

Transactions on Computer Systems and Networks

Jivan S. Parab ·
Madhusudan Ganuji Lanjewar ·
Marlon Darius Sequeira · Gourish Naik ·
Arman Yusuf Shaikh

Python Programming Recipes for IoT Applications

 Springer

Transactions on Computer Systems and Networks

Series Editor

Amlan Chakrabarti, Director and Professor, A. K. Choudhury School of
Information Technology, Kolkata, West Bengal, India

Jivan S. Parab · Madhusudan Ganuji Lanjewar ·
Marlon Darius Sequeira · Gourish Naik ·
Arman Yusuf Shaikh

Python Programming Recipes for IoT Applications

Jivan S. Parab
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Madhusudan Ganuji Lanjewar
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Marlon Darius Sequeira
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Gourish Naik
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Arman Yusuf Shaikh
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

ISSN 2730-7484

ISSN 2730-7492 (electronic)

Transactions on Computer Systems and Networks

ISBN 978-981-19-9465-4

ISBN 978-981-19-9466-1 (eBook)

<https://doi.org/10.1007/978-981-19-9466-1>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2023

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd. The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Contents

1	PYTHON Programming and IoT	1
1.1	Introduction to Python	1
1.2	Can Python Replace C/C++?	2
1.3	Overview of Python Programming	2
1.4	Python for Embedded System	23
1.5	Introduction to IoT	23
1.6	IoT Applications	25
	References	26
2	Configuring Raspberry Pi, MicroPython Pyboard, and Jetson Nano for Python	27
2.1	Raspberry Pi Board Features	27
2.1.1	Configuration of Raspberry Pi	29
2.2	MicroPython Pyboard Features	33
2.2.1	Configuration of MicroPython Pyboard	34
2.3	Jetson Nano Board Features	40
2.3.1	Configuration of Jetson Nano Board	41
	References	48
3	Simple Applications with Raspberry Pi	49
3.1	Blinking of LED	49
3.2	OLED Display Interface	55
3.3	Camera Interfacing	62
3.4	Motor Control (DC Motor, Stepper Motor, and Servo Motor)	69
3.5	Raspberry Pi and Mobile Interface Through Bluetooth	83
	References	87
4	MicroPython PyBoard for IoT	89
4.1	Home Automation	90
4.2	Smart e-waste Bin	96
4.3	Industrial Environmental Monitoring	105

4.4	Greenhouse Monitoring	111
4.5	Aquaculture Monitoring	116
	References	121
5	FoG and Cloud Computing with Jetson Nano	123
5.1	Introduction to FoG Computing	123
5.2	Architecture Model of FoG	126
5.3	Introduction to Cloud Computing	127
5.4	Cloud Computing Architecture	129
5.5	Role of FoG and Cloud Computing in IoT	131
5.6	Examples of FoG and Cloud Computing	131
5.6.1	Patient Monitoring system with Cloud	131
5.6.2	Home security with FoG	138
	References	165
6	Machine Learning (ML) in IoT with Jetson Nano	167
6.1	What is AI?	167
6.2	Concepts of Machine Learning (ML) and Deep Learning (DL)	168
6.3	Pattern Recognition Using ML with Cloud	171
6.4	Object Classification Using ML with FoG	178
6.5	Prediction of Unknown Glucose Concentration Using ML at EDGE	186
	References	192

Chapter 5

FoG and Cloud Computing with Jetson Nano



Abstract FoG and cloud paradigms are helping the IoT devices to offload the computing in order to reduce power consumption and increase battery life. The Jetson Nano SBC is an exciting platform which can act as a FoG node and as an intermediate layer between the cloud and IoT devices. The Jetson Nano module is a small Artificial Intelligence (AI) computer that has the performance and power efficiency needed to run modern AI workloads, multiple neural networks in parallel and process data from several sensors simultaneously. The ongoing chapter introduces the concepts of FoG and Cloud computing along with its role in IoT. At the end, this chapter covers implementation of FoG and Cloud computing.

Keywords FoG · Cloud · Patient monitoring · Home security · Fire alert system · Home safety lock · Surveillance

5.1 Introduction to FoG Computing

Cloud computing has gained widespread acceptance due to the services such as data storage, computing, etc. offered by it at a very affordable cost. To achieve this, centralized data centers are offered by big players. More businesses are able to adopt cloud computing for their operations due to the ready adaptability and reduced cost of cloud services (Chiang and Zhang 2016). This offloads a lot of workload and in turn increases productivity (Khan et al. 2017). To keep up with the market needs, cloud computing was able to evolve and offers better performance, security, and reliability. Today, cloud is able to offer various models for services such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

The world has seen an unprecedented rise in devices connected to the Internet over the decade. According to Cisco system inc, there are around 50 billion connected devices (Evans 2011). Naturally, there is high demand for quality cloud services and IoT devices. Since the overall operation of cloud data servers is centralized, it proves to be a hindrance for IoT devices spread over a large geographic area. Problems such as network congestion and high latency are common occurrences in such scenarios. FoG computing is an extension of the existing cloud platform which

was put forward by Cisco to resolve the issue faced and elaborated on above. Cisco was the first player to formally put forward the concept of FoG computing (Tang et al. 2015). An additional layer called the FoG layer is introduced in between the cloud data centers and the end-users. In principle, this layer is physically much closer to the end-user or the IoT nodes. It helps to achieve the decentralization of the data centers.

FoG computing together with cloud computing and IoT devices (End devices) forms three-tier architecture as shown in Fig. 5.1 (Taneja and Davy 2016). The bottom Tier 1 consists of the End /IoT devices which are the generators of the data. The IoT devices are equipped with sensors that generate the raw data. Tier 2 is the FoG computing layer. This computing layer generally consists of devices such as gateways, routers, switches, etc. which can provide processing and storage of information. Hence, this layer is sometimes referred to as FoG intelligence. If the devices which form the part of the FoG computing layer wishes to send data to the cloud they can do so on a periodic basis. In FoG computing architecture, not every data packet is redirected to the cloud, instead all real-time analysis and latency sensitive applications have a dependency to run from the FoG layer itself. Tier 3 is named as cloud computing layer, also referred to as the cloud intelligence, which leverages modern infrastructure such as data centers to provide enormous storing and processing capabilities to the lower layers.

Dastjerdi et al. defined FoG computing as a distributed computing environment extending the services of cloud computing to one-hop distance from the user (Dastjerdi et al. 2016). FoG network consists of multiple FoG nodes, which communicates with one another and also in sync with the cloud data center. FoG computing and cloud computing work together to enhance the quality and performance of any system. In this type of computing, one utilizes a network of FoG nodes which are placed at just one leap from the end-user or IoT device. This allows the local data processing on the FoG node rather than on the cloud data center which may be placed geographically at a distant location. In addition to data processing, FoG node may also provide

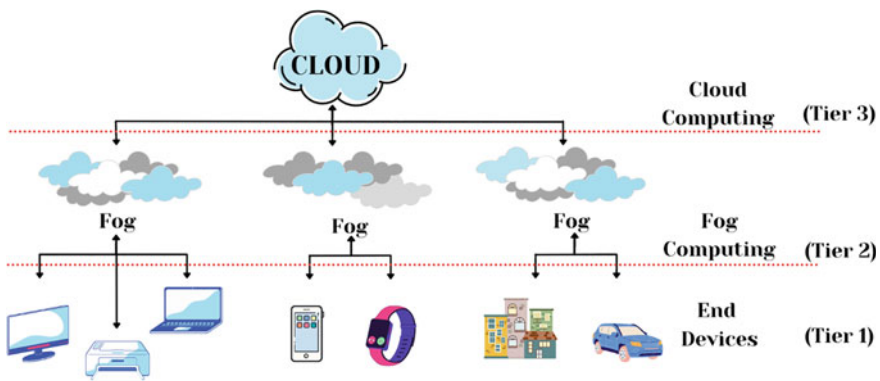


Fig. 5.1 Three-tier architecture

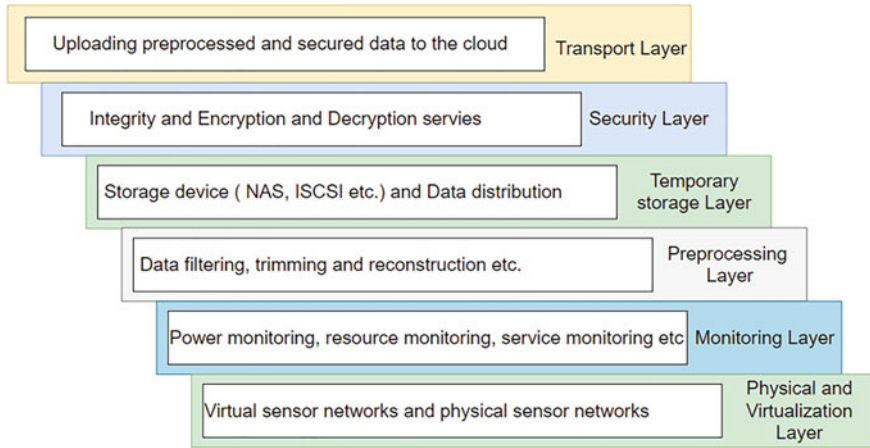


Fig. 5.2 Six-layer FoG architecture

networking between the end-users (edges) and the cloud data centers. The features like communication, mobility, and computing offered by FoG node aim to provide low latency.

FoG Computing Essential Characteristics

This section briefly discusses the essential characteristics of FoG computing. If a device possesses storage, computing capacity, and network connectivity, it can operate as a FoG node. They can be deployed anywhere within a network. Examples of such devices are routers, switches, embedded servers and industrial controllers, and cameras for video surveillance (Atlam et al. 2018). FoG Computing in general should have the below characteristics:

- *Location awareness and low latency:* FoG computing features location awareness and can be set up in any location. As FoG nodes are closer to the end devices, FoG computing gives low latency operation (Atlam et al. 2018).
- *Scalability:* One of the main features of FoG computing is distributed computing and storage networks which work as end devices/sensor networks. Additionally, FoG computing supports the ready addition of new end devices/sensor networks (Atlam et al. 2018).
- *Geographical distribution:* Services and applications given by FoG nodes are distributed and can be deployed anywhere, in contrast to the services of centralized data servers.
- *Heterogeneity and Interoperability:* FoG nodes or end devices are designed by different manufacturers but still possess the ability to work on different platforms and across different service providers (Atlam et al. 2018).
- *Real-time interactions:* FoG computing gives a real-time interaction between FoG nodes rather than the batch processing model provided by the cloud (Atlam et al. 2018).

- *Support for mobility:* Mobile devices can be connected directly in FoG applications (Atlam et al. 2018).

5.2 Architecture Model of FoG

In FoG computing, the facilities provided by the data centers of cloud computing are taken to the edge of the network. FoG nodes offer limited computing, storing, and networking capabilities. These capabilities are provided in a distributed manner that spans the Edge/IoT devices and the traditional cloud data centers. Here the sole objective is to give minimal latency for time-sensitive application running on the Edge/IoT devices (Atlam et al. 2018; Shi et al. 2015).

The architecture for FoG is made up of six layers as shown in the figure 5.2 (Mukherjee et al. 2018; Aazam and Huh 2014, 2015; Muntjir et al. 2017) namely, physical and virtualization, monitoring, pre-processing, temporary storage, security, and transport layer.

The lowest layer is the physical and virtualization layer and is made up of virtual sensor networks along with physical and wireless sensor networks. In this layer, myriad types of sensors are geographically distributed for environment sensing which generates raw data. This raw data is passed to the next layers via gateways for further processing and filtering (Liu et al. 2017).

The next layer is the monitoring layer which monitors the resource utilization and the availability of FoG nodes along with the other network elements. This layer is also responsible for monitoring services, performance, power consumption, and tracking the status of applications running on FoG nodes (Mukherjee et al. 2018; Aazam and Huh 2015).

Preprocessing and temporary storage layer handles the tasks of data management. The data forwarded by the lower layers are analyzed, trimmed, and filtered to extract meaningful information. This preprocessed data in the below layer is then required to be temporarily stored in the next layer (Aazam and Huh 2015; Muntjir et al. 2017).

The encryption/decryption of the data on the FoG nodes is handled at the security layer. Additional integrity mechanism is also applied to the data to protect the data from being tampered.

The topmost layer of the architecture is the transport layer where the data is uploaded to the cloud to generate more useful services (Aazam and Huh 2015; Muntjir et al. 2017). As the FoG nodes are constrained by limited resources, the protocols used for communication on the FoG nodes have to be lightweight, efficient, and customizable (Aazam et al. 2014; Marques et al. 2017).

5.3 Introduction to Cloud Computing

Prior to the introduction of cloud computing, if there is a need of computational facility, one has to invest in the hardware, software, networking, and storage requirements for such an endeavor. The cost includes the upfront cost of the real estate to house the hardware, the maintenance, and the operational cost. This becomes an enormous cost for an individual or enterprise needing a large computing facility (Chandrasekaran 2015). With the cloud, it is possible to use the computing facility of a provider as and when it is needed. Hence, cloud computing can be termed as utilizing the computing infrastructure made available by a service provider, to the extent needed, and paying only for the service consumed.

What is Cloud Computing?

The popular definitions among the experts on cloud computing were put forward by the National Institute of Standards and Technology (NIST). It states that “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” (Sunyaev 2020).

Cloud computing Essential Characteristics:

A cloud service should have below essential characteristics such as on-demand self-service, ubiquitous access, multi-tenancy, location independence, rapid elasticity, and metering.

On-demand self-service: A cloud service should allow a user/consumer to independently avail computing resources, such as network storage and server time, without human intervention (Mell and Grance 2011).

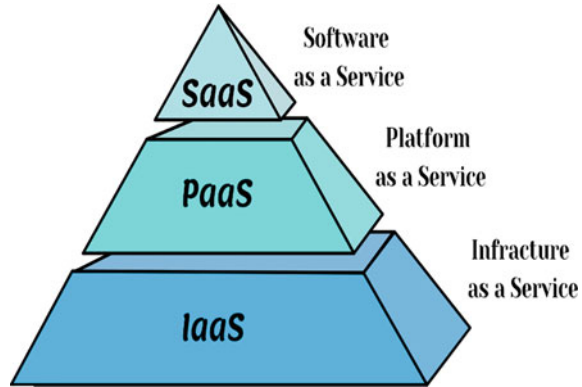
Ubiquitous access: The services are disseminated using a broadband network prominently using the Internet. The services can be availed on various devices such as smart phones, tablets, and workstations using standard communication interfaces. This enables the users to avail any cloud service from any device or platform at any given time (Mell and Grance 2011; Iyer and Henderson 2010).

Resource pooling and Multi-tenancy: Computing resource pooling is done at the providers end in order to service users/consumers using a multi-tenant architecture. Sharing computing resources is part of what could make cloud computing economically beneficial (Mell and Grance 2011; Arasaratnam 2011).

Location Independence: A user/consumer wanting to use cloud service is given a sense of location independence, where the user has no knowledge nor control of the physical location of the computing resources. The user may be provided a control of choosing at a higher level of abstraction, such as a choice of country, state, or data center (Mell and Grance 2011).

Rapid Elasticity: A sense of elasticity is provided to the user where the resource can be elastically commissioned and decommissioned to keep pace with the current

Fig. 5.3 Cloud service pyramid



resources needs. The rapid elasticity gives the users/customers an impression of the availability of unlimited resources.

Metering service: A metering capability is incorporated by the service provider in order to automatically control and optimize the resource utilization. The metering is done for the resources such as bandwidth, storage, processing, etc. This provides transparency to both the consumer and service provider.

Cloud Computing Service Models

The services provided by cloud computing have evolved into three typical service models, namely, (1) Infrastructure as a Service, (2) Platform as a Service, and (3) Software as a Service as shown in Fig. 5.3. One finds a hierarchical organization of these top three models based on the level of abstraction of the capabilities made available by individual layers (Kumar and DuPree 2011).

Infrastructure as a Service (IaaS): In the IaaS scheme, a consumer acquires processing, storage, networking, and other resources from the service provider. Here, the consumer is free to utilize the acquired resources at his discretion by running or deploying arbitrary software of his choice which can include applications and operating systems. In IaaS, the service provider gives to the consumer a large number of virtual resources by dividing a very large physical resource of the infrastructure (). The most popular IaaS service providers in the market are IBM, Microsoft, Rackspace, NTT, Oracle, Fujitsu, and Amazon (Marston et al. 2011).

Platform as a Service (PaaS): In PaaS cloud model, a user/consumer develops applications to run on the cloud infrastructure. The applications are developed using programming languages, services, tools, and libraries supported by the provider (Mell and Grance 2011). The user is not given the control or management of the cloud infrastructure, but given the control of configuration settings for the application hosting environment and applications. A PaaS provides a cloud developer freedom to design, build, test and deploy custom applications (). Leading providers offering PaaS are Amazon, Microsoft, Alibaba, Google, IBM, and Rackspace (Voorsluys et al. 2011).

Software as a Service (SaaS): As the name suggests the user/consumer uses the applications provided by the cloud service provider. These applications are running on the cloud infrastructure. The applications are accessed using a client device running a thin client interface such as a browser or a program interface. The management and control of the cloud infrastructure are not done by the user/consumer, while a limited user specific application configuration setting is controlled by the user (Mell and Grance 2011). Therefore, user/consumer enjoys abstraction from all the finer details of the current applications running behind the scenes (Arasaratnam 2011;). Leading SaaS providers are Salesforce, Microsoft, SAP, Oracle, Adobe Systems, and IBM.

Deployment Models

There are four different deployment models such as Private, Public, Community, and Hybrid cloud.

Private cloud: The infrastructure is exclusively reserved for use by the single organizations who's services are given to a large number of consumers, namely, business units. The cloud infrastructure may be housed on the premises of the organization or off premises (Mell and Grance 2011).

Community cloud: The infrastructure is exclusively reserved for a particular community of users from organizations that have common concerns. The cloud infrastructure may be housed on the premises of the organization or off premises (Mell and Grance 2011).

Public cloud: The infrastructure is not reserved for specific users and is open to the general public.

Hybrid cloud: Here, two or more distinct cloud deployment models such as community, public, or private are combined together. However, the individual entities remain unique.

5.4 Cloud Computing Architecture

The cloud architecture is split into two parts, namely, frontend and backend. Fig. 5.4 depicts an internal architectural view of cloud computing.

Cloud computing architecture consists of client infrastructure, application, service, runtime, storage, infrastructure, administration, and security (ITCandor 2018).

Frontend: The cloud architecture's frontend refers to the cloud's client infrastructure, which includes all user interfaces and applications.

Backend: Backend refers to the cloud itself which is used by the service provider. It controls resources and implements security measures. It also contains massive storage, virtual applications, virtual computers, traffic management techniques, deployment models, and so on.

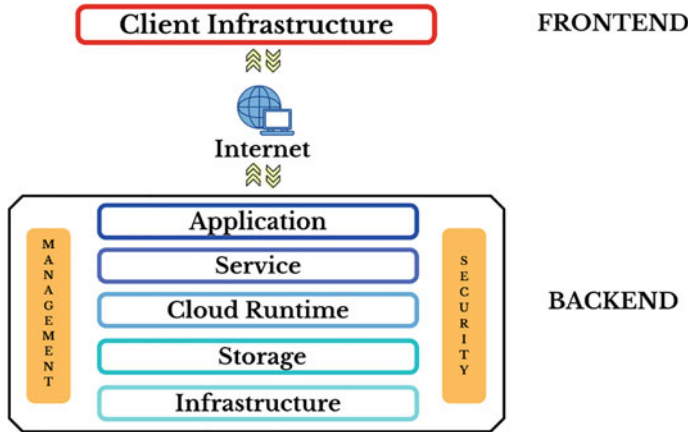


Fig. 5.4 Internal architectural of Cloud

- The Application Layer consists of a software platform that is accessed by the client and offers services in the backend based on the client's needs.
- The Service Layer includes three types of cloud-based services: SaaS, PaaS, and IaaS, and it offers selectable services based on customer requests.
- The Cloud Runtime Layer is in-charge of providing the virtual machine with an execution and runtime platform/environment.
- Storage Layer handles the management and scalability of storage services provided to clients.
- The Infrastructure Layer refers to cloud software and hardware such as servers, storage, network equipment, and so on.
- The Management Layer manages all the backend components.
- Security Layer is responsible to implement different security mechanisms in the backend.
- For interaction and communication, the Internet Layer serves as a bridge between the frontend and the backend.

There are various advantages of Cloud Computing Architecture, such as a simpler overall computing system, improved data processing needs, more security, improved disaster recovery, lower operational costs, and so on.

5.5 Role of FoG and Cloud Computing in IoT

The advent of IoT has seen smart devices permeating every home, vehicle, and workplace with connectivity to the Internet. Due to technological advancements, the things which surround us are potentially getting connected to the Internet and hence there is a huge stress on the current Internet and cloud infrastructure in place.

The conventional approach is to utilize a centralized cloud for processing which is localized at one site. But due to the proliferation of IoT devices, the sheer amount of data generated by these devices is increasing day by day. The existing cloud infrastructure will be severely strained by this massive explosion of data [29].

FoG computing allows computing to happen via IoT devices and only pushes relevant data to the cloud. The FoG brings the cloud closer to the objects that generate and act on IoT data. The FoG nodes can be deployed anywhere with a network between cloud and IoT devices. They analyze the most time-sensitive data at the network edge instead of sending vast amounts of IoT data to the cloud. Only required data can be sent to the cloud for historical analysis and longer term storage. It is very important to point out that FoG does not eliminate the cloud but complements it to improve the efficiency (Schneider and Sunyaev 2016).

5.6 Examples of FoG and Cloud Computing

The FoG and cloud computing has become ubiquitous in the modern computing scenario. On a regular basis, FoG and cloud service providers are adding exciting features to the existing platforms. Here, authors have brought out the salient features of both the paradigms especially relating to IoT with various applications. Under cloud computing, authors have implemented a Patient Monitoring system to monitor Oxygen Saturation, Pulse, and Body Temperature. FoG computing is leveraged for Home Security in which Home Surveillance, Home Safety Lock, and Fire Alert systems are implemented.

5.6.1 Patient Monitoring system with Cloud

In modern society, the incidence of chronic diseases has increased due to different risk factors such as dietary habits, physical inactivity, alcohol consumption, etc. It is said that in the next 10 years, deaths from chronic diseases will increase by 17%. If these diseases are not monitored and treated early, they can lead to several complications and even pose threat to life. Therefore, monitoring of health parameters is of utmost importance (Schneider and Sunyaev 2016).

This application demonstrates the Patient Monitoring system. Here, various health parameters such as Oxygen Saturation (SpO2), Pulse Rate, and Body Temperature are monitored using appropriate sensors. The main heart of this system is Jetson Nano SBC, which is interfaced with these sensors, as shown in Fig. 5.5. The Oxygen Saturation and Pulse Rate are obtained from the MAX30102 sensor which is interfaced with the Jetson nano SBC using I2C. The Body Temperature is obtained using the MLX90614 IR Temperature Sensor which is also interfaced using I2C. The health parameters are updated to ThingSpeak Cloud for monitoring from any geographic location. ThingSpeak is an open IoT platform for monitoring patients data online. Table 5.1 tabulates the pin connection of MAX30102 and MXL90614 with Jetson Nano.

MAX30102 is an integrated pulse oximetry and heart-rate sensor. It integrates two LEDs (IR and Red), a photodetector, optimized optics, and low-noise analog signal processing to detect pulse oximetry and heart-rate signals. It is fully configurable through software registers, and the digital output data is stored in a 32-deep FIFO within the device. It also has an ambient light cancellation (ALC), 18-bit sigma delta ADC, and discrete time filter. It has an ultra-low-power operation which makes it ideal for battery operated systems. MAX30102 operates within a supply range of 1.7 to 2 V. The same sensor can be used for various applications such as fitness assistant, wearable devices, etc.

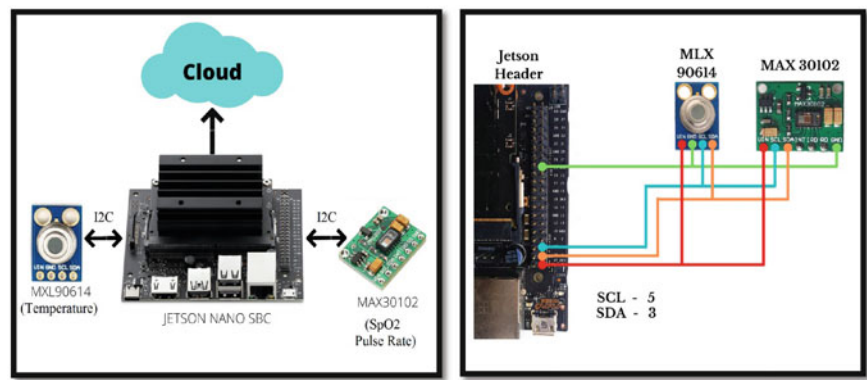


Fig. 5.5 Conceptual and pin diagram of Patient Monitoring system

Table 5.1 Pin connections for patient monitoring

Jetson Nano SBC	MAX30102	MXL90614
3.3 V	VIN	VIN
GND	GND	GND
PIN 3 (I2C 1 SDA)	SDA	SDA
PIN 5 (I2C 1 SCL)	SCL	SCL

MLX90614 sensor is manufactured by Melexis Microelectronics Integrated systems. It works on the principle of infrared thermopile sensor for temperature measurement. These sensors consist of two units embedded internally to give the temperature output. The first unit is the sensing unit which has an infrared detector, followed by the second unit which performs the computation of the data using Digital Signal Processing (DSP). This sensor works on Stefan-Boltzmann law which explains power radiated by a black body in terms of its temperature. MLX90614 sensor converts the computational value into 17-bit ADC and that can be accessed using the I2C protocol. This sensor measures the ambient temperature as well as object temperature with the resolution of 0.02 °C and can be operated with a supply ranging from 3.6 V to 5 V.

Implementation and Configuration steps:

Step 1: Before using I2C on Jetson Nano SBC, one need to install the relevant libraries by executing the following command on Jetson Nano terminals.

```
sudo apt-get install python-setuptools
sudo apt-get install -y i2c-tools
```

There are two I2C ports available on the J41 Header of the Jetson Nano SBC. Here, authors have used the I2C-1 for both the sensors as shown in Fig. 5.5. Pin 3 and Pin 5 of J41 header is SDA and SCL of I2C-1, respectively. One can test if the I2C device is properly connected by using the below command.

```
sudo i2cdetect -y -r 1
```

Step 2: Configuration of Library for MAX30102.

Create a folder named “Patient Monitoring system” → Download the library for MAX30102 from the github link: <https://github.com/doug-burrell/max30102> → Extract the downloaded file and place it in the above created folder. A few Python libraries, namely, 'smbus' and 'numpy', are required. Use “apt” to install the these libraries.

```
sudo apt install python-smbus
sudo apt install python-numpy
```

There are some changes to be done in the file '*heartrate_monitor.py*' of the downloaded library for efficient access to the parameters. The authors have introduced two variables, namely, '*self.print_bpm*' and '*self.print_spo2*' with an initial value as zero inside the class '*HeartRateMonitor (object)*'. These variables are updated later in the same class. The rest of the library files specific to MAX30102 are kept intact. To avoid confusion in the readers mind, authors recommend to adopt the below code.

```

# heartrate_monitor.py
from max30102 import MAX30102
import hrcalc
import threading
import time
import numpy as np

class HeartRateMonitor(object):
    """
    A class that encapsulates the max30102 device into a thread
    """
    LOOP_TIME = 0.01

    def __init__(self, print_raw=False, print_result=False):
        self.bpm = 0
        self.print_spo2 = 0
        self.print_bpm = 0
        if print_raw is True:
            print('IR, Red')
        self.print_raw = print_raw
        self.print_result = print_result

    def run_sensor(self):
        sensor = MAX30102()
        ir_data = []
        red_data = []
        bpms = []

        # run until told to stop
        while not self._thread.stopped:
            # check if any data is available
            num_bytes = sensor.get_data_present()
            if num_bytes > 0:
                # grab all the data and stash it into arrays
                while num_bytes > 0:
                    red, ir = sensor.read_fifo()
                    num_bytes -= 1
                    ir_data.append(ir)
                    red_data.append(red)
                    if self.print_raw:
                        print("{0}, {1}".format(ir, red))

                while len(ir_data) > 100:
                    ir_data.pop(0)
                    red_data.pop(0)

                if len(ir_data) == 100:
                    bpm, valid_bpm, spo2, valid_spo2 = hrcalc.calc_hr_and_spo2(ir_data,
                                                                              red_data)

                    if valid_bpm:
                        bpms.append(bpm)
                        while len(bpms) > 4:
                            bpms.pop(0)

```

```

        self.bpm = np.mean(bpm)
        if (np.mean(ir_data) < 50000 and np.mean(red_data) < 50000):
            self.bpm = 0
            if self.print_result:
                print("Finger not detected")
        if self.print_result:
            print("BPM: {0}, SpO2: {1}".format(self.bpm, spo2))
            self.print_spo2 = spo2
            self.print_bpm = self.bpm

        time.sleep(self.LOOP_TIME)

    sensor.shutdown()

    def start_sensor(self):
        self._thread = threading.Thread(target=self.run_sensor)
        self._thread.stopped = False
        self._thread.start()

    def stop_sensor(self, timeout=2.0):
        self._thread.stopped = True
        self.bpm = 0
        self._thread.join(timeout)

```

Step 3: Configuration of Library for MLX90614.

Download the library from the link: <https://pypi.org/project/PyMLX90614/#files>. Before using the library, it must be extracted to the same folder which was created earlier. There are no changes needed in this library.

Step 4: Configuration of ThingSpeak Channel

To create ThingSpeak channel, one has to first sign up on ThingSpeak (<https://thingspeak.com/>). In case one has an account on ThingSpeak just sign in using your id and password. For signup, fill your details, then verify with the received email and proceed. After this, click on 'New Channel' button which will subsequently ask for the 'Name and Description' of the data you want to upload on this channel as shown in Fig. 5.6.

In this application Pulse Rate, Oxygen Saturation, and Body Temperature are sent to ThingSpeak cloud. Hence, the authors have named the channel as "Pulse, Oxygen Saturation, and Body Temperature".

More than one field of data can be activated by checking the box next to the 'Field' option as shown in Fig. 5.6. The authors have created three fields, namely, BPM, SpO2, and Temp. After this, click on 'save channel' button to save the details.

ThingSpeak Channels Apps Devices Support Commercial Use How to Buy

New Channel

Name Pulse, Oxygen Saturation, Body Temperature

Description Updates the various parameters

Field 1 Pulse ☒

Field 2 Oxygen Saturation ☒

Field 3 Body Temperature ☒

Field 4 ☐

Field 5 ☐

Field 6 ☐

Field 7 ☐

Field 8 ☐

Metadata

Tags
(Tags are comma separated)

Link to External Site

Link to GitHub

Elevation

Show Channel Location ☐

Latitude

Longitude

Show Video ☐

☒ YouTube ☐ Vimeo

Video URL

Show Status ☐

Save Channel

Help

Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

Channel Settings

- Percentage complete:** Calculated based on data entered into the various fields of a channel. Enter the name, description, location, URL, video, and tags to complete your channel.
- Channel Name:** Enter a unique name for the ThingSpeak channel.
- Description:** Enter a description of the ThingSpeak channel.
- Field:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
- Metadata:** Enter information about channel data, including JSON, XML, or CSV data.
- Tags:** Enter keywords that identify the channel. Separate tags with commas.
- Link to External Site:** If you have a website that contains information about your ThingSpeak channel, specify the URL.
- Show Channel Location:**
 - Latitude:** Specify the latitude position in decimal degrees. For example, the latitude of the city of London is 51.5072.
 - Longitude:** Specify the longitude position in decimal degrees. For example, the longitude of the city of London is -0.1275.
 - Elevation:** Specify the elevation position meters. For example, the elevation of the city of London is 35.052.
- Video URL:** If you have a YouTube™ or Vimeo® video that displays your channel information, specify the full path of the video URL.
- Link to GitHub:** If you store your ThingSpeak code on GitHub®, specify the GitHub repository URL.

Using the Channel

You can get data into a channel from a device, website, or another ThingSpeak channel. You can then visualize data and transform it using ThingSpeak Apps.

See [Get Started with ThingSpeak™](#) for an example of measuring dew point from a weather station that acquires data from an Arduino® device.

[Learn More](#)

Fig. 5.6 Name and description of channel for Patient Monitoring system

Obtain the API Key

To send data to ThingSpeak, a unique API key is required, which is used in the main code to upload body parameters to ThingSpeak server. Navigate to 'API Keys' header under the newly created above channel to get a unique API key as shown in Fig. 5.7.

Now copy 'Write API Key' to use it in the main code. To set up the Python environment for sending data to ThingSpeak, one must install libraries using the below commands.

```
sudo apt-get install httpplib
sudo apt-get install urllib
```

The screenshot shows the ThingSpeak web interface. At the top, there's a navigation bar with 'Channels', 'Apps', 'Devices', and 'Support'. The main header displays the channel name 'Pulse, Oxygen Saturation, Body Temperature', its ID '1578444', author 'marlonseq', and access level 'Private'. Below this, there are tabs for 'Private View', 'Public View', 'Channel Settings', 'Sharing', 'API Keys' (which is active), and 'Data Import / Export'. The 'API Keys' section is divided into three parts: 'Write API Key' with a key input field containing '2TN5A7GW7ZYDBAL1' and a 'Generate New Write API Key' button; 'Read API Keys' with a key input field containing 'XIMKSQF1CI73LYK0', a 'Note' input field, 'Save Note' and 'Delete API Key' buttons, and an 'Add New Read API Key' button; and a 'Help' section with instructions on using API keys. To the right of the 'Help' section is an 'API Requests' section showing three example GET requests for writing and reading data from the channel.

Fig. 5.7 API Key for Patient Monitoring system

After completing these steps the channel is ready to send the body parameters. Fig. 5.8 gives the flow chart of Patient Monitoring System with Cloud.

Step 4:

Create a file 'main.py' in the project folder with the below code.

```
# Imports required for MAX30102 sensor
from heartrate_monitor import HeartRateMonitor
import time
import argparse

# Imports required for MLX90614 sensor
import smbus
from mlx90614 import MLX90614

# Imports required for Thingspeak cloud update
import os
import http.client
import urllib

key = "2TN5A7GW7ZYDBAL1" # Put your Thingspeakcloud API Key here
```

```

# MAX30102 sensor update section

parser = argparse.ArgumentParser(description="Read and print data from MAX30102")
parser.add_argument("-r", "--raw", action="store_true",
                    help="print raw data instead of calculation result")
parser.add_argument("-t", "--time", type=int, default=30,
                    help="duration in seconds to read from sensor, default 30")
args = parser.parse_args()

print('sensor starting...')
hrm = HeartRateMonitor(print_raw=args.raw, print_result=(not args.raw))
hrm.start_sensor()
try:
    time.sleep(args.time)
except KeyboardInterrupt:
    print('keyboard interrupt detected, exiting...')

hrm.stop_sensor()
print('sensor stoped!')
print(hrm.print_bpm,hrm.print_spo2)

# MLX90614 sensor update section

bus1 = smbus.SMBus(1)
sensor = MLX90614(bus1, address=0x5a)
print ("bodyTemperature :", sensor.get_object_1())
body_temp = sensor.get_object_1()
bus1.close()

# Thingspeak cloud upload section

params = urllib.parse.urlencode({'field1': hrm.print_bpm, 'field2': hrm.print_spo2, 'field3': body_temp, 'key':key })

headers = {"Content-type": "application/x-www-form-urlencoded","Accept": "text/plain"}
conn = http.client.HTTPConnection("api.thingspeak.com:80")
conn.request("POST", "/update", params, headers)
response = conn.getresponse()

print(response.status,response.reason)
data = response.read()
conn.close()

```

In the Heart Rate and SpO2 section of the code, authors instantiate an object of the class *HeartRateMonitor* found in *heartrate_monitor.py*. The thread is initiated and terminated by calling *start_sensor* and *stop_sensor*, respectively. When the thread is active one can access *bpm* to get beats per minute (BPM). The user has to wait for a few seconds to get a reliable BPM value. Also, make sure that there is minimum movement as the sensor is very sensitive to it. The program is designed in such a way that a keyboard interrupt (CNTRL + C) can stop the sensor and transit to the next section of the code. The last reading for Pulse rate and Oxygen saturation is written into *hrm.print_bpm* and *hrm.print_spo2* variables, respectively, which are then sent to ThingSpeak Cloud.

Step 5: To run the above *main.py* use the below command.

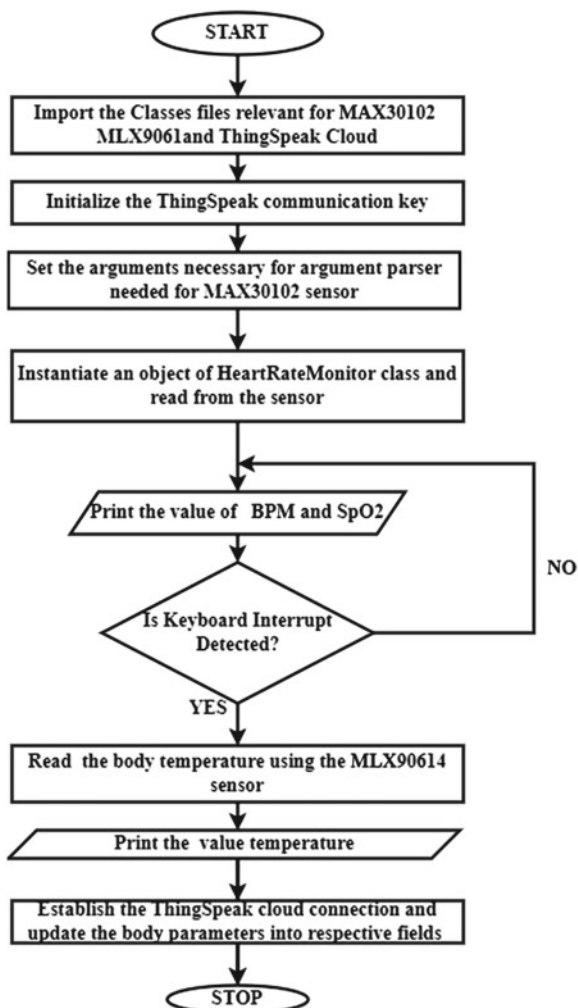
```
python3 main.py
```

The output after execution is shown on the terminal as in Fig. 5.9 and also the data visualization on the ThingSpeak server is shown in Fig. 5.10.

5.6.2 Home security with FoG

Certainly, the advent of the IoT and FoG computing paradigm has made it available for user to efficiently approach the problem of Home security. The authors have leveraged

Fig. 5.8 Flowchart for Patient Monitoring system with cloud

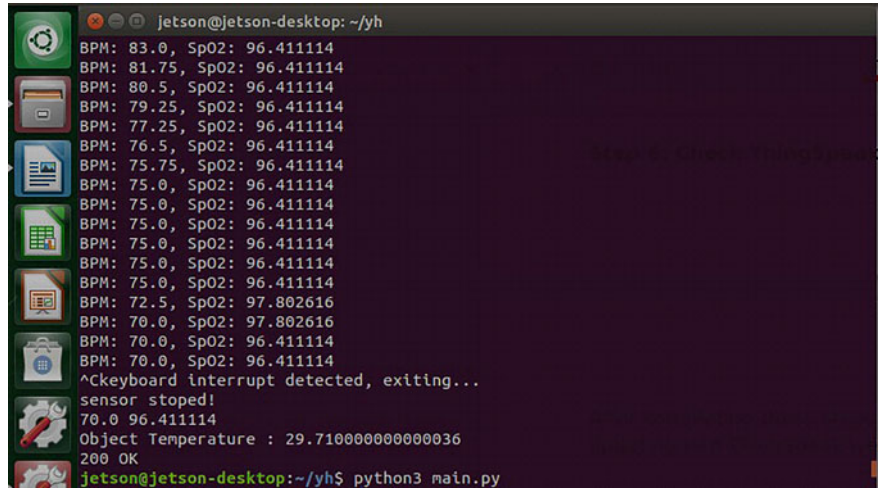


various security hardware such as PIR sensors, Surveillance cameras, Smart locks, etc. to safeguard property and human life from potential life threatening situations. In this section, authors have implemented Home Surveillance, Home Safety Lock, and Fire alert system. The programming of the IoT nodes is done by Thonny IDE using MicroPython programming as detailed in below steps.

Step 1: Download the Thonny IDE for windows from the link: <https://thonny.org>

Step 2: Download ESP 32 Firmware from the link <https://micropython.org/download/esp32/>

Step 3: Install appropriate MicroPython Firmware on ESP32:



```
jetson@jetson-desktop: ~/yh
BPM: 83.0, SpO2: 96.411114
BPM: 81.75, SpO2: 96.411114
BPM: 80.5, SpO2: 96.411114
BPM: 79.25, SpO2: 96.411114
BPM: 77.25, SpO2: 96.411114
BPM: 76.5, SpO2: 96.411114
BPM: 75.75, SpO2: 96.411114
BPM: 75.0, SpO2: 96.411114
BPM: 75.0, SpO2: 96.411114
BPM: 75.0, SpO2: 96.411114
BPM: 75.0, SpO2: 96.411114
BPM: 75.0, SpO2: 96.411114
BPM: 72.5, SpO2: 97.802616
BPM: 70.0, SpO2: 97.802616
BPM: 70.0, SpO2: 96.411114
BPM: 70.0, SpO2: 96.411114
^Ckeyboard interrupt detected, exiting...
sensor stopped!
70.0 96.411114
Object Temperature : 29.710000000000036
200 OK
jetson@jetson-desktop:~/yh$ python3 main.py
```

Fig. 5.9 Health parameters on terminal

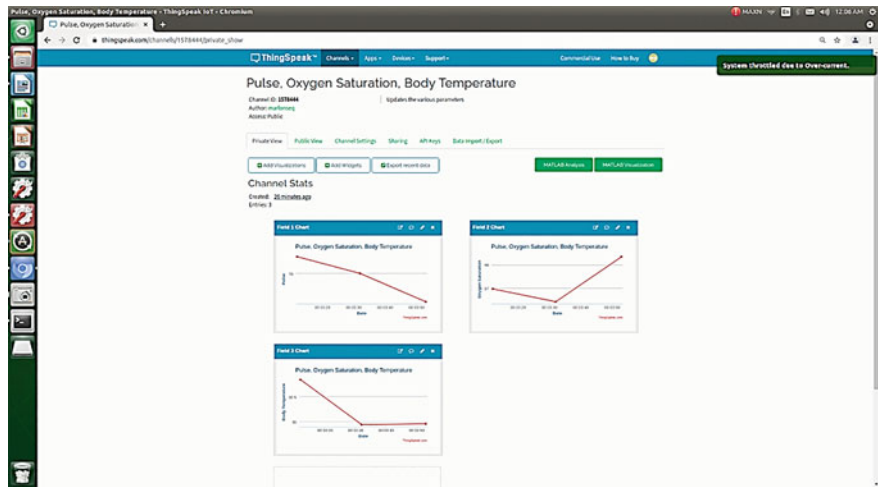


Fig. 5.10 Health parameters on ThingSpeak cloud

Open Thonny IDE → Tools → options → interpreter → select MicroPython (ESP32) → select COM port (Make sure the virtual com port drivers are installed) → click on “install and update firmware” → select port and firmware path (downloaded MicroPython Firmware path) → then click on 'install'.

Step 4: MicroPython interpreter selection:

Goto Tools → options → click on interpreter tab and select drop down for Micropython (ESP 32). Also select the proper COM port for communication.

5.6.2.1 Home Surveillance

Home Surveillance system consists of IoT device ESP32-CAM having PIR sensor interfaced to it. IoT device communicates with Jetson Nano SBC which is configured as FoG node as shown in Fig. 5.11. Intruder activity is sensed by a PIR sensor and an image of the activity is captured by a camera interfaced with IoT device. The captured image is sent over Wi-Fi using MQTT protocol to the Jetson Nano SBC. The Jetson Nano SBC(FoG node) stores the received image. The Jetson upon reception of the image will intimate the house owner with a prompt WhatsApp message to his mobile phone using the Twilio cloud python package. The FoG node will also upload the same image to the Google Drive cloud so that the house owner can monitor the house in real time. The detailed conceptual and pin diagram are shown in Fig. 5.11. Table 5.2 depicts the pin connection of PIR sensor to ESP32 CAM.

IoT Device (ESP32-CAM)

The ESP32-CAM is a fully-featured microcontroller that has an integrated video camera and microSD card socket. It is easy to use, and perfect for various IoT

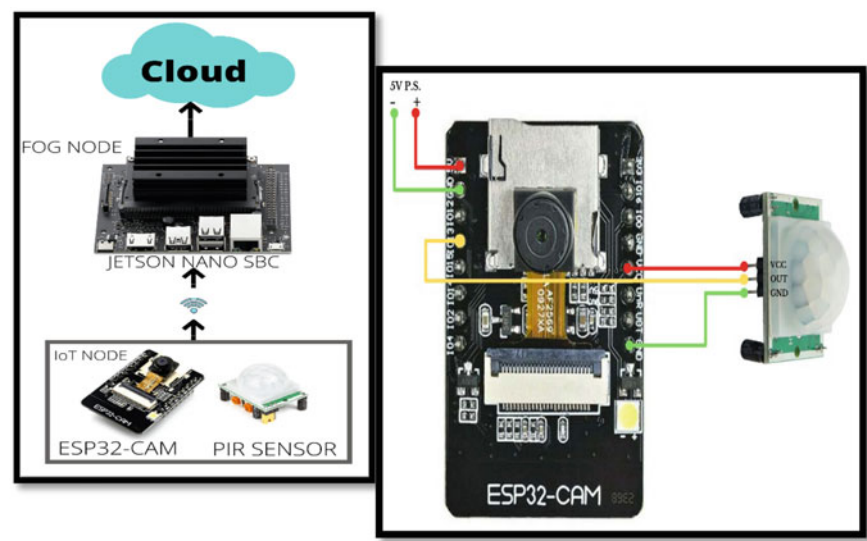


Fig. 5.11 Conceptual and pin diagram of home surveillance system

Table 5.2 Pin connection for Home Surveillance

ESP32-CAM	PIR sensor
VCC	VCC
GND	GND
IO13	OUT

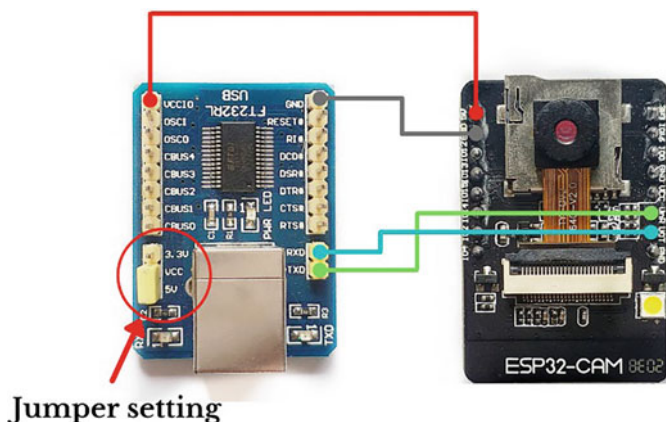


Fig. 5.12 ESP-32 CAM programming through FTDI 232

applications requiring a camera with advanced functions like image tracking and recognition.

Here, 2 megapixel camera module OV2640 is used, which gives outputs in various formats such as YUV422, YUV420, RGB565, RGB555, and 8-bit compressed data. This ESP32-CAM doesn't have a USB port to program, hence one has to use an FTDI adapter.

The IoT device is interfaced with HC-SR 501 (PIR sensor). The PIR sensor is used to detect animal/human movement. This pyroelectric PIR sensor detects motion based on emitted infrared radiation.

Programming the ESP32-CAM:

It is important to note that the FTDI adapter must be set for a 5V VCC output with proper jumper settings, as we are powering the ESP32-CAM using the 5V as shown in Fig. 5.12.

To upload the firmware, one has to short GPIO 0 pin to Ground on ESP32-CAM. This connection is required only for firmware upload to ESP32-CAM. For switching to programming mode, remove the short between GPIO 0 and ground.

The ESP32-CAM firmware required for this application can be downloaded from the link <https://github.com/leomariva/micropython-camera-driver/tree/master/firmware>. The firmware upload and programming for ESP32-CAM are done using MicroPython in the Thonny IDE.

In order to configure ESP32-CAM for MQTT protocol, one requires MicroPython MQTT client (umqtt) and camera package which can be downloaded using the steps outlined in Sect. 3.2 of Chapter 3. Now the ESP32-CAM is ready to capture the image and implement the MQTT protocol. Flowchart for configuring ESP32-CAM module for Home security is shown in Fig. 5.13.

Create a 'main.py' file in Thonny IDE and enter the below code in it. Then click the 'run' button to execute the code. Make sure before running the code, the Wi-Fi

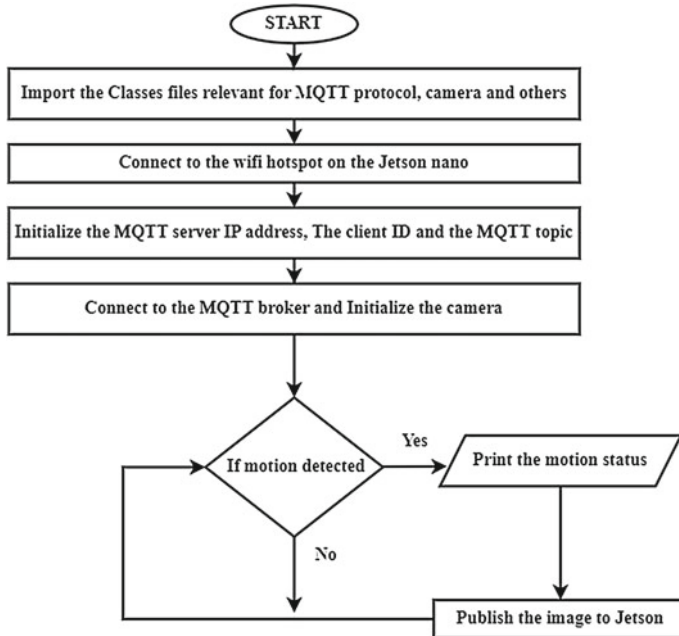


Fig. 5.13 Flowchart for configuring ESP32s-CAM module for Home security

hotspot created on FoG node is active. Here ESP32-CAM is the client which will publish the captured image to MQTT topic named as 'Image'.

```

from machine import Pin          #importing classes
from umqtt.simple import MQTTClient
import time
import os
import camera

# make sure wifi hotspot is enabled on jetson nano SBC(FoG Node)
import network
station = network.WLAN(network.STA_IF)
station.active(True)
station.connect("jetson-desktop", "11111111")

SERVER = '10.42.0.1' # MQTT Server Address (IP address of jetson hotspot)
CLIENT_ID = 'esp32-camera'
TOPIC = b'Image'

# Connect to MQTT broker
c = MQTTClient(CLIENT_ID, SERVER)
c.connect()

camera.init(0, format=camera.JPEG)

```

```

Motion_status = False    #Global variable to hold the state of motion sensor

def handle_interrupt(Pin):    #defining interrupt handling function
    global Motion_status
    Motion_status = True

PIR_Interrupt=Pin(13,Pin.IN)    # setting GPIO13 PIR_Interrupt as input

#Attach external interrupt to GPIO13 and rising edge as an external event source
PIR_Interrupt.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt)

while True:
    if Motion_status:
        print('Motion is detected!')
        buf = camera.capture()
        c.publish(TOPIC, buf)
        time.sleep_ms(100)
        Motion_status = False

```

Configuring the FoG Node

In order to use the Jetson Nano SBC as the FoG computing node, one need to enable the Wi-Fi hotspot on the Jetson Nano which will enable the image transfer using MQTT protocol.

Establishing MQTT on FoG:

MQTT is a lightweight publish-subscribe network protocol that transports messages between devices. The MQTT protocol defines two types of network entities: a broker and clients. An MQTT broker is a server responsible to receive and route all messages from the clients to the appropriate destination. An MQTT client is any device that connects to an MQTT broker over a network.

For MQTT on the FoG Node, Mosquitto library package is required to be installed. This can be done by running the below commands on the Jetson terminal.

```
sudo apt-get install mosquitto mosquitto-clients
```

One also requires the '*paho-mqtt*' library , which can be installed using the below command.

```
sudo apt-get install python3-pip
sudo pip3 install paho-mqtt
```

The Jetson FoG node is now ready for MQTT protocol communication.

Enabling Wi-Fi Hotspot on FoG node:

To enable the hotspot on the FoG node, select the settings menu on Jetson user interface. Under setting menu → Select 'Wi-Fi' → on the right-hand side, hover over the three horizontal lines (☰) → Turn on Wi-Fi hotspot.

It is not possible to access the Internet over wireless when hotspot is active. Hence a wired connectivity is provided for the Jetson Nano SBC to connect to the Internet for uploading the image to Google Drive and sending a message using Twilio cloud.

Configuring the FoG Node to Communicate with Cloud

In this application, authors have used two cloud platforms, namely, Twilio and Google Drive cloud. The Twilio communication platform is used to send the WhatsApp message indicating that there is an intruder activity. At the same time, the surveillance image is also uploaded to the Google Drive cloud. The configuration details of both these platforms are given below.

Twilio Configuration:

Next, one needs to prepare the Jetson Nano SBC (FoG node) to send the WhatsApp message using Twilio cloud communication platform. One has to create the Twilio account by visiting the website (<https://www.twilio.com/>)

Click on the signup option, and fill the respective user details. Once the account is created the verification of registered email and phone number is done.

Once the verification steps are done, one needs to specify the Twilio features required. Select the features using dropdown and radio buttons as shown in Fig. 5.14. At last click on button 'Get started with Twilio' which will take user to the console, wherein user needs to agree to activate the sandbox and click confirm.

Under the 'Manage account' navigate to 'general settings'. Next copy the Account SID and the Auth token which are highlighted in Fig. 5.15. These are needed in python code to send a WhatsApp message using the Twilio cloud.

Next step is to send a WhatsApp message with content "'join every-plain' to the phone number "+14155238886" (same number must be entered) from the registered phone number which will be used to receive the notification.

Setting up Jetson Nano SBC (FoG node) for Twilio

Before using the python code to receive and send WhatsApp messages we need to install the Twilio package on Jetson Nano SBC using the following command:

```
sudo pip install twilio
```

Now the system is ready to send and receive notifications on WhatsApp via the Twilio cloud platform.

Google Drive Configuration

In this implementation, the surveillance image is stored on to Google Drive. To do this, user need to install the Pydrive Python module using the following command.

```
pip install pydrive
```

To upload the image to Google Drive using python, one need an active google cloud account. User need to get the authentication files for Google Service API, so that the Python code can access Google Drive. To do that, one need to follow the below steps.

Step 1: Create a new project in Google Developer Console by clicking 'CREATE PROJECT' with the proper name of the project. The link for the

Fig. 5.14 Twilio feature selection

Ahoy! We're here, welcome to Twilio!

Tell us a bit about yourself so we can personalize your experience. You will have access to all Twilio products.

• Which Twilio product are you here to use?

WhatsApp

• What do you plan to build with Twilio?

Alerts & Notifications

• How do you want to build with Twilio?

☐ With code
Customize exactly what you want

☒ With minimal code
Build on top of our code samples

☐ With no code at all
Launch a starter app with no code

• What is your preferred coding language?

Python

• Would you like Twilio to host your code?
Host your Twilio app on our secure servers

☒ Yes, host my code on Twilio

☐ No, I want to use my own hosting service

Your billing country is India. [Change](#)

[Get Started with Twilio](#)

© Twilio, Inc. All rights reserved. [Privacy Policy](#) | [Terms of Service](#)

same is <https://console.cloud.google.com/cloud-resource-manager?organizationId=0&authuser=1&supportedpurview=project>.

Step 2: Next step is to Enable APIs and their Services by clicking the 'ENABLE APIS AND SERVICES'. This will bring you to the API library. Then Search 'Google Drive' → Click the 'Google Drive API' icon → Then click 'ENABLE', which will enable Google Drive API service.

Step 3: Creation of credentials: click on 'credentials' and select the options shown in Fig. 5.16 and click 'done'. Next hover over to 'create credentials' and select 'OAuth Client ID' as shown in Fig. 5.17.

Step 4: Next click on the button 'configure consent screen' → Navigate to OAuth consent screen → select 'External' and then click 'Create'.

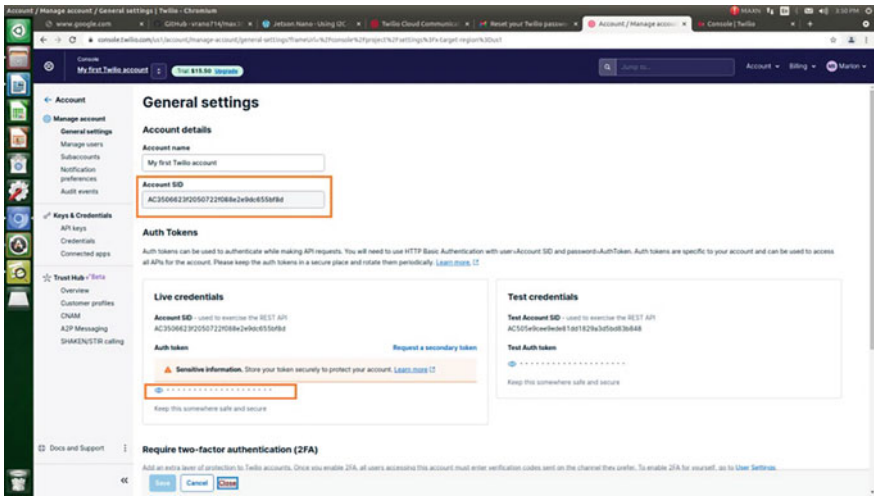


Fig. 5.15 Account SID and Auth token

On the next screen enter the 'app name' and the 'user support email id' as well as the developer contact information (developer email id) and then click 'save' and 'continue'. On the following screen, click on 'save and continue' as no changes are required. Next add a test user's email id and click 'add' → then click 'save and continue'. The next screen displays the summary of the OAuth consent.

Step 5: Next under Credential on left panel click 'Create Credentials' → select OAuth Client ID → Provide the settings as shown in Fig. 5.18 and click 'create'. Now the OAuth Client ID is created.

Step 6: After creating the OAuth client ID → click on 'download' button to get .JSON file. This downloaded JSON file is required for Python code to access Google Drive. After downloading the .JSON file, rename it to 'client_secrets.json' and make sure that it is placed in the same folder where the 'main.py' is present.

Step 7: The Python code for home surveillance running on Jetson Nano SBC (FoG node) requires the user to enter the google account and password. After authentication is completed browser displays 'The authentication flow has completed' and initiates the uploading of the image onto Google Drive.

The surveillance image sent by ESP32-CAM is received by FoG node will be saved in the folder named as 'photos' in the project folder of the FoG node. The same image is also uploaded to Google Drive in its root directory for remote monitoring. The flow chart for configuring Jetson Nano for Home security is given in Fig. 5.19.

Step 8: Create a 'main.py' to enter the below Python code on the Jetson Nano SBC and run it using the below command.

```
python3 main.py
```

Google Cloud Platform Home Surveillance

APIs and services

Create credentials

1 Credential Type

Which API are you using?

Different APIs use different auth platforms and some credentials can be restricted to only call certain APIs.

Select an API *
Google Drive API

What data will you be accessing? *

Different credentials are required to authorise access depending on the type of data that you request. [Learn more](#)

☐ User data
Data belonging to a Google user, like their email address or age. User consent required. This will create an OAuth client.

☒ Application data
Data belonging to your own application, such as your app's Cloud Firestore backend. This will create a service account.

Are you planning to use this API with Compute Engine, Kubernetes Engine, App Engine or Cloud Functions?

Applications running on GCE, GKE, GAE and GCF can use Application Default Credentials and don't require you to create a credential.

☒ Yes, I'm using one or more

☐ No, I'm not using them

NEXT

2 Your credentials

DONE CANCEL

Fig. 5.16 Creation of credentials

After successfully running the code the Twilio message notification is sent to the registered mobile number of the house owner as shown in Fig. 5.20.

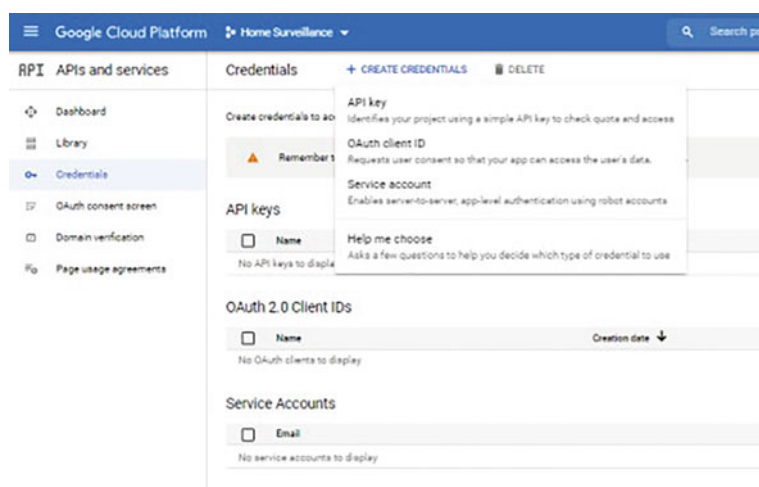


Fig. 5.17 OAuth client ID

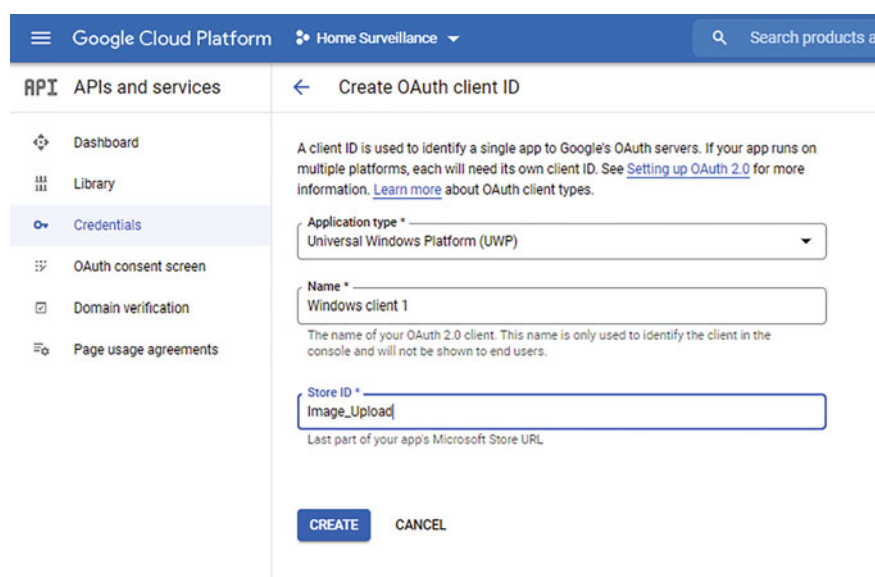


Fig. 5.18 OAuth client ID creation

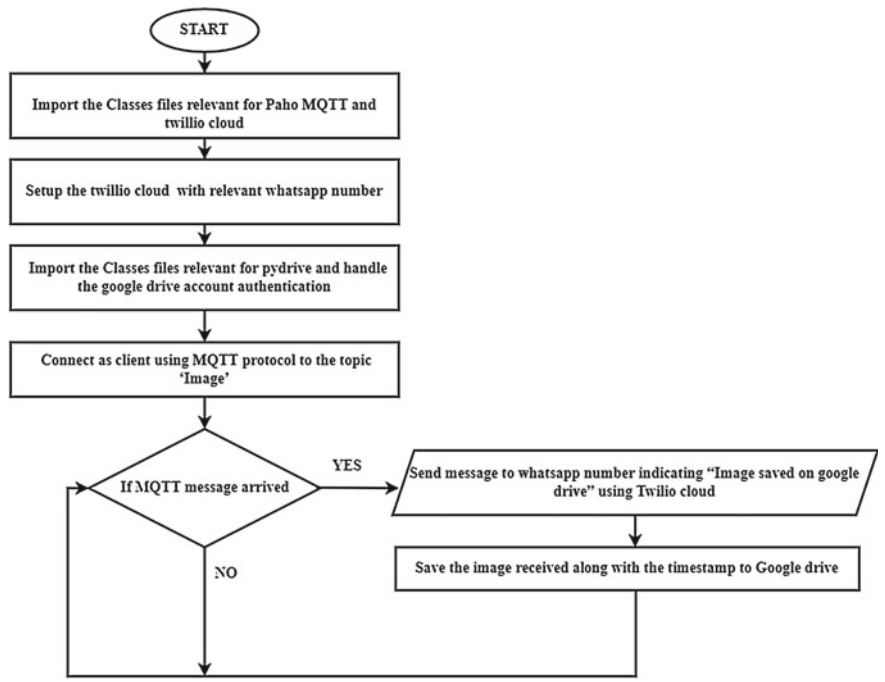


Fig. 5.19 Flow chart for using Twilio cloud and Google Drive from Jetson Nano

Fig. 5.20 Twilio message notification



```

import time
import paho.mqtt.client as mqtt

#twilio imports
import os
from twilio.rest import Client

# setuptwilio

twilio_client = Client('AC230101cb426d251b82479e33a992c149', '1d55483a6ab87ed20eac8531d70650c4');
# this is the Twilio sandbox testing number
from_whatapp_number='whatsapp:+14155238886'

# replace this number with your personal WhatsApp Messaging number
to_whatapp_number='whatsapp:+91705750xxxx'

# setuptwilio ends

# Google drive setup
from pydrive.drive import GoogleDrive
from pydrive.auth import GoogleAuth

# Google drive authentication
gauth = GoogleAuth()

# Creates local webserver to handle authentication
gauth.LocalWebserverAuth()
drive = GoogleDrive(gauth)

# Replace with the path, where the image is present
path = r"/home/jetson/MQTT-M5Camera/photos"

# Google drive setup ends here

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe('Image')

def on_message(client, userdata, msg):
    # generate filename
    timestamp = time.gmtime()
    time_str = '%d%d%d%d%d%d%d%d' % (timestamp[0], timestamp[1], timestamp[2], timestamp[4], timestamp[5], timestamp[6])
    # Create a file with write byte permission
    f = open('photos/'+time_str+'.jpg', "wb")
    f.write(msg.payload)
    f.close()
    print("Image received and saved!")

    # twilio message
    message = twilio_client.messages.create(body='Motion Detected : Image saved on GoogleDrive', from_=from_whatapp_number, to=to_whatapp_number)
    print(message.sid)

    # Google drive image upload
    t = drive.CreateFile({'title': time_str+'.jpg'})
    t.SetContentFile(os.path.join(path, time_str+'.jpg'))
    t.Upload()

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect('localhost', 1883, 60)

client.loop_forever()

```

5.6.2.2 Home Safety Lock

Safety is the most important requirement for any home. More than ever before, a need is felt for a home safety lock to safeguard one's house and the belonging therein. With the advent of IoT, one can give the connectivity to the door lock.

Here, authors have designed a smart home safety lock that communicates with the owner. It also informs the owner of an authentic entry into the house, or if an intruder is trying to enter the house. The solenoid lock is placed with IoT node configured on

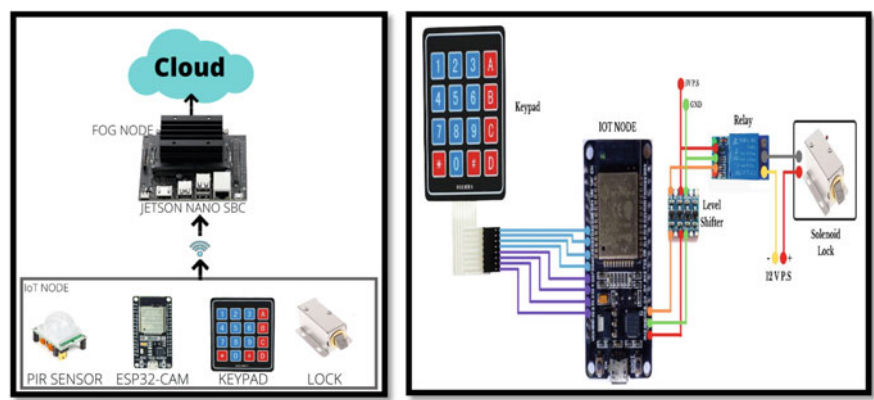


Fig. 5.21 Conceptual and pin diagram for home safety lock

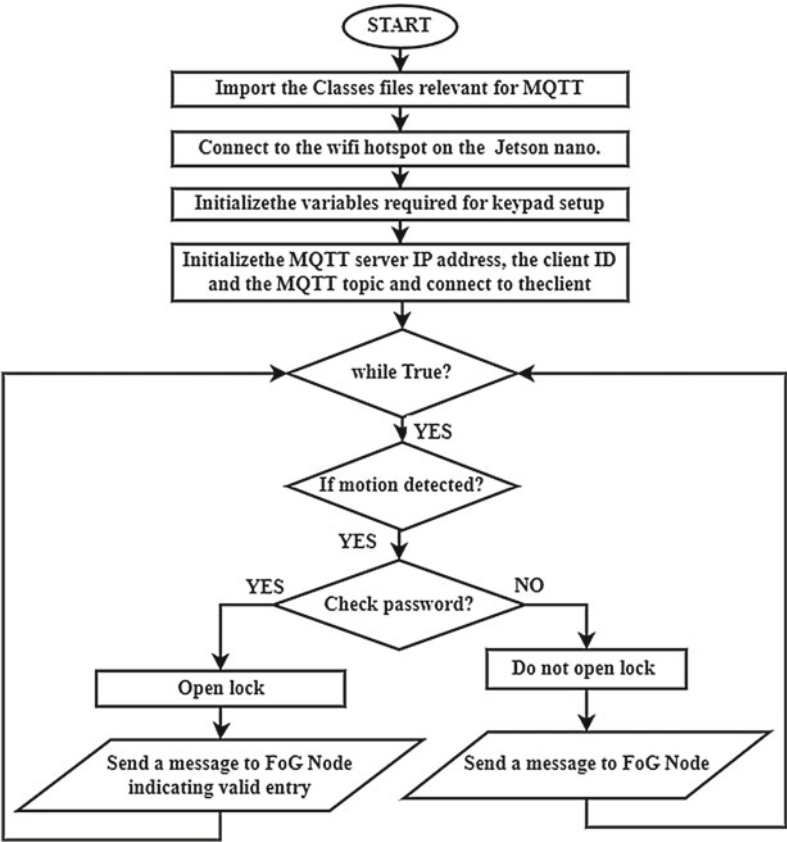


Fig. 5.22 Flow chart for home safety lock on ESP32 Wi-Fi module

Table 5.3 Pin connection for Home Safety lock

ESP3 Wi-Fi module	Keypad	Level shifter input
VCC	–	–
GND	–	–
3V3	–	LV
GND	–	GND
D15	–	LV1
D13	ROW 1	–
D12	ROW 2	–
D14	ROW 3	–
D27	ROW 4	–
D26	COL 1	–
D23	COL 2	–
D33	COL 3	–
D32	COL 4	–
Level shifter output		Relay module
HO		5 V(power supply)
HO1		IN
GND		GND
Relay module	Solenoid lock	12 V Power supply
–	RED PIN	+ PIN(RED)
NO	–	- PIN(BLACK)
COM	GND PIN	–

ESP32 Wi-Fi module. When PIR sensor detects motion, the IoT node prompts the user to enter the password. If the password is valid, only then the lock will open and the user is allowed to enter the house. The status of valid/invalid entry is informed to the Jetson Nano SBC configured as FoG node using MQTT protocol. The FoG node in turn sends an alert message to the house owner using Twilio cloud platform on WhatsApp. The conceptual diagram and pin diagram is shown in Fig. 5.21. Table 5.3 depicts the pin connection for the home safety lock.

To Configuring MQTT on ESP 32 Wi-Fi Module:

To configure, follow the steps already given in Sect. 5.6.2.1. This includes connecting to the hotspot hosted by Jetson Nano SBC.

Here, ESP32 is the client and publishes the sensor data to a topic called 'password_verify'. FoG node, i.e. the Jetson Nano SBC will be the broker and client. A Python MQTT client running on the FoG node will subscribe to the 'password_verify' topic and collect the results. After setting up the MQTT protocol on the IoT node, it is ready for communication. The flow chart for configuring ESP32 for Home Safety Lock is given in Fig. 5.22.

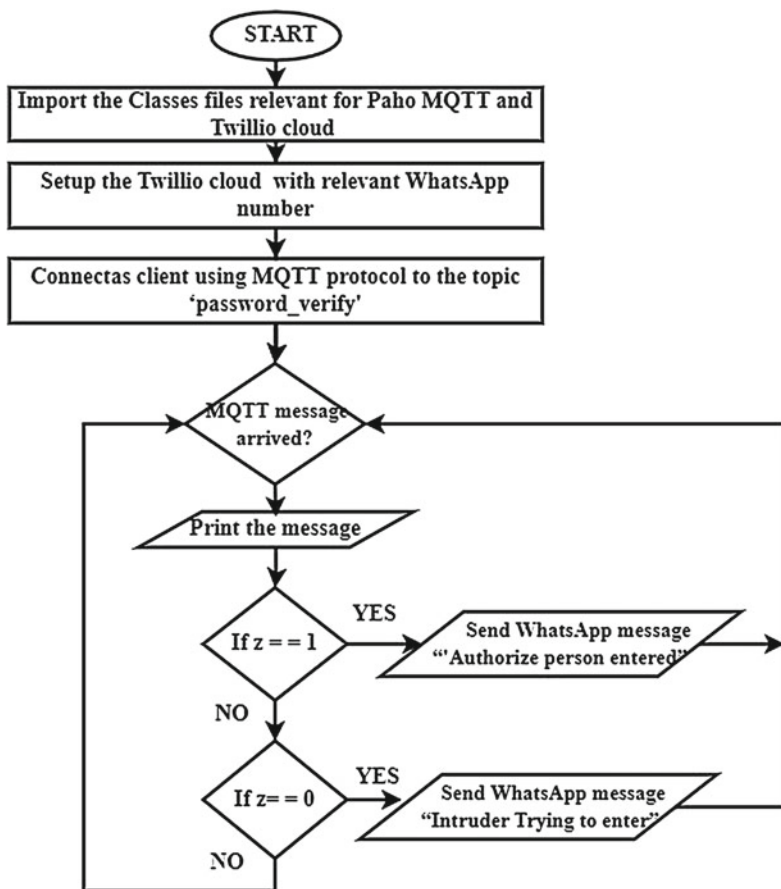


Fig. 5.23 Flow chart for configuring Home Safety Lock using Jetson Nano

Create a 'main.py' file in Thonny IDE and enter Python code. Then click the 'run' button to execute the code.

```

from machine import Pin          #importing classes
from time import sleep          #Import sleep from time class
from umqtt.simple import MQTTClient
import time
import machine

# make sure wifi hotspot is enabled on jetson nano SBC(FoG Node)
import network
station = network.WLAN(network.STA_IF)
station.active(True)
station.connect("jetson-desktop", "11111111")

KEY_UP   = const(0)
KEY_DOWN = const(1)

keys = [['1', '2', '3', 'A'], ['4', '5', '6', 'B'], ['7', '8', '9', 'C'], ['*', '0', '#', 'D']]

# Pin names
rows = [13,12,14,27]
cols = [26,25,33,32]

# set pins for rows as outputs
row_pins = [Pin(pin_name, mode=Pin.OUT) for pin_name in rows]

# set pins for cols as inputs
col_pins = [Pin(pin_name, mode=Pin.IN, pull=Pin.PULL_DOWN) for pin_name in cols]

def init():
    for row in range(0,4):
        for col in range(0,4):
            row_pins[row].value(0)

def scan(row, col):
    """ scan the keypad """
    # set the current column to high
    row_pins[row].value(1)
    key = None

    # check for keypressed events
    if col_pins[col].value() == KEY_DOWN:
        key = KEY_DOWN

    if col_pins[col].value() == KEY_UP:
        key = KEY_UP

    row_pins[row].value(0)
    # return the key state
    return key

SERVER = '10.42.0.1' # MQTT Server Address (Change to the IP address of your Pi)
CLIENT_ID = 'ESP32'

```

```

TOPIC = b'password_verify'

client = MQTTClient(CLIENT_ID, SERVER)
client.connect() # Connect to MQTT broker

Motion_status = False #Global variable to hold the state of motion sensor

def handle_interrupt(Pin): #defining interrupt handling function
    global Motion_status
    Motion_status = True

lock=Pin(15,Pin.OUT) #setting GPIO14 led as output
PIR_Interrupt=Pin(34,Pin.IN) # setting GPIO13 PIR_Interrupt as input

#Attach external interrupt to GPIO13 and rising edge as an external event source
PIR_Interrupt.irq(trigger=Pin.IRQ_RISING, handler=handle_interrupt)

password = ['1','0','0','0']
input_pass = ['0','0','0','0']

def password_verify(): #defining interrupt handling function
    print('Please enter the password')
    break_loop = 0
    i = 0
    while i<4:
        while break_loop == 0:
            for row in range(4):
                for col in range(4):
                    key = scan(row, col)
                    if key == KEY_DOWN:
                        print("Key Pressed", keys[row][col])
                        time.sleep(0.5)
                        last_key_press = keys[row][col]
                        break_loop = 1
            break_loop = 0
            input_pass[i]=last_key_press
            i=i+1

# set all the columns to low
init()
lock.value(1)# initially locked
while True:
    if Motion_status:
        print('Motion is detected!')
        password_verify()

    print(input_pass)

    if input_pass == password:
        print('yes')
        lock.value(0)

```



```

t = 1.0    # 1.0 indicates authentication done, lock opened
if isinstance(t, float): # Confirm sensor results are numeric
    msg = (b'{0:3.1f}'.format(t))
    client.publish(TOPIC, msg) # Publish data to MQTT
    print(msg)
else:
    print('Invalid sensor readings.')

sleep(10)
lock.value(1)
Motion_status = False
else:
    print('No')
    lock.value(1) # dont open door
    t = 0.0      # 0.0 indicates authentication not done
    if isinstance(t, float): # Confirm sensor results are numeric
        msg = (b'{0:3.1f}'.format(t))
        client.publish(TOPIC, msg) # Publish sensor data to MQTT
        print(msg)
    else:
        print('Invalid sensor readings.')

sleep(5)
Motion_status = False

```

Upon a valid password entry, the IoT node will unlock the lock and at the same time send an MQTT message '1' to the Jetson Nano SBC (FoG Node) indicating the same. If the intruder tries to enter the invalid password, the IoT node will send an MQTT message as '0' to indicate that it is an invalid entry and will not open the lock.

To configure the Jetson Nano (FoG Node) for MQTT and Twilio cloud follow the steps given in Sect. 5.6.2.1. The flow chart for configuring Jetson Nano for Home Safety Lock is given in Fig. 5.23.

Create a 'main.py' and enter Python code and run using the below command.

```
python3 main.py
```

After successfully running the code, the Twilio message notification is sent to the registered WhatsApp mobile number of the house owner.

```

import paho.mqtt.client as mqtt
global z

#twilio imports
import os
from twilio.rest import Client

# setup twilio

twilio_client = Client('AC230101cb426d251b82479e33a992c149', '1d55483a6ab87ed20eac8531d70650c4')
# this is the Twilio sandbox testing number
from_whatapp_number='whatsapp:+14155238886'

# replace this number with your personal WhatsApp Messaging number
to_whatapp_number= 'whatsapp:+91705750xxxx'

# setup twilio ends

```

```

# Callback fires when conected to MQTT broker.
def on_connect(client, userdata, flags, rc):
    print('Connected with result code {}'.format(rc))
    # Subscribe (or renew if reconnect).
    client.subscribe('password_verify')
# Callback fires when a published message is received.
def on_message(client, userdata, msg):
    print(msg)
    z = [float(x) for x in msg.payload.decode("utf-8").split(',')]
    if z[0] == 1.0:
        # twilio message
        message = twilio_client.messages.create(body='Authorized person
                                                entered',from_=from_whatsapp_number,
                                                to=to_whatsapp_number)

        print(message.sid)
    if z[0] == 0.0:
        message = twilio_client.messages.create(body='Intruder Trying to entered',
                                                from_=from_whatsapp_number,
                                                to=to_whatsapp_number)

        print(message.sid)

client = mqtt.Client()
client.on_connect = on_connect # Specify on_connect callback
client.on_message = on_message # Specify on_message callback
client.connect('localhost', 1883, 60) # Connect to MQTT broker (also running on Jetson).

# Processes MQTT network traffic, callbacks and reconnections (Blocking)
client.loop_forever()

```

Once the code is executed on Jetson SBC (FoG node), it will receive a message from the IoT node using MQTT protocol. If the message received indicates an authentic entry, it will inform the owner of the same. If there is invalid entry, it indicates that the intruder is trying to enter.

5.6.2.3 Fire Alert System

Fire is one of the most dangerous threats that home and business owners need to take into account. In the unfortunate case of a fire, there is a risk of losing the majority of your belongings. A lack of safety precautions may result in material and financial losses, as well as death. As a result, having a fire alert system installed is a great way to keep premises safe. This method provides early alert of the hazard, allowing adequate time to evacuate the premises and contact authorities before the fire spreads out of control.

Here, the authors have designed a Fire Alert system to mitigate such situation. In this implementation, authors have used the ESP32-CAM module configured as IoT device. MQ-4 Sensor Module is interfaced to detect the smoke in the air. The MQ-4 can detect smoke concentrations anywhere from 200 to 10,000 ppm. It provides analog as well as digital output corresponding to the concentration of the gases in the air.

When the IoT node detects a fire, it will inform the Jetson Nano SBC (FoG node) using Wi-Fi over MQTT protocol. On reception of a message from the IoT node, the FoG node will inform the house owner by a WhatsApp message using the Twilio Cloud Platform, that there is a fire in the house for preventive measures. The

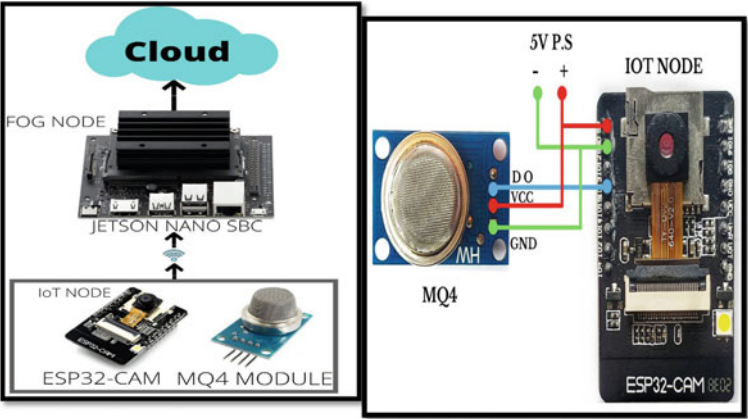


Fig. 5.24 Conceptual and pin diagram for Fire Alert system

Table 5.4 Pin connection for Fire Alert system

ESP32-CAM	MQ4 module
VCC	VCC
GND	GND
IO13	D O

conceptual diagram and pin diagram for Fire Alert system are shown in Fig. 5.24. Table 5.4 depicts the pin connection for the Fire Alert system.

To Configuring MQTT on ESP32-CAM

To configure ESP32-CAM module for MQTT follow the steps already given in Sect. 5.6.2.1. Here, ESP32 is the client and publishes the sensor data to a topic called 'smoke_alert'. FoG node, i.e. the Jetson Nano SBC will be the broker and client. A Python MQTT client running on the FoG node will subscribe to the 'smoke_alert' topic and collect the results. After setting up the MQTT protocol on the IoT node it is ready for communication. The flow chart for the above process is given in Fig. 5.25.

Create a 'main.py' file in Thonny IDE and enter the Python code . Then click the 'run' button to execute the code.

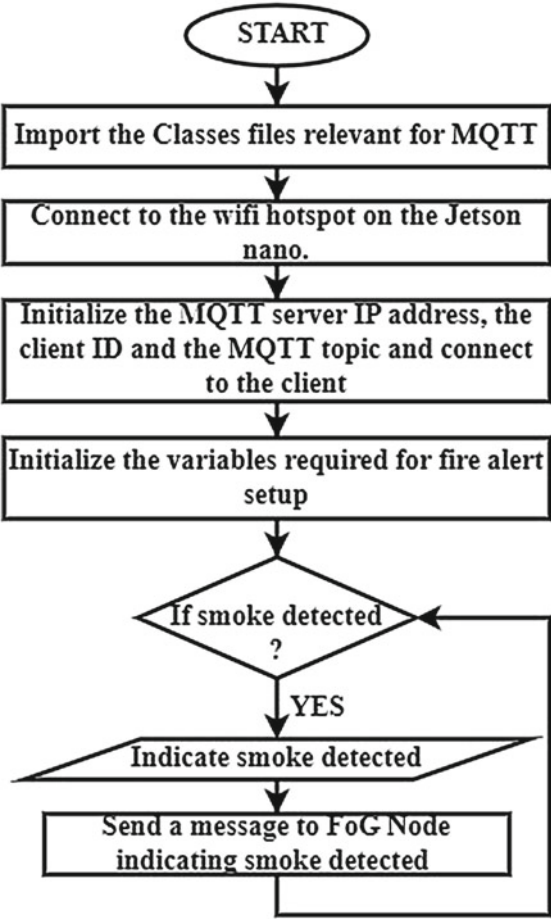


Fig. 5.25 Flow chart for configuring ESP 32 Wi-Fi module for Fire Alert system

```

import network
station = network.WLAN(network.STA_IF)
station.active(True)
station.connect("jetson-desktop", "11111111")

from machine import Pin          #importing classes
from time import sleep          #Import sleep from time class
from umqtt.simple import MQTTClient
import time
import machine

SERVER = '10.42.0.1' # MQTT Server Address (Change to the IP address of your Pi)
CLIENT_ID = 'ESP32'
TOPIC = b'smoke_alert'

client = MQTTClient(CLIENT_ID, SERVER)
client.connect() # Connect to MQTT broker

smoke_status = False # variable to hold the state of motion sensor

MQ4=Pin(13,Pin.IN) # setting GPIO13 as input

while True:
    if MQ4.value() == 0:
        smoke_status = True

        if smoke_status:
            print('smoke is detected!')

            t = 1.0 # 1.0 indicates authentication done
            if isinstance(t, float): # Confirm sensor results are numeric
                msg = (b'{0:3.1f}'.format(t))
                client.publish(TOPIC, msg) # Publish sensor data to MQTT topic
                print(msg)
            else:
                print('Invalid sensor readings.')
                sleep(5)
                smoke_status = False

```

To Configure the Jestson Nano (FoG Node) for MQTT and Twilio follow the steps given in Sect. 5.6.2.1. Now both the nodes are ready for communication. The flow chart for configuring Jetson Nano for the Fire Alert system is given in Fig. 5.26.

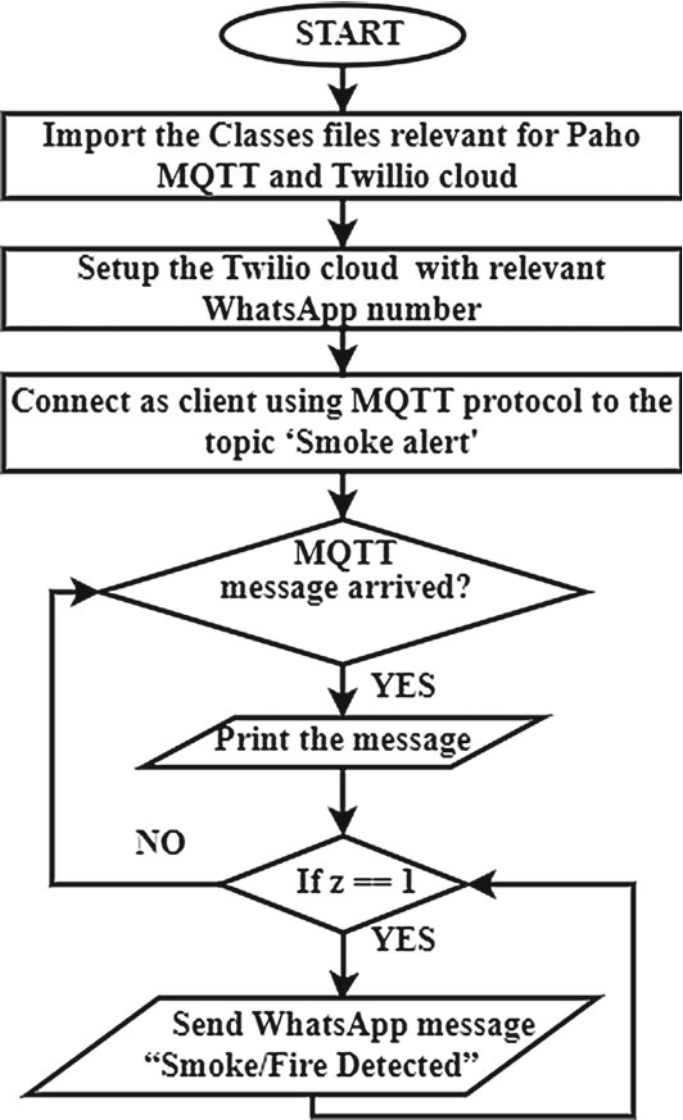


Fig. 5.26 Flow chart for configuring Jetson Nano for Fire Alert system

Next Create a 'main.py' with the below code on the Jetson Nano SBC and run it using the below command

```
python3 main.py
```

```
import paho.mqtt.client as mqtt
global z

#twilio imports
import os
from twilio.rest import Client

# setuptwilio
twilio_client =
Client('AC230101cb426d251b82479e33a992c149','1d55483a6ab87ed20eac8531d70650c4');
# this is the Twilio sandbox testing number
from_whatsapp_number='whatsapp:+14155238886'

# replace this number with your personal WhatsApp Messaging number
to_whatsapp_number='whatsapp:+917057504996'

# setuptwilio ends

# Callback fires when connected to MQTT broker.
def on_connect(client, userdata, flags, rc):
    print('Connected with result code {0}'.format(rc))
    # Subscribe (or renew if reconnect).
    client.subscribe('smoke_alert')

# Callback fires when a published message is received.
def on_message(client, userdata, msg):
    print(msg)
    z = [float(x) for x in msg.payload.decode("utf-8").split(',')]
    if z[0] == 1.0:
        message = twilio_client.messages.create(body='Smoke/Fire
                                                Detected',from_= from_whatsapp_number,
                                                to= to_whatsapp_number)
        print(message.sid)

client = mqtt.Client()
client.on_connect = on_connect# Specify on_connect callback
client.on_message = on_message# Specify on_message callback
client.connect('localhost', 1883, 60) # Connect to MQTT broker

# Processes MQTT network traffic, callbacks and reconnections. (Blocking)
client.loop_forever()
```

After successfully running the code, the Twilio message notification is sent to the registered mobile number of the house owner that there is the fire in the house and necessary action has to be taken.

Conclusion:

NVIDIA® Jetson Nano™ lets you bring incredible new capabilities to millions of small and power-efficient AI systems. It opens new worlds of embedded IoT applications. Jetson Nano is a powerful computer that lets run multiple codes for various applications. In this chapter, the authors have given a brief introduction to FoG and Cloud computing along with its architectures, characteristics, service models, and various models deployment. It also covers the role of FoG and Cloud computing in IoT applications. The authors have provided the detailed implementation steps for the Patient Monitoring system with Cloud, wherein the system monitors Oxygen saturation, Pulse, and Body Temperature. Authors have also implemented Home Security system such as Home Surveillance, Home Safety Lock, and Fire alert system by providing detailed steps. This Home Security system is implemented with Jetson Nano as a FoG node.

Exercise:

- (1) Similar to Sect. 5.6.2.3 (Fire alert system), using the ESP32-CAM module, design a system that reads the temperature from the DHT22 sensor and updates it over the FoG in regular intervals of 30 minutes.
- (2) Modify the system mentioned in Sect. 5.6.1 (Patient Monitoring System with Cloud) to work on Raspberry Pi instead of Jetson Nano.
- (3) Referring to Sect. 5.6.1 (Patient Monitoring System with Cloud) interface a MAX30102 Heart Rate sensor with Jetson Nano to monitor the patient's heart rate continuously. If the heart rate deviates outside the normal range and send an alert message via Twilio Cloud.
- (4) Interface ESP32-CAM module with 2 PIR sensors referring to Sect. 5.6.2.2 (Home Safety Lock). Also interface a light source through relay module. Write a Python program which can make the ESP32-CAM module to take inputs from both the PIR sensors. If both the PIR sensors detect movement, then send the signals indicating the same to Jetson Nano and turn on the light source. Further, send the message "Movement detected, Light On" through Twilio cloud to WhatsApp number.
- (5) Modify the setup in Sect. 5.6.2.1 (Home Surveillance) to capture images in interval of 10 minutes if there is no movement detected by the PIR sensor.
- (6) To enhance the Home Security, interface fingerprint sensor module with system.
- (7) Increase the number of IoT nodes with smoke sensors in Fire Alert system to detect fire in multiple rooms.

References

- Aazam M, Huh EN (2014) Fog computing and smart gateway based communication for cloud of things. In: Proceedings of the 2014 international conference on future internet of things cloud, FiCloud 2014, Barcelona, Spain, 27–29 August 2014, pp 464–470
- Aazam M, Huh EN (2015) Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT. In: Proceedings of international conference on advanced information network applications, AINA 2015, pp 687–694
- Aazam M, Hung PP, Huh E (2014) Smart gateway based communication for cloud of things. In: Proceedings of the 2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing, Singapore, 21–24 April 2014, pp 1–6
- Arasaratnam O (2011) Introduction to cloud computing. In: Halpert B (ed) Auditing cloud computing, a security and privacy guide. Wiley, Hoboken, NJ, pp 1–13
- Atlam HF, Walters RJ, Wills GB (2018) Fog computing and the internet of things: a review. *Big Data Cogn Comput* 2:10. <https://doi.org/10.3390/bdcc2020010>
- Cha H-J, Yang H-K, Song Y-J (2018) A study on the design of fog computing architecture using sensor networks. *Sensors* 18:3633. <https://doi.org/10.3390/s18113633>
- Chandrasekaran K (2015) Essentials of cloud computing. <http://www.crcnetbase.com/isbn/9781482205442>
- Chiang M, Zhang T (2016) Fog and IoT: an overview of research opportunities. *IEEE Internet Things J* 3(6):854–864
- Dastjerdi AV, Gupta H, Calheiros RN, Ghosh SK, Buyya R (2016) Fog computing: principles, architectures, and applications. In: Buyya R, Vahid Dastjerdi A (eds) Internet of things. Morgan Kaufmann, pp 61–75. <https://doi.org/10.1016/B978-0-12-805395-9.00004-6>
- Evans D (2011) The internet of things how the next evolution of the internet is changing everything. https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_041FINAL.pdf. Accessed 13 Nov 2021
- Fog Computing and Its Role in the Internet of Things. <https://readwrite.com/2020/10/30/fog-computing-and-its-role-in-the-internet-of-things/>
- ITCandor (2018) Distribution of cloud platform as a service (PaaS) market revenues worldwide from 2015 to June 2018, by vendor. <https://www.statista.com/statistics/540521/worldwidecloud-platform-revenue-share-by-vendor/>
- Iyer B, Henderson JC (2010) Preparing for the future: understanding the seven capabilities of cloud computing. *MIS Q Exec* 9(2):117–131
- Khan S, Parkinson S, Qin Y (2017) Fog computing security: a review of current applications and security solutions. *J Cloud Comput.* 6(1):19
- Kumar N, DuPree L (2011) Protection and privacy of information assets in the cloud. In: Halpert B (ed) Auditing cloud computing, a security and privacy guide. Wiley, Hoboken, NJ, pp 97–128
- Liu Y, Fieldsend JE, Min G (2017) A framework of fog computing: architecture challenges and optimization. *IEEE Access* 4:1–10
- Marques B, Machado I, Sena A, Castro MC (2017) A communication protocol for fog computing based on network coding applied to wireless sensors. In: Proceedings of the 2017 IEEE International symposium on high performance computer architecture, Vosendorf, Austria, 24–28 February 2017, pp 109–114
- Marston S, Li Z, Bandyopadhyay S, Zhang J, Ghalsasi A (2011) Cloud computing—the business perspective. *Decis Support Syst* 51(1):176–189
- Mell P, Grance T (2011) The NIST definition of cloud computing. National Institute of Standards and Technology, NIST Special Publication (SP), pp 800–145. <https://doi.org/10.6028/NIST.SP.800-145>
- Mukherjee M, Shu L, Wang D (2018) Survey of fog computing: fundamental, network applications, and research challenges. *IEEE Commun Surv Tutor*

- Muntjir M, Rahul M, Alhumyani HA (2017) An analysis of internet of things (IoT): novel architectures, modern applications, security aspects and future scope with latest case studies. *Int J Eng Res Technol* 6:422–447
- Ara R, Rahim MA, Roy S, Prodhan UK (2020) Cloud computing: architecture, services, deployment models, storage, benefits and challenges. *Int J Trend Sci Res Dev* 4(4):837–842
- Schneider S, Sunyaev A (2016) Determinant factors of cloud-sourcing decisions: reflecting on the IT outsourcing literature in the era of cloud computing. *J Inf Technol* 31(1):1–31
- Shi Y, Ding G, Wang H, Roman HE, Lu S (2015) The fog computing service for healthcare. In: *Proceedings of the 2015 2nd international symposium on future information and communication technologies for ubiquitous healthcare (Ubi-HealthTech)*, Beijing, China, 28–30 May 2015, pp 1–5
- Sunyaev A (ed) (2020) “Cloud computing,” in *internet computing: principles of distributed systems and emerging internet-based technologies*. Springer International Publishing, Cham, pp 195–236. https://doi.org/10.1007/978-3-030-34957-8_7
- Taneja M, Davy A (2016) Resource aware placement of data analytics platform in fog computing. *Proc Comput Sci* 97:153–156. <https://doi.org/10.1016/j.procs.2016.08.295>
- Tang B, Chen Z, Hefferman G, Wei T, He H, Yang Q (2015) A hierarchical distributed Fog computing architecture for big data analysis in smart cities. In: *Proceedings of the ASE Big Data & Social Informatics 2015*. ACM, p 28
- Voorsluys W, Broberg J, Buyya R (2011) Introduction to cloud computing. In: Buyya R, Broberg J, Goscinski A (eds) *Cloud computing*. Wiley, Hoboken, NJ, pp 3–42