_id, char* username, char* passwd);
int publishMQTTMessage(MQTTAsync* client, char *topic, char *payload);
int subscribe(MQTTAsync* client, char *topic);
int disconnect_mqtt_client(MQTTAsync* client);
int reconnect(MQTTAsync* client, int isRegistered,
                char* username, char* passwd);

int main(int argc, char **argv) {
```

```
	char* json;

	int lckStatus;
	int res;
	int sleepTimeout;
	struct config configstr;

	char *passwd;
	char *username;
	char msproxyUrl[MAXBUF];

	//setup the syslog logging
	setlogmask(LOG_UPTO(LOGLEVEL));
	openlog("iot", LOG_PID | LOG_CONS, LOG_USER);
	syslog(LOG_INFO, "**** IoT Raspberry Pi Sample has started ****");

	// register the signal handler for USR1-user defined signal 1
	if (signal(SIGUSR1, sig_handler) == SIG_ERR)
		syslog(LOG_CRIT, "Not able to register the signal handler\n");
	if (signal(SIGINT, sig_handler) == SIG_ERR)
		syslog(LOG_CRIT, "Not able to register the signal handler\n");

	//read the config file, to decide whether to goto quickstart or registered mode
of operation
	isRegistered = get_config(configFile, &configstr);

	if (isRegistered) {
		syslog(LOG_INFO, "Running in Registered mode\n");
		sprintf(msproxyUrl,
"ssl://%s.messaging.internetofthings.ibmcloud.com:8883", configstr.org);
		if(strcmp(configstr.authmethod ,"token") != 0) {
```

```
                        syslog(LOG_ERR, "Detected that auth-method is not token.
Currently other authentication mechanisms are not supported, IoT process will exit.");
                        syslog(LOG_INFO, "**** IoT Raspberry Pi Sample has ended
****");
                                closelog();
                                exit(1);
                } else {
                        username = "use-token-auth";
                        passwd = configstr.authtoken;
                }
        } else {
                syslog(LOG_INFO, "Running in Quickstart mode\n");

        strcpy(msproxyUrl,"tcp://quickstart.messaging.internetofthings.ibmcloud.com:
1883");
        }

        // read the events
        char* mac_address = getmac("eth0");
        getClientId(&configstr, mac_address);
        //the timeout between the connection retry
        int connDelayTimeout = 1;     // default sleep for 1 sec
        int retryAttempt = 0;

        // initialize the MQTT connection
        init_mqtt_connection(&client, msproxyUrl, isRegistered, clientId, username,
passwd);
        // Wait till we get a successful connection to IoT MQTT server
        while (!MQTTAsync_isConnected(client)) {
                connDelayTimeout = 1; // add extra delay(3,60,600) only when
reconnecting
                if (connected == -1) {
```

```
                        connDelayTimeout = reconnect_delay(++retryAttempt);
        //Try to reconnect after the retry delay
                        syslog(LOG_ERR,
                                        "Failed connection attempt #%d. Will try to
reconnect "
                                                        "in %d seconds\n", retryAttempt,
connDelayTimeout);
                        connected = 0;
                        init_mqtt_connection(&client, msproxyUrl, isRegistered,
clientId, username,
                                                passwd);
                }
                fflush(stdout);
                sleep(connDelayTimeout);
        }
        // resetting the counters
        connDelayTimeout = 1;
        retryAttempt = 0;

        // count for the sine wave
        int count = 1;
        sleepTimeout = EVENTS_INTERVAL;

        //subscribe for commands - only on registered mode
        if (isRegistered) {
                subscribe(&client, subscribeTopic);
        }
        while (1) {
                JsonMessage json_message = { DEVICE_NAME, getCPUTemp(),
sineVal(
                                MIN_VALUE, MAX_VALUE, 16, count),
GetCPULoad() };
```

```
                json = generateJSON(json_message);
                res = publishMQTTMessage(&client, publishTopic, json);
                syslog(LOG_DEBUG, "Posted the message with result code = %d\n",
res);
                if (res == -3) {
                        //update the connected to connection failed
                        connected = -1;
                        while (!MQTTAsync_isConnected(client)) {
                                if (connected == -1) {
                                        connDelayTimeout =
reconnect_delay(++retryAttempt); //Try to reconnect after the retry delay
                                        syslog(LOG_ERR, "Failed connection attempt
#%d. "
                                                        "Will try to reconnect in %d "
                                                        "seconds\n", retryAttempt,
connDelayTimeout);
                                        sleep(connDelayTimeout);
                                        connected = 0;
                                        reconnect(&client, isRegistered,
username,passwd);
                                }
                                fflush(stdout);
                                sleep(1);
                        }
                        // resetting the counters
                        connDelayTimeout = 1;
                        retryAttempt = 0;
                }
                fflush(stdout);
                free(json);
                count++;
                sleep(sleepTimeout);
```

```
        }

        return 0;
}

//This generates the clientID based on the tenant_prefix and mac_address(external Id)
void getClientId(struct config * configstr, char* mac_address) {

        char *orgId;
        char *typeId;
        char *deviceId;

        if (isRegistered) {

                orgId = configstr->org;
                typeId = configstr->type;
                deviceId = configstr->id;

        } else {

                orgId = "quickstart";
                typeId = "iotsample-raspberrypi";
                deviceId = mac_address;

        }
        sprintf(clientId, "d:%s:%s:%s", orgId, typeId, deviceId);
//      sprintf(clientId, "%s:%s", TENANT_PREFIX,mac_address);
}

//This function generates the sine value based on the interval specified and the
duration
float sineVal(float minValue, float maxValue, float duration, float count) {
```

```
	float sineValue;
	sineValue = sin(2.0 * PI * count / duration) * (maxValue - minValue) / 2.0;
	return sineValue;
}

// Signal handler to handle when the user tries to kill this process. Try to close down gracefully
void sig_handler(int signo) {
	syslog(LOG_INFO, "Received the signal to terminate the IoT process. \n");
	syslog(LOG_INFO,
			"Trying to end the process gracefully. Closing the MQTT connection. \n");
	int res = disconnect_mqtt_client(&client);

	syslog(LOG_INFO, "Disconnect finished with result code : %d\n", res);
	syslog(LOG_INFO, "Shutdown of the IoT process is complete. \n");
	syslog(LOG_INFO, "**** IoT Raspberry Pi Sample has ended ****");
	closelog();
	exit(1);
}
/* Reconnect delay time
 * depends on the number of failed attempts
 */
int reconnect_delay(int i) {
	if (i < 10) {
		return 3; // first 10 attempts try within 3 seconds
	}
	if (i < 20)
		return 60; // next 10 attempts retry after every 1 minute

	return 600;	// after 20 attempts, retry every 10 minutes
}
```

```
//Trimming characters
char *trim(char *str) {
	size_t len = 0;
	char *frontp = str - 1;
	char *endp = NULL;

	if (str == NULL)
		return NULL;

	if (str[0] == '\0')
		return str;

	len = strlen(str);
	endp = str + len;

	while (isspace(*(++frontp)))
		;
	while (isspace(*(--endp)) && endp != frontp)
		;

	if (str + len - 1 != endp)
		*(endp + 1) = '\0';
	else if (frontp != str && endp == frontp)
		*str = '\0';

	endp = str;
	if (frontp != str) {
		while (*frontp)
			*endp++ = *frontp++;
		*endp = '\0';
	}
```

```
	return str;
}


// This is the function to read the config from the device.cfg file
int get_config(char * filename, struct config * configstr) {

	FILE* prop;
	char str1[10], str2[10];
	prop = fopen(filename, "r");
	if (prop == NULL) {
		syslog(LOG_INFO,"Config file not found. Going to Quickstart
mode\n");
		return 0; // as the file is not present, it must be quickstart mode
	}
	char line[256];
	int linenum = 0;
	while (fgets(line, 256, prop) != NULL) {
		char* prop;
		char* value;

		linenum++;
		if (line[0] == '#')
			continue;

		prop = strtok(line, "=");
		prop = trim(prop);
		value = strtok(NULL, "=");
		value = trim(value);
		if (strcmp(prop, "org") == 0)
			strncpy(configstr->org, value, MAXBUF);
		else if (strcmp(prop, "type") == 0)
```

```
                strncpy(configstr->type, value, MAXBUF);
        else if (strcmp(prop, "id") == 0)
                strncpy(configstr->id, value, MAXBUF);
        else if (strcmp(prop, "auth-token") == 0)
                strncpy(configstr->authtoken, value, MAXBUF);
        else if (strcmp(prop, "auth-method") == 0)
                        strncpy(configstr->authmethod, value,
MAXBUF);
    }

    return 1;
}
```

- cpustat.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int PATHSIZE = 255;
int SIZE = 8;
char cputemploc[255] = "/sys/class/thermal/thermal_zone0/temp";
char cpuloadloc[255] = "/proc/loadavg";

float getCPUTemp();
float GetCPULoad();

float getCPUTemp() {
    FILE * cputemp = NULL;
    char buffer [SIZE];
    long tempinmillic;
    float tempinc;
```

```
        memset(buffer, 0, sizeof(buffer));
        cputemp = fopen(cputemploc, "r");
        fgets(buffer, SIZE, cputemp);
        tempinmillic = atol(buffer);
        tempinc = tempinmillic * 1.0 / 1000.0;
        fclose(cputemp);
        return tempinc;
}

float GetCPULoad() {
        FILE *f1;
      float load1,load5,load15;

      f1 = fopen(cpuloadloc, "r");
      fscanf(f1, "%f\t%f\t%f\t", &load1, &load5, &load15 );
        fclose(f1);
      return (load1);
}
```

- jsonator.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "iot.h"

char * generateJSON(JsonMessage passedrpi ) {
        char * jsonReturned;

        // 2 braces, 4 colons, 3 commas, 10 double-quotes makes it 19
        jsonReturned = calloc(1, sizeof passedrpi + sizeof(char) * 25);
```

```
strcat(jsonReturned, "{\"d\":");
strcat(jsonReturned, "{");

strcat(jsonReturned, "\"myName\":\"");
strcat(jsonReturned, passedrpi.myname);
strcat(jsonReturned, "\",");
char buffer[10];

strcat(jsonReturned, "\"cputemp\":");
sprintf(buffer, "%.2f", passedrpi.cputemp);
strcat(jsonReturned, buffer);
strcat(jsonReturned, ",");

strcat(jsonReturned, "\"cpuload\":");
sprintf(buffer, "%.2f", passedrpi.cpuload);
strcat(jsonReturned, buffer);
strcat(jsonReturned, ",");

strcat(jsonReturned, "\"sine\":");
sprintf(buffer, "%.2f", passedrpi.sine);
strcat(jsonReturned, buffer);

strcat(jsonReturned, "}");
strcat(jsonReturned, "}");

return jsonReturned;
```

- RegisterActivity.java

```
package com.androidexample.gcm;

import android.app.Activity;
import android.content.Intent;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class RegisterActivity extends Activity {

	// UI elements
	EditText txtName;
	EditText txtEmail;

	// Register button
	Button btnRegister;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_register);

		//Get Global Controller Class object (see application tag in
AndroidManifest.xml)
		final Controller aController = (Controller)
getApplicationContext();

		// Check if Internet Connection present
		if (!aController.isConnectingToInternet()) {

			// Internet Connection is not present
			aController.showAlertDialog(RegisterActivity.this,
					"Internet Connection Error",
					"Please connect to working Internet
connection", false);
```

```
                // stop executing code by return
                return;
            }

            // Check if GCM configuration is set
            if (Config.YOUR_SERVER_URL == null ||
Config.GOOGLE_SENDER_ID == null ||
Config.YOUR_SERVER_URL.length() == 0
                    || Config.GOOGLE_SENDER_ID.length() ==
0) {

                // GCM sernder id / server url is missing
                aController.showAlertDialog(RegisterActivity.this,
"Configuration Error!",
                        "Please set your Server URL and GCM
Sender ID", false);

                // stop executing code by return
                 return;
            }

            txtName = (EditText) findViewById(R.id.txtName);
            txtEmail = (EditText) findViewById(R.id.txtEmail);
            btnRegister = (Button) findViewById(R.id.btnRegister);

            // Click event on Register button
            btnRegister.setOnClickListener(new View.OnClickListener() {

                @Override
                public void onClick(View arg0) {
                    // Get data from EditText
```

```
                                String name = txtName.getText().toString();
                                String email = txtEmail.getText().toString();

                                // Check if user filled the form
                                if(name.trim().length() > 0 &&
email.trim().length() > 0){

                                        // Launch Main Activity
                                        Intent i = new
Intent(getApplicationContext(), MainActivity.class);

                                        // Registering user on our server

                                        // Sending registraiton details to
MainActivity
                                        i.putExtra("name", name);
                                        i.putExtra("email", email);
                                        startActivity(i);
                                        finish();

                                }else{

                                        // user doen't filled that data

        aController.showAlertDialog(RegisterActivity.this, "Registration
Error!", "Please enter your details", false);
                                }
                        }
                });
        }

}
```

- MainActivity.java

```
package com.androidexample.gcm;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.gcm.GCMRegistrar;

public class MainActivity extends Activity {
        // label to display gcm messages
        TextView lblMessage;
        Controller aController;

        // Asyntask
        AsyncTask<Void, Void, Void> mRegisterTask;

        public static String name;
        public static String email;

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
```

```
        //Get Global Controller Class object (see application tag in
AndroidManifest.xml)
        aController = (Controller) getApplicationContext();



        // Check if Internet present
        if (!aController.isConnectingToInternet()) {

            // Internet Connection is not present
            aController.showAlertDialog(MainActivity.this,
                    "Internet Connection Error",
                    "Please connect to Internet connection",
false);
            // stop executing code by return
            return;
        }

        // Getting name, email from intent
        Intent i = getIntent();

        name = i.getStringExtra("name");
        email = i.getStringExtra("email");

        // Make sure the device has the proper dependencies.
        GCMRegistrar.checkDevice(this);

        // Make sure the manifest permissions was properly set
        GCMRegistrar.checkManifest(this);

        lblMessage = (TextView) findViewById(R.id.lblMessage);
```

```
                // Register custom Broadcast receiver to show messages on
activity
                registerReceiver(mHandleMessageReceiver, new IntentFilter(
                                Config.DISPLAY_MESSAGE_ACTION));

                // Get GCM registration id
                final String regId = GCMRegistrar.getRegistrationId(this);

                // Check if regid already presents
                if (regId.equals("")) {

                                // Register with GCM
                                GCMRegistrar.register(this,
Config.GOOGLE_SENDER_ID);

                } else {

                                // Device is already registered on GCM Server
                                if (GCMRegistrar.isRegisteredOnServer(this)) {

                                                // Skips registration.
                                                Toast.makeText(getApplicationContext(),
"Already registered with GCM Server", Toast.LENGTH_LONG).show();

                                } else {

                                                // Try to register again, but not in the UI thread.
                                                // It's also necessary to cancel the thread
onDestroy(),
                                                // hence the use of AsyncTask instead of a raw
thread.
```

```
                final Context context = this;
                mRegisterTask = new AsyncTask<Void, Void,
Void>() {

                    @Override
                    protected Void doInBackground(Void...
params) {

                        // Register on our server
                        // On server creates a new user
                        aController.register(context,
name, email, regId);

                        return null;
                    }

                    @Override
                    protected void onPostExecute(Void
result) {
                        mRegisterTask = null;
                    }

                };

                // execute AsyncTask
                mRegisterTask.execute(null, null, null);
            }
        }
    }

    // Create a broadcast receiver to get message and show on screen
```

```
	private final BroadcastReceiver mHandleMessageReceiver = new
BroadcastReceiver() {

		@Override
		public void onReceive(Context context, Intent intent) {

			String newMessage =
intent.getExtras().getString(Config.EXTRA_MESSAGE);

			// Waking up mobile if it is sleeping

	aController.acquireWakeLock(getApplicationContext());

			// Display message on the screen
			lblMessage.append(newMessage + "\n");


			Toast.makeText(getApplicationContext(), "Got
Message: " + newMessage, Toast.LENGTH_LONG).show();

			// Releasing wake lock
			aController.releaseWakeLock();
		}
	};

	@Override
	protected void onDestroy() {
		// Cancel AsyncTask
		if (mRegisterTask != null) {
			mRegisterTask.cancel(true);
		}
		try {
```

```
                        // Unregister Broadcast Receiver
                        unregisterReceiver(mHandleMessageReceiver);

                        //Clear internal resources.
                        GCMRegistrar.onDestroy(this);

                } catch (Exception e) {
                        Log.e("UnRegister Receiver Error", "> " +
e.getMessage());
                }
                super.onDestroy();
        }

}
```

- GCMIntentService.java

```
package com.androidexample.gcm;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.util.Log;
import com.google.android.gcm.GCMBaseIntentService;

public class GCMIntentService extends GCMBaseIntentService {

        private static final String TAG = "GCMIntentService";

        private Controller aController = null;
```

```
public GCMIntentService() {
        // Call extended class Constructor GCMBaseIntentService
      super(Config.GOOGLE_SENDER_ID);
   }

   /**
    * Method called on device registered
    **/
   @Override
   protected void onRegistered(Context context, String registrationId) {

         //Get Global Controller Class object (see application tag in
AndroidManifest.xml)
          if(aController == null)
         aController = (Controller) getApplicationContext();

      Log.i(TAG, "Device registered: regId = " + registrationId);
      aController.displayMessageOnScreen(context, "Your device registred
with GCM");
      Log.d("NAME", MainActivity.name);
      aController.register(context, MainActivity.name, MainActivity.email,
registrationId);
   }

   /**
    * Method called on device unregistred
    * */
   @Override
   protected void onUnregistered(Context context, String registrationId) {
         if(aController == null)
         aController = (Controller) getApplicationContext();
      Log.i(TAG, "Device unregistered");
```

```
        aController.displayMessageOnScreen(context,
getString(R.string.gcm_unregistered));
        aController.unregister(context, registrationId);
    }

    /**
     * Method called on Receiving a new message from GCM server
     * */
    @Override
    protected void onMessage(Context context, Intent intent) {

            if(aController == null)
            aController = (Controller) getApplicationContext();

        Log.i(TAG, "Received message");
        String message = intent.getExtras().getString("price");

        aController.displayMessageOnScreen(context, message);
        // notifies user
        generateNotification(context, message);
    }

    /**
     * Method called on receiving a deleted message
     * */
    @Override
    protected void onDeletedMessages(Context context, int total) {

            if(aController == null)
            aController = (Controller) getApplicationContext();

        Log.i(TAG, "Received deleted messages notification");
```

```
        String message = getString(R.string.gcm_deleted, total);
        aController.displayMessageOnScreen(context, message);
        // notifies user
        generateNotification(context, message);
    }

    /**
     * Method called on Error
     * */
    @Override
    public void onError(Context context, String errorId) {

            if(aController == null)
            aController = (Controller) getApplicationContext();

        Log.i(TAG, "Received error: " + errorId);
        aController.displayMessageOnScreen(context,
getString(R.string.gcm_error, errorId));
    }

    @Override
    protected boolean onRecoverableError(Context context, String errorId) {

            if(aController == null)
            aController = (Controller) getApplicationContext();

        // log message
        Log.i(TAG, "Received recoverable error: " + errorId);
        aController.displayMessageOnScreen(context,
getString(R.string.gcm_recoverable_error,
                errorId));
        return super.onRecoverableError(context, errorId);
```

```
    }

    /**
     * Create a notification to inform the user that server has sent a message.
     */
    private static void generateNotification(Context context, String message) {
        int icon = R.drawable.ic_launcher;
        long when = System.currentTimeMillis();
        NotificationManager notificationManager = (NotificationManager)
                context.getSystemService(Context.NOTIFICATION_SERVICE);
        Notification notification = new Notification(icon, message, when);

        String title = context.getString(R.string.app_name);

        Intent notificationIntent = new Intent(context, MainActivity.class);
        // set intent so it does not start a new activity
        notificationIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
                Intent.FLAG_ACTIVITY_SINGLE_TOP);
        PendingIntent intent =
                PendingIntent.getActivity(context, 0, notificationIntent, 0);
        notification.setLatestEventInfo(context, title, message, intent);
        notification.flags |= Notification.FLAG_AUTO_CANCEL;

        // Play default notification sound
        notification.defaults |= Notification.DEFAULT_SOUND;

        //notification.sound = Uri.parse("android.resource://" +
context.getPackageName() + "your_sound_file_name.mp3");

        // Vibrate if vibrate is enabled
        notification.defaults |= Notification.DEFAULT_VIBRATE;
        notificationManager.notify(0, notification);
```

```
    }

}
```

- Controller.java

```
package com.androidexample.gcm;

import java.io.IOException;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Random;
import java.util.Map.Entry;

import com.google.android.gcm.GCMRegistrar;

import android.app.AlertDialog;
import android.app.Application;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.PowerManager;
import android.util.Log;

public class Controller extends Application{
```

```
        private  final int MAX_ATTEMPTS = 5;
    private  final int BACKOFF_MILLI_SECONDS = 2000;
    private  final Random random = new Random();



         // Register this account with the server.
    void register(final Context context, String name, String email, final String
regId) {

        Log.i(Config.TAG, "registering device (regId = " + regId + ")");

        String serverUrl = Config.YOUR_SERVER_URL;

        Map<String, String> params = new HashMap<String, String>();
        params.put("regId", regId);
        params.put("name", name);
        params.put("email", email);

        long backoff = BACKOFF_MILLI_SECONDS + random.nextInt(1000);

        // Once GCM returns a registration id, we need to register on our server
        // As the server might be down, we will retry it a couple
        // times.
        for (int i = 1; i <= MAX_ATTEMPTS; i++) {

            Log.d(Config.TAG, "Attempt #" + i + " to register");

            try {
            //Send Broadcast to Show message on screen
            displayMessageOnScreen(context, context.getString(
                    R.string.server_registering, i, MAX_ATTEMPTS));
```

```
            // Post registration values to web server
            post(serverUrl, params);

            GCMRegistrar.setRegisteredOnServer(context, true);

            //Send Broadcast to Show message on screen
            String message = context.getString(R.string.server_registered);
            displayMessageOnScreen(context, message);

            return;
          } catch (IOException e) {

            // Here we are simplifying and retrying on any error; in a real
            // application, it should retry only on unrecoverable errors
            // (like HTTP error code 503).

            Log.e(Config.TAG, "Failed to register on attempt " + i + ":" + e);

            if (i == MAX_ATTEMPTS) {
               break;
            }
            try {

               Log.d(Config.TAG, "Sleeping for " + backoff + " ms before
retry");
               Thread.sleep(backoff);

            } catch (InterruptedException e1) {
               // Activity finished before we complete - exit.
               Log.d(Config.TAG, "Thread interrupted: abort remaining
retries!");
```

```
            Thread.currentThread().interrupt();
            return;
          }

          // increase backoff exponentially
          backoff *= 2;
        }
      }

      String message = context.getString(R.string.server_register_error,
          MAX_ATTEMPTS);

      //Send Broadcast to Show message on screen
      displayMessageOnScreen(context, message);
    }

    // Unregister this account/device pair within the server.
    void unregister(final Context context, final String regId) {

      Log.i(Config.TAG, "unregistering device (regId = " + regId + ")");

      String serverUrl = Config.YOUR_SERVER_URL + "/unregister";
      Map<String, String> params = new HashMap<String, String>();
      params.put("regId", regId);

      try {
        post(serverUrl, params);
        GCMRegistrar.setRegisteredOnServer(context, false);
        String message = context.getString(R.string.server_unregistered);
        displayMessageOnScreen(context, message);
      } catch (IOException e) {
```

```
        // At this point the device is unregistered from GCM, but still
        // registered in the our server.
        // We could try to unregister again, but it is not necessary:
        // if the server tries to send a message to the device, it will get
        // a "NotRegistered" error message and should unregister the device.

        String message = context.getString(R.string.server_unregister_error,
                e.getMessage());
        displayMessageOnScreen(context, message);
    }
}

// Issue a POST request to the server.
private static void post(String endpoint, Map<String, String> params)
        throws IOException {

    URL url;
    try {

        url = new URL(endpoint);

    } catch (MalformedURLException e) {
        throw new IllegalArgumentException("invalid url: " + endpoint);
    }

    StringBuilder bodyBuilder = new StringBuilder();
    Iterator<Entry<String, String>> iterator = params.entrySet().iterator();

    // constructs the POST body using the parameters
    while (iterator.hasNext()) {
        Entry<String, String> param = iterator.next();
        bodyBuilder.append(param.getKey()).append('=')
```

```
            .append(param.getValue());
        if (iterator.hasNext()) {
            bodyBuilder.append('&');
        }
    }

    String body = bodyBuilder.toString();

    Log.v(Config.TAG, "Posting '" + body + "' to " + url);

    byte[] bytes = body.getBytes();

    HttpURLConnection conn = null;
    try {

        Log.e("URL", "> " + url);

        conn = (HttpURLConnection) url.openConnection();
        conn.setDoOutput(true);
        conn.setUseCaches(false);
        conn.setFixedLengthStreamingMode(bytes.length);
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type",
                "application/x-www-form-urlencoded;charset=UTF-8");
        // post the request
        OutputStream out = conn.getOutputStream();
        out.write(bytes);
        out.close();

        // handle the response
        int status = conn.getResponseCode();
```

```
        // If response is not success
        if (status != 200) {

          throw new IOException("Post failed with error code " + status);
        }
      } finally {
        if (conn != null) {
           conn.disconnect();
        }
      }
    }



        // Checking for all possible internet providers
public boolean isConnectingToInternet(){

    ConnectivityManager connectivity =
                        (ConnectivityManager) getSystemService(
                         Context.CONNECTIVITY_SERVICE);
      if (connectivity != null)
      {
         NetworkInfo[] info = connectivity.getAllNetworkInfo();
         if (info != null)
            for (int i = 0; i < info.length; i++)
               if (info[i].getState() == NetworkInfo.State.CONNECTED)
               {
                  return true;
               }

      }
      return false;
```

```
    }

   // Notifies UI to display a message.
   void displayMessageOnScreen(Context context, String message) {

       Intent intent = new Intent(Config.DISPLAY_MESSAGE_ACTION);
       intent.putExtra(Config.EXTRA_MESSAGE, message);

       // Send Broadcast to Broadcast receiver with message
       context.sendBroadcast(intent);

    }


   //Function to display simple Alert Dialog
   public void showAlertDialog(Context context, String title, String message,
                               Boolean status) {
                   AlertDialog alertDialog = new
AlertDialog.Builder(context).create();

                   // Set Dialog Title
                   alertDialog.setTitle(title);

                   // Set Dialog Message
                   alertDialog.setMessage(message);

                   if(status != null)
                           // Set alert dialog icon
                           alertDialog.setIcon((status) ? R.drawable.success :
R.drawable.fail);

                   // Set OK Button
```

```
                alertDialog.setButton("OK", new
DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int which)
{

                        }
                });

                // Show Alert Message
                alertDialog.show();
        }

    private PowerManager.WakeLock wakeLock;

    public  void acquireWakeLock(Context context) {
        if (wakeLock != null) wakeLock.release();

        PowerManager pm = (PowerManager)
context.getSystemService(Context.POWER_SERVICE);

        wakeLock = pm.newWakeLock(PowerManager.FULL_WAKE_LOCK |
                PowerManager.ACQUIRE_CAUSES_WAKEUP |
                PowerManager.ON_AFTER_RELEASE, "WakeLock");

        wakeLock.acquire();
    }

    public  void releaseWakeLock() {
        if (wakeLock != null) wakeLock.release(); wakeLock = null;
    }

}
```

- Config.java

```
package com.androidexample.gcm;

public interface Config {



        // CONSTANTS
        static final String YOUR_SERVER_URL =
"YOUR_SERVER_URL/gcm_server_files/register.php";
        // YOUR_SERVER_URL : Server url where you have placed your
server files
   // Google project id
   static final String GOOGLE_SENDER_ID = "9432966778899";  // Place
here your Google project id

   /**
    * Tag used on log messages.
    */
   static final String TAG = "GCM Android Example";

   static final String DISPLAY_MESSAGE_ACTION =
        "com.androidexample.gcm.DISPLAY_MESSAGE";

   static final String EXTRA_MESSAGE = "message";



}
```

- Config.php

```
<?php
/**
```

```
 * Database config variables
 */
define("DB_HOST", "localhost");
define("DB_USER", "YOUR-DATABASE-USER");
define("DB_PASSWORD", "YOUR-DATABASE-PASSWORD");
define("DB_DATABASE", "gcm");

/*
 * Google Cloud Messaging API Key
 */
define("GOOGLE_API_KEY", "XXXXXXXXXXXXXXXXXXXXXX"); //
Place your Google API Key

?>
```

- Function.php

```
<?php

  //Storing new user and returns user details

  function storeUser($name, $email, $gcm_regid) {

      // insert user into database
      $result = mysql_query("INSERT INTO gcm_users(name, email,
gcm_regid, created_at) VALUES('$name', '$email', '$gcm_regid', NOW())");

      // check for successful store
      if ($result) {

         // get user details
         $id = mysql_insert_id(); // last inserted id
```

```
            $result = mysql_query("SELECT * FROM gcm_users WHERE id =
$id") or die(mysql_error());
            // return user details
            if (mysql_num_rows($result) > 0) {
                return mysql_fetch_array($result);
            } else {
                return false;
            }

        } else {
            return false;
        }
    }

    /**
     * Get user by email
     */
    function getUserByEmail($email) {
        $result = mysql_query("SELECT * FROM gcm_users WHERE email =
'$email' LIMIT 1");
        return $result;
    }

    // Getting all registered users
    function getAllUsers() {
        $result = mysql_query("select * FROM gcm_users");
        return $result;
    }

    // Validate user
    function isUserExisted($email) {
```

```
        $result    = mysql_query("SELECT email from gcm_users WHERE
email = '$email'");
        $NumOfRows = mysql_num_rows($result);
        if ($NumOfRows > 0) {
            // user existed
            return true;
        } else {
            // user not existed
            return false;
        }
    }

            //Sending Push Notification
   function send_push_notification($registatoin_ids, $message) {



        // Set POST variables
        $url = 'https://android.googleapis.com/gcm/send';

        $fields = array(
            'registration_ids' => $registatoin_ids,
            'data' => $message,
        );

        $headers = array(
            'Authorization: key=' . GOOGLE_API_KEY,
            'Content-Type: application/json'
        );
                    //print_r($headers);
        // Open connection
        $ch = curl_init();
```

```
        // Set the url, number of POST vars, POST data
        curl_setopt($ch, CURLOPT_URL, $url);

        curl_setopt($ch, CURLOPT_POST, true);
        curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

        // Disabling SSL Certificate support temporarly
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);

        curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($fields));

        // Execute post
        $result = curl_exec($ch);
        if ($result === FALSE) {
            die('Curl failed: ' . curl_error($ch));
        }

        // Close connection
        curl_close($ch);
        echo $result;
    }
?>
```

- Index.php

```
<?php
  require_once('loader.php');

   $resultUsers =  getAllUsers();
        if ($resultUsers != false)
                $NumOfUsers = mysql_num_rows($resultUsers);
        else
```

```
                $NumOfUsers = 0;
?>
<!DOCTYPE html>
<html>
<head>
<title></title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script
>
<script type="text/javascript">
        $(document).ready(function(){

        });
        function sendPushNotification(id){
            var data = $('form#'+id).serialize();
            $('form#'+id).unbind('submit');
            $.ajax({
                url: "send_push_notification_message.php",
                type: 'GET',
                data: data,
                beforeSend: function() {

                },
                success: function(data, textStatus, xhr) {
                    $('.push_message').val("");
                },
                error: function(xhr, textStatus, errorThrown) {

                }
            });
            return false;
```

```
        }
</script>
<style type="text/css">

        h1{
            font-family:Helvetica, Arial, sans-serif;
            font-size: 24px;
            color: #777;
        }
        div.clear{
            clear: both;
        }

        textarea{
            float: left;
            resize: none;
        }

</style>
</head>
<body>


<table  width="910" cellpadding="1" cellspacing="1" style="padding-
left:10px;">
<tr>
<td align="left">
<h1>No of Devices Registered: <?php echo $NumOfUsers; ?></h1>
<hr/>
</td>
</tr>
<tr>
```

```
<td align="center">
<table width="100%" cellpadding="1" cellspacing="1" style="border:1px
solid #CCC;" bgcolor="#f4f4f4">
<tr>

<?php
            if ($NumOfUsers > 0) {
               $i=1;
               while ($rowUsers = mysql_fetch_array($resultUsers)) {
                                                   if($i%3==0)
                                                    print "</tr><tr><td
colspan='2'> </td></tr><tr>";
             ?>
<td align="left">
<form id="<?php echo $rowUsers["id"] ?>" name="" method="post"
onSubmit="return sendPushNotification('<?php echo $rowUsers["id"] ?>')">
<label><b>Name:</b></label><span><?php echo $rowUsers["name"]
?></span>
<div class="clear"></div>
<label><b>Email:</b></label><span><?php echo $rowUsers["email"]
?></span>
<div class="clear"></div>
<div class="send_container">
<textarea rows="3" name="message" cols="25" class="push_message"
placeholder="Type push message here"></textarea>
<input type="hidden" name="regId" value="<?php echo
$rowUsers["gcm_regid"] ?>"/>
<input type="submit"  value="Send Push Notification" onClick=""/>
</div>
</form>
</td>
<?php }
```

```
                } else { ?>
    <td>
                    User not exist.
    </td>
    <?php } ?>

    </tr>
    </table>
    </td>
    </tr>
    </table>



    </body>
    </html>
```

- Loader.php

```
    <?php
    require_once('config.php');
    require_once('function.php');

    // connecting to mysql
    $conn = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD);
    // selecting database
    if(!mysql_select_db(DB_DATABASE))
      print "Not connected with database.";



    ?>
```

- Register.php

```
    <?php
    require_once('loader.php');
```

```
// return json response
$json = array();

$nameUser  = $_POST["name"];
$nameEmail = $_POST["email"];
$gcmRegID  = $_POST["regId"]; // GCM Registration ID got from device

/**
 * Registering a user device in database
 * Store reg id in users table
 */
if (isset($nameUser) && isset($nameEmail) && isset($gcmRegID)) {

        // Store user details in db
   $res = storeUser($nameUser, $nameEmail, $gcmRegID);

   $registatoin_ids = array($gcmRegID);
   $message = array("product" => "shirt");

   $result = send_push_notification($registatoin_ids, $message);

   echo $result;
} else {
   // user details not found
}
?>
```

- Send_push_notification.php

```
<?php

 require_once('loader.php');
```

```
        $gcmRegID    = $_GET["regId"]; // GCM Registration ID got from
device
        $pushMessage = $_GET["message"];

        if (isset($gcmRegID) && isset($pushMessage)) {


                $registatoin_ids = array($gcmRegID);
                $message = array("price" => $pushMessage);

                $result = send_push_notification($registatoin_ids, $message);

                echo $result;
        }
?>
```

8. REFERENCES

8.1 Books:

- Matt Richardson & Shawn Wallace, "Getting Started with Raspberry Pi",O'Reilly Media, December 2012 , First Edition.

8.2 Research Paper:

- Lei Zang, Gaofeng Wang, "Design and Development of Automatic Fire Alarm System based on Wireless Sensor Networks", Proceedings of 2009 International Symposium on Information Processing, August 21-23 ,2009, pp. 410-413
- Shi, Jianyong ,"Application of computer integration technology for the fire safety analysis",Tsinghua Science and Technology Journal, Volume 13 , 2008, pp. 387-392
- Z.Liu, J.Makar and A.K. Kim, "Development of Fire Detection Systems in the Intelligent Building", 12th International Conference on Automatic Fire detection, March 25-28, 2001, pp. 1-14
- Huber Flores, Mobile Cloud Messaging Supported by XMPP Primitives, ACM Digital Library, pp. 17-24, 2013.
- Dmitry Namiot, Local Area Messaging for Smartphones, International Journal of Open Information Technologies, 2013

8.3 Technical Report/Article:

- Fire Alarm System Research – Where it's been and where it's going

8.4 Official Reports (By respective Govt. for statistics):

- Sushil Gupta , RMSI, "Fire Hazard and Risk Analysis in the country for Revamping the Fire Services in the Country", Directorate General NDRF & Civil Defence (Fire), Ministry of Home Affairs, New Delhi , November 2012.
- Tropical Fire Report Series, "Thanksgiving Day Fires in Residential Buildings", Department of Homeland Security, U.S. Fire Administration, National Fire Center , Emmitsburg Maryland, November 2010 , Volume 2, Issue 5

8.5 Web References:

- Fire statistics : http://www.delhi.gov.in/wps/wcm/connect/doit_fire/FIRE/Home/About+Us/Statistical+Report+of+Delhi+Fire
- Interfacing sensor to Raspberry Pi :https://learn.adafruit.com/adafruits-raspberry-pi-lesson-11-ds18b20-temperature-sensing?view=all

8.6 Official Documentation:

- Raspberry Pi : http://www.raspberrypi.org/documentation/
- IBM Bluemix Cloud: https://www.ng.bluemix.net/docs/#
- Android Developer Docs: https://developer.android.com/
- GCM Docs : https://developer.android.com/google/gcm/index.html
- Android Developer Location Docs: https://developer.android.com/google/play-services/location.html